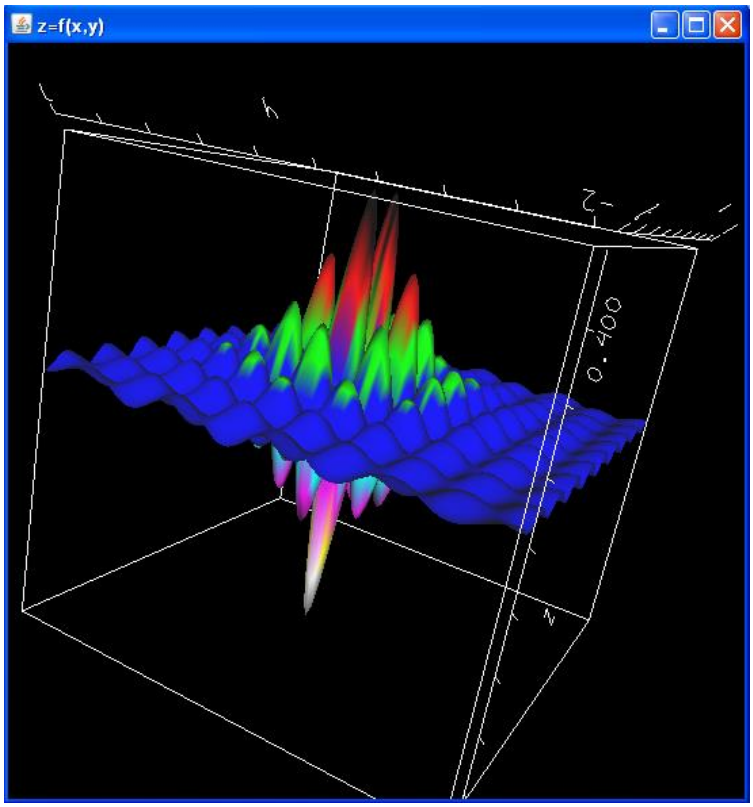
NUMERICAL ANALYSIS



M. Turhan ÇOBAN

EGE UNIVERSITY

School of Engineering

Department of Mechanical Engineering

Bornova, İZMİR, TURKEY

turhan.coban@turhancoban.com

www.turhancoban.com

INDEX

1. **Introductory Concepts**
   - 1.1 Programming Languages
   - 1.2 Taylor series and error terms in the equations
   - 1.3 Series solutions of functions

2. **Finding the roots of non linear equations of one independent variable**
   - 2.1 Bisection Method
   - 2.2 False position method (Regula falsi)
   - 2.3 Fixed position method wıth aitken extrapolation (Stephensen's method)
   - 2.4 Newton-Raphson method
   - 2.5 Secant method
   - 2.6 Muller method
   - 2.7 Newton-Raphson (secant) – bisection combined method
   - 2.8 Inverse quadratic Lagrange interpolation method
   - 2.9 Brent method (ınverse quadratic lagrange interpolatıon – bisectıon and sekant combined method)
   - 2.10 Roots of a complex number by Newton-Raphson method
   - 2.11 Higher order methods, Halley, Abbasbandy Thiele1, Thiele2 and Hasan methods
   - 2.12 Finding the roots limits
   - 2.13 Ridder method
   - 2.14 Roots of the second and third degree polynomials
   - 2.15 Complex roots of the polynomials by utilising synthetic division process
   - 2.16 Complex roots of the polynomials by Bairstow's method
   - 2.17 Root finding by using software packages
   - 2.18 Problems

3. **Linear system of equations**
   - 3.1 Linear system of equations, general information
   - 3.2 Gauss elimination method
   - 3.3 Gauss elimination with partial pivoting
   - 3.4 Gauss-Jordan method
   - 3.5 Gauss-Jordan method with partial pivoting
   - 3.6 LU-decomposition- gauss (dolittle) method
   - 3.7 Crut LU decomposition method
   - 3.8 Cholesky symmetric matrix LU decomposition method
   - 3.9 Jacobi gauss -Seidel and successive relaxation methods
   - 3.10 Methods for band marices
   - 3.11 Conjugate gradient method
   - 3.12 The residual correction method
   - 3.13 Norms
   - 3.14 Unit matrix, transpose matrix, Determinant and matrix inversion
   - 3.15 Cramer'rule
   - 3.16 Problems

4. **Nonlinear system of equations (roots of system of equations)**
   - 4.1 Newton-Raphson method

4.2 Broyden secant method with Sharman-Morrison formula
4.3 homotopy or continuation methods
4.4 Jacobi gauss -Seidel and successive relaxation methods
4.4 Problems

5. Optimisation

5.1 Definition of optimization and graphic methods
5.2 one dimensional non-linear fibonnachi search (golden search) method
5.3 One dimensional non-linear quadratic polynomial method
5.4 One dimensional non-linear cubic polynomial method
5.5 One dimensional non-linear Newton-Raphson method
5.6 One dimensional non-linear Brent method
5.7 Improving one dimensional optimization bracket values
5.8 Newton-Raphson method for nonlinear multivariable optimization
5.9 Nelder – Mead simplex nonlinear mulitivariable optimization method
5.10 Non-linear multidimensional steepest descent method
5.11 Modified newton method(Newton-Raphson steepest descent combined method)
5.12 Non-linear multidimensional Fletcher-Reeves and Polak-Ribiere conjugate gradient method
5.13 Non-linear multidimensional Polak-Ribiere conjugate gradient method
5.14 Non-linear multidimensional nonlinear Davidon-Fletcher-Powell method
5.15 Multivariable non-linear Broyden-Fletcher-Goldberg-Shanno (bfgs) method
5.16 Homotopy or continuation methods
5.17 Multivariable nonlinear broyden secant method with Sherman Morrison formula
5.18 Scholastic methods : Genetic algorithms
5.19 Scholastic methods: Monte-Carlo method
5.20 Stochastic methods: iterative hill climbing
5.21 Stochastic methods: simulated annealing
5.22 Stochastic methods: Firefly algorithm
5.23 Stochastic methods: Particle swarm algorithm
5.24 Stochastic methods: Bath algorithm
5.25 Stochastic methods: Harmony and Melody Search
5.26 Stochastic methods: Cuckoo search
5.27 Stochastic methods:Hunting search
5.28 Stochastic methods:Harris' hawks algorithm
5.29 Stochastic methods: Thermal Exchange algorithm
5.30 Non linear optimization with boundary restrictions
5.31 Travelling salesman problem (TSP)
5.32 Linear multivariable optimization, Simplex method
5.33 Solution of the nonlinear system of equations by using optimization algorithms
5.34 Problems

6. **Curve fitting and function approximation**

6.1 Linear curve fitting : least square methods with one variable
6.2 Ortogonal polynomial least square
6.3 Curve fitting : least square method with multi variables
6.4 Interpolation (Newton, Lagrange and Hermit interpolation formulas)
6.5 Vandermonde interpolation
6.6 Cubic spline interpolation
6.7 B-spline interpolation
6.8 Least square curve fitting of nonlinear equations

6.9 Usage of parametric variables in curve fitting
6.10 Moving least squares, with one and multi dimensional variables
6.11 Bernstein polinomials and Bezier curves
6.12 Curve fitting by using partially continous functions
6.13 Rational function approximations : Pade approximation method
6.14 Rational function approximations : Chebyschev approximation method
6.15 Trigonometric function approximation and discrete fourier series approximation
6.16 Finite element (simplex) methods for surface curve fitting
6.17 Neural network
6.18 Problems

7. **Integration and derivatives**
7.1 Newton-Cotes formulations : Trapezoidal, Bole, Simpson 1/3 ve 3/8 method
7.2 Richardson extrapolation (Romberg integration) formula and Aitken extrapolation with Newton – Cotes integration formulas
7.3 Quadratures: Gauss-Legendre, Gauss - Chebychev, Gauss -Jacobi, Gauss -Hermit, Gauss-Lequerre Integration formulas
7.4 Integration formulation with adjustable error
7.5 Clenshaw-Curtis integration formula
7.6 One and multi-dimensional Monte-Carlo integral
7.7 Multidimensional integration by parametric methods
7.8 Direct integration of data
7.9 Derivation
7.10 Derivation , Pad'e Approximation
7.11 Direct derivation of the numerical data
7.12 Problems

8. Differential equations
8.1 Euler equation
8.2 Heun Equation
8.3 Runge-Kutta Methods
8.4 Multistep (open) Methods
8.5 Differential equations with more then one set of equation
8.6 Boundary value problems and shooting methods in differential equations
8.7 Problems

9. Eigenvalue problem
9.1 Introduction to eigenvalue problems
9.2 Dominant eigenvalue calculation (Power method)
9.3 Smallest magnitude eigen value calculation (inverse power method)
9.4 Computation of intermediate eigenvalues by power method
9.5 Jacobi method and Householder transformation
9.6 Characteristic polynomial and Leverier Method
9.7 QL Algorithm
9.8 QR algorithm
9.9 Roots of polynomials and companion matrix
9.10 Singular value decomposition (SVD) for solving system of equations
9.11 Singular value decomposition (SVD) for solving least square curve fitting problems
9.12 Problems

10. Finite Difference Solutions of partial differential equations
10.1 Introduction to finite difference solutions

11. Finite element solution of partial differential equations
11.1 Finite elements
11.2 Application of Integral equations to finite elements
12. Method of Lines solutions of partial differential equations
13. Approximate methods as solutions of partial differential equations
14. Homework problems and exams
15. References

Mustafa Turhan Çoban, Borned in Bolu, Seben, Turkey in 1957. He received his BSc degree from Ege University School of Mechanical Engineering, department of mechanical engineering in 1978, his M.Sc. degree in Mechanical Engineering from Michigan Technological University(U.S.A.) in 1982. He Graduated From University of Utah(U.S.A.) in 1986 with a Ph.D. in Mechanical Engineering. In 1995, he received Postgraduate degree in Computer Science from Victoria Technological University(Australia). He worked as a teacher and researcher at University of Nebraska, Lincoln(U.S.A.), Victoria Technological University(Australia), Ballarat University (Australia), Dokuz Eylül University, School of Engineering, Department of mechanical engineering(Turkey), Gebze Yüksek Teknoloji Enstitüsü, Computer science(Turkey), Ege University, School of Engineering, Department of mechanical engineering. He worked at Aras compressors (Turkey), Mineral Research and Exploration Institute of Turkey, division of drilling, section of geothermal energy, TUBITAK (Turkish Technological and Scientific Research Foundation) Institute of Energy, Imperial Chemical Industries (Australia), Ceramic Fuel Cells Limited (Australia) as an engineer/senior engineer/researcher; TUBITAK UME (National Metrology Institute of Turkey) as technical vice-chairman.

PREFACE

This book came from the combination of four courses I've been given in Ege University, mechanical engineering. Graduate level numerical analysis, optimisation, applied mathematics and undergraduate level numerical analysis. As a basic approach, I tried to combine computer codes with algorithms, so that students can directly try computer codes in computer environment. While developing the basic codes java language is used as prefered computer language, but to show independence of algorithms from computer languages codes on severel other languages such as python, octave-matlab, C++ , MS excel and Fortran are introduced in starting chapters. While developing codes, function inputs are kept independent of the code so that users can introduced their own functions easily when they are utilising the codes. Science of numerical mathematics are developing very fast in recent years, therefore many algorithms are not existed on the books. I've taken some algoritms from books, but a considerable few of them coming from recent papers. One interesting point about the book, Russian peasant multiplication algorithm in introductory first chapter is taken from the book "The Last Theorem" by Arthur C. Clarke and Fredirich Pohl. In algorithms two and three dimensional graphic codes also used extensively. These codes are mostly developed by author, but some codes are taken from the internet environment and further developed to fit our purposes. For example 3D surfacePlot program is developed by using free software developed by Yanto Suryono yanto@fedu.uec.ac.jp . Thanks guys for this, important contribution. The codes given in here can also be used as free, please do give a reference. All the codes are not included in the book due to space requirement and long size of the programs. Please refer to SCO1.rar file given in my internet site www.turhancoban.com to get access to the complete list of the files, including my other publications such as numerical heat transfer, numerical thermodynamis and java programming language. I utilised the basic algorithms and codes also in the other places and the books for further scientific calculations (mostly in energy field). I am hoping that it would be a valuable source for all researchers and scientists utilising this book and the codes given with the book. One more point, some of the codes are developed by my students as their research or course assignments. You might find their names in the related programs or chapters. I would like to mention here Hamza Taş, my ph.D students Mert Sinan Turgut, Oğuz Emrah Turgut and Halil Atalay. Thanks guys. One of my surprise just today: While I was looking some reference related to some numerical analysis topic, I found out that this book was given as a first reference by Google search. was a nice surprise for me. I wish an enjoyable trip for everyone who will attempt to utilise this book. Sharing the knowledge will enhanced the education and education is an important tool to make a better world for all of us.

M. Turhan ÇOBAN, Ege University, 5.12.2019

1 INTRODUCTORY CONCEPTS

1.1 WORKING ENVIRONMENTS OF NUMERICAL CODES

As a basic programming language, java will be used in most of the example codes in this book, but in the years of computer numerical analysis classes, ı've seen that my students found java language a bit difficult basically due to class concept and relatively long class structures. Therefore, at least in starting sections I'll use examples by using other languages such as ptyhon, octave(matlab), excel and C++ as well as java. They are all third generation computer languages and except small differences, they all use the same logical structures. Computer compilers and interpreters and math calculation packages can be found as free Access in internet environment. In order to develop computer codes you also need a development environment. They are called editors. Editors can be specifically designed for a certain computer programming language or could be designed to serve with any language. Some of them even wrte codes automatically. As a learning environment using simple editors will be much usefull for a student. Note that internet adresses changes by time, if any adress is not existed do your own search. It should be somewhere in the internet universe. You should also note that, this boook asume students has knowledge of computer programming. If you do not have enough programming experience please refer to programming books first. For java language you can check out my book (http://www.turhancoban.com/kitap/COMPUTER%20PROGRAMMING.pdf)

As this book is written several useful internet sites available can be listed as:

Apache Open Office	http://www.openoffice.org/download/
Anaconda python	https://anaconda.org/anaconda/python
Python oficial site	https://www.python.org/
Java jdk (java development kit)	https://www.oracle.com/java/technologies/javase-downloads.html
Mingw c++ compiler(C++ and fortran)	http://mingw.org/
Gnu octave (math sooftware similar to matlab)	https://www.gnu.org/software/octave/
Scilab (math sooftware similar to matlab)	https://www.scilab.org/
Cedit editor (general)	https://sourceforge.net/projects/cedit/
BlueJ editor (java editor)	https://www.bluej.org/
Maxima (math software)	http://maxima.sourceforge.net/download.html
Eclipse editor (java +)	https://www.eclipse.org

1.2 TAYLOR SERIES AND EROR TERM IN THE EQUATIONS

Taylor equation can be considered the roots of many numerical analysis methods. It will give us serial. **Taylor series** is a representation of a function as an infinite sum of terms calculated from the values of its derivatives at a single point. It is named after the English mathematician Brook Taylor. If the series

is centered at zero, the series is also called a **Maclaurin series**, named after the Scottish mathematician Colin Maclaurin. It is common practice to use a finite number of terms of the series to approximate a function. Taylor series around a point a can be written as

$$f(x) = f(a) + \frac{f'(a)}{1!}(x-a) + \frac{f"(a)}{2!}(x-a)^2 + \frac{f^{(3)}(a)}{3!}(x-a)^3 + \cdots$$

Or if it is written in summation notation

$$f(x) = \sum_{n=0}^{\infty} \frac{f^{(n)}(a)}{n!}(x-a)^n$$

Or if a is replaces with x^k and x is replaced with x^{k+1} equation becomes

$$f(x^{k+1}) = \sum_{n=0}^{\infty} \frac{f^{(n)}(x^k)}{n!}(x^{k+1}-x^k)^n$$

+..

Computers could not calculate anything more complex than addition, the rest usually accomplish through Taylor series approximations. For example function $f(x)=e^x$ can be approximated as

$$f(x) = e^x = 1 + \frac{x}{1!} + \frac{x^2}{2!} + \frac{x^3}{3!} + \cdots = 1 + x + \frac{x^2}{2} + \frac{x^3}{6} + \frac{x^4}{24} + \frac{x^5}{120}$$

Some of the formulation used in Numerical analysis is direct derivation from the Taylor series. For example for the Newton root finding equation, above linear terms of the equation is ignorede and function value is replaced with 0

$f(x^{k+1}) = 0 = f(x^k) + f'(x^k)(x^{k+1}-x^k)$ Will yield to

$$x^{k+1} = x^k - \frac{f(x^k)}{f'(x^k)}$$

Taylor series can also be used to estimate error. The series usually used up to a certain term and error will be proportinal to one upper term in the series. For example If exponetial function is taken as

$$f(x) = e^x = 1 + x + \frac{x^2}{2} + \frac{x^3}{6} + \frac{x^4}{24}$$

The error term will be proportional to the next term of the series

$$Error \cong \frac{x^5}{120}$$

1.3 SERİES SOLUTIONS OF FUNCTIONS

Digital computers can only calculate addition process, therefore most of the mathematical expressions are calculated by using series. This series mostly created by using Taylor series expansion. Series solutions of a few functions will be listed in here.

1. $\ln(x) = 2\sum_{k=1}^{\infty} \frac{1}{2k-1}\left(\frac{x-1}{x+1}\right)^{2k-1}$ $[0 < x]$
2. $\exp(x) = \sum_{k=1}^{\infty} \frac{x^k}{k!}$
3. $\exp(-x) = \sum_{k=1}^{\infty} (-1)^k \frac{x^k}{k!}$
4. $\exp(-x^2) = \sum_{k=1}^{\infty} (-1)^k \frac{x^{2k}}{k!}$
5. $\exp(x)(1+x) = \sum_{k=1}^{\infty} \frac{x^k(k+1)}{k!}$
6. $a^x = e^{(xln(a))} = \sum_{k=1}^{\infty} \frac{[xln(a)]^k}{k!}$
7. $\sum_{k=0}^{\infty} a^{kx} = \frac{1}{1-a^x}$ [a>1 and x<0 or 0<a<1 x>0]

8. $\sin(x) = \sum_{k=0}^{\infty}(-1)^k \frac{x^{2k+1}}{(2k+1)!}$
9. $\sin h(x) = \sum_{k=0}^{\infty} \frac{x^{2k+1}}{(2k+1)!}$
10. $\cos(x) = \sum_{k=0}^{\infty}(-1)^k \frac{x^{2k}}{(2k)!}$
11. $\cos h(x) = \sum_{k=0}^{\infty} \frac{x^{2k}}{(2k)!}$
12. $\tan(x) = \frac{\sin(x)}{\cos(x)}$
13. $\tanh(x) = \frac{\sinh(x)}{\cos h(x)}$
14. $\arcsin(x) = \sum_{k=0}^{\infty} \frac{(2k)!x^{2k+1}}{2^{2k}(k!)^2(2k+1)} \quad x^2 \leq 1$
15. $\operatorname{arcsinh}(x) = \sum_{k=0}^{\infty}(-1)^k \frac{(2k)!x^{2k+1}}{2^{2k}(k!)^2(2k+1)} \quad x^2 \leq 1$
16. $\arctan(x) = \sum_{k=0}^{\infty}(-1)^k \frac{x^{2k+1}}{(2k+1)} \quad x^2 \leq 1$
17. $\operatorname{arctanh}(x) = \sum_{k=0}^{\infty} \frac{x^{2k+1}}{(2k+1)} \quad x^2 \leq 1$
18. $\operatorname{erf}(x) = \frac{2}{\sqrt{\pi}}\sum_{n=0}^{\infty}(-1)^n \frac{x^{2n+1}}{n!(2n+1)}$
19. $\pi = \sum_{k=0}^{\infty} \frac{1}{16^k}\left(\frac{4}{8k+1} - \frac{2}{8k+4} - \frac{1}{8k+5} - \frac{2}{8k+6}\right)$ Bailey, Borwein Plouffe(BBP) formula[90]
20. $J_n(z) = \left(\frac{1}{2z}\right)\sum_{k=1}^{\infty}(-1)^k \frac{\left(\frac{1}{2z}\right)^{2k}}{k!\,\Gamma(n++1)}$ Bessel function
21. $\Gamma(z) = \int_{z=0}^{\infty} t^{z-1}e^{-t}dt = \frac{e^{-\gamma z}}{z}\prod_{n=1}^{\infty}\left(1+\frac{z}{n}\right)^{-1} e^{t/n}$ where $\gamma =$ 0.57721566490153286… is the Euler-Mascheroni constant
22. $Y_n(z) = \frac{J_n(z)\cos(n\pi) - J_{-n}(z)}{\sin(n\pi)}$ Bessel function

Now we would like to convert some of these mathematical series formulations into computer programming codes in popular computer languages. As a first series let us look at a simple form as: $f(n) = sum(n) = \sum_{k=1}^{n} k = 1 + 2 + .. + n$ can be approximated in computer environment as

Program 1.3.1 sum(n) function series solution, Java Version

```
public class sum1
{ public static double sum(int n)
  {double total=0;
   for(int k=1;k<=n;k++)
   {total+=k;}
   return total;
  }
  public static void main(String arg[])
   {System.out.println(sum(100));}
}
```

```
---------- Capture Output ----------
> "D:\co\java\bin\javaw.exe" sum1
5050.0
> Terminated with exit code 0.
```

Program 1.3.2 sum(n) function series solution, C++ Version

```
#include <stdio.h>
#include <iostream>
#include <math.h>
```

```
using namespace std;

double sum(int n)
  {double total=0;
   for(int k=1;k<=n;k++)
   {total+=k;}
   return total;
  }
int main()
{printf("sum=%20.15f\n",sum(100));
 cout<<sum(100);
}
```

```
---------- Capture Output ----------
> "D:\okul\SCO1\run.bat" sum1

D:\okul\SCO1>sum1
sum=5050.000000000000000
5050
> Terminated with exit code 0.
```

Program 1.3.3 sum(x) function series solution, Python Version

```
from math import *

def sum1(n):
    total=0;
    i=1;
    while i<=n :
        total=total+i;
        i=i+1;
    return total;

n=int(input("n = "));
r=sum1(n);
print ("total=",r);
```

```
---------- Capture Output ----------
> "D:\co\anaconda\pythonw.exe" sum1.py
n = 100
total= 5050

> Terminated with exit code 0.
```

Program 1.3.4 sum(x) function series solution, Octave(matlab) Version

```
function total = sum1 (n)
  total=0;
  for k=1:n
      total=total+k;
   endfor
end
```

```
>> sum1(100)
ans =  5050
>>
```

Program 1.3.5 sum(x) function series solution, Fortran 90 Version

```
program SUM1
IMPLICIT NONE
INTEGER n
n=100
```

```
print *,SUM(n)

CONTAINS
  FUNCTION SUM(n)
  INTEGER n,I
  REAL SUM
  !summation of integer numbers from 1 to n
  SUM=0
  DO I = 1, n
  SUM=SUM+I
  END DO
  END FUNCTION SUM

end program SUM1
```

```
---------- Capture Output ----------
> "E:\okul\SCO1\run.bat" SUM1

E:\okul\SCO1>SUM1
   5050.00000

> Terminated with exit code 0.
```

f(n) $= average(n) = \frac{1}{n}\sum_{k=1}^{n} k = \frac{1}{n}(1 + 2 + .. + n)$ can be approximated in computer environment as

Program 1.3.6 average(x) function series solution, java Version

```
public class average1
{ public static double average(int n)
  {double total=0;
   for(int k=1;k<=n;k++)
   {total+=k;}
   return total/n;
  }
  public static void main(String arg[])
   {System.out.println(average(100));}
}
```

```
---------- Capture Output ----------
> "D:\co\java\bin\javaw.exe" average1
50.5
> Terminated with exit code 0.
```

Program 1.3.7 average(x) function series solution, C++ Version

```
#include <stdio.h>
#include <iostream>
#include <math.h>
#include <iomanip>
using namespace std;

double average(int n)
  {double total=0;
   for(int k=1;k<=n;k++)
   {total+=k;}
   return total/n;
  }
int main()
{printf("average =%20.15f\n",average(100));
 cout<<setprecision(20);
 cout<<average(100);
}
```

```
---------- Capture Output ----------
> "D:\okul\SCO1\run.bat" average1

D:\okul\SCO1>average1
average =  50.500000000000000
50.5
> Terminated with exit code 0.
```

Program 1.3.8 average(x) function series solution, python Version

```
from math import *

def average(n):
    total=0;
    i=1;
    while i<=n :
        total=total+i;
        i=i+1;
    return total/n;

n=int(input("n = "));
r=average(n);
print ("average=",r);
```

```
---------- Capture Output ----------
> "D:\co\anaconda\pythonw.exe" average.py
n = 100
average= 50.5
> Terminated with exit code 0.
```

Program 1.3.9 average(x) function series solution, octave(matlab) Version

```
function total = average(n)
  total=0;
              for k=1:n
   total=total+k;
    endfor
    total=total/n;
end
```

```
>> average(100)
ans =  50.500
>>
```

Program 1.3.10 average(x) function series solution, Fortran 90 Version

```
 program average1
IMPLICIT NONE
INTEGER n
n=100
print *,'average = ',average(n)


CONTAINS
  FUNCTION average(n)
  INTEGER n,I
  REAL SUM,average
  !summation of integer numbers from 1 to n
  SUM=0
  DO I = 1, n
  SUM=SUM+I
  END DO
```

```
  average=SUM/n
  END FUNCTION average

end program average1
```

```
---------- Capture Output ----------
> "E:\okul\SCO1\run.bat" average1

E:\okul\SCO1>average1
 average =    50.5000000

> Terminated with exit code 0.
```

f(n) $= average(x_k) = \frac{\sum_{k=0}^{n} x_k}{n}$ can be approximated in computer environment as

Program 1.3.10 average(x_k) function series solution, java Version

```
public class average2
{ public static double average(double x[])
  {double total=0;
   int n=x.length;
   for(int k=0;k<n;k++)
   {total+=x[k];}
   return total/n;
  }
  public static void main(String arg[])
   {   double x[]={1.11,2.21,3.31,4.41,5.51,6.61};
                System.out.println(average(x));}
}
```

```
---------- Capture Output ----------
> "D:\co\java\bin\javaw.exe" average2
3.86
> Terminated with exit code 0.
```

Program 1.3.11 average(x_k) function series solution, java Version

```
public class average2
{ public static double average(double x[])
  {double total=0;
   int n=x.length;
   for(int k=0;k<n;k++)
   {total+=x[k];}
   return total/n;
  }
  public static void main(String arg[])
   {   double x[]={1.11,2.21,3.31,4.41,5.51,6.61};
                System.out.println(average(x));}
}
```

```
---------- Capture Output ----------
> "D:\co\java\bin\javaw.exe" average2
3.86
> Terminated with exit code 0.
```

Program 1.3.12 average(x_k) function series solution, C++ Version

```
#include <stdio.h>
#include <iostream>
#include <math.h>
```

```
#include <iomanip>
using namespace std;

double average(double x[],int n)
  {double total=0;
   for(int k=0;k<n;k++)
   {total+=x[k];}
   return total/n;
  }
int main()
{   double x[]={1.11,2.21,3.31,4.41,5.51,6.61};
    int n=sizeof x/sizeof(double);
    printf("average =%20.15f\n",average(x,n));
    cout<<setprecision(20);
    cout<<average(x,n);
}
```

```
---------- Capture Output ----------
> "D:\okul\SCO1\run.bat" average2
D:\okul\SCO1>average2
average =   3.860000000000000
3.8599999999999999
> Terminated with exit code 0.
```

Program 1.3.13 average(x_k) function series solution, python Version

```
from math import *

def average(x):
    n=len(x);
    total=0;
    i=1;
    for i in range(0,n):
        total=total+x[i];
    return total/n;

x=[1.11,2.21,3.31,4.41,5.51,6.61];
r=average(x);
print ("average=",r);
```

```
---------- Capture Output ----------
> "D:\co\anaconda\pythonw.exe" average.py
average= 3.86

> Terminated with exit code 0.
```

Program 1.3.14 average(x_k) function series solution, Octave(Matlab) Version

```
from math import *

def average(x):
    n=len(x);
    total=0;
    i=1;
    for i in range(0,n):
        total=total+x[i];
    return total/n;

x=[1.11,2.21,3.31,4.41,5.51,6.61];
r=average(x);
print ("average=",r);
```

```
>> x=[1.11 2.21 3.31 4.41 5.51 6.61];
```

```
>> average1(x)
ans =  3.8600
>>
```

Program 13.15 average(x_k) function series solution, Fortran 90 Version

```
 program average2
IMPLICIT NONE
REAL, DIMENSION(:), ALLOCATABLE :: X
X=(/1.11,2.21,3.31,4.41,5.51,6.61/)
print *,'average = ',average(X)
CONTAINS
  FUNCTION average(Y)
  REAL, DIMENSION(:), INTENT(INOUT):: Y
  INTEGER I,n
  REAL SUM
  REAL average
  n=SIZE(Y)
  !summation of real array
  SUM=0
  DO I = 1, n
  SUM=SUM+Y(I)
  END DO
  average=SUM/n
  END FUNCTION average
end program average2
```

```
---------- Capture Output ----------
> "E:\okul\SCO1\run.bat" average2
E:\okul\SCO1>average2
 average =    3.85999990
> Terminated with exit code 0.
```

The function f(x) = exp(x) = $\sum_{k=1}^{\infty} \frac{x^k}{k!}$ Or in open form:

f(x) = $e^x = 1 + \frac{x}{1!} + \frac{x^2}{2!} + \frac{x^3}{3!} + \frac{x^4}{4!} + \frac{x^5}{5!} + \cdots$ can be approximated in computer environment as

Program 1.3.16 exp(x) function series solution, Java Version

```
public class func
{
   public static double exp(double x)
   {double total=0;
    double factorial=1;
    double power=1;
    for(int i=1;i<=200;i++)
    {total=total+power/factorial;
     power=power*x;
     factorial=factorial*i;
    }
    return total;
   }
   public static void main(String arg[])
   {System.out.println(exp(1));}
}
```

```
---------- Capture Output ----------
> "D:\co\java\bin\javaw.exe" func
2.7182818284590455
> Terminated with exit code 0.
```

Program 1.3.17 exp(x) function series solution, C++ Version

```
#include <stdio.h>
#include <iostream>
```

```
#include <math.h>
using namespace std;
   double exp(double x)
   {double total=0;
    double factorial=1;
    double power=1;
    for(int i=1;i<=200;i++)
    {total=total+power/factorial;
     power=power*x;
     factorial=factorial*i;
    }
    return total;}
   int main()
   { double r=exp(1.0);
     printf("r=%20.15f\n",r);
     cout<<"r="<<r;
     return 0;
   }
```

```
---------- Capture Output ----------
> "D:\okul\SCO1\run.bat" func

D:\okul\SCO1>func
r=   2.718281828459046
r=2.71828
> Terminated with exit code 0.
```

Program 1.3.18 exp(x) function series solution, Python version

```
from math import *
def exp(x):
      total=0;
      factorial=1;
      power=1;
      for i in range(1,100):
        total=total+power/factorial;
        power=power*x;
        factorial=factorial*i;
      return total;
r=exp(1.0);
print("r=",r);
```

```
---------- Capture Output ----------
> "D:\co\python\pythonw.exe" func.py
r= 2.7182818284590455
> Terminated with exit code 0.
```

Program 1.3.19 exp(x) function series solution, Octave (Matlab) version

```
 function total=exp1(x)
      total=0;
      factorial=1;
      power=1;
      for i=1:100
        total=total+power/factorial;
        power=power*x;
        factorial=factorial*i;
      endfor
 endfunction
```

```
>> exp1(1)
ans =  2.71828182845905
>>
```

Note: the function name exp is not used due to predefined function is already existed in Octave

Program 1.3.20 exp(x) function series solution, Fortran 90 version

```
program exp1
```

```
IMPLICIT NONE
INTEGER n
REAL(KIND = SELECTED_REAL_KIND(10,34))  x
x=1.0
print *,'exp(',x,') = ',exp(x)
CONTAINS
  FUNCTION exp(x)
  REAL(KIND = SELECTED_REAL_KIND(10,34))  exp,x,total,factorial,power
  INTEGER i
  total=0
  factorial=1
  power=1
  DO i = 1, 200
  total=total+power/factorial
  power=power*x
  factorial=factorial*i
  END DO
  exp=total
  END FUNCTION exp
end program exp1
```

```
---------- Capture Output ----------
> "E:\okul\SCO1\run.bat" exp1
E:\okul\SCO1>exp1
 exp(   1.0000000000000000      ) =    2.7182818284590455
> Terminated with exit code 0.
```

	x=		1
n	x^n	n!	total
0	1	1	1
1	1	1	2
2	1	2	2,5
3	1	6	2,66666667
4	1	24	2,70833333
5	1	120	2,71666667
6	1	720	2,71805556
7	1	5040	2,71825397
8	1	40320	2,71827877
9	1	362880	2,71828153
10	1	3628800	2,7182818
11	1	39916800	2,71828183
12	1	479001600	2,71828183
13	1	6227020800	2,71828183
14	1	8,718E+010	2,71828183

Program 1.3.21 exp(x) function series solution, OpenOffice Calc (excel) version

function f(x)=ln(x) can be approximated as $\mathbf{ln}(\boldsymbol{x}) = \mathbf{2}\sum_{\boldsymbol{k}=\mathbf{1}}^{\infty}\frac{\mathbf{1}}{\mathbf{2}\boldsymbol{k}-\mathbf{1}}\left(\frac{\boldsymbol{x}-\mathbf{1}}{\boldsymbol{x}+\mathbf{1}}\right)^{\mathbf{2}\boldsymbol{k}-\mathbf{1}}$ $[\mathbf{0} < \boldsymbol{x}]$

Program 1.3.22 ln(x) function series solution, java Version

```
public class func2
{
   public static double ln(double x)
```

```
  {double total=0;
   double y=(x-1.0)/(x+1.0);
   double power=y;
   for(int k=1;k<=200;k++)
   {  total=total+power/(2.0*k-1.0);
      power=power*y*y;
   }
   return 2.0*total;
  }
  public static void main(String arg[])
  {   double r=2.7182818284590455;
                System.out.println(ln(r));
  }
}
```

```
---------- Capture Output ----------
> "D:\co\java\bin\javaw.exe" func2
0.9999999999999999
> Terminated with exit code 0.
```

Program 1.3.23 ln(x) function series solution,C++ Version

```
#include <stdio.h>
#include <iostream>
#include <math.h>
using namespace std;
double ln(double x)
  {double total=0;
   double y=(x-1.0)/(x+1.0);
   double power=y;
   for(int k=1;k<=200;k++)
   {  total=total+power/(2.0*k-1.0);
      power=power*y*y;
   }
   return 2.0*total;
  }
  int main()
  {   double r=2.7182818284590455;
              cout<<ln(r);}
```

```
---------- Capture Output ----------
> "D:\okul\SCO1\run.bat" func2
D:\okul\SCO1>func2
1
> Terminated with exit code 0.
```

Program 1.3.24 ln(x) function series solution,Python version

```
from math import *
def ln(x):
    total=0;
    y=(x-1.0)/(x+1.0);
    power=y;
    for k in range(1,100):
       total=total+power/(2.0*k-1.0);
       power=power*y*y;
    return 2*total;
e=r=2.7182818284590455;
r=ln(e);
print("r=",r);
```

```
---------- Capture Output ----------
> "D:\co\python\pythonw.exe" func2.py
r= 0.9999999999999999
> Terminated with exit code 0.
```

Program 1.3.26 ln(x) function series solution, Fortran 90 version

```
program ln1
IMPLICIT NONE
INTEGER n
REAL(KIND = SELECTED_REAL_KIND(10,34))  x
x=2.7182818284590455
print *,'ln(',x,') = ',ln(x)

CONTAINS
  FUNCTION ln(x)
  REAL(KIND = SELECTED_REAL_KIND(10,34))  ln,x,y,total,power
  INTEGER k
  total=0
  y=(x-1.0)/(x+1.0)
  power=y
  DO k = 1, 200
  total=total+power/(2.0*k-1.0)
  power=power*y*y
  END DO
  ln=2.0*total
  END FUNCTION ln
end program ln1
```

```
---------- Capture Output ----------
> "E:\okul\SCO1\run.bat" ln1
E:\okul\SCO1>ln1
 ln(   2.7182817459106445      ) =   0.99999996963213977
> Terminated with exit code 0.
```

Program 1.3.27 ln(x) function series solution, OpenOffice Calc (excel) version

ln(x)

k	x=	2,7182818285
	y=(x-1)/(x+1)	0,4621171573
	power	total
0	0,4621171573	0
1	0,0986861666	0,9242343145
2	0,0210746546	0,9900250922
3	0,0045005403	0,9984549541
4	0,0009611006	0,9997408227
5	0,0002052452	0,9999544006
6	4,38306E-005	0,9999917179
7	9,36012E-006	0,9999984611
8	1,99887E-006	0,9999997091
9	4,26864E-007	0,9999999443
10	9,11578E-008	0,9999999892
11	1,94670E-008	0,9999999979
12	4,15721E-009	0,9999999996
13	8,87782E-010	0,9999999999
14	1,89588E-010	1
15	4,04869E-011	1
16	8,64608E-012	1
17	1,84639E-012	1
18	3,94301E-013	1
19	8,42038E-014	1
20	1,79819E-014	1

Program 1.3.28 ln(x) function series solution, Octave (Matlab) version

```
function total=ln(x)
    total=0;
    y=(x-1.0)/(x+1.0);
    power=y;
    for k=1:100
      total=total+power/(2.0*k-1.0);
```

```
      power=power*y*y;
    endfor
    total=total*2;
endfunction
```

```
>> ln(e)
ans =  1.000000000000000
>>
```

Some of the functions are defined as combination of more than one function (or series solution). For example power function is defined as:

$$f(x) = a^x = e^{(xln(a))} = \sum_{k=1}^{\infty} \frac{[xln(a)]^k}{k!}$$

Program 1.3.23 $f(x) = a^x$ function series solution, Java Version

```
public class func3
{
  public static double ln(double x)
  {double total=0;
   double y=(x-1.0)/(x+1.0);
   double power=y;
   for(int k=1;k<=200;k++)
   {   total=total+power/(2.0*k-1.0);
       power=power*y*y;
   }
   return 2.0*total;
  }
  public static double exp(double x)
  {double total=0;
   double factorial=1;
   double power=1;
   for(int i=1;i<=200;i++)
   {total=total+power/factorial;
    power=power*x;
    factorial=factorial*i;
   }
   return total;
  }
  public static double pow(double a,double x)
  {return exp(x*ln(a));}

  public static void main(String arg[])
  {   System.out.println(pow(2.0,3.0));}}
```

```
---------- Capture Output ----------
> "D:\co\java\bin\javaw.exe" func3
7.999999999999997
> Terminated with exit code 0.
```

Program 1.3.24 $f(x) = a^x$ function series solution, C++ Version

```
#include <stdio.h>
#include <iostream>
#include <math.h>
using namespace std;
  double exp(double x)
  {double total=0;
   double factorial=1;
   double power=1;
   for(int i=1;i<=200;i++)
   {total=total+power/factorial;
    power=power*x;
    factorial=factorial*i;
   }
   return total;
  }
  double ln(double x)
  {double total=0;
   double y=(x-1.0)/(x+1.0);
   double power=y;
```

```
    for(int k=1;k<=200;k++)
    {  total=total+power/(2.0*k-1.0);
       power=power*y*y;
    }
    return 2.0*total;
   }
   double pow(double a,double x)
   {return exp(x*ln(a));}
   int main()
   { double r=pow(2.0,3.0);
     printf("r=%20.15f\n",r);
     cout<<"r="<<r;
     return 0;
}
```

```
---------- Capture Output ----------
> "D:\okul\SCO1\run.bat" func3
D:\okul\SCO1>func3
r=   7.999999999999997
r=8
> Terminated with exit code 0.
```

Program 1.3.25 $f(x) = a^x$ function series solution, Python version

```
from math import *

def ln(x):
    total=0;
    y=(x-1.0)/(x+1.0);
    power=y;
    for k in range(1,100):
       total=total+power/(2.0*k-1.0);
       power=power*y*y;
    return 2*total;
def exp(x):
      total=0;
      factorial=1;
      power=1;
      for i in range(1,100):
        total=total+power/factorial;
        power=power*x;
        factorial=factorial*i;
      return total;
def pow(a,x):
    return exp(x*log(a));

r=pow(2.0,3.0);
print("r=",r);
```

```
---------- Capture Output ----------
> "D:\co\python\pythonw.exe" func3.py
r= 7.999999999999999

> Terminated with exit code 0.
```

Program 1.3.26 $f(x) = a^x$ function series solution,Octave (Matlab) version

```
 function total=pow(a,x)
     total=exp1(x*ln(a));
 endfunction
```

```
>> pow(2,3)
ans =  8.00000000000000
>>
```

Note: the function name exp is not used due to predefined function is already existed in Octave

Program 1.3.27 $f(x) = a^x$ function series solution, Fortran 90 version

```
program power1
IMPLICIT NONE
INTEGER n
REAL(KIND = SELECTED_REAL_KIND(10,34))  a,x
a=2.0
x=3.0
print *,'pow(',a,',',x,') = ',pow(a,x)
CONTAINS
 FUNCTION ln(x)
 REAL(KIND = SELECTED_REAL_KIND(10,34))  ln,x,y,total,power
 INTEGER k
 total=0
 y=(x-1.0)/(x+1.0)
 power=y
 DO k = 1, 200
 total=total+power/(2.0*k-1.0)
 power=power*y*y
 END DO
 ln=2.0*total
 END FUNCTION ln
 FUNCTION exp(x)
 REAL(KIND = SELECTED_REAL_KIND(10,34))  exp,x,total,factorial,power
 INTEGER i
 total=0
 factorial=1
 power=1
 DO i = 1, 200
 total=total+power/factorial
 power=power*x
 factorial=factorial*i
 END DO
 exp=total
 END FUNCTION exp
 FUNCTION pow(a,x)
 REAL(KIND = SELECTED_REAL_KIND(10,34))  pow,a,x
 pow=exp(x*ln(a))
 END FUNCTION pow
end program power1
```

```
---------- Capture Output ----------
> "E:\okul\SCO1\run.bat" power1
E:\okul\SCO1>power1
 pow(   2.0000000000000000     ,   3.0000000000000000     ) =    7.9999999999999973
> Terminated with exit code 0.
```

Program 1.3.28 $f(x) = a^x$ function series solution,OpenOffice Calc (excel) version

ln(a)

k	a=	2
	x=	3
	y=(x-1)/(x+1)	0,3333333333

	power	total
0	0,3333333333	0
1	0,037037037	0,6666666667
2	0,0041152263	0,6913580247
3	0,0004572474	0,6930041152
4	5,08053E-005	0,6931347573
5	0,000005645	0,6931460474
6	6,27225E-007	0,6931470738
7	6,96917E-008	0,6931471703
8	7,74352E-009	0,6931471795
9	8,60392E-010	0,6931471805
10	9,55991E-011	0,6931471805
11	1,06221E-011	0,6931471806
12	1,18024E-012	0,6931471806
13	1,31137E-013	0,6931471806
14	1,45708E-014	0,6931471806
15	1,61898E-015	0,6931471806
16	1,79887E-016	0,6931471806
17	1,99874E-017	0,6931471806
18	2,22082E-018	0,6931471806
19	2,46758E-019	0,6931471806
20	2,74175E-020	0,6931471806

exp(z) = exp(x*ln(a))

z=x*ln(a)	2,0794415417

power	factorial	total
1	1	1
2,0794415417	1	3,0794415417
4,3240771253	2	5,2414801043
8,9916656037	6	6,7400910383
18,697642985	24	7,519159496
38,880655555	120	7,8431649589
80,850050329	720	7,9554566955
168,1229533	5040	7,9888144243
349,6018532	40320	7,9974851052
726,9766166	362880	7,9994884579
1511,7053764	3628800	7,9999050434
3143,5029584	39916800	7,9999837947
6536,7306382	479001600	7,9999974413
13592,749236	6227020800	7,9999996242
28265,327427	87178291200	7,9999999484
58776,09604	1,3077E+012	7,9999999934
122221,45576	2,0923E+013	7,9999999992
254152,3724	3,5569E+014	7,9999999999
528495,00108	6,4024E+015	8
1098974,4598	1,2165E+017	8
2285253,145	2,4329E+018	8

Some of the series solutions might not be as clear as a directly given function. For example let us look at the **Russian peasant multiplication problem:**

Take two integer for example 21 and 37 then double the first one(left hand side) and divide by two the second one(right side) ignore remaining in division process and stop when the right hand side reached to 1 and then cross each line where the right side number is even add up remaining left hand side numbers for example. If the given integer numbers to be multiplied is 21 and 17, the process will follow the path given below:

```
21 37
~~42 18~~ cross out (18 is even)
84 9
~~168 4~~ cross out(4 is even)
~~336 2~~ cross out(2 is even)
672 1
then 21+84+672=777
21*37=777
```

Now let us convert this to an algorithmic structure:

Program 1.3.28 Russian peasant multiplication, Java version

```
public  class russian
{
public static int multiply(int x1,int x2)
{ int y2=x2;
  int y1=x1;
  int total=0;
  while(y2>=1)
  {  if(y2%2 != 0) total+=y1;
     System.out.println("y1 = "+y1+" y2 = "+y2);
     y1=y1*2;y2=y2/2;
  }
  return total;
}
public static void main(String arg[])
{System.out.println(multiply(21,37));}
}
```

```
---------- Capture Output ----------
> "D:\co\java\bin\javaw.exe" russian
y1 = 21 y2 = 37
y1 = 42 y2 = 18
y1 = 84 y2 = 9
y1 = 168 y2 = 4
y1 = 336 y2 = 2
y1 = 672 y2 = 1
777

> Terminated with exit code 0.
```

Program 1.3.29 Russian peasant multiplication, C++ version

```
#include <stdio.h>
#include <iostream>
#include <math.h>
using namespace std;

int multiply(int x1,int x2)
{ int y2=x2;
```

```
  int y1=x1;
  int total=0;
  while(y2>=1)
  {  if(y2%2 != 0) total+=y1;
              cout<<"y1 = "<<y1<<" y2 = "<<y2<<endl;
    y1*=2;y2/=2;
  }
  return total;
}

int main()
{ cout << "multiplication = "<<multiply(21,37);
  return 0;
}
```

```
---------- Capture Output ----------
> "D:\okul\SCO1\run.bat" russian

D:\okul\SCO1>russian
y1 = 21 y2 = 37
y1 = 42 y2 = 18
y1 = 84 y2 = 9
y1 = 168 y2 = 4
y1 = 336 y2 = 2
y1 = 672 y2 = 1
multiplication = 777
> Terminated with exit code 0.
```

Program 1.3.30 Russian peasant multiplication, Python version

```
from math import *

def multiply(x1,x2):
  y2=x2;
  y1=x1;
  b2=0;
  total=0;
  while y2>=1:
    if y2 %2 != 0 :
      total=total+y1;
    print("total=",total,"y1=",y1,"y2=",y2);
    y1=y1*2;
    y2=(int)(y2/2);
  return total;

x1=21;
x2=37;
x=multiply(x1,x2);
print("Russian multiplication :\nx = ",x)
```

```
---------- Capture Output ----------
> "D:\co\python\pythonw.exe" russian.py
total= 21 y1= 21 y2= 37
total= 21 y1= 42 y2= 18
total= 105 y1= 84 y2= 9
total= 105 y1= 168 y2= 4
total= 105 y1= 336 y2= 2
total= 777 y1= 672 y2= 1
Russian multiplication :
x =  777

> Terminated with exit code 0.
```

Program 1.3.31 Russian peasant multiplication, Octave(Matlab) version

```
function total=russian(x1,x2)
    y2=x2;
    y1=x1;
```

```
    total=0;
    while y2>=1
     if mod(y2,2)!=0
     total=total+y1;
     endif
     y1=y1*2;y2=fix(y2/2);
    endwhile
endfunction
```

```
>> russian(21,37)
ans =  777
>>
```

Program 1.3.32 Russian peasant multiplication, Fortran 90 version with function sub program

```
program russian2
IMPLICIT NONE
INTEGER x,y
x=21
y=37
print *,multiply(x,y)
CONTAINS
  FUNCTION multiply(x1,x2)
  INTEGER  x1,x2,y1,y2,total,z,multiply
  total=0
  y1=x1
  y2=x2
  do
    z=y2-(y2/2)*2
    if (z/=0) then
      total=total+y1
    end if
    y1=y1*2
    y2=y2/2
    if(y2<1) EXIT
  end do
  multiply=total
  END FUNCTION multiply
end program russian2
```

```
---------- Capture Output ----------
> "E:\okul\SCO1\run.bat" russian2
E:\okul\SCO1>russian2
        777
> Terminated with exit code 0.
```

Program 1.3.33 Russian multiplication, Fortran 90 version with subroutine sub program

```
 program russian3a
IMPLICIT NONE
INTEGER x,y,total
x=21
y=37
call multiply(x,y,total)
print *,total

CONTAINS
  SUBROUTINE multiply(x1,x2,total)
  INTEGER  x1,x2,y1,y2,total,z
  total=0
  y1=x1
  y2=x2
```

```
  do
    print *, 'y1=',y1,'y2=',y2
    z=y2-(y2/2)*2
    if (z/=0) then
      total=total+y1
    end if
    y1=y1*2
    y2=y2/2

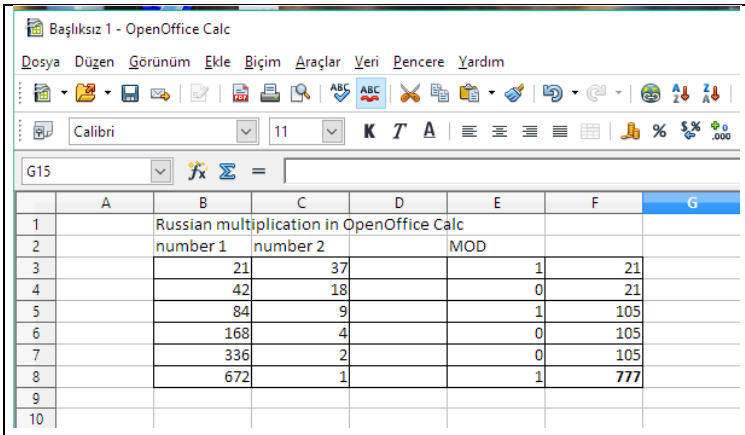
    if(y2<1) EXIT
  end do
  END SUBROUTINE multiply
end program russian3a
```

```
 ---------- Capture Output ----------
> "E:\okul\SCO1\run.bat" russian3a

E:\okul\SCO1>russian3a
 y1=         21 y2=         37
 y1=         42 y2=         18
 y1=         84 y2=          9
 y1=        168 y2=          4
 y1=        336 y2=          2
 y1=        672 y2=          1
        777

> Terminated with exit code 0.
```

Program 1.3.34 Russian multiplication, OpenOffice calc (excel) version



	A	B	C	D	E	F	G
1		Russian multiplication in OpenOffice Calc					
2		number 1	number 2		MOD		
3		21	37		1	21	
4		42	18		0	21	
5		84	9		1	105	
6		168	4		0	105	
7		336	2		0	105	
8		672	1		1	**777**	
9							
10							

Program 1.3.35 Java sin(x) function

```
public class sin
{
  public static double sin(double x)
  {double total=0;
   double factorial=1;
   double power=x;
   int plusminus=1;
   int i=0;
   for(int k=1;k<=200;k++)
   {total=total+power/factorial*plusminus;
    power=power*x*x;
    i=2*k+1;
```

```
 factorial=factorial*(i-1)*i;
 plusminus*=-1;
 }
 return total;
}
public static void main(String arg[])
{   double x=Math.PI/3.0;
            System.out.println("sin = "+sin(x)+" Math.sin = "+Math.sin(x));}
}
```

```
---------- Capture Output ----------
> "E:\co\java\bin\java.exe" sin
sin = 0.8660254037844385 Math.sin = 0.8660254037844386

> Terminated with exit code 0.
```

Program 1.3.36 C++ sin(x) function

```
#include <stdio.h>
#include <iostream>
#include <math.h>
#include <iomanip>
using namespace std;

  double sin1(double x)
  {double total=0;
   double factorial=1;
   double power=x;
   int plusminus=1;
   int i=0;
   for(int k=1;k<=200;k++)
   {total=total+power/factorial*plusminus;
    power=power*x*x;
    i=2*k+1;
    factorial=factorial*(i-1)*i;
    plusminus*=-1;
   }
   return total;
  }
  int main()
  {   double PI=atan(1.0)*4.0;
            double x=PI/3.0;
            cout<<setprecision(12)<<"sin = "<<sin1(x)<<" Math.sin = "<<sin(x);
  }
```

```
---------- Capture Output ----------
> "E:\okul\SCO1\run.bat" sin

E:\okul\SCO1>sin
sin = 0.866025403784 Math.sin = 0.866025403784
> Terminated with exit code 0.
```

Program 1.3.37 Python sin(x) function

```
import math;

def sin(x):
   total=0;
   factorial=1;
   power=x;
   plusminus=1;
   i=0;
   for k in range(1,80):
      total=total+power/factorial*plusminus;
```

```
      power=power*x*x;
      i=2*k+1;
      factorial=factorial*(i-1)*i;
      plusminus*=-1;
   return total;

pi=math.pi;
x=pi/3.0;
print ("sin="+str(sin(x))+"math.sin = "+str(math.sin(x)));
```

```
---------- Capture Output ----------
> "E:\co\python\pythonw.exe" sin.py
sin=0.8660254037844385math.sin = 0.8660254037844386

> Terminated with exit code 0.
```

Program 1.3.38 Octave (Mathlab) sin(x) function

```
function total=sin1(x)
% sin function
   total=0;
   factorial=1;
   power=x;
   plusminus=1;
   for k=1:100
      total=total+power/factorial*plusminus;
      power=power*x*x;
      i=2*k+1;
      factorial=factorial*(i-1)*i;
      plusminus=-1*plusminus;
   end
end
```

```
>> format long
>> x=pi/3
x =  1.047197551196598
>> sin1(x)
ans =    8.660254037844385e-01
>> sin(x)
ans =    8.660254037844386e-01
>>
```

Program 1.3.38 Fortran 90 version sin(x) function

```
program sin1
IMPLICIT NONE
INTEGER n
REAL(KIND = SELECTED_REAL_KIND(10,34))  x,PI
PI=4.0*Atan(1.0)
x=PI/3.0
print *,'sin(',x,') = ',sin(x)
CONTAINS
  FUNCTION sin(x)
  REAL(KIND = SELECTED_REAL_KIND(10,34))  sin,total,factorial,power,x
  INTEGER plusminus,k,i
  total=0
  factorial=1
  power=x
  plusminus=1
  DO k = 1, 200
     total=total+power/factorial*plusminus
     power=power*x*x
     i=2*k+1
     factorial=factorial*(i-1)*i
```

```
    plusminus=-1*plusminus
  END DO
  sin=total
  END FUNCTION sin
end program sin1
```

```
---------- Capture Output ----------
> "E:\okul\SCO1\run.bat" sin1

E:\okul\SCO1>sin1
 sin(   1.0471975803375244      ) =   0.86602541835490165

> Terminated with exit code 0.
```

Program 1.3.39 java version arcsin(x) function

```
public class asin
{
  public static double asin(double x)
  {double total=0;
   double factorial1=1;
   double factorial2=1;
   double power=x;
   double power2=1;
   int i=0;
   for(int k=1;k<=80;k++)
   {total=total+power*factorial1/(power2*factorial2*factorial2)/(2.0*(k-1)+1.0);
    power*=x*x;
    power2*=4;
    i=2*k;
    factorial1=factorial1*(i-1)*i;
    factorial2=factorial2*k;
   }
   return total;
  }
  public static void main(String arg[])
  {   double x=Math.sin(Math.PI/3.0);
              System.out.println("asin = "+asin(x)+" Math.asin = "+Math.asin(x));}
  }
```

```
---------- Capture Output ----------
> "E:\co\java\bin\java.exe" asin
asin = 1.047197551196467 Math.asin = 1.0471975511965976

> Terminated with exit code 0.
```

Program 1.3.40 C++ version arcsin(x) function

```
#include <stdio.h>
#include <iostream>
#include <math.h>
#include <iomanip>
using namespace std;
   double asin1(double x)
  {double total=0;
   double factorial1=1;
   double factorial2=1;
   double power=x;
   double power2=1;
   int i=0;
   for(int k=1;k<=80;k++)
   {total=total+power*factorial1/(power2*factorial2*factorial2)/(2.0*(k-1)+1.0);
    power*=x*x;
    power2*=4;
```

```
    i=2*k;
    factorial1=factorial1*(i-1)*i;
    factorial2=factorial2*k;
   }
   return total;
  }
  int main()
  {   double PI=atan(1.0)*4.0;
              double x1=PI/3.0;
              double x2=sin(x1);
              cout<<setprecision(20)<<"asin = "<<asin1(x2)<<" math.asin = "<<asin(x2);
  }
```

```
---------- Capture Output ----------
> "E:\okul\SCO1\run.bat" asin

E:\okul\SCO1>asin
asin = 1.0471975511964670691 math.asin = 1.0471975511965976313
> Terminated with exit code 0.
```

Program 1.3.41 python version arcsin(x) function

```
import math;

def asin(x):
   total=0;
   factorial1=1;
   factorial2=1;
   power=x;
   power2=1;
   i=0;
   for k in range(1,80):
      total=total+power*factorial1/(power2*factorial2*factorial2)/(2.0*(k-1)+1.0);
      power*=x*x;
      power2*=4;
      i=2*k;
      factorial1=factorial1*(i-1)*i;
      factorial2=factorial2*k;
   return total;

pi=math.pi;
x1=pi/3.0;
x2=math.sin(x1);
print ("asin="+str(asin(x2))+" math.asin = "+str(math.asin(x2)));
```

```
---------- Capture Output ----------
> "E:\co\python\pythonw.exe" asin.py
asin=1.0471975511964204 math.asin = 1.0471975511965976

> Terminated with exit code 0.
```

Program 1.3.41 java version arcsinh(x) function

```
public class asinh
{  public static double asinh1(double x)
   {return Math.log(x + Math.sqrt(x*x + 1.0));}
   public static double asinh(double x)
   {double total=0;
    double factorial1=1;
    double factorial2=1;
    double power=x;
    double power2=1;
    double artieksi=1;
    int i=0;
```

```
    for(int k=1;k<=80;k++)
    {total=total+power*factorial1*artieksi/(power2*factorial2*factorial2)/(2.0*(k-1)+1.0);
     power*=x*x;
     power2*=4;
     i=2*k;
     factorial1=factorial1*(i-1)*i;
     factorial2=factorial2*k;
     artieksi*=-1;
    }
    return total;
   }
   public static void main(String arg[])
   {   double x=0.1;
                System.out.println("asinh = "+asinh(x)+"asinh1 = "+asinh1(x));}
   }
```

```
---------- Capture Output ----------
> "E:\co\java\bin\java.exe" asinh
asinh = 0.09983407889920756asinh1 = 0.09983407889920758

> Terminated with exit code 0.
```

Program 1.3.42 java version erf(x) , error function

```
public class erf1
{  public static double erf(double x)
   {double A=2.0/Math.sqrt(Math.PI);
    double total=0;
    double plusminus=1;
    double power=x;
    double factorial=1;
    for(int n=1;n<200;n++)
    {total+=power*plusminus/factorial/(2.0*(n-1)+1.);
     power*=x*x;
     factorial*=n;
     plusminus*=-1;
     }
    return A*total;
   }
   public static void main(String arg[])
   {   double x=0.1;
                System.out.println("erf = "+erf(x)+"Mathd.erf="+Mathd.erf(x));
   }
   }
```

```
---------- Capture Output ----------
> "E:\co\java\bin\java.exe" erf1
erf = 0.1124629160182849Mathd.erf=0.11246291599933575

> Terminated with exit code 0.
```

Program 1.3.43 python version erf(x) , error function

```
import  math

def erf(x):
   A=2.0/math.sqrt(math.pi)
   total=0.0
   plusminus=1
   power=x
   factorial=1.0
   for n in range(1,200):
        total=total+power*plusminus/factorial/(2.0*(n-1)+1.0)
        power=power*x*x
```

```
        factorial=factorial*n
        plusminus*=plusminus*(-1);
    return A*total

x=0.1
y1=erf(x)
y=math.erf(x)
print("x=",x,"y1=",y,"y=",y)
```

```
runfile('E:/okul/SCO1/erf1.py', wdir='E:/okul/SCO1')
x= 0.1 y1= 0.1124629160182849 y= 0.1124629160182849
```

Program 1.3.43 python version erf(x) , error function (lngamma is called from math library)

```
from math import *

def J(v,x):
    JJ=0
    a=1.0
    fact=1
    b=0.0
    if x <=20:
        x5=1
        lna=0.0;
        x1=v*log(0.5*x)
        lna1=log(0.25*x*x)
        for k in range(0,100):
            x2=lna
            x3=lgamma(k+1)
            x4=lgamma(v+k+1)
            b=x1+x2-x3-x4
            b=x5*exp(b)
            lna=lna+lna1
            x5=-1*x5
            JJ=JJ+b
    elif (v==0 and x==0):
        JJ=1
    else:
        JJ=sqrt(2.0/PI/x)*cos(x-0.5*v*PI-0.25*PI)
    return JJ

x=0.1
y=J(0.0,x)
print("J(0.0,",x,") = ",y)
```

```
runfile('E:/okul/SCO1/bessel.py', wdir='E:/okul/SCO1')
J(0.0, 0.1 ) =  0.9975015620660401
```

The basic reason of this chapter is introducing to you algorithm concept and independence of algorithms from computer languages. We also try to show you that all the third generation languages are quite similar to each other in structure. If you investigate Taylor series solutions with library based results, you might see small differences in further digits. This is due to approximation and accuracy of computer double numbers.

1.4 COMPUTER LANGUAGES

There are several languges existed in computer environment. In this work most of the codes of numerical algorithms will be given in java language. The basic approach for the compiler and computer languge selection in this text is to be free for the use in learning environment. As additional to java language other languages are also defined in here. We will be given some details in here where you can find each computer language and how to compile and run them.

Computer language	İnternet adress to upload language source	compile	run
java	https://www.java.com	**javac** name.java	**java** name
octave	https://www.gnu.org/software/octave/	Write as functions in editor and save	Call functions in run window
C++	Octave above is written in c++ so you can find it under mingw64/bin subdirectory as c++ or g++ or gcc Alternatively https://mingww64.org/doku.php/download	C++ name.cpp –o name.exe	name
Fortran	Octave above has under mingw64/bin subdirectory as gfortran Alternatively https://mingww64.org/doku.php/download	gfroran name.f90 –o name.exe	name
Python(Anaconda)	https://www.anaconda.com/	-	pythonw name.py

In first chapters some of the codes will be given in multiple languages. The basic reason fort his to Show the studends similarities and differences of third generation computer languages. In the starting chapters codes are relatively small so we can afford to give them in several languages.

1.5 ABSTRACT CLASSES, INTERFACES AND LAMBDA VARIABLES

An abstract class or interface is an empty class for reference purposes, although they can have additional methods. Most of the modern third geneation languages such as java, C++, python has abstract class and/or interface buid into the language. Lambda variables a short-cut definition to define a new class derived from an interface. This concepts are very useful for us in numerical methods, becouse function definitions in our numerical codes become independent of actual function. As an input function an object of abstract class or interface can be defined, and when we will use the numerial analysis codes (suc as root finding, integration etc.) actual function can be defined as derived form abstract/class and/or interface. Abstract class/interface examples will be given here in several programming languages for simple function f(x). Please note that inside of these classes some additional codes for derivatives, root finding methods and integrals might exist. In order to see details of these codes please refer to related chapters in this book.

Abstract class example in java language. Function f(x)

Program 1.5.1 f_x.java Abstract class for function f(x)

```
 import static java.lang.Math.*;

abstract public class f_x extends Mathd
{// single function single independent variable
  // example f=x*x
  // Reference :"Generation of Finite Difference Formulas on Arbitrary Spaced Grids",
  // Bength Fornberg, Mathematics of Computation, Volume 51, Number 184, October 1988
  // pages 699-706
// abstract function func
abstract public double func(double x);
// 1st order derivative
public double dfunc(double x)
{double h=1.0e-5;
 int n=1;
 int M=10;
 return dfunc(x,n,M,h);
}
//second order derivative
public double dfunc2(double x)
{double h=1.0e-3;
```

```
 int n=2;
 int M=10;
 return dfunc(x,n,M,h);
}
// nth order derivative
public double dfunc(double x,int N,int Mi,double hi)
{// order of the maximum derivative
 // N order of derivative
 // M degree of difference formula
double c[][][];
double alpha[];
double h;
int M;
double a[]=new double[0];
h=0.01;
double x0=0;
M=20;
double alphai[]={0,1,-1,2,-2,3,-3,4,-4,5,-5,6,-6,7,-7,8,-8,9,-9,10,-10,11,-11,12,-12,13,-13,14,-14,15,
-15,16,-16,17,-17,18,-18,19,-19,20,-20,21,-21,22,-22,23,-23,24,-24,25,-25,26,-26,27,-27,28,-28,29,-29,30,-30,
31,-31,32,-32,33,-33,34,-34,35,-35,36,-36,37,-37,38,-38,39,-39,40,-40,41,-41,42,-42,43,-43,44,-44,45,-45,46,
-46,47,-47,48,-48,49,-49,50,-50,51,-51,52,-52,53,-53,54,-54,55,-55,56,-56,57,-57,58,-58,59,-59,60,-60,
-61,61,-62,62,-63,63,-64,64,-65,65,-66,66,-67,67,-68,68,-70,70,-71,71,-72,72,-73,73,-74,74,-75,75,
-76,76,-77,77,-78,78,-79,79,-80,80,-81,81,-82,82,-83,83,-84,84,-85,85,-86,86,-87,87,
-88,88,-89,89,-90,90,-91,91,-92,92,-93,93,-94,94,-95,95,-96,96,-97,97,-98,98,-99,99,-100,100};
alpha=alphai;
int N1=alpha.length-1;
// M   degree of highest derivative
// N+1 number of coefficients
double delta[][][]=new double[N1+1][N1+1][M+1];
double c1,c2,c3;
delta[0][0][0]=1.0;
c1=1.0;
for(int n=1;n<=N1;n++)
{  c2=1;
   for(int nu=0;nu<=(n-1);nu++)
   {c3=alpha[n]-alpha[nu];
    c2=c2*c3;
    if(n<=M) delta[n-1][nu][n]=0.0;
    for(int m=0;m<=Math.min(n,M);m++)
    {
       if(m==0)
       { delta[n][nu][m]=((alpha[n]-x0)*delta[n-1][nu][m])/c3;}
       else
       {delta[n][nu][m]=((alpha[n]-x0)*delta[n-1][nu][m]-m*delta[n-1][nu][m-1])/c3;}
    }//next m
   }//next nu
   for(int m=0;m<=Math.min(n,M);m++)
   {   if(m==0)
                {delta[n][n][m]=c1/c2*(-(alpha[n-1]-x0)*delta[n-1][n-1][m]);}
                else
                {delta[n][n][m]=c1/c2*(m*delta[n-1][n-1][m-1]-(alpha[n-1]-x0)*delta[n-1][n-1][m]);}
   }//next m
   c1=c2;
}//next n
c=delta;
  if(Mi<N) M=N;
  else M=Mi;
  h=hi;
  double deriv=0;
  double h1=1/h;
  double h2=1;
  for(int j=0;j<N;j++)
  {h2*=h1;}
  for(int i=0;i<c[0].length;i++)
  {   deriv+=c[M][i][N]*func(x+alpha[i]*h);}
  return deriv*h2;}
public  double dfunc(double x,int N)
{int M=30;double h=0.05*N;return dfunc(x,N,M,h);}
}
```

```
---------- Capture Output ----------
> "E:\co\java\bin\javaw.exe" f_x_test
x = 1.2 y = -0.56
```

```
> Terminated with exit code 0.
```

Program 1.5.2 Usage example of Abstract class f_x

```
class ff1 extends f_x
{ public double func(double x)
  {return x*x-2.0;}
}

public class f_x_test
{
  public static void main(String arg[])
  {
  ff1 f=new ff1();
  double x=1.2;
  System.out.println("x = "+x+" y = "+f.func(x));
  }
}
```

Interface example in java language. Function f(x)

Program 1.5.3 if_x.java Interface for function f(x)

```
import static java.lang.Math.*;
import java.io.*;
import java.util.*;
import javax.swing.*;
import javax.swing.table.*;
import java.awt.*;
import static java.lang.Math.*;
  // single function single independent variable
  // example f=x*x
  // includes full set of derivatives
  // Reference :"Generation of Finite Difference Formulas on Arbitrary Spaced Grids",
  // Bength Fornberg, Mathematics of Computation, Volume 51, Number 184, October 1988
  // pages 699-706
  //interface (empty function) version
@FunctionalInterface
interface if_x extends iMathd
{  public double func(double x);
//first order derivative
default double dfunc(double x)
{double h=1.0e-4;
 int n=1;
 int M=10;
 return dfunc(x,n,M,h);
}
default double dfunc(double x,double h)
{
 int n=1;
 int M=10;
 return dfunc(x,n,M,h);
}
//second order derivative
default double dfunc2(double x)
{double h=1.0e-6;
 int n=2;
 int M=10;
 return dfunc(x,n,M,h);
```

```
}
default double dfunc(double x,int N,int Mi,double hi)
{// order of the maximum derivative
 // N order of derivative
 // M degree of difference formula
double c[][][];
double alpha[];
double h;
int M=Mi;
double a[]=new double[0];
h=0.01;
double x0=0;
double alphai[]={0,1,-1,2,-2,3,-3,4,-4,5,-5,6,-6,7,-7,8,-8,9,-9,10,-10,11,-11,12,-12,13,-13,14,-14,15,
-15,16,-16,17,-17,18,-18,19,-19,20,-20,21,-21,22,-22,23,-23,24,-24,25,-25,26,-26,27,-27,28,-28,29,-29,30,-30,
31,-31,32,-32,33,-33,34,-34,35,-35,36,-36,37,-37,38,-38,39,-39,40,-40,41,-41,42,-42,43,-43,44,-44,45,-45,46,
-46,47,-47,48,-48,49,-49,50,-50,51,-51,52,-52,53,-53,54,-54,55,-55,56,-56,57,-57,58,-58,59,-59,60,-60,
-61,61,-62,62,-63,63,-64,64,-65,65,-66,66,-67,67,-68,68,-70,70,-71,71,-72,72,-73,73,-74,74,-75,75,
-76,76,-77,77,-78,78,-79,79,-80,80,-81,81,-82,82,-83,83,-84,84,-85,85,-86,86,-87,87,
-88,88,-89,89,-90,90,-91,91,-92,92,-93,93,-94,94,-95,95,-96,96,-97,97,-98,98,-99,99,-100,100};
alpha=alphai;
int N1=alpha.length-1;
// M   degree of highest derivative
// N+1 number of coefficients
double delta[][][]=new double[N1+1][N1+1][M+1];
double c1,c2,c3;
delta[0][0][0]=1.0;
c1=1.0;
for(int n=1;n<=N1;n++)
{  c2=1;
   for(int nu=0;nu<=(n-1);nu++)
   {c3=alpha[n]-alpha[nu];
    c2=c2*c3;
    if(n<=M) delta[n-1][nu][n]=0.0;
    for(int m=0;m<=Math.min(n,M);m++)
    {
       if(m==0)
       { delta[n][nu][m]=((alpha[n]-x0)*delta[n-1][nu][m])/c3;}
       else
       {delta[n][nu][m]=((alpha[n]-x0)*delta[n-1][nu][m]-m*delta[n-1][nu][m-1])/c3;}
    }//next m
   }//next nu
   for(int m=0;m<=Math.min(n,M);m++)
   {   if(m==0)
               {delta[n][n][m]=c1/c2*(-(alpha[n-1]-x0)*delta[n-1][n-1][m]);}
               else
               {delta[n][n][m]=c1/c2*(m*delta[n-1][n-1][m-1]-(alpha[n-1]-x0)*delta[n-1][n-1][m]);}
   }//next m
   c1=c2;
}//next n
c=delta;
  if(Mi<N) M=N;
  else M=Mi;
  h=hi;
  double deriv=0;
  double h1=1/h;
  double h2=1;
  for(int j=0;j<N;j++)
  {h2*=h1;}
  for(int i=0;i<c[0].length;i++)
```

```
  {   deriv+=c[M][i][N]*func(x+alpha[i]*h);}
  return deriv*h2;}
default double dfunc(double x,int N)
{int M=30;double h=0.01*N*N;return dfunc(x,N,M,h);}
default double[][] gauss_legendre_coefficients(double x1,double x2,int n)
{   //calculates legendre gauss-coefficients as coefficients of the integral
            //for n terms
            double EPS=3.0e-8;
            int m,j,i;
            double z1,z,xm,xl,pp,p3,p2,p1;
    //double x[]=new double[n];
    //double w[]=new double[n];
    double a[][]=new double[2][n];//a[0][i]=x[i]   a[1][i]=w[i]
            m=(n+1)/2;
            xm=0.5*(x2+x1);
            xl=0.5*(x2-x1);
            for (i=1;i<=m;i++)
            { z=Math.cos(Math.PI*((i-0.25)/(n+0.5)));
                        do {
                                    p1=1.0;
                                    p2=0.0;
                                    for (j=1;j<=n;j++) {
                                                p3=p2;
                                                p2=p1;
                                                p1=((2.0*j-1.0)*z*p2-(j-1.0)*p3)/j;
                                    }
                                    pp=n*(z*p1-p2)/(z*z-1.0);
                                    z1=z;
                                    z=z1-p1/pp;
                        } while (Math.abs(z-z1) > EPS);
                        a[0][i-1]=xm-xl*z;
                        a[0][n-i]=xm+xl*z;
                        a[1][i-1]=2.0*xl/((1.0-z*z)*pp*pp);
                        a[1][n-i]=a[1][i-1];
            }
return a;
}
default double integral(double x1,double x2)
{ int n=10;
  return integral(x1,x2,n);
}
default double integral(double x1,double x2,int n)
{
//n : number of integral coefficients
// this routine first generates gauss legendre coefficients
// for [x1,x2] band
// then calculates gauss legendre integral
double a[][]=new double[2][n];
a=gauss_legendre_coefficients(x1,x2,n);
double z=0;
for(int i=0;i<n;i++)
{ z+=a[1][i]*func(a[0][i]);
}
return z;
}
default double bisection(double a,double b)
  {double b1=1.1*b;
   double r=(a+b)/2.0;
   double eps=1.0e-6;
```

```
   int nmax=100;
   int i=1;
   while(Math.abs(func(r))>eps && i<nmax)
   {if(func(a)*func(r)<0) b=r;
    else a=r;
    r=(a+b)/2.0;;
    i++;
   }
   if(i>=nmax) r=bisection(a,b1);
   return r;
  }
default double secant(double x)
{ double eps=1.0e-10;
  double y=1.0;
  double dy=0.0;
  int miter=100;
  int i=0;
  while(abs(y)>eps && i<miter)
  {y=func(x);
   dy=dfunc(x);
   x=x-y/dy;
   i++;
  }
  if(i>=miter)
    System.out.println("i="+i+"results may not be valid");
  return x;
}
default double illinois(double xa,double xu)
{
//Modified False position root finding: Illinois method
double test;
double p=0;
double es,ea;
double fa,fp,fu;
int maxit=100,iter=0;
es=0.000001;
ea=1.1*es;
double xold;
int ia=0,iu=0;
fa=func(xa);
fu=func(xu);
p=xu;
double xpold;
while(iter<maxit && ea>es)
{
xpold=p;
p=xu-fu*(xa-xu)/(fa-fu);
fp=func(p);
if(signum(fp)!=signum(fu))
{xa=xu;fa=fu;}
else
{fa=fa/2;}
xu=p;fu=fp;
iter++;
if(p!=0) ea=abs((p-xpold)/p)*100.0;
}
if(iter>=maxit) System.out.println("Maximum number of iteration is exceeded ");
return p;
}
```

```
default double newton_bisection(double x1, double x2)
{
double xacc=1.0e-10; //accuracy
double h=0.0001; //derivative function step
int MAXIT=100;  //Maximum number of iterations
int j;
double dfr,dx,dxold,fr,fh,fl;
double temp,b,a,r;
fl=func(x1);
fh=func(x2);
//Check if a root is existed in the given region
if ((fl > 0.0 && fh > 0.0) || (fl < 0.0 && fh < 0.0))
System.out.println("There are no root in the given region or double root?");
if (fl == 0.0) return x1;
if (fh == 0.0) return x2;
if (fl < 0.0)   { a=x1;b=x2;}
else           { b=x1;a=x2;}
r=0.5*(x1+x2); //midpoint value
dxold=Math.abs(x2-x1);
dx=dxold;
fr=func(r);   //function value at midpoint
dfr=dfunc(r); //derivative of function value at midpoint
for (j=1;j<=MAXIT;j++)
{
if ((((r-b)*dfr-fr)*((r-a)*dfr-fr) > 0.0) || (Math.abs(2.0*fr) > Math.abs(dxold*dfr)))
   { //bisection step
            dxold=dx;
     dx=0.5*(b-a);
     r=a+dx;
     if (a == r) return r; //çözüm!!!!
   }
else{ //Newton-Raphson (Secant) step
            dxold=dx;
     dx=fr/dfr;
     temp=r;
     r -= dx;
     if (temp == r) return r;  //solution!!!!!
   }
if (Math.abs(dx) < xacc) return r; //solution!!!!!
fr=func(r);
dfr=dfunc(r);
if (fr < 0.0)
a=r;
else
b=r;
}
System.out.println("Maximum number of iterations are exceeded");
return 0.0; //program should never reach here
          //dummy return
}
default double brent(double a,double b)
{
double test;
double p=0;
double es,ea;
double f1,f2,f3,fp;
int maxit=500,iter=0;
double tol=1.0e-15;
es=0.0000001;
```

```
double x1=a;f1=func(x1);
double x2=b;f2=func(x2);
double x3=(x1+x2)/2.0;f3=func(x3);
if(f1==0) return x1;
else if(f2==0) return x2;
else if(f3==0) return x3;
if(f1*f2>0) System.out.println("No root is existed in the given region");
p=-(f2*f3*x1*(f2-f3)+f3*f1*x2*(f3-f1)+f1*f2*x3*(f1-f2))/((f1-f2)*(f2-f3)*(f3-f1));
fp=func(p);
ea=Math.abs(x3-p);
while((ea>es)&&(iter<maxit))
{           if(Math.abs(f3)<tol) return x3;
   if((p<x1) && (p>x2))
   {p=(x1+x2)/2.0;
    if(p>x3) {x1=x3;f1=f3;x3=p;f3=fp;}
    else if(p<x3) {x2=x3;f2=f3;x3=p;f3=fp;}
   }
   else
   {
   if(p>x3) {x1=x3;f1=f3;x3=p;f3=fp;}
   else if(p<x3) {x2=x3;f2=f3;x3=p;f3=fp;}
   p=-(f2*f3*x1*(f2-f3)+f3*f1*x2*(f3-f1)+f1*f2*x3*(f1-f2))/((f1-f2)*(f2-f3)*(f3-f1));
   fp=func(p);
   ea=Math.abs(x3-p);
   }
   iter++;
}
if(iter>=maxit) JOptionPane.showMessageDialog(null,"Warning : MAximum number of iteration is exceeded
\n"+
  " result may not be valid","MAXIMUM ITERATION NUMBER
WARNING",JOptionPane.WARNING_MESSAGE);
return p;
}
default double enlarge(double x0,double dx)
{  //enlarge region untill a root is existed
            double x1=x0;
            double x2=x1+dx;
            int NTRY=200;
            double a[]=new double[2];
            double FACTOR=1.001;
            int j;
            double f1,f2;
            if (x1 == x2) System.out.println("Giriş değerleri yanlış verildi");
            f1=func(x1);
            f2=func(x2);
            for (j=1;j<=NTRY;j++)
            { if (f1*f2 < 0.0) {break;}
              else {x2+=dx;f2=func(x2);}
            }
            return x2;
}
default void plot(double x1,double x2)
   {Plot p=new Plot(this,x1,x2);
    double x[]={x1,x2};
    double y[]={0,0};
    p.addData(x,y);
    p.plot();}
}
```

Program 1.5.4 Usage example of Interface if_x

```
class ff1 extends f_x
{ public double func(double x)
  {return x*x-2.0;}
}

public class f_x_test
{
  public static void main(String arg[])
  {
  ff1 f=new ff1();
  double x=1.2;
  System.out.println("x = "+x+" y = "+f.func(x));
  }
}
```

2.0 FINDING THE ROOTS OF EQUATION NON-LINEAR FUNCTIONS WITH ONE VARIABLE

Root finding is one of the most used numerical methods. For the one independent variable functions, it requires to find x value where the function is zero f(x)=0. If multidimensional (has more than one independent value) functions are involved, it requires to find $x_1,x_2,x_3,\ldots,x_n$ roots of n equation $f_i(x_1,x_2,x_3,\ldots,x_n)=0$, i=1..n

Some of the root finding methods will be investigated. As a starting point, we will start a simpler form of a function y=f(x). In this equation you have one root value..

2.1 BISECTION METHOD

In the bisection method root of the functions is searched in a given region $a \leq x \leq b$ If a single root is existed in the given region equation $f(a)f(b) < 0$ (2.1.1) will be satisfied. If this condition is not satisfied, it can be assumed that there is no root in the given region. Of course the reason will be existance of double root in the given region. If the root existed region is divide to two equal parts as

$$p = \frac{a+b}{2} \quad (2.1.2)$$

and function is evaluated in the middle. If

$$f(a) * f(p) < 0 \quad (2.1.3)$$

root should be located in a-p else it is located in p-b region. Rarely it is also possible that

$$f(a) * f(p) = 0 \quad (2.1.4)$$

can happen, that will indicate the root is located at point p. The new root region is used in further iterations. In order to use iterative process an exceptible criteria for error limit can be assumed

$$\left|\frac{(b-a)}{(b+a)}\right| < \epsilon \quad \text{or} \quad |f(p)| < \epsilon \quad (2.1.5)$$

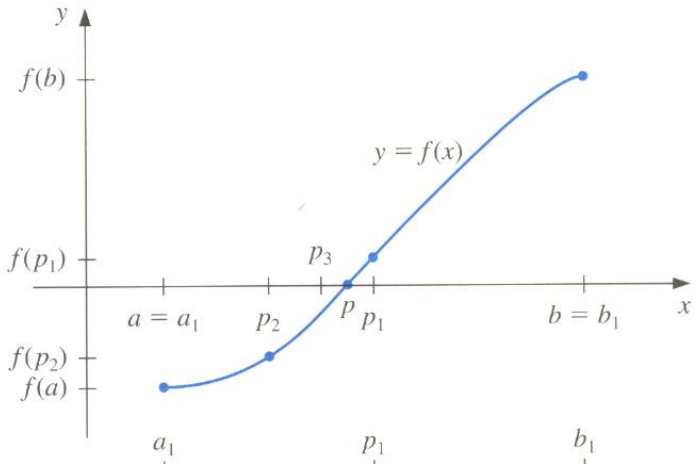


Figure 2.1 Convergence of bisection Method

An example code is created for the Bisection method. In order to define function as a general definition abstract class f_x defined in the following code. There are also defined methods for the derivatives of to function in this class. The details of the numerical derivatives will be defined in related chapter.

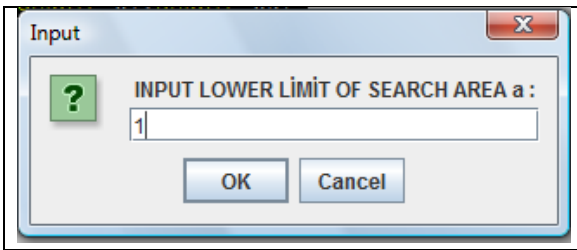
Program 2.1-2 Bisection method NA1

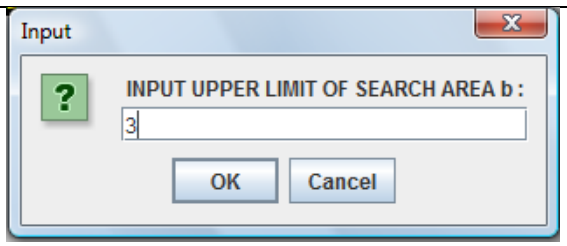
```
import javax.swing.*;

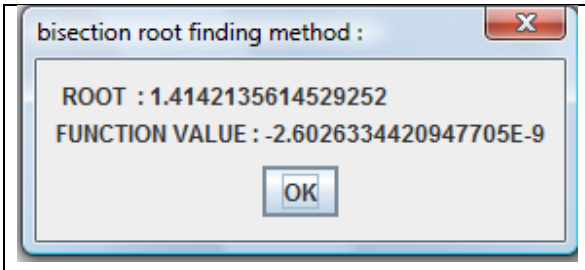
class f1 extends f_x
{
```

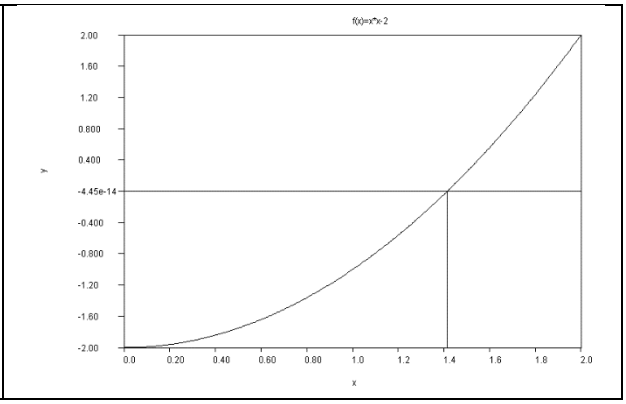
```
public double func(double x)
{return x*x-2;}
}
public class NA1
{

public static double bisection(f_x f,double xl,double xu)
{
//bisection root finding method
double test;
double xr=0;
double es,ea;
double fxl,fxr,fxu;
double xold=0;
int maxit=100,iter=0;
es=0.0000001;
ea=1.1*es;
String s="";
fxl= f.func(xl);
fxu= f.func(xu);
if(fxl*fxu>0) System.out.println("xl="+xu+"xr="+xr+"fxl="+fxl+"fxu="+fxu+"error");
while((ea>es)&&(iter<maxit))
{   xold=xr;
    xr=(xl+xu)/2.0;
    fxr= f.func(xr);
    iter++;
    if(xr!=0)
        { ea=Math.abs((xr-xold)/xr)*100;}
    test= fxl*fxr;
    if(test==0.0)      ea=0;
    else if(test<0.0) {xu=xr;fxu=fxr;}
    else if(test>0)   {xl=xr;fxl=fxr;}
    else              {ea=0;}
}
if(iter>=maxit) JOptionPane.showMessageDialog(null,"Maximum number of iteration is exceeded \n"+
   " result might not be valid","MAKSİMUM NUMBER OF ITERATION
WARNING",JOptionPane.WARNING_MESSAGE);
//JOptionPane.showMessageDialog(null,s,
//     "bisection root finding method : ",JOptionPane.PLAIN_MESSAGE);
  return xr;
}
     public static void main (String args[])
     {
     double a,b;
     a=Double.parseDouble(JOptionPane.showInputDialog(" INPUT LOWER LİMİT OF SEARCH AREA a : "));
     b=Double.parseDouble(JOptionPane.showInputDialog(" INPUT UPPER LIMIT OF SEARCH AREA b : "));
     double r;
     f1 f=new f1();
     r= bisection(f,a,b);
     JOptionPane.showMessageDialog(null," ROOT  : "+r+"\nFUNCTION VALUE : "+f.func(r),
     "bisection root finding method : ",JOptionPane.PLAIN_MESSAGE);
     System.exit(0);
     }
}
```









Bisection function definitions can also be given as **lambda expressions.** Lambda expressions are defined by using an empty referance class called **interface**. An interface class equivalent of the abstract class f_x is listed below

Program 2.1-2a Bisection method NA1a (lambda definitions)

```
 import static java.lang.Math.*;
import java.io.*;
import java.util.*;
import javax.swing.*;
import javax.swing.table.*;
import java.awt.*;

public class NA1a
{
public static double bisection(if_x f,double xl,double xu)
{
//bisection root finding method
double test;
double xr=0;
double es,ea;
double fxl,fxr,fxu;
double xold=0;
int maxit=100,iter=0;
es=0.0000001;
ea=1.1*es;
String s="";
fxl= f.func(xl);
fxu= f.func(xu);
if(fxl*fxu>0) System.out.println("xl="+xu+"xr="+xr+"fxl="+fxl+"fxu="+fxu+"error");
while((ea>es)&&(iter<maxit))
{   xold=xr;
   xr=(xl+xu)/2.0;
   fxr= f.func(xr);
   iter++;
   if(xr!=0)
      { ea=Math.abs((xr-xold)/xr)*100;}
   test= fxl*fxr;
   if(test==0.0)      ea=0;
   else if(test<0.0) {xu=xr;fxu=fxr;}
   else if(test>0)   {xl=xr;fxl=fxr;}
   else              {ea=0;}
}
if(iter>=maxit) JOptionPane.showMessageDialog(null,"Maximum number of iteration is exceeded \n"+
   " result might not be valid","MAKSİMUM NUMBER OF ITERATION WARNING",JOptionPane.WARNING_MESSAGE);
//JOptionPane.showMessageDialog(null,s,
//     "bisection root finding method : ",JOptionPane.PLAIN_MESSAGE);
   return xr;
```

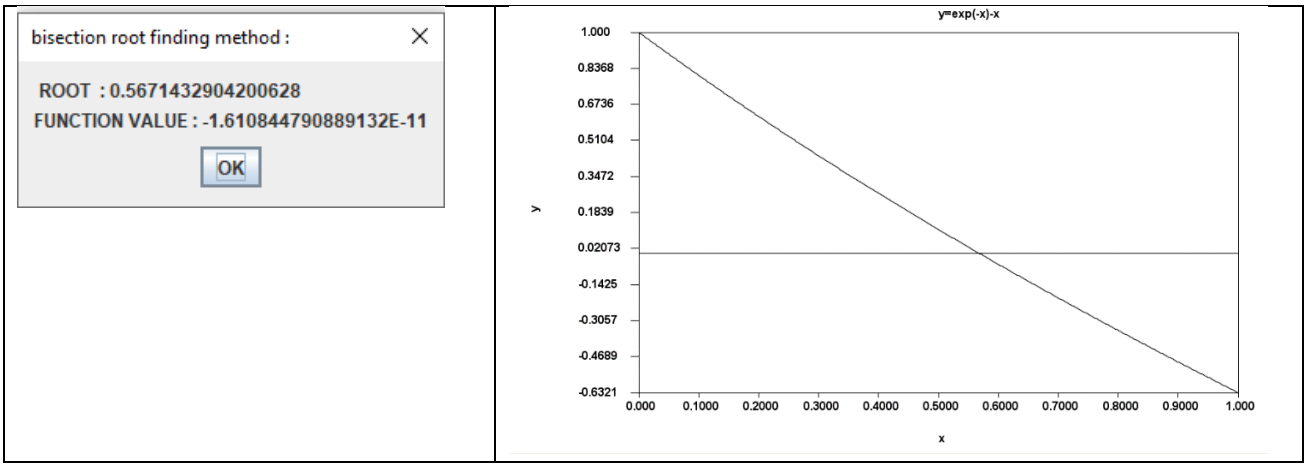
```
}
    public static void main (String args[]) throws IOException
    {
    double a,b;
    a=Double.parseDouble(JOptionPane.showInputDialog(" INPUT LOWER LİMİT OF SEARCH AREA a : "));
    b=Double.parseDouble(JOptionPane.showInputDialog(" INPUT UPPER LIMIT OF SEARCH AREA b : "));
    double r;
    if_x f1=x->x*x-2.0;
    r= bisection(f1,a,b);
    JOptionPane.showMessageDialog(null," ROOT  : "+r+"\nFUNCTION VALUE : "+f1.func(r),
    "bisection root finding method : ",JOptionPane.PLAIN_MESSAGE);
    System.exit(0);
    } }
```

In the next example root of $f(x) = e^{-x} - 1$ will be found by using bisectionmethod and lambda variable willbe used, but exp(x) function defined inside of the code and bisection method is called from NA1a class above.

```
import javax.swing.*;

public class NA1x
{
   public static double exp(double x)
   {double total=0;
    double factorial=1;
    double power=1;
    for(int i=1;i<=200;i++)
    {total=total+power/factorial;
     power=power*x;
     factorial=factorial*i;
    }
    return total;
   }

     public static void main (String args[])
     {
     double a,b;
     a=Double.parseDouble(JOptionPane.showInputDialog(" INPUT LOWER LİMİT OF SEARCH
AREA a : "));
     b=Double.parseDouble(JOptionPane.showInputDialog(" INPUT UPPER LIMIT OF SEARCH
AREA b : "));
     double r;
     if_x f1=x->exp(-x)-x;
     r= NA1a.bisection(f1,a,b);
     JOptionPane.showMessageDialog(null," ROOT  : "+r+"\nFUNCTION VALUE : "+f1.func(r),
     "bisection root finding method : ",JOptionPane.PLAIN_MESSAGE);
     System.exit(0);
     } }
```



Steps of bisection method is shown in detail in the following table(a spreadsheet solution) for function **$f(x)=x^2-2$**

Program 2.1-2b Bisection method in excel environment

xl	xu	xr	fxl	fxu	fxr	fxr*fxl
1	2	1.5	-1	2	0.25	-0.25
1	1.5	1.25	-1	0.25	-0.4375	0.4375
1.25	1.5	1.375	-0.4375	0.25	-0.10938	0.047852
1.375	1.5	1.4375	-0.10938	0.25	0.066406	-0.00726
1.375	1.4375	1.40625	-0.10938	0.066406	-0.02246	0.002457
1.40625	1.4375	1.421875	-0.02246	0.066406	0.021729	-0.00049
1.40625	1.421875	1.4140625	-0.02246	0.021729	-0.00043	9.6E-06
1.4140625	1.421875	1.41796875	-0.00043	0.021729	0.010635	-4.5E-06
1.4140625	1.41796875	1.416015625	-0.00043	0.010635	0.0051	-2.2E-06
1.4140625	1.416015625	1.415039063	-0.00043	0.0051	0.002336	-1E-06
1.4140625	1.415039063	1.414550781	-0.00043	0.002336	0.000954	-4.1E-07
1.4140625	1.414550781	1.414306641	-0.00043	0.000954	0.000263	-1.1E-07
1.4140625	1.414306641	1.41418457	-0.00043	0.000263	-8.2E-05	3.5E-08
1.41418457	1.414306641	1.414245605	-8.2E-05	0.000263	9.06E-05	-7.4E-09

In order to see the difference in the code if different languages is used an Octave (Matlab) code of bisection method is given below:

Program 2.1-2c Bisection method in Matlab(Octave) environment

```
function xr=bisection(f,xl,xu)
% bisection root finding method
maxit=100;
iter=0;
es=0.0000001;
ea=1.1*es;
while((ea>es)&&(iter<maxit))
   xr=(xl+xu)/2.0;
   iter=iter+1;
   if xr~=0 ea=abs((xu-xl)/(xu+xl))*100;end
   fxl= f(xl);
   fxr= f(xr);
   test= fxl*fxr;
   if test == 0.0      ea=0;
   elseif test < 0.0  xu=xr;
   else               xl=xr;
   end
end
if(iter>=maxit) fprintf('Maximum number of iteration is exceeded result might not be valid","MAKSİMUM NUMBER OF ITERATION WARNING');
end
end
```

To use the function defined above in Matlab editor, for the function f(x)=x*x-2. Note that function f1 is given as lambda definition also in matlab code

```
>> f1=@(x) x*x-2;
>> y=bisection(f1,0,2)
```

```
y =1.414213561452925
```

As it is seen function definition here is similar to lambda expressions in Java language.The same program given in C++ language. **Abstract class f_x** is also given as a part of the code.

Program 2.1-2d Bisection method in C++ language (lambda variable version)

```
  #include <stdio.h>
#include <iostream>
#include <math.h>
#include <iomanip>
using namespace std;
class f_x{
public: virtual double func(double x)=0;
public: double dfunc(double x)
{ double h=0.0001;
  double hh=1.0/h;
  double df=(3.0*func(x-4.0*h)-32.0*func(x-3.0*h)+168.0*func(x-2.0*h)-672.0*func(x-h)+672.0*func(x+h)-168.0*func(x+2.0*h)
+32.0*func(x+3.0*h) - 3.0*func(x+4.0*h))/840.0*hh;
  return df;
}
};
double bisection(f_x &f,double xl,double xu)
{
//bisection root finding method
double test;
double xr=0;
double es,ea;
double fxl,fxr;
int maxit=100;
int iter=0;
es=0.00000001;
ea=1.1*es;
while((ea>es)&&(iter<maxit))
{
    xr=(xl+xu)/2.0;
    iter++;
    if(xr!=0)
        { ea=fabs((xu-xl)/(xu+xl))*100.0;}
    fxl= f.func(xl);
    fxr= f.func(xr);
    test= fxl*fxr;
    if(test==0.0)      ea=0;
    else if(test<0.0)  xu=xr;
    else               {xl=xr;}
}
if(iter>=maxit) cout <<("Maximum number of iteration is exceeded \n result might not be valid");
return xr;
}
int main()
{  double a,b,r;
   class f1:public f_x{public:double func(double x){ return x*x-2; };};
   f1 f;
   cout << " INPUT LOWER LIMIT OF SEARCH AREA a : ";
   cin >>a;
   cout << " INPUT UPPER LIMIT OF SEARCH AREA b : ";
   cin >>b;
   r=bisection(f,a,b);
   cout<<setprecision(20);
   cout<<"r="<<r;
   return 0;
}
```

```
 ---------- Capture Output ----------
> "D:\okul\SCO1\run.bat" bisection

D:\okul\SCO1>bisection
 INPUT LOWER LIMIT OF SEARCH AREA a : 1
 INPUT UPPER LIMIT OF SEARCH AREA b : 3
r=1.4142135622678325
> Terminated with exit code 0.
```

Pyton definition of abstract if_x class

Program 2.1-1e Interface(abstract class) if_x in python language

```
 #include <stdio.h>
#include <iostream>
#include <math.h>
#include <iomanip>
using namespace std;
class if_x{
public: virtual double func(double x)=0;
public: double dfunc(double x)
{ double h=0.0001;
  double hh=1.0/h;
  double df=(3.0*func(x-4.0*h)-32.0*func(x-3.0*h)+168.0*func(x-2.0*h)-672.0*func(x-h)+672.0*func(x+h)-168.0*func(x+2.0*h)
+32.0*func(x+3.0*h) - 3.0*func(x+4.0*h))/840.0*hh;
  return df;
}
};
double bisection(if_x &f,double a,double b)
  {double b1=1.1*b;
   double r=(a+b)/2.0;
   double eps=1.0e-6;
   int nmax=100;
   int i=0;
   while(abs(f.func(r))>eps && i<nmax)
   {if(f.func(a)*f.func(r)<0) b=r;
    else a=r;
    r=(a+b)/2.0;;
    i++;
   }
   if(i>=nmax) r=bisection(f,a,b1);
   return r;
  }
int main()
{  double a,b,r;
   class f1:public if_x{public:double func(double x){ return x*x-2; };};
   f1 f;
   cout << " INPUT LOWER LIMIT OF SEARCH AREA a : ";
   cin >>a;
   cout << " INPUT UPPER LIMIT OF SEARCH AREA b : ";
   cin >>b;
   r=bisection(f,a,b);
   cout<<setprecision(20);
   cout<<"r="<<r;
   return 0;
}
```

Program 2.1-2f Bisection method in python language class f1 is defined as seperate entity

```
 from math import *
from if_x import *;

class f1(if_x):
  def func(self,x):
```

```
    y=x*x-2;
    return y;

def bisection(f,xl,xu):
  # bisection root finding method
  maxit=100
  iter=0
  es=0.0000001
  ea=1.1*es
  s=""
  fxl= f.func(xl)
  fxu= f.func(xu)
  xr=xl
  while ea>es and iter<maxit:
    xold=xr
    xr=(xl+xu)/2.0;
    fxr= f.func(xr);
    iter=iter+1;
    if xr!=0:
      ea=abs((xr-xold)/xr)*100
    test= fxl*fxr;
    if test==0.0 :      ea=0;
    elif test<0.0: xu=xr;fxu=fxr;
    elif test>0:    xl=xr;fxl=fxr;
    else :          ea=0;
    if iter>=maxit: print("Maximum number of iteration is exceeded \n result might not be valid")
  return xr

f=f1();
a=float(input("enter lower limit : "))
b=float(input("enter upper limit : "))
r=bisection(f,a,b);
print(" ROOT  : ",r,"\nFUNCTION VALUE : ",f.func(r));
```

Pyton version of the bisection program: Note that function f1 is given as lambda definition but not as a separate class

Program 2.1-3g Bisection method in python language

```
from math import *

def bisection(f,xl,xu):
  # bisection root finding method
  maxit=100
  iter=0
  es=0.0000001
  ea=1.1*es
  s=""
  fxl= f(xl)
  fxu= f(xu)
  xr=xl
  while ea>es and iter<maxit:
    xold=xr
    xr=(xl+xu)/2.0;
    fxr= f(xr);
    iter=iter+1;
    if xr!=0:
      ea=abs((xr-xold)/xr)*100
    test= fxl*fxr;
    if test==0.0 :      ea=0;
    elif test<0.0: xu=xr;fxu=fxr;
    elif test>0:    xl=xr;fxl=fxr;
    else :          ea=0;
    if iter>=maxit: print("Maximum number of iteration is exceeded \n result might not be valid")
  return xr
```

```
f1 = lambda x: x*x-2;
a=float(input("enter lower limit : "))
b=float(input("enter upper limit : "))
r=bisection(f1,a,b);
print(" ROOT  : ",r,"\nFUNCTION VALUE : ",f1(r));x = np.arange(0, 3, 0.1);
```

```
runfile('D:/okul/SCO1/bisection.py', wdir='D:/okul/SCO1')
enter lower limit : 1
enter upper limit : 3
ROOT  :  1.4142135614529252
FUNCTION VALUE :  -2.6026334420947705e-09
```

Program 2.1-3h Bisection method in python language class f1 is defined internally with lambda variable

```
class f1(if_x):func=lambda self,x:x*x-2;
a=float(input("enter lower limit : "))
b=float(input("enter upper limit : "))
r=bisection(f,a,b);
print(" ROOT  : ",r,"\nFUNCTION VALUE : ",f.func(r));
```

Program 2.1-3i Bisection method Fortran 90 version

```
program bisectionB
IMPLICIT NONE
REAL(KIND = SELECTED_REAL_KIND(10,34)) a,b
a=1.0
b=3.0
print *,'bisection(',a,',',b,') = ',bisection(a,b)

CONTAINS
  FUNCTION func(x)
  REAL(KIND = SELECTED_REAL_KIND(10,34)) func,x
  func=x*x-2.0
  END FUNCTION func

  FUNCTION bisection(a,b)
  REAL(KIND = SELECTED_REAL_KIND(10,34)) bisection,a,b,b1,r,eps,fa,fr
  INTEGER nmax,i
  eps=1.0e-8;
  nmax=100;
  i=0;
  fa=func(a)
  DO i=1,nmax
    r=(a+b)/2.0
    fr=func(r)
    IF (fr*fa<0) THEN
      b=r
    ELSE
      a=r
      fa=fr
    END IF
    IF (abs(fr)<eps) EXIT
  END DO
  bisection=r
  END FUNCTION bisection
END PROGRAM bisectionB
```

```
---------- Capture Output ----------
> "E:\okul\SCO1\run.bat" bisectionB

E:\okul\SCO1>bisectionB
```

```
bisection(   1.0000000000000000     ,   3.0000000000000000     ) =    1.4142135605216026

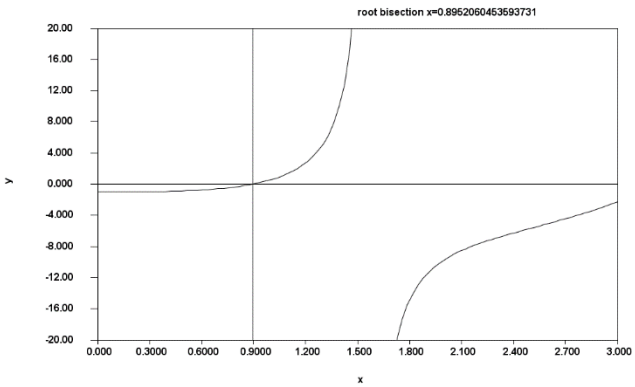
> Terminated with exit code 0.
```

From program in different programming languages you might notice that numerical analysis algorithms are almost independent of computer languages, and beside in all popular languages the code is very similar to each other.

Example Problem:

Use Bisection Method to Find the smallest positive root of $y = f(x) = x^2 \tan(x) - 1$

```
import java.io.*;
class PlotT01
{
public static double bisection(if_x f,double a,double b)
  {double b1=1.1*b;
   double r=(a+b)/2.0;
   double eps=1.0e-8;
   int nmax=100;
   int i=0;
   while(Math.abs(f.func(r))>eps && i<nmax)
   {if(f.func(a)*f.func(r)<0) b=r;
    else a=r;
    r=(a+b)/2.0;;
    i++;
   }
   if(i>=nmax) r=bisection(f,a,b1);
   return r;
  }
public static void main (String args[]) throws java.io.IOException
{  double ymin=-20;
   double ymax=20;
   double xmin=0;
   double xmax=3;
   int N=100;
   if_x f1=x->x*x*Math.tan(x)-1.0;
   if_x f2=x->0.0;
   Plot p=new Plot(f1,xmin,xmax,N);
   p.addFunction(f2,xmin,xmax,N);
   double x=bisection(f1,xmin,xmax);
   double xx[]={x,x};
   double yy[]={ymin,ymax};
   p.addData(xx,yy);
   p.setPlabel("root bisection x="+x);
   p.setYMinMax(ymin,ymax);
   p.plot();
}
}
```



2.2 FALSE POSITION METHOD (REGULA FALSI)

In bisection method search region was devided by two in each time. It is caused a relatively long interpolation period. In order to improve interpolation time, two points can be joined with a line and root of the line can be taken as the next iteration point .

$$p = b - f(b)\frac{(a-b)}{f(a)-f(b)} = \frac{bf(a)-af(b)}{f(a)-f(b)} \quad (2.2.1)$$

Region selection process is same as bisection process. If f(a)*f(p) < 0 root should be located in a-p else it is located in p-b region. The new root region is used in further iterations. In order to use iterative process an exceptible criteria for error limit can be assumed

$$\left|\frac{(b-a)}{(b+a)}\right| < \epsilon \quad \text{or} \quad |f(p)| < \epsilon \quad (2.2.2)$$

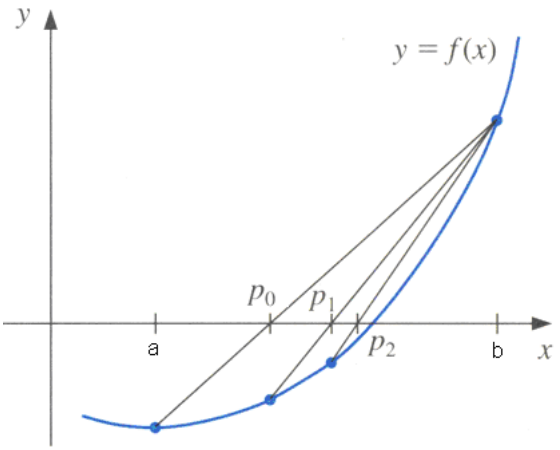


Figure 2.2 Convergence of the False position Method

Program 2.2-1 False position (Regula falsi) NA2

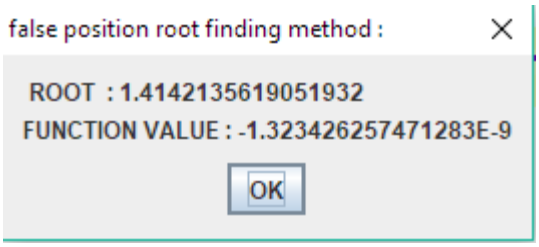
```
import static java.lang.Math.*;
import java.io.*;
import java.util.*;
import javax.swing.*;
import javax.swing.table.*;
import java.awt.*;

class f1 extends f_x
{
public double func(double x)
{ return x*x-2;}        }

public class NA2
{

public static double false_position(f_x f,double xl,double xu)
{
//false position root finding method
double test;
double xr=0;
double es,ea;
double fxl,fxr,fxu;
double xold=0;
int maxit=100,iter=0;
es=0.0000001;
ea=1.1*es;
String s="";
fxl= f.func(xl);
```

```
fxu= f.func(xu);
if(fxl*fxu>0) System.out.println("xl="+xu+"xr="+xr+"fxl="+fxl+"fxu="+fxu+"error");
while((ea>es)&&(iter<maxit))
{   xold=xr;
    xr=xu-fxu*(xl-xu)/(fxl-fxu);
    fxr= f.func(xr);
    iter++;
    if(xr!=0)
       { ea=Math.abs((xr-xold)/xr)*100;}
    test= fxl*fxr;
    if(test==0.0)      ea=0;
    else if(test<0.0) {xu=xr;fxu=fxr;}
    else if(test>0)   {xl=xr;fxl=fxr;}
    else              {ea=0;}
}
if(iter>=maxit) JOptionPane.showMessageDialog(null,"Maximum number of iteration is exceeded \n"+
   " result might not be valid","MAKSIMUM NUMBER OF ITERATION WARNING",JOptionPane.WARNING_MESSAGE);
   return xr;
}
     public static void main (String args[]) throws IOException
     {
     double a,b;
     a=Double.parseDouble(JOptionPane.showInputDialog(" INPUT LOWER LİMİT OF SEARCH AREA a : "));
     b=Double.parseDouble(JOptionPane.showInputDialog(" INPUT UPPER LIMIT OF SEARCH AREA b : "));
     double r;
     f1 f=new f1();
     r= false_position(f,a,b);
     JOptionPane.showMessageDialog(null," ROOT  : "+r+"\nFUNCTION VALUE : "+f.func(r),
     "false position root finding method : ",JOptionPane.PLAIN_MESSAGE);
     System.exit(0);
     }
}
```



Plot of the iterations through this example is shown in the plot.

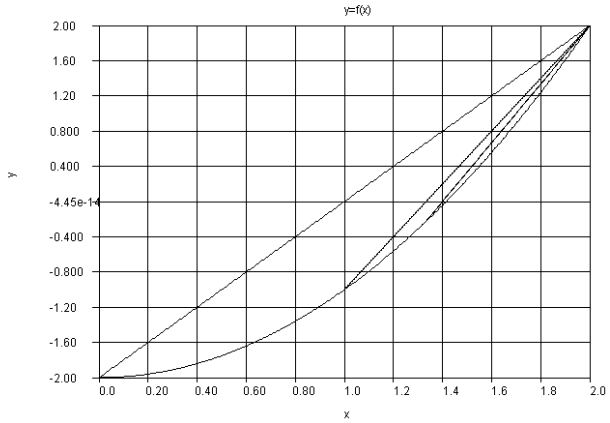


Figure 2.3 Convergence of the False position Method example program function $f(x)=x^2-2$

Steps of False position method is shown in detail in the following open Office calc (excel) program for function **$f(x)=x^2-2$**

Program 2.2-2 False position (Regula falsi) open Office calc (excel)

xl	xu	xr	fxl	fxu	fxr	fxr*fxl
1	2	1.333333333	-1	2	-0.22222	0.222222
1.333333333	2	1.4	-0.22222	2	-0.04	0.008889
1.4	2	1.411764706	-0.04	2	-0.00692	0.000277
1.411764706	2	1.413793103	-0.00692	2	-0.00119	8.23E-06
1.413793103	2	1.414141414	-0.00119	2	-0.0002	2.43E-07
1.414141414	2	1.414201183	-0.0002	2	-3.5E-05	7.14E-09
1.414201183	2	1.414211438	-3.5E-05	2	-6E-06	2.1E-10
1.414211438	2	1.414213198	-6E-06	2	-1E-06	6.19E-12
1.414213198	2	1.4142135	-1E-06	2	-1.8E-07	1.82E-13
1.4142135	2	1.414213552	-1.8E-07	2	-3E-08	5.37E-15
1.414213552	2	1.414213561	-3E-08	2	-5.2E-09	1.58E-16
1.414213561	2	1.414213562	-5.2E-09	2	-8.9E-10	4.65E-18
1.414213562	2	1.414213562	-8.9E-10	2	-1.5E-10	1.37E-19
1.414213562	2	1.414213562	-1.5E-10	2	-2.6E-11	4.03E-21
1.414213562	2	1.414213562	-2.6E-11	2	-4.5E-12	1.19E-22
1.414213562	2	1.414213562	-4.5E-12	2	-7.7E-13	3.49E-24
1.414213562	2	1.414213562	-7.7E-13	2	-1.3E-13	1.03E-25
1.414213562	2	1.414213562	-1.3E-13	2	-2.3E-14	3.04E-27
1.414213562	2	1.414213562	-2.3E-14	2	-4.2E-15	9.65E-29

As it is seen from the plot and table for iteration steps 2.0 is always stays and root search goes on between this two points. Even if solution is close to first point iterations did not cut from the left side of the search area. It may cause a long number of iteration steps.

Program 2.2-3 False position (Regula falsi) in C++

```
 #include <stdio.h>
#include <iostream>
#include <math.h>
#include <iomanip>
using namespace std;

class f_x{
public: virtual double func(double x)=0;
public: double dfunc(double x)
{ double h=0.0001;
  double hh=1.0/h;
  double df=(3.0*func(x-4.0*h)-32.0*func(x-3.0*h)+168.0*func(x-2.0*h)-672.0*func(x-h)+672.0*func(x+h)-168.0*func(x+2.0*h)
+32.0*func(x+3.0*h) - 3.0*func(x+4.0*h))/840.0*hh;
  return df;
}
};

double false_position(f_x &f,double xa,double xu)
{
//Modified False position root finding
double test;
double p=0;
double es,ea;
double fa,fp,fu;
int maxit=100,iter=0;
es=0.000001;
ea=1.1*es;
int ia=0,iu=0;
fa=f.func(xa);
fu=f.func(xu);
p=xu;
double xpold;
while(iter<maxit && ea>es)
```

```
{
xpold=p;
p=xu-fu*(xa-xu)/(fa-fu);
fp=f.func(p);
iter++;
if(p!=0) ea=fabs((p-xpold)/p)*100.0;
test=fa*fp;
if(test<0) {xu=p;fu=f.func(xu);iu=0;ia++;if(ia>=2) fa/=2.0;}
else if(test>0) {xa=p;fa=f.func(xa);ia=0;iu+=1;if(iu>=2)fu/=2.0;}
else ea=0;
}
if(iter>=maxit) std::cout <<("Maximum number of iteration is exceeded \n result might not be valid");
return p;
}

int main()
{  double a,b,r;
  class f1:public f_x{public:double func(double x){ return x*x-2; };};
  f1 f;
  cout << " INPUT LOWER LIMIT OF SEARCH AREA a : ";
  cin >>a;
  cout << " INPUT UPPER LIMIT OF SEARCH AREA b : ";
  cin >>b;
  r=false_position(f,a,b);
  cout <<setprecision(20)<<"r="<<r;
  return 0;
}
```

```
 ---------- Capture Output ----------
> "D:\okul\SCO1\run.bat" false_position

D:\okul\SCO1>false_position
 INPUT LOWER LIMIT OF SEARCH AREA a : 1
 INPUT UPPER LIMIT OF SEARCH AREA b : 3
r=1.4142135623730951
> Terminated with exit code 0.
```

I will try the same code in Python language now

Program 2.2-4 False position (Regula falsi) false_position.py python code

```
 from math import *

def false_position(f,xl,xu):
  # bisection root finding method
  maxit=100
  iter=0
  es=0.0000001
  ea=1.1*es
  s=""
  fxl= f(xl)
  fxu= f(xu)
  xr=xl
  while ea>es and iter<maxit:
    xold=xr
    xr=xu-fxu*(xl-xu)/(fxl-fxu);
    fxr= f(xr);
    iter=iter+1;
    if xr!=0:
      ea=abs((xr-xold)/xr)*100
    test= fxl*fxr;
    if test==0.0 :      ea=0;
    elif test<0.0: xu=xr;fxu=fxr;
    elif test>0:    xl=xr;fxl=fxr;
```

```
    else :          ea=0;
    if iter>=maxit: print("Maximum number of iteration is exceeded \n result might not be valid")
  return xr

f1 = lambda x: x*x-2;
a=float(input("enter lower limit : "))
b=float(input("enter upper limit : "))
r=false_position(f1,a,b);
print(" ROOT  : ",r,"\nFUNCTION VALUE : ",f1(r))
```

```
---------- Capture Output ----------
> "D:\co\anaconda\pythonw.exe" false_position.py
enter lower limit : 1
enter upper limit : 3
 ROOT  :  1.4142135617510516
FUNCTION VALUE :  -1.759404621992644e-09

> Terminated with exit code 0.
```

Program 2.2-5 False position (Regula falsi) fortran 90 code

```
program false_position
IMPLICIT NONE
REAL(KIND = SELECTED_REAL_KIND(10,34)) a,b
a=1.0
b=2.15
print *,'false position(',a,',',b,') = ',false(a,b)

CONTAINS
  FUNCTION func(x)
  REAL(KIND = SELECTED_REAL_KIND(10,34)) func,x
  func=x*x-2.0
  END FUNCTION func

  FUNCTION false(a,b)
  REAL(KIND = SELECTED_REAL_KIND(10,34)) false,a,b,b1,r,eps,fa,fb,fr
  INTEGER nmax,i
  eps=1.0e-8;
  nmax=100;
  i=0;
  fa=func(a)
  fb=func(b)
  DO i=1,nmax
    r=b-fb*(a-b)/(fa-fb)
    fr=func(r)
    IF (fr*fa<0) THEN
      b=r
      fb=fr
    ELSE
      a=r
      fa=fr
    END IF
    IF (abs(fr)<eps) EXIT
  END DO
  false=r
  END FUNCTION false
END PROGRAM false_position
```

```
---------- Capture Output ----------
> "E:\okul\SCO1\run.bat" false_position

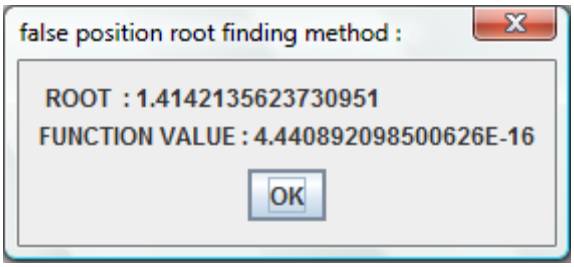
E:\okul\SCO1>false_position
 false position(   1.0000000000000000      ,   2.1500000953674316      ) =    1.4142135594660559
```

```
> Terminated with exit code 0.
```

In order to overcome the difficulty of one sided cut of the region a bisection step can be put in with some regular intervals. For example once in every 4 false position steps. This is called **modified false position** method. The next example looks the root of $f(x)=x^2-2$ with modified false position method.

Program 2.2-6 Modified False position (Regula falsi) NA3 java code

```
 import static java.lang.Math.*;
import java.io.*;
import java.util.*;
import javax.swing.*;
import javax.swing.table.*;
import java.awt.*;
public class NA3
{
public static double false_position(f_x f,double xl,double xu)
{
//false position root finding method
double test;
double xr=0;
double es,ea;
double fxl,fxr,fxu;
double xold=0;
int maxit=100,iter=0;
es=0.0000001;
ea=1.1*es;
String s="";
fxl= f.func(xl);
fxu= f.func(xu);
if(fxl*fxu>0) System.out.println("xl="+xu+"xr="+xr+"fxl="+fxl+"fxu="+fxu+"error");
while((ea>es)&&(iter<maxit))
{   xold=xr;
   if(iter%4!=0)
   {xr=xu-fxu*(xl-xu)/(fxl-fxu);}
   else
   {xr=(xl+xu)/2.0;}
   fxr= f.func(xr);
   iter++;
   if(xr!=0)
      { ea=Math.abs((xr-xold)/xr)*100;}
   test= fxl*fxr;
   if(test==0.0)      ea=0;
   else if(test<0.0) {xu=xr;fxu=fxr;}
   else if(test>0)   {xl=xr;fxl=fxr;}
   else              {ea=0;}
}
if(iter>=maxit) JOptionPane.showMessageDialog(null,"Maximum number of iteration is exceeded \n"+
  " result might not be valid","MAKSIMUM NUMBER OF ITERATION WARNING",JOptionPane.WARNING_MESSAGE);
  return xr;
}
    public static void main (String args[]) throws IOException
    {
    double a,b;
    class f1 extends f_x{public double func(double x){return x*x-2;}};
    f1 f=new f1();
    a=Double.parseDouble(JOptionPane.showInputDialog(" INPUT LOWER LİMİT OF SEARCH AREA a : "));
    b=Double.parseDouble(JOptionPane.showInputDialog(" INPUT UPPER LIMIT OF SEARCH AREA b : "));
    double r;
    r= false_position(f,a,b);
    JOptionPane.showMessageDialog(null," ROOT  : "+r+"\nFUNCTION VALUE : "+f.func(r),
    "false position root finding method : ",JOptionPane.PLAIN_MESSAGE);
    System.exit(0);
    } }
```



Program 2.2-7 Modified False position (Regula falsi) in python code

```
import numpy as np
import matplotlib.pyplot as plt
from math import *

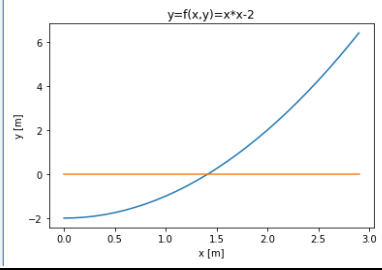
def f1(x):
  return x*x-2.0

def false_position(xl,xu):
  # bisection root finding method
  maxit=100
  iter=0
  es=0.0000001
  ea=1.1*es
  s=""
  fxl= f1(xl)
  fxu= f1(xu)
  xr=xl
  while ea>es and iter<maxit:
    xold=xr
    if iter%4!=0:
      xr=xu-fxu*(xl-xu)/(fxl-fxu);
    else:
      xr=(xl+xu)/2.0;
    fxr= f1(xr);
    iter=iter+1;
    if xr!=0:
      ea=abs((xr-xold)/xr)*100
    test= fxl*fxr;
    if test==0.0 :      ea=0;
    elif test<0.0: xu=xr;fxu=fxr;
    elif test>0:    xl=xr;fxl=fxr;
    else :          ea=0;
    if iter>=maxit: print("Maximum number of iteration is exceeded \n result might not be valid")
  return xr

x = np.arange(0, 3, 0.1);
y = f1(x)
y1=0.0*x
plt.title('y=f(x,y)=x*x-2')
plt.xlabel('x [m]')
plt.ylabel('y [m]')
plt.plot(x, y,x,y1)
print(" Modified false position: ")
a=float(input("enter lower limit : "))
b=float(input("enter upper limit : "))
r=false_position(a,b);
print(" ROOT  : ",r,"\nFUNCTION VALUE : ",f1(r))
```

```
Modified false position:
```

```
enter upper limit : 3
 ROOT   :  1.4142135623729 46
FUNCTION VALUE :  -4.2166270475263445e-13
```



As it is seen no problem arise finding the root of this function in modified . Another modification of false position method is the **Illinois method**. In this method function value is halved (divided by two) whenever end point is reused.

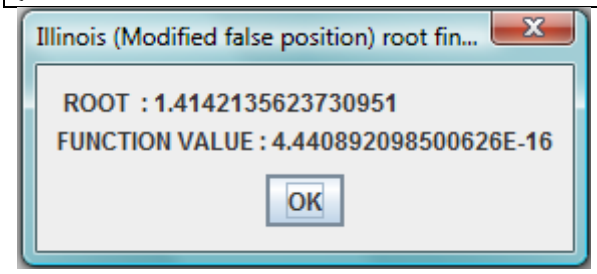
Program 2.2-8 Modified False position Illinois method (Regula falsi) NA3A (java code)

```
 import java.util.*;
import java.awt.*;
import java.applet.Applet;
import java.awt.event.*;
import javax.swing.*;
//============= Function to be define ================

class fb extends f_x
{
  public double func (double x)
  { return x*x-2.0;}
}
//============================================================
public class NA3A
{
public static double illinois(f_x f,double xa,double xu)
{
//Modified False position root finding: Illinois method
double test;
double p=0;
double es,ea;
double fa,fp,fu;
int maxit=100,iter=0;
es=0.000001;
ea=1.1*es;
double xold;
int ia=0,iu=0;
fa=f.func(xa);
fu=f.func(xu);
p=xu;
double xpold;
while(iter<maxit && ea>es)
{
xpold=p;
p=xu-fu*(xa-xu)/(fa-fu);
fp=f.func(p);
if(Math.signum(fp)!=Math.signum(fu))
{xa=xu;fa=fu;}
else
{fa=fa/2;}
xu=p;fu=fp;
iter++;
if(p!=0) ea=Math.abs((p-xpold)/p)*100.0;
}
```

```
if(iter>=maxit) JOptionPane.showMessageDialog(null,"Maximum number of iteration is exceeded \n"+
   " result might not be valid","MAKSİMUM NUMBER OF ITERATION WARNING",JOptionPane.WARNING_MESSAGE);
return p;
}

    public static void main (String args[])
    {
   double a,b;
    a=Double.parseDouble(JOptionPane.showInputDialog(" INPUT LOWER LİMİT OF SEARCH AREA a : "));
    b=Double.parseDouble(JOptionPane.showInputDialog(" INPUT UPPER LIMIT OF SEARCH AREA b : "));
    double r;
    fb f=new fb();
    r= illinois(f,a,b);
    JOptionPane.showMessageDialog(null," ROOT  : "+r+"\nFUNCTION VALUE : "+f.func(r),
    "Illinois (Modified false position) root finding method : ",JOptionPane.PLAIN_MESSAGE);
    System.exit(0);
    }
}
```



Modified fixed position method code in matlab is given below to compare:

Program 2.2-9 Modified False position (Regula falsi) Matlab (Octave) code: false_positon.m

```
function p=false_position(f,xa,xu)
% modified false position root finding method
maxit=100;
iter=0;
es=0.000001;
ea=1.1*es;
ia=0;
iu=0;
fa=f(xa);
fu=f(xu);
p=xu;
double xpold;
while(iter<maxit && ea>es)
xpold=p;
p=xu-fu*(xa-xu)/(fa-fu);
fp=f(p);
iter=iter+1;
if p~=0 ea=abs((p-xpold)/p)*100.0;
test=fa*fp;
if test<0
   xu=p;
   fu=f(xu);
   iu=0;ia=ia+1;
   if ia >= 2 fa=fa/2.0;end
elseif test>0  xa=p;fa=f(xa);ia=0;iu=iu+1;
      if iu>=2 fu=fu/2.0; end
elseif test==0 ea = 0; end
end
if iter>=maxit fprintf('Maximum number of iteration is exceeded result might not be valid'); end
end
```

```
>> f1=@(x) x*x-2;
>> false_position(f1,0,3)
ans =1.414213562373074
>>
```

Modified false position method given as c++ code is listed below:

Program 2.2-10 Modified False position (Regula falsi) c++ code false_position.cpp

```
 #include <stdio.h>
#include <iostream>
#include <math.h>
#include <iomanip>
using namespace std;

class f_x{
public: virtual double func(double x)=0;
public: double dfunc(double x)
{ double h=0.0001;
  double hh=1.0/h;
  double df=(3.0*func(x-4.0*h)-32.0*func(x-3.0*h)+168.0*func(x-2.0*h)-672.0*func(x-h)+672.0*func(x+h)-168.0*func(x+2.0*h)
+32.0*func(x+3.0*h) - 3.0*func(x+4.0*h))/840.0*hh;
  return df;
}
};

class f1:public f_x
{ public: double func(double x)
  { double ff=x*x-2.0;
                return ff;
  }
};

double false_position(f_x &f,double xa,double xu)
{
//Modified False position root finding
double test;
double p=0;
double es,ea;
double fa,fp,fu;
int maxit=100,iter=0;
es=0.000001;
ea=1.1*es;
int ia=0,iu=0;
fa=f.func(xa);
fu=f.func(xu);
p=xu;
double xpold;
while(iter<maxit && ea>es)
{
xpold=p;
p=xu-fu*(xa-xu)/(fa-fu);
fp=f.func(p);
iter++;
if(p!=0) ea=fabs((p-xpold)/p)*100.0;
test=fa*fp;
if(test<0) {xu=p;fu=f.func(xu);iu=0;ia++;if(ia>=2) fa/=2.0;}
else if(test>0) {xa=p;fa=f.func(xa);ia=0;iu+=1;if(iu>=2)fu/=2.0;}
else ea=0;
}
if(iter>=maxit) std::cout <<("Maximum number of iteration is exceeded \n result might not be valid");
return p;
}

int main()
{  double a,b,r;
   f1 f;
   cout << " INPUT LOWER LIMIT OF SEARCH AREA a : ";
   cin >>a;
   cout << " INPUT UPPER LIMIT OF SEARCH AREA b : ";
```

```
  cin >>b;
  r=false_position(f,a,b);
  cout<<setprecision(20);
  cout<<"r="<<r;
  return 0;
}
```

```
---------- Capture Output ----------
> "D:\okul\SCO1\run.bat" false_position1

D:\okul\SCO1>false_position1
 INPUT LOWER LIMIT OF SEARCH AREA a : 1
 INPUT UPPER LIMIT OF SEARCH AREA b : 3
r=1.4142135623730951
> Terminated with exit code 0.
```

Program 2.2-11 Modified False position (Regula falsi) Fortran 90 code

```
program modified_false_position
IMPLICIT NONE
REAL(KIND = SELECTED_REAL_KIND(10,34)) a,b
a=1.0
b=2.15
print *,'false position(',a,',',b,') = ',false(a,b)

CONTAINS
 FUNCTION func(x)
 REAL(KIND = SELECTED_REAL_KIND(10,34)) func,x
 func=x*x-2.0
 END FUNCTION func

 FUNCTION false(a,b)
 REAL(KIND = SELECTED_REAL_KIND(10,34)) false,a,b,b1,r,eps,fa,fb,fr,remain
 INTEGER nmax,i
 eps=1.0e-8;
 nmax=100;
 i=0;
 fa=func(a)
 fb=func(b)
 DO i=1,nmax
  remain=i-(i/4)*4
  if(remain==0) then
   r=(a+b)/2.0
  else
   r=b-fb*(a-b)/(fa-fb)
  end if
  fr=func(r)
  IF (fr*fa<0) THEN
   b=r
   fb=fr
  ELSE
   a=r
   fa=fr
  END IF
  IF (abs(fr)<eps) EXIT
 END DO
 false=r
 END FUNCTION false
END PROGRAM modified_false_position
```

```
---------- Capture Output ----------
> "E:\okul\SCO1\run.bat" modified_false_position
```

```
E:\okul\SCO1>modified_false_position
 false position(  1.0000000000000000   ,  2.1500000953674316   ) =  1.4142135609373101

> Terminated with exit code 0.
```

2.3 FIXED POSITION METHOD WITH AITKEN EXTRAPOLATION (STEPHENSEN METHOD)

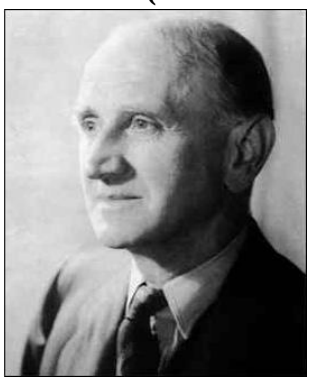


Figure 2.3.1 Alexander Craig Aitken

Fixed position method is relatively simple. When a function f(x) is given to find the root, the function f(x) converted with algebraic manupulations and converted to f(x)=g(x) – x form. Since this equation can be written in as g(x)=x, starting from an initial guess, iteration continued till x and g(x) gives the same x value. Even though the relative simplicity, this method is not used much due to high failure possibility. The equation can diverge from the root sometimes. In order to overcome this problem aitken extrapolation process will be combined with fixed point , iteration. Method is called Stephensen method.

Aitken extrapolation process is a error reduction method. For a very big k number

$$\frac{x_{k+2}-a}{x_{k+1}-a} \approx \frac{x_{k+1}-a}{x_k-a} \quad k \gg 1 \qquad (2.3.1)$$

Relation can be valid. If the value of fourth point desired to be calculated from this equation

$$x = \frac{x_k x_{k+2} - x_{k+1}^2}{2x_k + 2x_{k+1} + 2x_{k+2}} \qquad (2.3.2)$$

When 3 points are previously known, a third one can be evaluated

$$x_k = x_{k+1} - x_k \qquad (2.3.3)$$

$$\Delta^2 x_k = x_{k+2} - 2x_{k+1} + x_k \qquad (2.3.4)$$

$$x_e = x_{k+2} - \frac{(\Delta x_k)^2}{\Delta^2 x_k} \qquad (2.3.5)$$

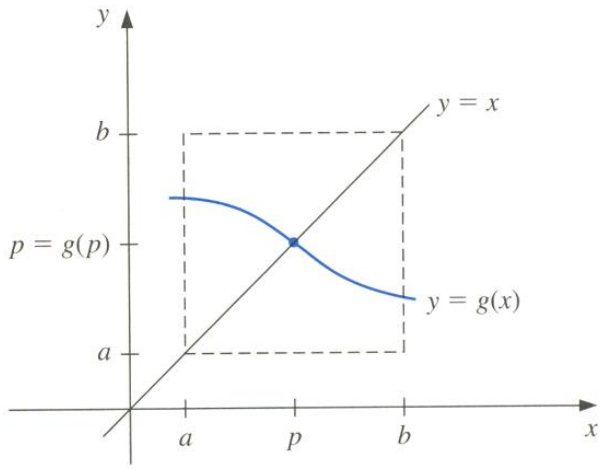


Figure 2.4 Convergence of the Fixed iteration Method

Stephensen method evaluates equation by using an aitken extrapolation step after two step of fixed point iteration.

Program 2.3-1 : Fixed point iteration (Stephensen method) : (java code)

```
// Stephensen method x=g(x)

import java.util.*;
import java.awt.*;
import java.applet.Applet;
import java.awt.event.*;
import javax.swing.*;
//============= Tanimi gereken fonksiyon ================
class fb extends f_x
{  //x = g(x) fonction
   //f(x)=x-Math.exp(-x)
   public double func (double x)
   { return Math.exp(-x);}
}

//=========================================================
public class NA4
{

public static double stephensen(f_x f,double x0)
{
//Stephensen root finding method
double es,ea;
int maxit=100,iter=0;
es=0.000001;
ea=1.1*es;
double x,x1,x2;
x=0;
while(iter<maxit && ea>es)
{
x1=f.func(x0);
x2=f.func(x1);
x=x0-(x1-x0)*(x1-x0)/(x2-2*x1+x0);
ea=Math.abs(x-x0)*100.0;
x0=x;
iter++;
}
if(iter>=maxit) JOptionPane.showMessageDialog(null,"Maximum number of iteration is exceeded \n"+
   " result might not be valid","MAKSİMUM NUMBER OF ITERATION WARNING",JOptionPane.WARNING_MESSAGE);
return x;
}
    public static void main (String args[])
    {
    fb g=new fb();
    double xa,xu;
    xa=Double.parseDouble(JOptionPane.showInputDialog(" first guess for the root value : "));
    double r= stephensen(g,xa);
    double fr=g.func(r);
    String s=" root value  : "+r+"\ng(x) = "+fr+"\nf(x)=g(x)-x = "+(fr-r);
    JOptionPane.showMessageDialog(null,s,
    "Stephensen root finding method : ",JOptionPane.PLAIN_MESSAGE);
    } }
```

Stephensen Method is simple enough to calculate in excel format: The example problem is :

$$f(x) = x - \sqrt{\frac{5}{x+1}} = x - g(x) \qquad\qquad g(x) = \sqrt{\frac{5}{x+1}}$$

Program 2.3-2 : Fixed point iteration (Stephensen method) with excel:

Fixedİteration

f(x)=x-(5/(x+1)^0.5

Aitken		dxk	d2xk	xe=xk-(dxk)^2/d2xk	g(x)=(5/(x+1))^0.5
	x				
0	0.5				1.825741858
0	1.825741858	1.325741858			1.330205563
0	1.330205563	-0.495536296	-1.821278154	1.465031877	1.464832223
1	1.465031877	0.134826314	0.63036261	1.436194292	1.424209021
0	1.424209021	-0.040822856	-0.17564917	1.433696715	1.436150556
0	1.436150556	0.011941535	0.052764391	1.433447971	1.432626367

0	1.432626367	-0.003524189	-0.015465724	1.433429427	1.433663727
1	1.433429427	0.00080306	0.004327249	1.433280394	1.433427145
0	1.433427145	-2.28264E-06	-0.000805343	1.433427151	1.433427817
0	1.433427817	6.72303E-07	2.95494E-06	1.433427664	1.433427619
0	1.433427619	-1.98012E-07	-8.70315E-07	1.433427664	1.433427677
1	1.433427664	4.50514E-08	2.43064E-07	1.433427656	1.433427664
0	1.433427664	-7.32747E-15	-4.50514E-08	1.433427664	1.433427664
0	1.433427664	1.9984E-15	9.32587E-15	1.433427664	1.433427664
0	1.433427664	0	-2.44249E-15	1.433427664	1.433427664

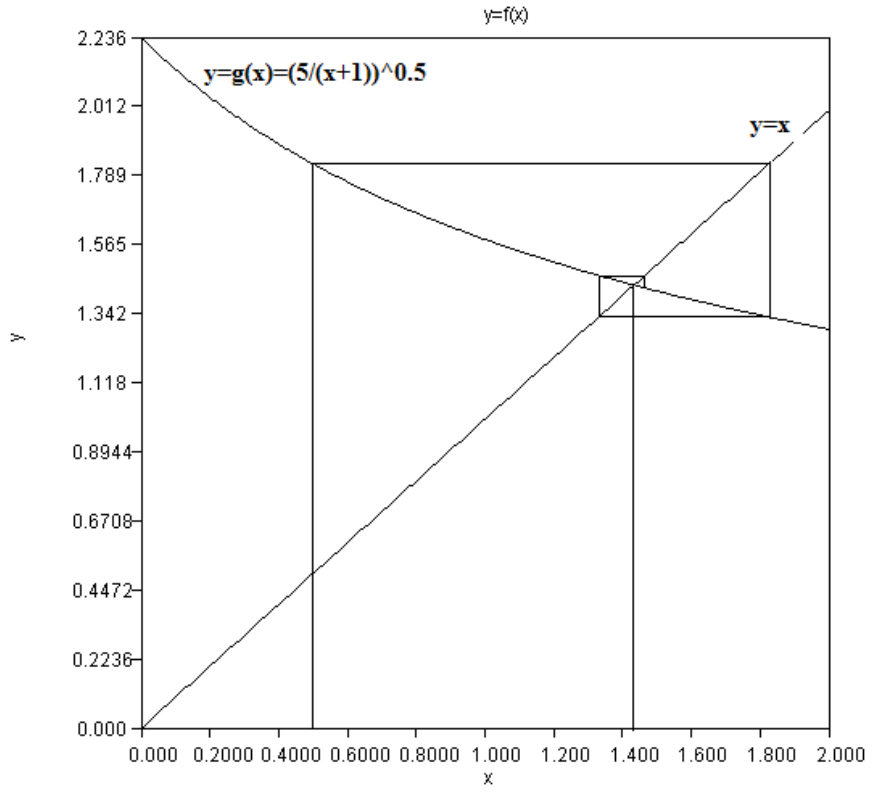


Figure 2.5 Convergence of the Fixed position Method

If the same problem is worked through the Matlab:

Program 2.3-3 : Fixed point iteration (Stephensen method) with Matlab(Octave):

```
function x = stephensen(f,x0)
%Fixed point iteration with Aitken extrapolation
%Stephensen Method
maxit=100;
iter=0;
es=0.000001;
ea=1.1*es;
x=0;
while iter<maxit && ea>es
x1=f(x0);
x2=f(x1);
x=x0-(x1-x0)*(x1-x0)/(x2-2*x1+x0);
ea=abs(x-x0)*100.0;
x0=x;
iter=iter+1;
end
if iter>=maxit  fprintf('Maximum number of iteration is exceeded  result might not be valid');end
end
```

```
>> g=@(x) sqrt(5/(x+1));
>> stephensen(g,1)
ans =
  1.433427663863820
```

C++ code of the stephensen method is given below:

Program 2.3-4 : Fixed point iteration (Stephensen method) in c++ language

```
 #include <stdio.h>
#include <iostream>
#include <math.h>
#include <iomanip>
using namespace std;

class f_x{
public: virtual double func(double x)=0;
public: double dfunc(double x)
{ double h=0.0001;
  double hh=1.0/h;
  double df=(3.0*func(x-4.0*h)-32.0*func(x-3.0*h)+168.0*func(x-2.0*h)-672.0*func(x-h)+672.0*func(x+h)-
168.0*func(x+2.0*h) +32.0*func(x+3.0*h) - 3.0*func(x+4.0*h))/840.0*hh;
  return df;
}};

double stephensen(f_x &f,double x0)
{
//Stephensen root finding method
double es,ea;
int maxit=100,iter=0;
es=0.000001;
ea=1.1*es;
double x,x1,x2;
x=0;
while(iter<maxit && ea>es)
{
x1=f.func(x0);
x2=f.func(x1);
x=x0-(x1-x0)*(x1-x0)/(x2-2*x1+x0);
ea=fabs(x-x0)*100.0;
x0=x;
iter++;
}
if(iter>=maxit) std::cout <<("Maximum number of iteration is exceeded \n result might not be valid");
return x;
}

int main()
{  double xa,r;
   class f1:public f_x{ public: double func(double x) {return sqrt(5.0/(x+1.0));}};
   f1 f;
   cout << " first guess for the root value : ";
   cin >>xa;
   r=stephensen(f,xa);
   cout<<setprecision(20);
   cout<<"root value : = "<<r<<" fr = "<<f.func(r)-r;
   return 0;
}
```

```
 ---------- Capture Output ----------
> "D:\okul\SCO1\run.bat" stephensen
D:\okul\SCO1>stephensen
```

```
 first guess for the root value : 1
root value : = 1.4334276638638197 fr = 0
> Terminated with exit code 0.
```

Program 2.3-5 : Fixed point iteration (Stephensen method) in Python code

```
 import numpy as np
import matplotlib.pyplot as plt
from math import *

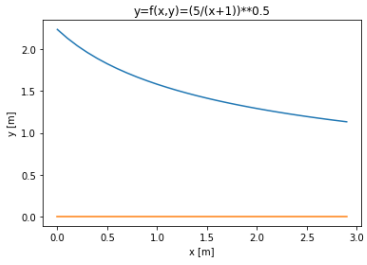
def f1(x):
  return (5.0/(x+1.0))**0.5

def stephensen(x0):
  # Fixed point iteration with Aitken extrapolation
  # Stephensen Method
  maxit=100;
  iter=0;
  es=0.000001;
  ea=1.1*es;
  x=0;
  while iter<maxit and ea>es:
    x1=f1(x0);
    x2=f1(x1);
    x=x0-(x1-x0)*(x1-x0)/(x2-2*x1+x0);
    ea=abs(x-x0)*100.0;
    x0=x;
    iter=iter+1;
    if iter>=maxit:  print("Maximum number of iteration is exceeded  result might not be valid")
  return x

x = np.arange(0, 3, 0.1);
y = f1(x)
y1 = 0.0*x
plt.title('y=f(x,y)=(5/(x+1))**0.5')
plt.xlabel('x [m]')
plt.ylabel('y [m]')
plt.plot(x, y,x,y1)
print(" Stephensen method: ")
a = float(input("enter initial value : "))
rr=stephensen(a)
print(" ROOT  : ",rr,"\nFUNCTION VALUE : ",f1(rr))
```

```
In [21]: runfile('D:/okul/SC01/stephensen.py', wdir='D:/okul/SC01')
 Stephensen method:

enter initial value : 1
 ROOT  :  1.4334276638638197
FUNCTION VALUE :  1.4334276638638197
```




```
In [22]:
```

2.4 NEWTON-RAPHSON METHOD



2.4.1 Isaac Newton



2.4.2 Joseph Raphson

Taylor Formula can be written as

$$f(x_{n+1}) = f(x_n) + \frac{f'(x_n)}{1!}(x_{n+1} - x_n) + \frac{f''(x_n)}{2!}(x_{n+1} - x_n)^2 + \frac{f^{(3)}(x_n)}{3!}(x_{n+1} - x_n)^3 + \cdots \quad (2.4.1)$$

We would like to find $f(x_{n+1}) = 0$. If We substitude it into the Taylor Formula, and also cancel out terms above the first degree , The equation becomes

$$0 = f(x_n) + f'(x_n)(x_{n+1} - x_n) \quad (2.4.2)$$

If x_{n+1} taken to left hand side Newton-Raphson Formula is obtained

$$x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)} \quad (2.4.3)$$

We can find the root iteratively starting from a first guess. Newton Formula aproached to the root quickly if the first guess is close to the actual root. But if the first gues is far away from the root, it might take longer or it might fail to reach to the root. Another difficulty in Newton-Raphson Formula is the requirement of knowing the actual derivative of the function

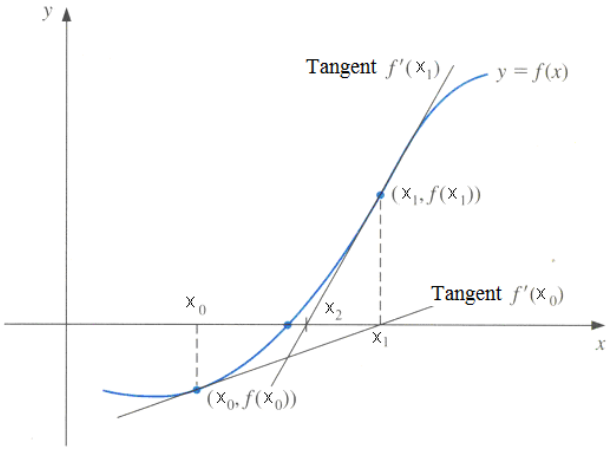


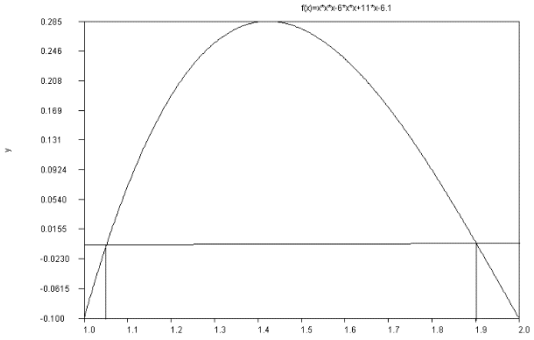
Figure 2.5 Convergence of the Newton-Raphson Method

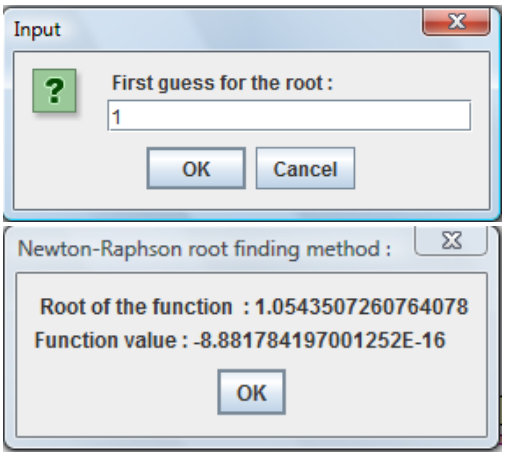
PROGRAM 2.4-1 Newton-Raphson method (java code)

```
// Newton-Raphson root finding
//Numerical Analysis
//
import java.util.*;
import java.awt.*;
import java.applet.Applet;
import java.awt.event.*;
```

```
import javax.swing.*;
//============= The function ================

class fb extends f_x
{
   public double func (double x)
   { return x*x*x-6*x*x+11*x-6.1;}
}
//============= Derivative of the function ================
class dfb extends f_x
{
   public double func (double x)
   { return 3.0*x*x-12.0*x+11;}
}
public class NA5
{
public static double newton(f_x f,f_x df,double x)
{
int nmax=100;
double tolerance=1.0e-10;
double fx,dfx;
for(int i=0;i<nmax;i++)
{
  fx=f.func(x);
  dfx=df.func(x);
  x-=fx/dfx;
  System.out.println("i="+i+"x="+x+"fx="+fx+"dfx = "+dfx);
  if(Math.abs(fx)<tolerance) { return x;}
}
JOptionPane.showMessageDialog(null,"Warning : MAximum number of iteration is exceeded \n"+
   " çözüm geçerli olmiyabilir","MAXİMUM ITERATION NUMBER
WARNING",JOptionPane.WARNING_MESSAGE);
return x;
}
     public static void main (String args[])
     { fb f=new fb();
        dfb df=new dfb();
       double x0;
       x0=Double.parseDouble(JOptionPane.showInputDialog(" First guess for the root : "));
       double r= newton(f,df,x0);
       JOptionPane.showMessageDialog(null," Root of the function  : "+r+"\nFunction value : "+f.func(r),
       "Newton-Raphson root finding method : ",JOptionPane.PLAIN_MESSAGE);
     } }
```






```
---------- Capture Output ----------
> "C:\turhan\java\bin\javaw.exe" NA5
i=0x=1.0499999999999998fx=-0.09999999999999964dfx = 2.0
i=1x=1.054319180087848fx=-0.0073750000000005755dfx = 1.7075000000000014
i=2x=1.0543507243988341fx=-5.30870767248004E-5dfx = 1.68293663944916
i=3x=1.0543507260764082fx=-2.82295076203809E-9dfx = 1.6827576573350296
i=4x=1.0543507260764078fx=8.881784197001252E-16dfx = 1.6827576478166488
> Terminated with exit code 0.
```

In order to see step calculations of Newton-Raphson method, root of $f(x)=x^2-2$ is calculated in the spreadsheet environment as well

PROGRAM 2.4-2 Newton-Raphson method in excel

Newton-Raphson root finding $f(x)=x^2-a$

a	2	
x	f(x)=x*x-a	fx/dx=2*x
1.00000000000000000	-1	2
1.50000000000000000	0.25	3
1.41666666666667000	0.006944444	2.833333333
1.41421568627451000	6.0073E-06	2.828431373
1.41421356237469000	4.51061E-12	2.828427125
1.41421356237310000	0	2.828427125
1.41421356237310000	0	2.828427125
1.41421356237310000	0	2.828427125

PROGRAM 2.4-3 Newton-Raphson method in Matlab(Octave)

```
function p=newton_raphson(f,df,x,tolerance,nmax)
% secant method for root finding
if nargin<5 nmax=100;end
if nargin<4 tolerance=1e-10;end
i=0;
fx=f(x);
while (abs(fx) > tolerance) && i<nmax
   fx=f(x);
   dfx=df(x);
   x=x-fx/dfx;
   i=i+1;
end
p=x;
if i>=nmax fprintf('Maximum number of iterations is exceeded the result may not be valid');end
end
```

```
>> f=@(x) x*x-2;
>> df=@(x) 2*x;
>> newton_raphson(f,df,1)
ans =
  1.414213562373095
```

PROGRAM 2.4-4 Newton-Raphson method in C++

```
#include <stdio.h>
```

```
#include <iostream>
#include <math.h>
#include <iomanip>
using namespace std;

class f_x{
public: virtual double func(double x)=0;
public: double dfunc(double x){ double h=0.0001;double hh=1.0/h;
 double df=(3.0*func(x-4.0*h)-32.0*func(x-3.0*h)+168.0*func(x-2.0*h)-672.0*func(x-h)+672.0*func(x+h)-168.0*func(x+2.0*h)
+32.0*func(x+3.0*h) - 3.0*func(x+4.0*h))/840.0*hh;
  return df;}};

class f1:public f_x
{ public: double func(double x) { double ff=x*x-2.0; return ff;}};

class df1:public f_x
{ public: double func(double x) { double ff=2.0*x;return ff;}};

double newton(f_x &f,f_x &df,double x)
{
int nmax=100;
double tolerance=1.0e-8;
double fx,dfx;
for(int i=0;i<nmax;i++)
{
  fx=f.func(x);
  dfx=df.func(x);
  x-=fx/dfx;
  cout <<"i="<<i<<"x="<<x<<"fx="<<fx<<"dfx="<<dfx<<"\n";
  if(fabs(fx)<tolerance) { return x;}
}
cout <<("Maximum number of iteration is exceeded \n result might not be valid");
return x;
}

int main()
{  double xa,r;
   cout << " first guess for the root value : ";
   cin >>xa;
   f1 f;
   df1 df;
   cout<<setprecision(20);
   r=newton(f,df,xa);
   cout<<"root value : ="<<r;
   return 0;
}
```

```
 ---------- Capture Output ----------
> "D:\okul\SCO1\run.bat" newton

D:\okul\SCO1>newton
 first guess for the root value : 1
i=0x=1.5fx=-1dfx=2
i=1x=1.4166666666666667fx=0.25dfx=3
i=2x=1.4142156862745099fx=0.006944444444444654dfx=2.8333333333333335
i=3x=1.4142135623746899fx=6.0073048829714475e-006dfx=2.8284313725490198
i=4x=1.4142135623730951fx=4.5108350648498385e-012dfx=2.8284271247493797
root value : =1.4142135623730951
> Terminated with exit code 0.
```

Python version of the Newton-Raphson method.

PROGRAM 2.4-5 Newton-Raphson method in Python

```
 from math import *

def f1(x):
   return x*x-2.0

def df1(x):
   return 2.0*x

def newton(x):
   # Newton-Raphson root finding method
  nmax=100
  tolerance=1.0e-10
  for i in range(0,nmax):
      fx= f1(x);
      dfx=df1(x)
      x=x-fx/dfx
      print("fx",fx,"dfx",dfx,"x=",x)
      if abs(fx)<tolerance: return x
  return x

a=float(input("enter initial guess : "))
r=newton(a)
print(" ROOT  : ",r,"\nFUNCTION VALUE : ",f1(r))
```

```
---------- Capture Output ----------
> "D:\co\python\pythonw.exe" NA5.py
enter initial guess : 1
fx -1.0 dfx 2.0 x= 1.5
fx 0.25 dfx 3.0 x= 1.4166666666666667
fx 0.006944444444444642 dfx 2.8333333333333335 x= 1.4142156862745099
fx 6.007304882871267e-06 dfx 2.8284313725490198 x= 1.4142135623746899
fx 4.510614104447086e-12 dfx 2.8284271247493797 x= 1.4142135623730951
 ROOT  :  1.4142135623730951
FUNCTION VALUE :  4.440892098500626e-16

> Terminated with exit code 0.
```

PROGRAM 2.4-6 Newton-Raphson method in Fortran 90

```
program newton_raphson
IMPLICIT NONE
REAL(KIND = SELECTED_REAL_KIND(10,34)) a
a=1.0
print *,'Newton_Raphson(',a,') = ',newton(a)

CONTAINS
  FUNCTION func(x)
  REAL(KIND = SELECTED_REAL_KIND(10,34)) func,x
  func=x*x-2.0
  END FUNCTION func

  FUNCTION dfunc(x)
  REAL(KIND = SELECTED_REAL_KIND(10,34)) dfunc,x
  dfunc=2.0*x
  END FUNCTION dfunc

  FUNCTION newton(a)
  REAL(KIND = SELECTED_REAL_KIND(10,34)) newton,a,x,eps,fx,dfx
  INTEGER nmax,i
  eps=1.0e-8;
  nmax=100;
  x=a
  DO i=1,nmax
    fx=func(x)
```

```
    dfx=dfunc(x)
    x=x-fx/dfx
    IF (abs(fx)<eps) EXIT
  END DO
  newton=x
  END FUNCTION newton
END PROGRAM newton_raphson
```

```
---------- Capture Output ----------
> "E:\okul\SCO1\run.bat" newton_raphson

E:\okul\SCO1>newton_raphson
 Newton_Raphson(   1.0000000000000000      ) =    1.4142135623730951

> Terminated with exit code 0.
```

Another program for Python version of the Newton-Raphson method. Please refer to **Program 2.1-1e Abstract class f_x in python language.** In this program function defined as a class function which is derived from the abstract class f_x

```
 from math import *
from f_x import *;

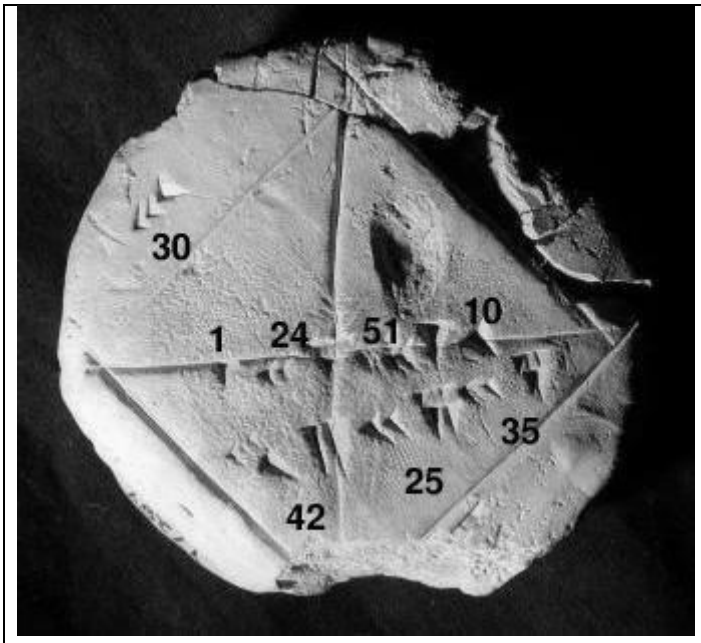
class f1(f_x):
  def func(self,x):
      y=x*x-2;
      return y;

class df1(f_x):
  def func(self,x):
      y=2.0*x;
      return y;

def newton(f,df,x):
   # Newton-Raphson root finding method
  nmax=100
  tolerance=1.0e-10
  for i in range(0,nmax):
      fx= f.func(x);
      dfx=df.func(x);
      x=x-fx/dfx
      print("fx",fx,"dfx",dfx,"x=",x)
      if abs(fx)<tolerance: return x
  return x


f=f1();
df=df1();
a=float(input("enter initial guess : "))
r=newton(f,df,a);
print(" ROOT  : ",r,"\nFUNCTION VALUE : ",f.func(r));
```

As it is seen from the results, convergence is quite fast for this problem.



Babylonian clay tablet YBC 7289 with annotations. The diagonal displays an approximation of the square root of 2 in four sexagesimal figures, 1 24 51 10, which is good to about six decimal digits.
$1 + 24/60 + 51/60^2 + 10/60^3 = 1.41421296...$
Babylonian algorithm is actually newton-raphson method.

$$x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)}$$
$$f(x_n) = x^2 - 2$$
$$f'(x_n) = 2x$$
$$x = x - \frac{x^2 - 2}{2x} = \frac{x}{2} + \frac{1}{x}$$

Babylonian algorithm in java format:

```
public class sqrt1
{
//Babylonian algorithm
public static double sqrt(double xi)
{
  double x = xi;
  double y =x*x-2;
  double e = 1.0e-10;
  while(Math.abs(y) > e)
  { x =x /2.0+ 1.0/x;
     y =x*x-2.0;
    System.out.println("x="+x+"y="+y);
  }
  return x;
}
Public
public static void main(String arg[])
{
System.out.println(sqrt(2.0));
}
}
```

```
---------- Capture Output ----------
> "E:\co\java\bin\javaw.exe" sqrt1
```

```
x=1.5y=0.25
x=1.4166666666666665y=0.006944444444444198
x=1.4142156862745097y=6.007304882427178E-6
x=1.4142135623746899y=4.510614104447086E-12
1.4142135623746899

> Terminated with exit code 0.
```

2.5 SECANT METHOD

One of the difficulty of the Newton-Raphson method is the requirement to find the derivative. If difference equation (Numerical derivative) is used instead of derivative (tangent to the function), a secant to the function is drawn to the function, so the method is called secant method. If derivative is approximated with a first degree difference equation

$$f'(x_n) \cong \frac{f(x_n)-f(x_{n-1})}{(x_n-x_{n-1})} \quad (2.5.1)$$

Newton –Raphson Formula is converted to

$$x_{n+1} = x_n - \frac{f(x_n)(x_n-x_{n-1})}{f(x_n)-f(x_{n-1})} \quad (2.5.2)$$

Two first estimation is needed to start iteration by using this formula $f(x_0)$ and $f(x_{-1})$

If we would like to establish a one estimation secant Formula, the previous difference equation can be written for a small Δx as (central difference Formula)

$$f'(x_n) \cong \frac{f(x_n+\Delta x)-f(x_n-\Delta x)}{2\Delta x} \quad (2.5.3)$$

Second equation becomes:

$$x_{n+1} = x_n - \frac{2f(x_n)\Delta x}{f(x_n+\Delta x)-f(x_n-\Delta x)} \quad (2.5.4)$$

In order to minimize error Δx should be relatively small, to have a further approximation a second or higher derivative (difference) formulas can also be used

$$f'(x_n) \cong \frac{-f(x_n+2\Delta x)+8f(x_n+\Delta x)-8f(x_n-\Delta x)+f(x_n-2\Delta x)}{12\Delta x} \quad (2.5.5)$$

$$f'(x_n) \cong \frac{f(x_n+3\Delta x)-9f(x_n+2\Delta x)+45f(x_n+\Delta x)-45f(x_n-\Delta x)+9f(x_n-2\Delta x)-f(x_n-3\Delta x)}{60\Delta x} \quad (2.5.6)$$

$$f'(x_n) \cong \frac{-3f(x_n+4\Delta x)+32f(x_n+3\Delta x)-168f(x_n+2\Delta x)+672f(x_n+\Delta x)-672f(x_n-\Delta x)+168f(x_n-2\Delta x)-32f(x_n-3\Delta x)+3f(x_n-4\Delta x)}{840\Delta x}$$

(2.5.7)

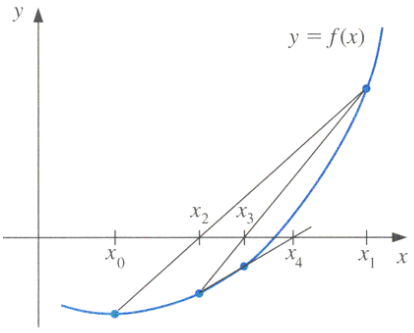


Figure 2.6 : Second method approximation with linear difference

PROBLEM 2.5-1 NA7 : Secant method with two initial root estimation (java code)

```
// Numerical Analysis
//
//
// Root finding with secant method
// two initia root estimation
//
```

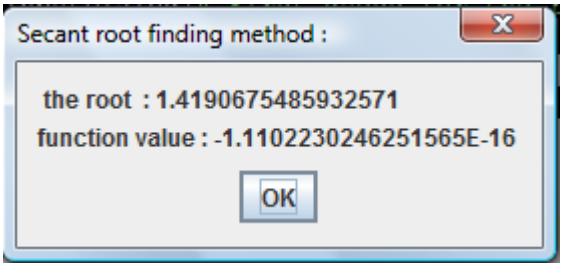
```
import java.util.*;
import java.awt.*;
import java.applet.Applet;
import java.awt.event.*;
import javax.swing.*;

//============= Function definition ================

class fb extends f_x
{
  public double func (double x)
  { return Math.sin(Math.sqrt(x))-x;}
}

public class NA7
{
public static double secant(f_x f,double x0,double x1)
{
int nmax=500;
double tolerance=1.0e-10;
double fx0,fx1;
double x;
fx0=f.func(x0);
for(int i=0;i<nmax;i++)
{
 fx1=f.func(x1);
 x=x1-fx1*(x1-x0)/(fx1-fx0);
 x0=x1;
 fx0=fx1;
 x1=x;
 if(Math.abs(fx1)<tolerance) { return x1;}
}
JOptionPane.showMessageDialog(null,"Warning : MAximum number of iteration is exceeded \n"+
  " çözüm geçerli olmiyabilir","MAXİMUM ITERATION NUMBER WARNING",JOptionPane.WARNING_MESSAGE);
return x1;
}

    public static void main (String args[])
    {
            fb f=new fb();
    double x0,x1;
    x0=Double.parseDouble(JOptionPane.showInputDialog(" First guess for the root : "));
    x1=Double.parseDouble(JOptionPane.showInputDialog(" Second guess for the root : "));
    double r= secant(f,x0,x1);
    JOptionPane.showMessageDialog(null," the root  : "+r+"\nfunction value : "+f.func(r),
    "Secant root finding method : ",JOptionPane.PLAIN_MESSAGE);
    System.exit(0);
    }
}
```



Re-write the program to find root of f(x)=tan(x-1)

In order to see calculation steps of the method a spreadsheet program is introduced, for the function $f(x)=x^2-2$, initial values $x_{-1}=0$, $x_0=1$ and $f'(x_n) \cong \frac{f(x_n)-f(x_{n-1})}{(x_n-x_{n-1})}$

PROBLEM 2.5-2 Secant method with two initial root estimation (octave-excel code)

Secant 1 $f(x)=x^2-a$

a	2	
x	f(x)=x*x-a	$fx/dx=(f(x_n)-f(x_n-1))/(x_n-x_{n-1})$

0.00000000000000000	-2	
1.00000000000000000	-1	1
2.00000000000000000	2	3
1.33333333333333000	-0.222222222	3.333333333
1.40000000000000000	-0.04	2.733333333
1.41463414634146000	0.001189768	2.814634146
1.41421143847487000	-6.00729E-06	2.828845585
1.41421356205732000	-8.93146E-10	2.828425001
1.41421356237310000	0	2.828424244
1.41421356237310000	0	

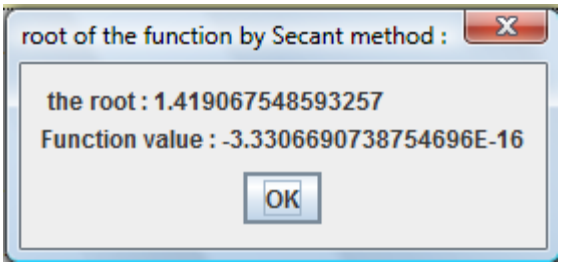
In the next java example $f'(x_n) \cong \frac{f(x_n+\Delta x)-f(x_n-\Delta x)}{2\Delta x}$ approach is used to estimate the derivative

PROBLEM 2.5-3 NA8 : Secant method with one initial root estimation, linear difference Formula (java code)

```
///Numerical Analysis
// Secant method with one root estimation
// Linear difference formula
import java.util.*;
import java.awt.*;
import java.applet.Applet;
import java.awt.event.*;
import javax.swing.*;

//============= function definition ================
class fb extends f_x
{
  public double func (double x)
  { return x*x-2;}
}
public class NA8
{
public static double secant(f_x f,double x)
{
int nmax=100;
double tolerance=1.0e-10;
double fx,dfx;
double dx=0.01;
for(int i=0;i<nmax;i++)
{
 fx=f.func(x);
 dfx=(f.func(x+dx)-f.func(x-dx))/(2.0*dx);
 x-=fx/dfx;
 if(Math.abs(fx)<tolerance) { return x;}
}
JOptionPane.showMessageDialog(null,"Warning : Maximum number of iteration is exceeded \n"+
  " çözüm geçerli olmiyabilir","MAXİMUM ITERATION NUMBER WARNING",JOptionPane.WARNING_MESSAGE);
return x;
}

    public static void main (String args[])
    {
            fb f=new fb();
    double x0;
    x0=Double.parseDouble(JOptionPane.showInputDialog(" First estimation for the root : "));
    double r= secant(f,x0);
    JOptionPane.showMessageDialog(null," the root : "+r+"\nFunction value : "+f.func(r),
    "root of the function by Secant method : ",JOptionPane.PLAIN_MESSAGE);
    System.exit(0);
    }
}
```



Find roos of f(x)=x*x*x-2

to see calculation steps of the method a spreadsheet program is introduced, for the function $f(x)=x^2-2$, initial values $x_0=1$, $\Delta x=0.1$ and $f'(x_n) \cong \frac{f(x_n+\Delta x)-f(x_n-\Delta x)}{2\Delta x}$

PROBLEM 2.5-3 Secant method with one initial root estimation, linear difference Formula (octave-excel code)

Secant 2 f(x)=x*x-a

a	2					
x	$f(x)=x^2-a$	x+Δx	x-Δx	f(x+Δx)	f(x-Δx)	f'(x)
Δx	0.1					
1	-1	1.1	0.9	-0.79	-1.19	2
1.5	0.25	1.6	1.4	0.56	-0.04	3
1.416666667	0.00694	1.51667	1.31667	0.30028	-0.26639	2.83333
1.414215686	6E-06	1.51422	1.31422	0.29285	-0.27284	2.82843
1.414213562	4.5E-12	1.51421	1.31421	0.29284	-0.27284	2.82843
1.414213562	0	1.51421	1.31421	0.29284	-0.27284	2.82843

PROBLEM 2.5-6 NA9 : Secant method with one initial root estimation, quadratic difference Formula (java code)

```
///Numerical Analysis
//
//
// Secant method with one root estimation
// quadratic difference formula
//

import java.util.*;
import java.awt.*;
import java.applet.Applet;
import java.awt.event.*;
import javax.swing.*;

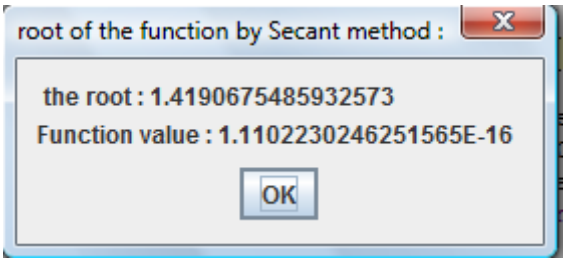
//============= function definition ================
class fb extends f_x
{
  public double func (double x)
  { return Math.log(x*x)-0.7;}
}
//=============================================================
public class NA9
{

public static double df(f_x f,double x,double h)
{ // derivative formula
  double hh=1.0/h;
  return (-f.func(x+2.0*h)+8.0*f.func(x+h)-8.0*f.func(x-h)+f.func(x-2.0*h))/12.0*hh;
}

public static double secant(f_x f,double x)
{
int nmax=500;
double tolerance=1.0e-15;
double fx,dfx;
for(int i=0;i<nmax;i++)
{
  fx=f.func(x);
```

```
  dfx=df(f,x,0.00001);
  x-=fx/dfx;
  if(Math.abs(fx)<tolerance) { return x;}
}
JOptionPane.showMessageDialog(null,"Warning : Maximum number of iteration is exceeded \n"+
   " çözüm geçerli olmiyabilir","MAXİMUM ITERATION NUMBER WARNING",JOptionPane.WARNING_MESSAGE);
return x;
}

    public static void main (String args[])
    {
              fb f=new fb();
    double x0;
    x0=Double.parseDouble(JOptionPane.showInputDialog(" First estimation for the root : "));
    double r= secant(f,x0);
    JOptionPane.showMessageDialog(null," the root : "+r+"\nFunction value : "+f.func(r),
    "root of the function by Secant method : ",JOptionPane.PLAIN_MESSAGE);
    } }
```



PROBLEM 2.5-6 NA9 : Secant method with one initial root estimation, An alternative formula for the difference equation (4th order) (java code)

```
public static double df(f_x f,double x,double h)
{ // derivative formula
  double hh=1.0/h;
  double df=(3.0*f.func(x-4.0*h)-32.0*f.func(x-3.0*h)+168.0*f.func(x-2.0*h)-672.0*f.func(x-h)+672.0*f.func(x+h)-168.0*f.func(x+2.0*h)
+32.0*f.func(x+3.0*h) - 3.0*f.func(x+4.0*h))/840.0*hh;
  return df;
}
```

C++ version of the secant method:

PROBLEM 2.5-7 secant2.cpp : Secant method with one initial root estimation, An alternative formula for the difference equation (4th order) (C++ code)

```
 #include <stdio.h>
#include <iostream>
#include <math.h>
#include <iomanip>
using namespace std;
class f_x{
public: virtual double func(double x)=0;
public: double dfunc(double x) { double h=0.0001;double hh=1.0/h;
double df=(3.0*func(x-4.0*h)-32.0*func(x-3.0*h)+168.0*func(x-2.0*h)-672.0*func(x-h)+672.0*func(x+h)-168.0*func(x+2.0*h)
+32.0*func(x+3.0*h) - 3.0*func(x+4.0*h))/840.0*hh;
return df;}};
class f1:public f_x{ public: double func(double x)
  { double ff=x*x-2.0;return ff;}};
double secant(f_x &f,double x){
int nmax=100;
double tolerance=1.0e-8;
double fx,dfx;
for(int i=0;i<nmax;i++)
{
  fx=f.func(x);
  dfx=f.dfunc(x);
```

```
  x-=fx/dfx;
  cout <<"i="<<i<<"x="<<x<<"fx="<<fx<<"dfx="<<dfx<<"\n";
  if(fabs(fx)<tolerance) { return x;}
}
cout <<("Maximum number of iteration is exceeded \n result might not be valid");
return x;}
    int main()
    {
    double x0;
    cout << "root estimation x0 : ";
    cin >>x0;
    double r;
    f1 f;
    cout<<setprecision(20);
    r= secant(f,x0);
    cout<<" Value of the root  : "<<r<<"\nFunction value : "<<f.func(r);
    }
```

```
 ---------- Capture Output ----------
> "D:\okul\SCO1\run.bat" secant1

D:\okul\SCO1>secant1
root estimation x0 : 1
i=0x=1.4999999999999951fx=-1dfx=2.0000000000000191
i=1x=1.4166666666666554fx=0.24999999999998535dfx=2.999999999999599
i=2x=1.4142156862745094fx=0.006944444444444125685dfx=2.8333333333330035
i=3x=1.4142135623746899fx=6.0073048817152908e-006dfx=2.8284313725487062
i=4x=1.4142135623730951fx=4.5108350648498385e-012dfx=2.8284271247490693
 Value of the root  : 1.4142135623730951
Function value : 2.7343578790084422e-016
> Terminated with exit code 0.
```

Utilize the method to find root of f(x)=x*x*x-3

In the next example secant solution of the function $f(x)=x^2-2$ will be inroduced with initial values $x_0=1$, $\Delta x=0.1$ and $f'(x_n) \cong \frac{-f(x_n+2\Delta x)+8f(x_n+\Delta x)-8f(x_n-\Delta x)+f(x_n-2\Delta x)}{12\Delta x}$ in the spreadsheet environment (excel)

PROBLEM 2.5-8 Secant method with one initial root estimation, quadratic difference Formula in excel (or open ofice calc) spreadsheet

Secant 3 $f(x)=x^2-a$

a	2									
Δx	0.1									
x	f(x)=x*x-a	x+2Δx	x+Δx	x-Δx	x-2Δx	f(x+2Δx)	f(x+Δx)	f(x-Δx)	f(x-2Δx)	f'(x)
1	-1	1.2	1.1	0.9	0.8	-0.56	-0.79	-1.19	-1.36	2.99167
1.334261838	-0.21975	1.53426	1.43426	1.23426	1.13426	0.35396	0.05711	-0.4766	-0.71345	2.08121
1.439846995	0.07316	1.63985	1.53985	1.33985	1.23985	0.6891	0.37113	-0.20481	-0.46278	1.71622
1.397218859	-0.04778	1.59722	1.49722	1.29722	1.19722	0.55111	0.24166	-0.31722	-0.56667	1.86805
1.422795983	0.02435	1.6228	1.5228	1.3228	1.2228	0.63347	0.31891	-0.25021	-0.50477	1.77768
1.409099259	-0.01444	1.6091	1.5091	1.3091	1.2091	0.5892	0.27738	-0.28626	-0.53808	1.82635
1.417005353	0.0079	1.61701	1.51701	1.31701	1.21701	0.61471	0.30131	-0.2655	-0.5189	1.79833
1.412610076	-0.00453	1.61261	1.51261	1.31261	1.21261	0.60051	0.28799	-0.27705	-0.52958	1.81393
1.415108941	0.00253	1.61511	1.51511	1.31511	1.21511	0.60858	0.29556	-0.27049	-0.52351	1.80507
1.413705498	-0.00144	1.61371	1.51371	1.31371	1.21371	0.60405	0.2913	-0.27418	-0.52692	1.81005
1.414499268	0.00081	1.6145	1.5145	1.3145	1.2145	0.60661	0.29371	-0.27209	-0.52499	1.80724
1.414052077	-0.00046	1.61405	1.51405	1.31405	1.21405	0.60516	0.29235	-0.27327	-0.52608	1.80882
1.414304575	0.00026	1.6143	1.5143	1.3143	1.2143	0.60598	0.29312	-0.2726	-0.52546	1.80793
1.414162185	-0.00015	1.61416	1.51416	1.31416	1.21416	0.60552	0.29269	-0.27298	-0.52581	1.80843
1.414242539	8.2E-05	1.61424	1.51424	1.31424	1.21424	0.60578	0.29293	-0.27277	-0.52562	1.80815
1.414197211	-4.6E-05	1.6142	1.5142	1.3142	1.2142	0.60563	0.29279	-0.27289	-0.52573	1.80831
1.414222786	2.6E-05	1.61422	1.51422	1.31422	1.21422	0.60572	0.29287	-0.27282	-0.52566	1.80822
1.414208358	-1.5E-05	1.61421	1.51421	1.31421	1.21421	0.60567	0.29283	-0.27286	-0.5257	1.80827

1.414216498	8.3E-06	1.61422	1.51422	1.31422	1.21422	0.60569	0.29285	-0.27283	-0.52568	1.80824

Now if the same problem is solved by uisng Matlab functions:

PROBLEM 2.5-9 Secant method with one initial root estimation in Matlab(octave) environment

```
function p=df(f,x,dx)
% derivative of a function
if nargin<3
   dx=0.001;
end
p = ( -3*f(x+4*dx)+32*f(x+3*dx)-168*f(x+2*dx)+672*f(x+dx)-672*f(x-dx) + 168*f(x-2*dx)-32*f(x-3*dx)+3*f(x-4*dx) )/(dx*840);
end
```

```
function p=secant(f,x,tolerance,nmax)
% secant method for root finding
if nargin<4 nmax=100;end
if nargin<3 tolerance=1e-10;end
i=0;
fx=f(x);
while (abs(fx) > tolerance) && i<nmax
   fx=f(x);
   dfx=df(f,x);
   x=x-fx/dfx;
   i=i+1;
end
p=x;
if i>=nmax fprintf('Maximum number of iterations is exceeded the result may not be valid');end
end
```

```
f=@(x) x*x-2;
>> secant(f,1)
ans =1.414213562373095
```

Secant method given as python code:

PROBLEM 2.5-10 Secant method with one initial root estimation, quadratic difference Formula in python language

```
 from math import *

#def df(f,x):
#  h=0.001
#   return (f(x+dx)-f(x-dx))/(2.0*h)
def df(f,x):
   h=0.001
   return (-f(x+2.0*h)+8.0*f(x+h)-8.0*f(x-h)+f(x-2.0*h))/(12.0*h);

def secant(f,x):
   # secant root finding method
  nmax=100
  tolerance=1.0e-10
  for i in range(0,nmax):
      fx= f(x);
      dfx=df(f,x)
      x=x-fx/dfx
      print("fx",fx,"dfx",dfx,"x=",x)
      if abs(fx)<tolerance: return x
  return x

f1 = lambda x: x*x-2
a=float(input("enter initial guess : "))
r=secant(f1,a)
print(" ROOT  : ",r,"\nFUNCTION VALUE : ",f1(r))
```

```
 ---------- Capture Output ----------
> "D:\co\anaconda\pythonw.exe" secant.py
enter initial guess : 1
```

```
fx -1.0 dfx 1.9999999999998537 x= 1.5000000000000366
fx 0.25000000000011013 dfx 2.9999999999995217 x= 1.4166666666666532
fx 0.00694444444440645 dfx 2.8333333333329103 x= 1.4142156862745094
fx 6.0073048815389996e-06 dfx 2.828431372548642 x= 1.4142135623746899
fx 4.510614104447086e-12 dfx 2.828427124748826 x= 1.4142135623730951
 ROOT  :  1.4142135623730951
FUNCTION VALUE :  4.440892098500626e-16

> Terminated with exit code 0.
```

Note: lambda variable type function is used to define function in the last python code.

Second version of the python secant method (utilising class f_x as abstract root class)

```
from math import *
from f_x import *;

class f1(f_x):
  def func(self,x):
      y=x*x-2;
      return y;

def secant(f,x):
   # secant root finding method
  nmax=100
  tolerance=1.0e-10
  for i in range(0,nmax):
      fx= f.func(x);
      dfx=f.dfunc(x,1);
      x=x-fx/dfx
      print("fx",fx,"dfx",dfx,"x=",x)
      if abs(fx)<tolerance: return x
  return x

f=f1();
a=float(input("enter initial guess : "))
r=secant(f,a);
print(" ROOT  : ",r,"\nFUNCTION VALUE : ",f.func(r));
```

PROBLEM 2.5-11 Secant method with one initial root estimation, Fortran 90 version

```
 program secant1
IMPLICIT NONE
REAL(KIND = SELECTED_REAL_KIND(10,34)) a
a=1.0
print *,'secant(',a,') = ',secant(a)

CONTAINS
  FUNCTION func(x)
  REAL(KIND = SELECTED_REAL_KIND(10,34)) func,x
  func=x*x-2.0
  END FUNCTION func

  FUNCTION dfunc(x)
  REAL(KIND = SELECTED_REAL_KIND(10,34)) dfunc,x,h,df,f1,f2,f3,f4,f5,f6,f7,f8
  h=1.12314e-4
  f1=3.0*func(x-4.0*h)
  f2=-32.0*func(x-3.0*h)
  f3=168.0*func(x-2.0*h)
  f4=-672.0*func(x-h)
  f5=672.0*func(x+h)
  f6=-168.0*func(x+2.0*h)
  f7=32.0*func(x+3.0*h)
  f8=-3.0*func(x+4.0*h)
  dfunc=(f1+f2+f3+f4+f5+f6+f7+f8)/(840.0*h)
  END FUNCTION dfunc

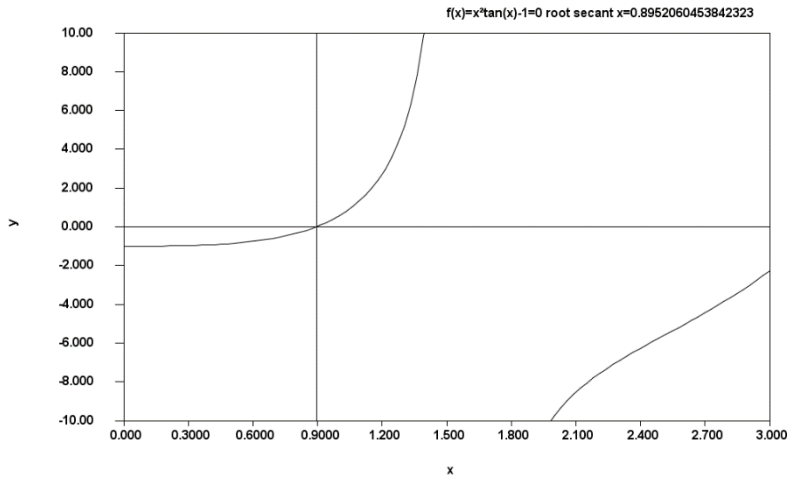
  FUNCTION secant(a)
  REAL(KIND = SELECTED_REAL_KIND(10,34)) secant,a,x,eps,fx,dfx
  INTEGER nmax,i
  eps=1.0e-8;
```

```
  nmax=100;
  x=a
  DO i=1,nmax
    fx=func(x)
    dfx=dfunc(x)
    x=x-fx/dfx
    IF (abs(fx)<eps) EXIT
  END DO
  secant=x
  END FUNCTION secant
END PROGRAM secant1
```

```
---------- Capture Output ----------
> "E:\okul\SCO1\run.bat" secant1
E:\okul\SCO1>secant1
 Newton_Raphson(   1.0000000000000000      ) =    1.4142135623730951
> Terminated with exit code 0.
```

Example problem: Use Secant method to find the smallest positive root of $f(x) = x^2 \tan(x) - 1$

```
 import java.io.*;
class PlotT01
{
public static double secant(if_x f,double x)
{ double eps=1.0e-10;
  double y=f.dfunc(x);
  double dy=f.dfunc2(x);
  int miter=100;
  int i=0;
  while(Math.abs(y)>eps && i<miter)
  {x=x-y/dy;
   y=f.func(x);
   dy=f.dfunc(x);
   i++;
  }
  return x;
}
public static void main (String args[]) throws java.io.IOException
{  double ymin=-10;
   double ymax=10;
   double xmin=0;
   double xmax=3;
   int N=100;
   if_x f1=x->x*x*Math.tan(x)-1.0;
   if_x f2=x->0.0;
   Plot p=new Plot(f1,xmin,xmax,N);
   p.addFunction(f2,xmin,xmax,N);
   double x=secant(f1,(xmin+xmax)/2.0);+
   double xx[]={x,x};
   double yy[]={ymin,ymax};
   p.addData(xx,yy);
   p.setPlabel("f(x)=x"+"\u00B2'+"tan(x)-1=0 root secant x="+x);
   p.setYMinMax(ymin,ymax);
   p.plot();
} }
```



2.6 MULLER'S METHOD



Figure 2.7 Werner Muller, German Mathematician

In false position method roots were found by using the root of the linear line passing through the two interval points. The same concept can be use to fit a quadratic curve and find the roots of the quadratic function. But quadratic function fitting is required 3 estimation point. The method consist of deriving the coefficients of a quadratic curve(parabola) that goes through three points. If function evaluation of these 3 points are $f(x_0)$, $f(x_1)$ and $f(x_2)$, enough information will be available to make a quadratic curve root estimation. Assuming quadratic function defined b the following equation :

$f(x) = a(x - x_2)^2 + b(x - x_2) + c$ (2.6.1)

Evaluation of the function in 3 given points will be

$f(x_0) = a(x_0 - x_2)^2 + b(x_0 - x_2) + c$ (2.6.2)

$f(x_1) = a(x_1 - x_2)^2 + b(x_1 - x_2) + c$ (2.6.3)

$f(x_2) = a(x_2 - x_2)^2 + b(x_2 - x_2) + c = c$ (2.6.4)

Considering the difference of the functions:

$f(x_0) - f(x_2) = a(x_0 - x_2)^2 + b(x_0 - x_2)$ (2.6.5)

$f(x_1) - f(x_2) = a(x_1 - x_2)^2 + b(x_1 - x_2)$ (2.6.6)

$h_0 = (x_1 - x_0)$ (2.6.7)

$h_1 = (x_2 - x_0)$ (2.6.8)

$d_0 = \frac{f(x_1) - f(x_0)}{h_0}$ (2.6.9)

$d_1 = \frac{f(x_2) - f(x_0)}{h_1}$ (2.6.10)

When these are substituted back into the main equation

$(h_0 + h_1)b - (h_0 + h_1)^2 a = h_0 d_0 + h_1 d_1$ (2.6.11)

$h_1 b - (h_1)^2 a = h_1 d_1$ (2.6.12)

is obtained. a and b can be solved from here.

$a = \frac{d_1 - d_0}{h_1 - h_0}$ (2.6.13)

$b = ah_1 + d_1$ (2.6.14)

$c = f(x_2)$ (2.6.16)

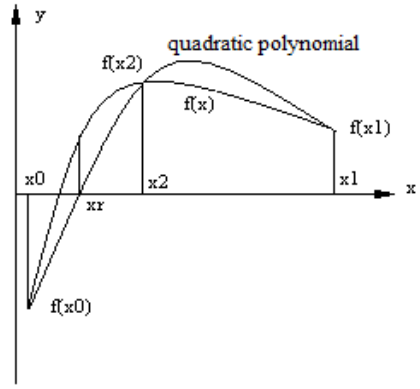


Figure 2.8 graphic representation of Muller root finding method.

to find root root :

$\Delta = \sqrt{b^2 - 4ac}$ (2.6.17)

if

$|b + \Delta| > |b + \Delta| \quad then\ e = b + \Delta$ (2.6.18)

Else

$e = b + \Delta$ (2.6.19)

$x_r = x_2 - \frac{2c}{e}$ (2.6.20)

$x_0 = x_1 \quad x_1 = x_2 \quad x_2 = x_r$ (2.6.21)

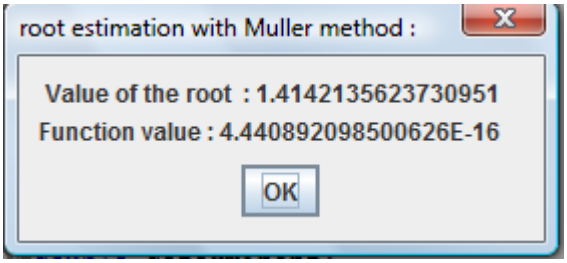
x_2 will be substituted with x_r and iteration continues. It should be noted that in a quadratic formula complex roots can be located as well as the real roots, therefore this method can be used to find complex roots as well. The program NA10 is developed to calculate real roots by

Program 2.6-1 NA10 : Muller method root finding with quadratic formula

```
//Numerical Analysis
// Muller method for root finding
import java.util.*;
import java.awt.*;
import java.applet.Applet;
import java.awt.event.*;
import javax.swing.*;
//============= Function definition ================
class fb extends f_x
{public double func (double x)
  { return x*x-2;}
}
//============================================================
public class NA10
{public static double muller(f_x f,double x0,double x1,double x2)
{ //Finding real roots by using Muller method
  //orta xr noktasindan +/- h kadar giderek diğer 2 noktaya ulaşiyoruz.
int maxit=5;
double xr=x0;
int iter=0;
double es1=0.001;
double es=es1;
double ea=1.1*es;
double h0,h1;
double d0,d1;
```

```
double a,b,c;
double fx0,fx1,fx2;
double determinant;
double den;
double dxr;
while(iter<maxit && ea>es)
{
   fx0=f.func(x0);
   fx1=f.func(x1);
   fx2=f.func(x2);
   h0=x1-x0;
   h1=x2-x1;
   d0=(fx1-fx0)/h0;
   d1=(fx2-fx1)/h1;
   a=(d1-d0)/(h1+h0);
   b=a*h1+d1;
   c=fx2;
   determinant=b*b-4.0*a*c;
   if(determinant<=0) determinant=0;//we do not want to calculate complex roots here
   else determinant=Math.sqrt(determinant);
   if(Math.abs(b+determinant)>Math.abs(b-determinant)) {den=b+determinant;;}
   else {den=b-determinant;}
   dxr=-2*c/den;
   xr=x2+dxr;
   System.out.println("a="+a+"b="+b+"c="+c+"disc="+determinant+"xr="+xr+"iter="+iter);
   ea=Math.abs((xr-x2)/xr)*100;
   iter++;
   x0=x1;
   x1=x2;
   x2=xr;
}
if(iter>=maxit) JOptionPane.showMessageDialog(null,"Warning : MAximum number of iteration is exceeded \n"+
  " çözüm geçerli olmiyabilir","MAXİMUM ITERATION NUMBER WARNING",JOptionPane.WARNING_MESSAGE);
return xr;
}

    public static void main (String args[])
    {
            fb f=new fb();
    double x0,x1,x2;
    x0=Double.parseDouble(JOptionPane.showInputDialog(" root estimation x0 : "));
    x1=Double.parseDouble(JOptionPane.showInputDialog(" root estimation x1 : "));
    x2=Double.parseDouble(JOptionPane.showInputDialog(" root estimation x2 : "));
    //h=Double.parseDouble(JOptionPane.showInputDialog(" h=(küçük bir rakam örneğin 0.1"));
    double r;
    r= muller(f,x0,x1,x2);
    JOptionPane.showMessageDialog(null," Value of the root  : "+r+"\nFunction value : "+f.func(r),
    "root estimation with Muller method : ",JOptionPane.PLAIN_MESSAGE);
    System.exit(0);
    }
}
```



Program 2.6-3 Muller method root finding with quadratic formula in C++

```
#include <stdio.h>
```

```
#include <iostream>
#include <math.h>
#include <iomanip>
using namespace std;

class f_x{
public: virtual double func(double x)=0;
public: double dfunc(double x)
{ double h=0.0001;
  double hh=1.0/h;
  double df=(3.0*func(x-4.0*h)-32.0*func(x-3.0*h)+168.0*func(x-2.0*h)-672.0*func(x-h)+672.0*func(x+h)-168.0*func(x+2.0*h)
+32.0*func(x+3.0*h) - 3.0*func(x+4.0*h))/840.0*hh;
  return df;
}
};

class f1:public f_x
{ public: double func(double x){ double ff=x*x-2.0;return ff;}};

double muller(f_x &f,double x0,double x1,double x2)
{
//Finding real roots by using Muller method
//orta xr noktasından +/- h kadar giderek diğer 2 noktaya ulaşıyoruz.
int maxit=5;
double xr=x0;
int iter=0;
double es1=0.001;
double es=es1;
double ea=1.1*es;
double h0,h1;
double d0,d1;
double a,b,c;
double fx0,fx1,fx2;
double determinant;
double den;
double dxr;
while(iter<maxit && ea>es)
{
   fx0=f.func(x0);
   fx1=f.func(x1);
   fx2=f.func(x2);
   h0=x1-x0;
   h1=x2-x1;
   d0=(fx1-fx0)/h0;
   d1=(fx2-fx1)/h1;
   a=(d1-d0)/(h1+h0);
   b=a*h1+d1;
   c=fx2;
   determinant=b*b-4.0*a*c;
   if(determinant<=0) determinant=0;//we do not want to calculate complex roots here
   else determinant=sqrt(determinant);
   if(fabs(b+determinant)>fabs(b-determinant)) {den=b+determinant;;}
   else {den=b-determinant;}
   dxr=-2*c/den;
   xr=x2+dxr;
   ea=fabs((xr-x2)/xr)*100;
   iter++;
   x0=x1;
   x1=x2;
   x2=xr;
}
if(iter>=maxit) cout<<"Warning : Maximum number of iteration is exceeded \n";
return xr;
}
```

```
int main()
{  f1 f;
double x0,x1,x2;
cout << "root estimation x0 : ";
cin >>x0;
cout << "root estimation x1 : ";
cin >>x1;
cout << "root estimation x2 : ";
cin >>x2;
double r;
cout<<setprecision(20);
r= muller(f,x0,x1,x2);
cout<<" Value of the root  : "<<r<<"\nFunction value : "<<f.func(r);
}
```

```
---------- Capture Output ----------
> "D:\okul\SCO1\run.bat" NA10

D:\okul\SCO1>NA10
root estimation x0 : 1
root estimation x1 : 3
root estimation x2 : 4
 Value of the root  : 1.41421
Function value : 2.73436e-016
> Terminated with exit code 0.
```

Program 2.6-3 Muller method root finding with quadratic formula in Matlab (Octave)

```
function xr=muller(f,x0,x1,x2)
% Finding real roots by using Muller method
%
iter=0;
es1=0.001;
es=es1;
ea=1.1*es;
maxit = 100;
xr=x0;
while (iter<maxit) && (ea>es)
   fx0=f(x0);
   fx1=f(x1);
   fx2=f(x2);
   h0=x1-x0;
   h1=x2-x1;
   d0=(fx1-fx0)/h0;
   d1=(fx2-fx1)/h1;
   a=(d1-d0)/(h1+h0);
   b=a*h1+d1;
   c=fx2;
   determinant=b*b-4.0*a*c;
   if determinant<=0  determinant=0;
   else determinant=sqrt(determinant);end
   if abs(b+determinant)>  abs(b-determinant) den=b+determinant;
   else den=b-determinant;end
   dxr=-2*c/den;
   xr=x2+dxr;
   ea=abs((xr-x2)/xr)*100;
   iter=iter+1;
   x0=x1;
   x1=x2;
   x2=xr;
end
if iter>=maxit  fprintf('Warning : Maximum number of iteration is exceeded result may not be valis');end
end
```

```
>> f=@(x) x*x-2;
>> muller(f,0,1,2)
```

```
ans =1.414213562373095
```

Program 2.6-4 Muller method root finding with quadratic formula in Python code

```
# Muller method for root finding
# python version

from math import *
from f_x import *;

class f1(f_x):
  def func(self,x):
      y=x*x-2;
      return y;

def muller(f,x0,x1,x2):
  # Finding real roots by using Muller method
  iter=0;
  es1=0.001;
  es=es1;
  ea=1.1*es;
  maxit = 100;
  xr=x0;
  for iter in range(0,maxit):
    fx0=f.func(x0);
    fx1=f.func(x1);
    fx2=f.func(x2);
    h0=x1-x0;
    h1=x2-x1;
    d0=(fx1-fx0)/h0;
    d1=(fx2-fx1)/h1;
    a=(d1-d0)/(h1+h0);
    b=a*h1+d1;
    c=fx2;
    determinant=b*b-4.0*a*c;
    if determinant<=0:  determinant=0;
    else: determinant=sqrt(determinant);
    if abs(b+determinant)>  abs(b-determinant): den=b+determinant;
    else: den=b-determinant;
    dxr=-2*c/den;
    xr=x2+dxr;
    ea=abs((xr-x2)/xr)*100;
    x0=x1;
    x1=x2;
    x2=xr;
    if ea<es: return xr
  if iter>=maxit:  print('Warning : Maximum number of iteration is exceeded result may not be valis');
  return xr;


f=f1();
x0=float(input("enter first number : "));
x1=float(input("enter second number : "));
x2=float(input("enter third number : "));
r=muller(f,x0,x1,x2);
print("muller method root  : ",r);
```

```
---------- Capture Output ----------
> "D:\co\python\pythonw.exe" muller.py
enter first number : 0
enter second number : 1
enter third number : 3
muller method root  :  1.4142135623730951
```

```
> Terminated with exit code 0.
```

Program 2.6-5 Muller method root finding with quadratic formula in Fortran 90

```
program muller1
IMPLICIT NONE
REAL(KIND = SELECTED_REAL_KIND(10,34)) x0,x1,x2
x0=1.0
x1=1.15
x2=3
print *,'muller(',x0,',',x1,',',x2,') = ',muller(x0,x1,x2)

CONTAINS
 FUNCTION func(x)
 REAL(KIND = SELECTED_REAL_KIND(10,34)) func,x
 func=x*x-2.0
 END FUNCTION func

 FUNCTION muller(x0,x1,x2)
 REAL(KIND = SELECTED_REAL_KIND(10,34)) muller,eps,x0,x1,x2,xr,dxr,fx0,fx1,fx2,h0,h1,d0,d1,a,b,c,determinant,den
 INTEGER maxit,i
 eps=1.0e-8;
 maxit=100
 i=0;
 fx0=func(x0)
 fx1=func(x1)
 fx2=func(x2)
 DO i=1,maxit
  h0=x1-x0
  h1=x2-x1
  d0=(fx1-fx0)/h0
  d1=(fx2-fx1)/h1
  a=(d1-d0)/(h1+h0)
  b=a*h1+d1
  c=fx2
  determinant=b*b-4.0*a*c
  IF (determinant<=0) THEN
   determinant=0
  ELSE
   determinant=sqrt(determinant)
  END IF
  IF (abs(b+determinant)>abs(b-determinant)) THEN
   den=b+determinant
  ELSE
   den=b-determinant
  END IF
  dxr=-2.0*c/den
  xr=x2+dxr
  x0=x1
  x1=x2
  x2=xr
  fx0=func(x0)
  fx1=func(x1)
  fx2=func(x2)
  IF (abs(fx2)<eps) EXIT
 END DO
 muller=xr
 END FUNCTION muller
END PROGRAM muller1
```

```
---------- Capture Output ----------
> "E:\okul\SCO1\run.bat" muller1

E:\okul\SCO1>muller1
```

```
 muller(  1.0000000000000000   ,  1.1499999761581421   ,  3.0000000000000000   ) =   1.4142135623730949

> Terminated with exit code 0.
```

2.7 NEWTON-RAPHSON (SECANT) – BISECTION COMBINED METHOD

Bisection method is good in getting closer to the root. But when it get close to root it takes a long time to achieve the final accuracy. On the other hand, Newton-Raphson method is very efficient when the estimation is close to the actual root but rather poor performer when the root estimation is not very close to the actual root. By combining two methods advantages of both methods will be maximized while inefficiencies will be eliminated. Therefore a combine method is a good candidate for an efficient algorithm.

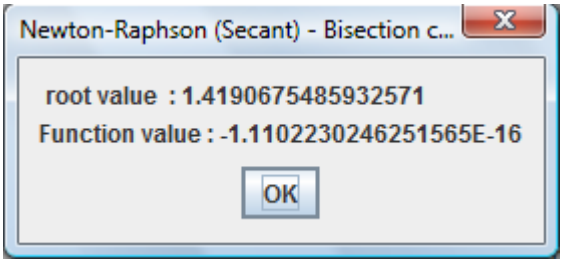
Program 2.7-1 NA11 : Newton-Raphson (Secant)-Bisection combined method

```
//
// Newton-Raphson (Secant)-Bisection combined root finding method
//
import java.util.*;
import java.awt.*;
import java.applet.Applet;
import java.awt.event.*;
import javax.swing.*;
//============= Function definition ================
class fb extends f_x
{
   public double func (double x)
   { return Math.log(x*x)-0.7;}
}
//============================================================

public class NA11
{
public static double dfx(f_x f,double x,double h)
{ //Derivative formula
  double hh=1.0/h;
  return (-f.func(x+2.0*h)+8.0*f.func(x+h)-8.0*f.func(x-h)+f.func(x-2.0*h))/12.0*hh;
}

public static double newton_bisection(f_x fx, double x1, double x2)
{
double xacc=1.0e-10; //accuracy
double h=0.0001; //derivative function step
int MAXIT=100;  //Maximum number of iterations
int j;
double dfr,dx,dxold,fr,fh,fl;
double temp,b,a,r;
fl=fx.func(x1);
fh=fx.func(x2);
//Check if a root is existed in the given region
if ((fl > 0.0 && fh > 0.0) || (fl < 0.0 && fh < 0.0))
System.out.println("There are no root in the given region or double root?");
if (fl == 0.0) return x1;
if (fh == 0.0) return x2;
if (fl < 0.0)   { a=x1;b=x2;}
else            { b=x1;a=x2;}
r=0.5*(x1+x2); //midpoint value
dxold=Math.abs(x2-x1);
dx=dxold;
fr=fx.func(r);   //function value at midpoint
dfr=dfx(fx,r,h); //derivative of function value at midpoint
for (j=1;j<=MAXIT;j++)
```

```
{
if ((((r-b)*dfr-fr)*((r-a)*dfr-fr) > 0.0) || (Math.abs(2.0*fr) > Math.abs(dxold*dfr)))
   { //bisection step
                dxold=dx;
     dx=0.5*(b-a);
     r=a+dx;
     if (a == r) return r; //çözüm!!!!
   }
else{ //Newton-Raphson (Secant) step
                dxold=dx;
     dx=fr/dfr;
     temp=r;
     r -= dx;
     if (temp == r) return r;  //solution!!!!!
   }
if (Math.abs(dx) < xacc) return r; //solution!!!!!
fr=fx.func(r);
dfr=dfx(fx,r,h);
if (fr < 0.0)
a=r;
else
b=r;
}
System.out.println("Maximum number of iterations are exceeded");
return 0.0; //program should never reach here
          //dummy return
}
    public static void main (String args[])
    {   fb f=new fb();
    double x0,x1;
    x0=Double.parseDouble(JOptionPane.showInputDialog(" lower root limit : "));
    x1=Double.parseDouble(JOptionPane.showInputDialog(" higher root limit : "));
    double r= newton_bisection(f,x0,x1);
    JOptionPane.showMessageDialog(null," root value  : "+r+"\nFunction value : "+f.func(r),
    "Newton-Raphson (Secant) - Bisection combined root finding method : ",JOptionPane.PLAIN_MESSAGE);
    System.exit(0);
    }
}
```



C++ version of the code:

Program 2.7-2 : Newton-Raphson (Secant)-Bisection combined method C++ version

```
#include <stdio.h>
#include <iostream>
#include <math.h>
#include <iomanip>
using namespace std;

class f_x{
public: virtual double func(double x)=0;
public: double dfunc(double x)
{ double h=0.0001;
  double hh=1.0/h;
  double df=(3.0*func(x-4.0*h)-32.0*func(x-3.0*h)+168.0*func(x-2.0*h)-672.0*func(x-h)+672.0*func(x+h)-168.0*func(x+2.0*h)
+32.0*func(x+3.0*h) - 3.0*func(x+4.0*h))/840.0*hh;
```

```
  return df;
}
};

class f1:public f_x
{ public: double func(double x)
  { double ff=log(x*x)-0.7;return ff;}};

double newton_bisection(f_x &f, double x1, double x2)
{
double xacc=1.0e-10; //accuracy
double h=0.0001; //derivative function step
int MAXIT=100;  //Maximum number of iterations
int j;
double dfr,dx,dxold,fr,fh,fl;
double temp,b,a,r;
fl=f.func(x1);
fh=f.func(x2);
//Check if a root is existed in the given region
if ((fl > 0.0 && fh > 0.0) || (fl < 0.0 && fh < 0.0))
cout<<"There are no root in the given region or double root?";
if (fl == 0.0) return x1;
if (fh == 0.0) return x2;
if (fl < 0.0)   { a=x1;b=x2;}
else            { b=x1;a=x2;}
r=0.5*(x1+x2); //midpoint value
dxold=fabs(x2-x1);
dx=dxold;
fr=f.func(r);   //function value at midpoint
dfr=f.dfunc(r); //derivative of function value at midpoint
for (j=1;j<=MAXIT;j++)
{
if ((((r-b)*dfr-fr)*((r-a)*dfr-fr) > 0.0) || (fabs(2.0*fr) > fabs(dxold*dfr)))
   { //bisection step
                dxold=dx;
     dx=0.5*(b-a);
     r=a+dx;
     if (a == r) return r; //çözüm!!!!
   }
else{ //Newton-Raphson (Secant) step
                dxold=dx;
     dx=fr/dfr;
     temp=r;
     r -= dx;
     if (temp == r) return r;  //solution!!!!!
   }
if (fabs(dx) < xacc) return r; //solution!!!!!
fr=f.func(r);
dfr=f.dfunc(r);
if (fr < 0.0)
a=r;
else
b=r;
}
cout <<("Maximum number of iteration is exceeded \n result might not be valid");
return 0.0; //program should never reach here
          //dummy return
}

int main()
{  double x1,x2,r;
   cout << " first guess for the root value : ";
   cin >>x1;
   cout << " second guess for the root value : ";
```

```
  cin >>x2;
  f1 f;
  cout<<setprecision(20);
  r=newton_bisection(f,x1,x2);
  cout<<"root value : ="<<r;
  return 0;
}
```

```
 ---------- Capture Output ----------
> "D:\okul\SCO1\run.bat" newton_bisection
D:\okul\SCO1>newton_bisection
 first guess for the root value : 1
 second guess for the root value : 3
root value : =1.4190675485932571
> Terminated with exit code 0.
```

Program 2.7-3 : Newton-Raphson (Secant)-Bisection combined method Fortran 90 version

```
program newton_bisection1
IMPLICIT NONE
REAL(KIND = SELECTED_REAL_KIND(10,34)) a,b
a=1.0
b=2.15
print *,'newton_bisection(',a,',',b,') = ',newton_bisection(a,b)

CONTAINS
 FUNCTION func(x)
 REAL(KIND = SELECTED_REAL_KIND(10,34)) func,x
 func=x*x-2.0
 END FUNCTION func
 FUNCTION dfunc(x)
 REAL(KIND = SELECTED_REAL_KIND(10,34)) dfunc,x,dx,xp,xn
 dx=1.12314e-4
 xp=x+dx
 xn=x-dx
 dfunc=(func(xp)-func(xn))/(2.0*dx)
 END FUNCTION dfunc
 FUNCTION newton_bisection(a,b)
 REAL(KIND = SELECTED_REAL_KIND(10,34)) newton_bisection,a,b,b1,r,eps,fa,fb,fr,dfr,remain
 INTEGER nmax,i
 eps=1.0e-8;
 nmax=100;
 i=0;
 fa=func(a)
 fb=func(b)
 DO i=1,nmax
  remain=i-(i/4)*4
  if(remain==0) then
   fr=func(r)
   dfr=dfunc(r)
   r=r-fr/dfr
  else
   r=(a+b)/2.0
   fr=func(r)
   IF (fr*fa<0) THEN
     b=r
     fb=fr
   ELSE
     a=r
     fa=fr
   END IF
  end if
  IF (abs(fr)<eps) EXIT
 END DO
```

```
  newton_bisection=r
  END FUNCTION newton_bisection
END PROGRAM newton_bisection1
```

```
---------- Capture Output ----------
> "E:\okul\SCO1\run.bat" newton_bisection1

E:\okul\SCO1>newton_bisection1
 newton_bisection(   1.0000000000000000     ,   2.1500000953674316     ) =    1.4142135609714437

> Terminated with exit code 0.
```

Another problem will be defined as an engineering problem from heat transfer:
Time dependent temperature of a flat wall will be given with the differential equation :
In this equation T is temperature (degree Kelvin), x is the distance(meter), t is the time(second)

$$\frac{\partial T(x,t)}{\partial t} = \alpha \frac{\partial^2 T(x,t)}{\partial x^2} \quad (2.7.1)$$

$$\alpha = \frac{k}{\rho C_p} \quad (2.7.2)$$

And α is thermal diffusivity, where k is thermal conductivity(W/(mK)), ρ is density (kg/m3) and C_p is specific heat(J/(kgK))

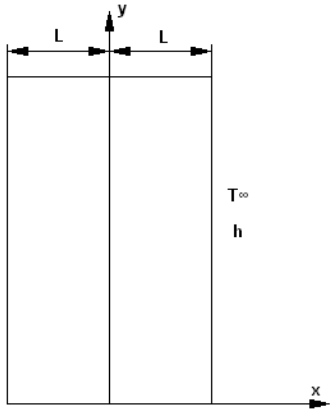


Figure 2.9 geometry of wall in time dependent heat transfer problemT_∞

Boundary conditions:

$$T(x,0) = T_i \quad (2.7.3)$$

$$\left.\frac{\partial T(x,t)}{\partial x}\right|_{x=0} = 0 \quad (2.7.4)$$

$$-k\left.\frac{\partial T(x,t)}{\partial x}\right|_{x=L} = [\![hT(L,t) - T_\infty]\!] \quad (2.7.5)$$

Converted dimensionless form of the equation and boundary conditions
Boundary conditions:

$$\theta(\bar{x},0) = 1 \quad (2.7.6)$$

$$\left.\frac{\partial \theta(\bar{x},t)}{\partial \bar{x}}\right|_{\bar{x}=0} = 0 \quad (2.7.7)$$

$$\left.\frac{\partial \theta(\bar{x},\bar{t})}{\partial \bar{x}}\right|_{\bar{x}=L} = -Bi * \theta(1,Fo) \quad (2.7.8)$$

$$\bar{x} = \frac{x}{L} \quad \bar{t} = Fo = \frac{\alpha t}{L^2} \quad \theta(\bar{x},\bar{t}) = \frac{T(x,t)-T_\infty}{T_i-T_\infty} \quad (2.7.9)$$

Where Bi is called Biot Number, and Fo is caled Fourier number, t is time α is thermal diffusivity coefficient, h is thermal convectivity coefficient, k is thermal conductivity coefficient, T_i is initial time and $T_\infty$is environmental temperature

$$T = T(x,t,T_i,T_\infty,L,k,h,\alpha) \quad \theta = T(\bar{x},Fo,Bi) \quad (2.7.10)$$

Analytic solution of this differential equation :

$$\theta = \sum_{n=1}^{\infty} C_n e^{-\zeta_n^2 Fo} \cos(\zeta_n \bar{x}) \quad (2.7.11)$$

Where

$$C_n = \frac{4\sin(\zeta_n)}{2\zeta_n + \sin(2\zeta_n)} \qquad (2.7.12)$$

And

$$\zeta_n \tan(\zeta_n) = Bi \qquad (2.7.13)$$

As it is seen from the last equation, this equation can only be solved by using numerical root finding methods. After finding the roots coefficients C_n's can be evaluated and heat transfer equation is solved. Program calculated the heat transfer is given as a whole below:

Program 2.7-2 HT_wall_x_t : Newton-Raphson (Secant)-Bisection combined method

```
import java.util.*;
import java.awt.*;
import java.applet.Applet;
import java.awt.event.*;
import javax.swing.*;

public class HT_duvar
{

public static double fx(double lambda,double Bi)
{return lambda*Math.tan(lambda)-Bi;}

public static double dfx(double x,double h,double Bi)
{ //Türev hesabi
  return (-fx((x+2.0*h),Bi)+8.0*fx((x+h),Bi)-8.0*fx((x-h),Bi)+fx((x-2.0*h),Bi))/(12.0*h);
}

public static double Cn(double ln,double Bi)
{
 return 4.0*Math.sin(ln)/(2.0*ln+Math.sin(2.0*ln));
}

public static double lambda(double n,double Bi)
{
 //linear olmayan denklemin kökü : ikiyebölme-Newton Raphson metodu
 //lambda*tan(lambda)-Bi=0 denkleminin köklerini bulur
 double x1=(n-1)*Math.PI;
 double x2=          x1+Math.PI/2-0.000001;
double xacc=1.0e-10; //iterasyonlar arasindaki fark, bu rakamdan
// küçük olduğunda çözüme ulaştiğimizi kabul edeceğiz
double h=0.0001; //türev step büyüklüğü
int MAXIT=100;  //Maksimum iterasyon sayisi
int j;
double dfr,dx,dxold,fr,fh,fl;
double temp,b,a,r;
fl=fx(x1,Bi);
fh=fx(x2,Bi);
//System.out.println("x1 ="+x1+"fl="+fl+"x2="+x2+"fh="+fh);
//Bölgemizde kök değeri varmi kontrol et
if ((fl > 0.0 && fh > 0.0) || (fl < 0.0 && fh < 0.0))
System.out.println("n = "+n+"x1="+x1+"x2="+x2+"verilen sinirlar içinde kök değeri yok");
if (fl == 0.0) return x1;
if (fh == 0.0) return x2;
if (fl < 0.0)   { a=x1;b=x2;}
else          { b=x1;a=x2;}
r=0.5*(x1+x2); //orta nokta değeri
```

```
dxold=Math.abs(x2-x1);
dx=dxold;
fr=fx(r,Bi);   //fonksiyonun r noktasindaki değeri
dfr=dfx(r,h,Bi); //fonksiyonun türevinin r noktasindaki değeri
for (j=1;j<=MAXIT;j++)
{
if ((((r-b)*dfr-fr)*((r-a)*dfr-fr) > 0.0) || (Math.abs(2.0*fr) > Math.abs(dxold*dfr)))
   { //ikiye bölme
            dxold=dx;
     dx=0.5*(b-a);
     r=a+dx;
     if (a == r) return r; // solution!!!!!
   }
else{ //Newton-Raphson
            dxold=dx;
     dx=fr/dfr;
     temp=r;
     r -= dx;
     if (temp == r) return r;  //solution!!!!!
   }
if (Math.abs(dx) < xacc) return r; // solution!!!!!
fr=fx(r,Bi);
dfr=dfx(r,h,Bi);
if (fr < 0.0)
a=r;
else
b=r;
}
System.out.println("Maximum iteration number is exceeded");
return 0.0; //program never reach here
}

public static double teta(double Fo,double Bi,double x)
{double tt=0;double ln=0;
 for(int n=1;n<20;n++)
 {ln=lambda(n,Bi);tt+=Cn(ln,Bi)*Math.exp(-ln*ln*Fo)*Math.cos(ln*x);}
 return tt;
}

    public static void main (String args[])
    { fb f=new fb();
      double Bi,Fo,x;
      Fo=Double.parseDouble(JOptionPane.showInputDialog(" Forier sayisi Fo : "));
      Bi=Double.parseDouble(JOptionPane.showInputDialog(" Biot sayisi   Bi : "));
      x=Double.parseDouble(JOptionPane.showInputDialog("        x* = x/L : "));
      JOptionPane.showMessageDialog(null," teta = "+teta(Fo,Bi,x),
      "duvarda zamana bağli isi transferi  : ",JOptionPane.PLAIN_MESSAGE);
      System.exit(0);
    }
}
```

2.8 INVERSE QUADRATIC LAGRANGE INTERPOLATION

Lagrange interpolation will be investigated further in the interpolation section. In here, the Formula for the quadratic inverse lagrange polinomial will be introduced to utilise in finding roots of a function.

$$x = \frac{[y-f(x_1)][y-f(x_2)]x_3}{[f(x_3)-f(x_1)][f(x_3)-f(x_2)]} + \frac{[y-f(x_2)][y-f(x_3)]x_1}{[f(x_1)-f(x_2)][f(x_1)-f(x_3)]} + \frac{[y-f(x_3)][y-f(x_1)]x_2}{[f(x_2)-f(x_3)][f(x_2)-f(x_1)]} \quad (2.8.1)$$

In this equation if y=0 is taken, the value x will give us a quadratic approximation to the root of the function

$$x = \frac{f(x_1)f(x_2)x_3}{[f(x_3)-f(x_1)][f(x_3)-f(x_2)]} + \frac{f(x_2)f(x_3)x_1}{[f(x_1)-f(x_2)][f(x_1)-f(x_3)]} + \frac{f(x_3)f(x_1)x_2}{[f(x_2)-f(x_3)][f(x_2)-f(x_1)]} \quad (2.8.2)$$

In order to make the equation simpler further definations will apply as:

$R = \frac{f(x_2)}{f(x_3)}$(2.8.4) $S = \frac{f(x_2)}{f(x_1)}$(2.8.5) $T = \frac{f(x_1)}{f(x_3)}$(2.8.6)

$$P = S[T(R-T)(x_3-x_2) - (1-R)(x_2-x_1)] \quad (2.8.7)$$

$$Q = (T-1)(R-1)(S-1) \quad (2.8.8)$$

Now the root can be written as:

$$x = x_2 + \frac{P}{Q} \quad (2.8.9)$$

When a new root is found the neigbour region of the root is left for the next iteration and the region further away to the root is eliminated.

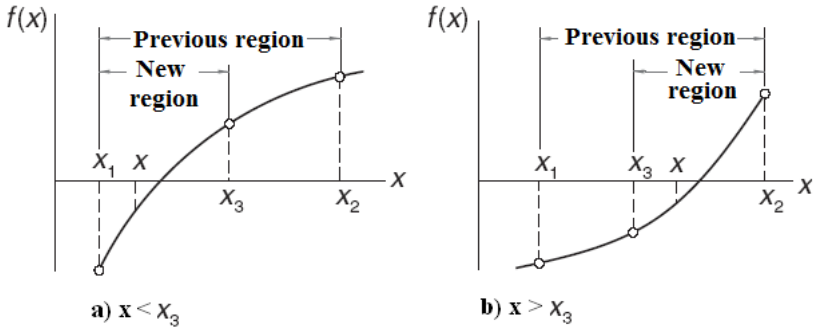
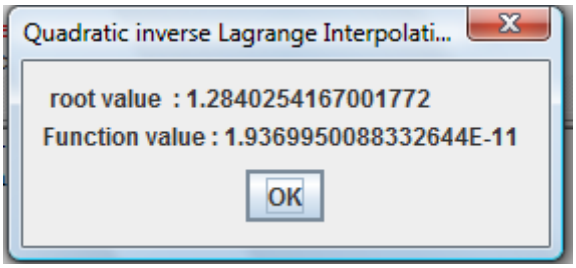


Figure 2.10 Quadratic inverse Lagrange interpolation region selection

Program 2.8-1 NA12 : quadratic inverse lagrange interpolation formula method (java code)

```
public class NA12
{public static double inverseinter2Droot(f_x f,double x1,double x3)
{double x2=(x1+x3)/2.0;
return inverseinter2Droot(f,x1,x2,x3);}
public static double inverseinter2Droot(f_x f,double x1,double x2,double x3)
{// Inverse quadratic Lagrange interpolation
double test;
double p=0;
double es,ea;
double f1,f2,f3,fp;
int maxit=100,iter=0;
double tol=1.0e-10;
es=0.00000001;
ea=0.001;
f1=f.func(x1);
f2=f.func(x2);
f3=f.func(x3);
if(f1==0) return x1;
else if(f2==0) return x2;
```

```
else if(f3==0) return x3;
if(f1*f3>0) System.out.println("Root is not existed in the given region");
while((ea>es)&&(iter<maxit))
{               p=-(f2*f3*x1*(f2-f3)+f3*f1*x2*(f3-f1)+f1*f2*x3*(f1-f2))/((f1-f2)*(f2-f3)*(f3-f1));
   fp=f.func(p);
   ea=Math.abs(x3-p);
               if(Math.abs(f3)<tol) return x3;
   if(p>x3) {x1=x3;f1=f3;x3=p;f3=fp;}
   else if(p<x3) {x2=x3;f2=f3;x3=p;f3=fp;}
   iter++;}
if(iter>=maxit) JOptionPane.showMessageDialog(null,"Warning : MAximum number of iteration is exceeded \n"+
  " çözüm geçerli olmiyabilir","MAXİMUM ITERATION NUMBER WARNING",JOptionPane.WARNING_MESSAGE);
return p;}
    public static void main (String args[])
    {
               fc f=new fc();
    double x1,x2,x3;
    x1=Double.parseDouble(JOptionPane.showInputDialog(" root estimation x1 : "));
    x3=Double.parseDouble(JOptionPane.showInputDialog(" root estimation x2 : "));
    double r;
    r= inverseinter2Droot(f,x1,x3);
    JOptionPane.showMessageDialog(null," root value  : "+r+"\nFunction value : "+f.func(r),
    "Quadratic inverse Lagrange Interpolation method : ",JOptionPane.PLAIN_MESSAGE);
    } }
```



Program 2.8-2 quadratic inverse lagrange interpolation formula method in C++

```
#include <stdio.h>
#include <iostream>
#include <math.h>
#include <iomanip>
using namespace std;

class f_x{
public: virtual double func(double x)=0;
public: double dfunc(double x)
{ double h=0.0001;
  double hh=1.0/h;
  double df=(3.0*func(x-4.0*h)-32.0*func(x-3.0*h)+168.0*func(x-2.0*h)-672.0*func(x-h)+672.0*func(x+h)-168.0*func(x+2.0*h)
+32.0*func(x+3.0*h) - 3.0*func(x+4.0*h))/840.0*hh;
  return df;
}
};

class f1:public f_x
{ public: double func(double x)
  { double ff=x*x-2.0;return ff;}};

double inverseinter2Droot(f_x &f,double x1,double x2,double x3)
{
// Inverse quadratic Lagrange interpolation
double test;
double p=0;
double es,ea;
```

```
double f1,f2,f3,fp;
int maxit=100,iter=0;
double tol=1.0e-10;
if(x1==0) x1=1.0e-10;
if(x2==0) x2=1.0e-10;
if(x3==0) x3=1.0e-10;
es=0.00000001;
ea=0.001;
f1=f.func(x1);
f2=f.func(x2);
f3=f.func(x3);
if(f1==0) return x1;
else if(f2==0) return x2;
else if(f3==0) return x3;
if(f1*f3>0) cout<<"Root is not existed in the given region";
while((ea>es)&&(iter<maxit))
{               p=-(f2*f3*x1*(f2-f3)+f3*f1*x2*(f3-f1)+f1*f2*x3*(f1-f2))/((f1-f2)*(f2-f3)*(f3-f1));
   fp=f.func(p);
   ea=fabs(x3-p);
               if(fabs(f3)<tol) return x3;
   if(p>x3) {x1=x3;f1=f3;x3=p;f3=fp;}
   else if(p<x3) {x2=x3;f2=f3;x3=p;f3=fp;}
   iter=iter+1;
   cout<<"iter="<<iter<<"p="<<p<<"fp="<<fp<<endl;
}
if(iter>=maxit) cout<<"Warning : MAximum number of iteration is exceeded \n";
return p;
}
    int main ()
    { double x1,x2,x3,r;
      cout << " first guess for the root value : ";
      cin >>x1;
      cout << " second guess for the root value : ";
      cin >>x3;
      x2=(x1+x3)/2.0;
      f1 f;
      cout<<setprecision(20);
      r=inverseinter2Droot(f,x1,x2,x3);
      cout<<"root value : ="<<r;
      return 0;
}
```

```
---------- Capture Output ----------
> "D:\okul\SCO1\run.bat" inverseinter2Droot

D:\okul\SCO1>inverseinter2Droot
 first guess for the root value : 1
 second guess for the root value : 3
iter=1p=1.3666666666666667fp=-0.13222222222222213
iter=2p=1.419336630469842fp=0.014516470593484942
iter=3p=1.4141618310219912fp=-0.00014631568052914416
iter=4p=1.4142135655958628fp=9.1153636039027069e-009
iter=5p=1.4142135623730949fp=-3.546425306200085e-016
root value : =1.4142135623730949
> Terminated with exit code 0.
```

Program 2.8-2 quadratic inverse lagrange interpolation formula method in Matlab(Octave)

```
function p = inverseinter2Droot(f,x1,x2,x3)
%Inverse quadratic Lagrange erpolation
p=1;
maxit=100,iter=0;
tol=1.0e-10;
es=0.00000001;
ea=0.001;
```

```
f1=f(x1);
f2=f(x2);
f3=f(x3);
if f1==0 p=x1;
elseif f2==0 p=x2;
elseif f3==0 p=x3;end
if f1*f3>0 fprintf('Root is not existed in the given region');end
while (ea>es) && (iter<maxit) && f1~=0 && f2~=0 && f3~=0
   p=-(f2*f3*x1*(f2-f3)+f3*f1*x2*(f3-f1)+f1*f2*x3*(f1-f2))/((f1-f2)*(f2-f3)*(f3-f1));
   fp=f(p);
   ea=abs(x3-p);
   if abs(f3)<tol  p=x3; end
   if p>x3  x1=x3;f1=f3;x3=p;f3=fp;
   elseif(p<x3) x2=x3;f2=f3;x3=p;f3=fp;end
   iter=iter+1;
   end
if(iter>=maxit) fprintf('Warning : Maximum number of iteration is exceeded \nresult may not be valid');end
end
```

Program 2.8-3 quadratic inverse lagrange interpolation formula method in Python language

```
 from math import *

from math import *
from f_x import *;

class f1(f_x):
  def func(self,x):
     y=x*x-2;
     return y;

def inverseinter2Droot(f,x1,x2,x3):
  #Inverse quadratic Lagrange erpolation
  p=1;
  maxit=100
  iter=0;
  tol=1.0e-10;
  es=0.00000001;
  ea=0.001;
  f1=f.func(x1);
  f2=f.func(x2);
  f3=f.func(x3);
  if f1==0: p=x1;
  elif f2==0: p=x2;
  elif f3==0: p=x3;
  if f1*f3>0: print("Root is not existed in the given region");
  while (ea>es) and (iter<maxit) and f1!=0 and f2!=0 and f3!=0:
    p=-(f2*f3*x1*(f2-f3)+f3*f1*x2*(f3-f1)+f1*f2*x3*(f1-f2))/((f1-f2)*(f2-f3)*(f3-f1));
    fp=f.func(p);
    ea=abs(x3-p);
    if abs(f3)<tol:  p=x3;
    if p>x3:  x1=x3;f1=f3;x3=p;f3=fp;
    elif p<x3: x2=x3;f2=f3;x3=p;f3=fp;
    iter=iter+1;
  if iter>=maxit:  print("Warning : Maximum number of iteration is exceeded");
  return p;

f=f1();
r=inverseinter2Droot(f,1.0,2.0,3.0);
print("r=",r);
```

```
---------- Capture Output ----------
> "D:\co\python\pythonw.exe" inverseinter2Droot.py
r= 1.4142135623730954
> Terminated with exit code 0.
```

```
program quadratic_inverse_lagrange
IMPLICIT NONE
REAL(KIND = SELECTED_REAL_KIND(10,34)) x1,x2,x3
x1=1.0
x2=1.15
x3=2.5
print *,'inverse quadratic lagrange(',x1,',',x2,',',x3,') = ',quad_lagrange(x1,x2,x3)

CONTAINS
 FUNCTION func(x)
 REAL(KIND = SELECTED_REAL_KIND(10,34)) func,x
 func=x*x-2.0
 END FUNCTION func
 FUNCTION dfunc(x)
 REAL(KIND = SELECTED_REAL_KIND(10,34)) dfunc,x,h,df,f1,f2,f3,f4,f5,f6,f7,f8
 h=1.12314e-4
 f1=3.0*func(x-4.0*h)
 f2=-32.0*func(x-3.0*h)
 f3=168.0*func(x-2.0*h)
 f4=-672.0*func(x-h)
 f5=672.0*func(x+h)
 f6=-168.0*func(x+2.0*h)
 f7=32.0*func(x+3.0*h)
 f8=-3.0*func(x+4.0*h)
 dfunc=(f1+f2+f3+f4+f5+f6+f7+f8)/(840.0*h)
 END FUNCTION dfunc
 FUNCTION quad_lagrange(x1,x2,x3)
 REAL(KIND = SELECTED_REAL_KIND(10,34)) quad_lagrange,x1,x2,x3,test,p,es,ea,f1,f2,f3,fp,tol
 INTEGER maxit,iter,i
 p=0
 maxit=100
 iter=0
 if(x1==0) then
  x1=1e-10
 end if
 if(x2==0) then
  x2=1e-10
 end if
 if(x2==0) then
  x2=1e-10
 end if
 es=1e-8
 ea=0.00001
 f1=func(x1)
 f2=func(x2)
 f3=func(x3)
 DO i=1,maxit
  p=-(f2*f3*x1*(f2-f3)+f3*f1*x2*(f3-f1)+f1*f2*x3*(f1-f2))/((f1-f2)*(f2-f3)*(f3-f1))
  fp=func(p)
  ea=abs(x3-p)
  if(abs(f3)<tol) EXIT
  if(p>x3) then
   x1=x3
   f1=f3
   x3=p
   f3=fp
  else if(p<x3) then
   x2=x3
   f2=f3
   x3=p
   f3=fp
  end if
  IF (ea<es) EXIT
 END DO
```

```
  quad_lagrange=p
  END FUNCTION quad_lagrange
END PROGRAM quadratic_inverse_lagrange
```

```
---------- Capture Output ----------
> "E:\okul\SCO1\run.bat" quadratic_inverse_lagrange

E:\okul\SCO1>quadratic_inverse_lagrange
 inverse quadratic lagrange(   1.0000000000000000     ,   1.1499999761581421     ,   2.5000000000000000     ) =
1.4142135623730951

> Terminated with exit code 0.
```

2.9 BRENT METHOD (INVERSE QUADRATIC LAGRANGE INTERPOLATION – BISECTION AND SECANT COMBINED METHOD)



Figure 2.9.1 Richard P Brent

Being a quadratic method, Inverse quadratic Lagrange interpolation is an efficient method by combining it with bi-section and secant method one of the most used root finding method is created. Two versions of the program code and algorithm of the code is given below.

Program 2.9-1 NA13.java : BRENT METHOD (java language version)

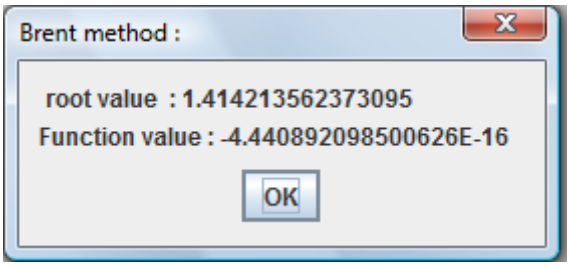
```
/ Numerical Analysis package
// Brent root finding method
import java.util.*;
import java.awt.*;
import java.applet.Applet;
import java.awt.event.*;
import javax.swing.*;
//============== Function Definition ================
class fc extends f_x
{
public double func(double x)
{return x*x-2;}
}
//============================================================

public class NA13
{

public static double brent(f_x f,double a,double b)
{
double test;
double p=0;
double es,ea;
double f1,f2,f3,fp;
int maxit=500,iter=0;
double tol=1.0e-15;
es=0.0000001;
```

```
double x1=a;f1=f.func(x1);
double x2=b;f2=f.func(x2);
double x3=(x1+x2)/2.0;f3=f.func(x3);
if(f1==0) return x1;
else if(f2==0) return x2;
else if(f3==0) return x3;
if(f1*f2>0) System.out.println("No root is existed in the given region");
p=-(f2*f3*x1*(f2-f3)+f3*f1*x2*(f3-f1)+f1*f2*x3*(f1-f2))/((f1-f2)*(f2-f3)*(f3-f1));
fp=f.func(p);
ea=Math.abs(x3-p);
while((ea>es)&&(iter<maxit))
{               if(Math.abs(f3)<tol) return x3;
   if((p<x1) && (p>x2))
   {p=(x1+x2)/2.0;
    if(p>x3) {x1=x3;f1=f3;x3=p;f3=fp;}
    else if(p<x3) {x2=x3;f2=f3;x3=p;f3=fp;}
   }
   else
   {
   if(p>x3) {x1=x3;f1=f3;x3=p;f3=fp;}
   else if(p<x3) {x2=x3;f2=f3;x3=p;f3=fp;}
   p=-(f2*f3*x1*(f2-f3)+f3*f1*x2*(f3-f1)+f1*f2*x3*(f1-f2))/((f1-f2)*(f2-f3)*(f3-f1));
   fp=f.func(p);
   ea=Math.abs(x3-p);
   }
   iter++;
}
if(iter>=maxit) JOptionPane.showMessageDialog(null,"Warning : MAximum number of iteration is exceeded \n"+
  " result may not be valid","MAXIMUM ITERATION NUMBER WARNING",JOptionPane.WARNING_MESSAGE);
return p;
}

    public static void main (String args[])
    {
               fc f=new fc();
    double x1,x2,x3;
    x1=Double.parseDouble(JOptionPane.showInputDialog(" root estimation x1 : "));
    x2=Double.parseDouble(JOptionPane.showInputDialog(" root estimation x2 : "));
    double r;
    r= brent(f,x1,x2);
     JOptionPane.showMessageDialog(null," root value  : "+r+"\nFunction value : "+f.func(r),
    "Brent method : ",JOptionPane.PLAIN_MESSAGE);
    System.exit(0);
    }
}
```



Program 2.9-2 brent.cpp : BRENT METHOD (C++ version)

```
 #include <stdio.h>
#include <iostream>
#include <math.h>
#include <iomanip>
using namespace std;
```

```
class f_x{
public: virtual double func(double x)=0;
public: double dfunc(double x)
{ double h=0.0001;
  double hh=1.0/h;
  double df=(3.0*func(x-4.0*h)-32.0*func(x-3.0*h)+168.0*func(x-2.0*h)-672.0*func(x-h)+672.0*func(x+h)-168.0*func(x+2.0*h)
+32.0*func(x+3.0*h) - 3.0*func(x+4.0*h))/840.0*hh;
  return df;
}
};

class f1:public f_x
{ public: double func(double x)
  { double ff=x*x-2.0;
                return ff;
  }
};

double brent(f_x &f,double a,double b)
{
double test;
double p=0;
double es,ea;
double f1,f2,f3,fp;
int maxit=500,iter=0;
double tol=1.0e-15;
es=0.0000001;
double x1=a;f1=f.func(x1);
double x2=b;f2=f.func(x2);
double x3=(x1+x2)/2.0;f3=f.func(x3);
if(f1==0) return x1;
else if(f2==0) return x2;
else if(f3==0) return x3;
if(f1*f2>0) cout<<"No root is existed in the given region";
p=-(f2*f3*x1*(f2-f3)+f3*f1*x2*(f3-f1)+f1*f2*x3*(f1-f2))/((f1-f2)*(f2-f3)*(f3-f1));
fp=f.func(p);
ea=fabs(x3-p);
while((ea>es)&&(iter<maxit))
{               if(fabs(f3)<tol) return x3;
   if((p<x1) && (p>x2))
   {p=(x1+x2)/2.0;
    if(p>x3) {x1=x3;f1=f3;x3=p;f3=fp;}
    else if(p<x3) {x2=x3;f2=f3;x3=p;f3=fp;}
   }
   else
   {
   if(p>x3) {x1=x3;f1=f3;x3=p;f3=fp;}
   else if(p<x3) {x2=x3;f2=f3;x3=p;f3=fp;}
   p=-(f2*f3*x1*(f2-f3)+f3*f1*x2*(f3-f1)+f1*f2*x3*(f1-f2))/((f1-f2)*(f2-f3)*(f3-f1));
   fp=f.func(p);
   ea=fabs(x3-p);
   }
   iter++;
}
if(iter>=maxit) cout<<"Warning : MAximum number of iteration is exceeded \n";
return p;
}
    int main ()
    {
    double x1,x2,x3;
    cout << "root estimation x1 : ";
    cin >>x1;
    cout << "root estimation x2 : ";
    cin >>x2;
```

```
    double r;
    f1 f;
    cout<<setprecision(20);
    r= brent(f,x1,x2);
    cout<<" root value  : "<<r<<"\nFunction value : "<<f.func(r);
    }
```

```
---------- Capture Output ----------
> "D:\okul\SCO1\run.bat" brent

D:\okul\SCO1>brent
root estimation x1 : 1
root estimation x2 : 3
 root value  : 1.4142135623730951
Function value : 2.7343578790084422e-016
> Terminated with exit code 0.
```

The same code given in python language

Program 2.9-3 brent.py : BRENT METHOD (FIRST VERSION) python code

```
from math import *

def brent(f,a,b):
    #brent root finding method
    maxit=500;
    iter=0;
    tol=1.0e-15;
    es=0.0000001;
    x1=a;f1=f(x1);
    x2=b;f2=f(x2);
    x3=(x1+x2)/2.0;f3=f(x3);
    if f1==0:    return x1;
    elif f2==0:  return x2;
    elif f3==0:  return x3;
    if f1*f2>0: print("No root is existed in the given region");
    p=-(f2*f3*x1*(f2-f3)+f3*f1*x2*(f3-f1)+f1*f2*x3*(f1-f2))/((f1-f2)*(f2-f3)*(f3-f1));
    fp=f(p);
    ea=abs(x3-p);
    while (ea>es) and (iter<maxit):
      if abs(f3)<tol: return x3;
      if (p<x1) and (p>x2):
        p=(x1+x2)/2.0;
        if p>x3:   x1=x3;f1=f3;x3=p;f3=fp;
        elif p<x3: x2=x3;f2=f3;x3=p;f3=fp;
      else:
        if p>x3: x1=x3;f1=f3;x3=p;f3=fp;
        elif p<x3: x2=x3;f2=f3;x3=p;f3=fp;
        p=-(f2*f3*x1*(f2-f3)+f3*f1*x2*(f3-f1)+f1*f2*x3*(f1-f2))/((f1-f2)*(f2-f3)*(f3-f1));
        fp=f(p);
        ea=abs(x3-p);
      iter=iter+1;
    if iter>=maxit: print("Warning : Maximum number of iteration is exceeded");
    return p;

f1 = lambda x: x*x-2
a=float(input("enter lower limit : "))
b=float(input("enter upper limit : "))
r=brent(f1,a,b);
print(" ROOT  : ",r,"\nFUNCTION VALUE : ",f1(r))
```

```
runfile('D:/okul/SCO1/brent.py', wdir='D:/okul/SCO1')
enter lower limit : 1
enter upper limit : 3
```

```
ROOT :  1.414213562373095
FUNCTION VALUE :  -4.440892098500626e-16
```

Program 2.9-4 NA13A.java : BRENT METHOD (SECOND VERSION) java language

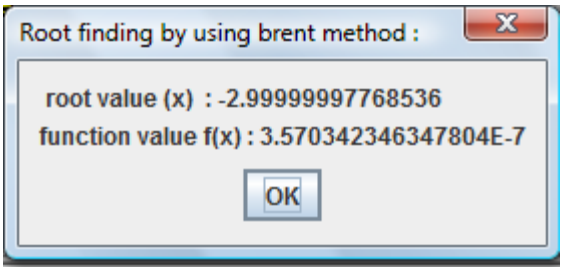
```
// Brent root finding method
import java.util.*;
import java.awt.*;
import java.applet.Applet;
import java.awt.event.*;
import javax.swing.*;

class fc extends f_x
{
public double func(double x)
{return (x+3)*(x-1)*(x-1);}
}

public class NA13A
{

public static double brent(f_x f,double x1,double x2)
{
   // roots of the nonlinear equation: Brent method
   // referance : Numerical Recipes in C, second edition, William H. Press,
   // Saul A. Teukolsky, William T. Vetterling, Brian P. Flannery
   // cambridge university press
            double ITMAX=100;
           double EPS=2.22044604925031e-16;
            int iter;
            double a=x1,b=x2,c,d,e,min1,min2;
            double tol=1.0e-5;
            c=0;
            d=0;
            e=0;
            double fa=f.func(a),fb=f.func(b),fc,p,q,r,s,tol1,xm;
            if (fb*fa > 0.0) JOptionPane.showMessageDialog(null,"Root limit selection is uncorrect \n result may be wrong"
            ,"ROOT LIMITS WARNING",JOptionPane.WARNING_MESSAGE);
            fc=fb;
            for (iter=1;iter<=ITMAX;iter++) {
                        if (fb*fc > 0.0) {
                                    c=a;
                                    fc=fa;
                                    e=d=b-a;
                        }
                        if (Math.abs(fc) < Math.abs(fb)) {
                                    a=b;
                                    b=c;
                                    c=a;
                                    fa=fb;
                                    fb=fc;
                                    fc=fa;
                        }
                        tol1=2.0*EPS*Math.abs(b)+0.5*tol;
                        xm=0.5*(c-b);
                        if (Math.abs(xm) <= tol1 || fb == 0.0) return b;
                        if (Math.abs(e) >= tol1 && Math.abs(fa) > Math.abs(fb)) {
                                    s=fb/fa;
                                    if (a == c) {
                                                p=2.0*xm*s;
                                                q=1.0-s;
                                    } else {
                                                q=fa/fc;
                                                r=fb/fc;
```

```
                                        p=s*(2.0*xm*q*(q-r)-(b-a)*(r-1.0));
                                        q=(q-1.0)*(r-1.0)*(s-1.0);
                                }
                                if (p > 0.0)  q = -q;
                                p=Math.abs(p);
                                min1=3.0*xm*q-Math.abs(tol1*q);
                                min2=Math.abs(e*q);
                                if (2.0*p < (min1 < min2 ? min1 : min2)) {
                                        e=d;
                                        d=p/q;
                                } else {
                                        d=xm;
                                        e=d;
                                }
                        } else {
                                d=xm;
                                e=d;
                        }
                        a=b;
                        fa=fb;
                        if (Math.abs(d) > tol1)
                                b += d;
                        else
                                b += (xm > 0.0 ? Math.abs(tol1) : -Math.abs(tol1));
                        fb=f.func(b);
                }
JOptionPane.showMessageDialog(null,"Maximum number of iterations are exceeded \n"+
  " Result mAy not be valid","MAXIMUM ITERATION NUMBER WARNING",JOptionPane.WARNING_MESSAGE);
return b;
}
    public static void main (String args[])
    {
    double a,b;
    a=Double.parseDouble(JOptionPane.showInputDialog(" Fonksiyonun alt sinirini giriniz a : "));
    b=Double.parseDouble(JOptionPane.showInputDialog(" Fonksiyonun üst sinirini giriniz b : "));
    double r;
    fc f=new fc();
    r= brent(f,a,b);
    JOptionPane.showMessageDialog(null," root value (x)  : "+r+"\nfunction value f(x) : "+f.func(r),
    "Root finding by using brent method : ",JOptionPane.PLAIN_MESSAGE);
    System.exit(0);
    }
}
```



A third version of the Brent method is given below

Program 2.9-5 NA13A : BRENT METHOD (THIRD VERSION)

```
/
//
//
// Brent root finding method
//
//
```

```
import java.util.*;
import java.awt.*;
import java.applet.Applet;
import java.awt.event.*;
import javax.swing.*;

class fc extends f_x
{
public double func(double x)
{return (x+3)*(x-1)*(x-1);}
}
public class NA13F
{
public static double brent(f_x f,double xl,double xu)
{
   // roots of the nonlinear equation: Brent method
   // referances : Numerical Methods for Engineers,
   // Steven C. Chapra, Raymond P. Chanale, sixth edition
   // Mc-Graw-Hill
   // Moler,C.B. Numerical Computing with MATLAB, SIAM,
   // Philedelphia, 2004
   double eps=2.22044604925031e-16;
   double tol=0.000001;
   double s,p,q,r;
   double m;
   double a=xl;double b=xu;double fa=f.func(a);double fb=f.func(b);
   double c=a;double fc=fa;double d=b-c;double e=d;
   int MAXITER=100;
   for(int i=0;i<MAXITER;i++)
   {
   if(fb==0) return b;
   if(Math.signum(fa)==Math.signum(fb))
   {a=c;fa=fc;d=b-c;e=d;
   if(Math.abs(fa)<Math.abs(fb))
   {c=b;b=a;a=c;fc=fb;fb=fa;fa=fc;}
   m=0.5*(a-b);
   tol=2*eps*Math.max(Math.abs(b),1);
   if(Math.abs(m)<=tol || fb==0) return b; //exit
   if(Math.abs(e)>=tol && Math.abs(fc)>Math.abs(fb))
   {s=fb/fc;
    if(a==c) {p=2*m*s;q=1-s;} //secant method
    else     {q=fc/fa;r=fb/fa;p=s*(2*m*q*(q-r)-(b-c)*(r-1));//İnverse quadratic interpolation
             q=(q-1)*(r-1)*(s-1);
             }
    if(p>0) q=-q;
    else    p=-p;
    if(2*p<(3*m*q-Math.abs(tol*q)) && p<Math.abs(0.5*e*q))
    {e=d;d=p/q;}
    else
    {d=m;e=m;}
    }
    c=b;fc=fb;
    if(Math.abs(d)>tol) {b=b+d;}
    else {b=b-Math.signum(b-a)*tol;}
    fb=f.func(b);
   }
   }
   return b;
   }
    public static void main (String args[])
    {
    double a,b;
    a=Double.parseDouble(JOptionPane.showInputDialog(" Enter lower limit of the function a : "));
    b=Double.parseDouble(JOptionPane.showInputDialog(" Enter upper limit of the function  b : "));
```

```
    double r;
    fc f=new fc();
    r= brent(f,a,b);
    JOptionPane.showMessageDialog(null," root value (x)  : "+r+"\nfunction value f(x) : "+f.func(r),
    "Root finding by using brent method : ",JOptionPane.PLAIN_MESSAGE);
    System.exit(0);
    } }
```

Algorithm

Input a,b and a function f
Calculate f(a) and f(b)
if(f(a)*f(b)<0 then give no root error
if |f(a)| < |f(b)| then swap(a,b)
c=a
repeat untill f(b)=0 or s=0 or |b-a| is small enough (convergence)
if(f(a)≠f(c) and f(b)≠f(c) then

$$s = \frac{af(b)f(c)}{(f(a)-f(b))(f(a)-f(c))} + \frac{bf(a)f(c)}{(f(b)-f(a))(f(b)-f(c))} + \frac{cf(a)f(b)}{(f(c)-f(a))(f(c)-f(b))}$$

(inverse quadratic interpolation)
else

$$s = b - f(b)\frac{b-a}{(f(b)-f(a))}$$ (secant rule)

end if
if *(condition 1) s* is not between (3*a* + *b*)/4 and *b*
or *(condition 2)* (mflag is set and |*s*−*b*| ≥ |*b*−*c*| / 2)
or *(condition 3)* (mflag is cleared and |*s*−*b*| ≥ |*c*−*d*| / 2)
or *(condition 4)* (mflag is set and |*b*−*c*| < |δ|)
or *(condition 5)* (mflag is cleared and |*c*−*d*| < |δ|)
then

$$s := \frac{a+b}{2}$$ *(bisection method)*

set mflag
else
clear mflag
end if
calculate *f*(*s*)
d := *c (d is assigned for the first time here; it won't be used above on the first iteration because mflag is set)*
c := *b*
if *f*(*a*) *f*(*s*) < 0 then *b* := *s* else *a* := *s* end if
if |*f*(*a*)| < |*f*(*b*)| then swap (*a*,*b*) end if
end repeat
output *b or s (return the root*

2.10 ROOTS OF A COMPLEX FUNCTION BY NEWTON_RAPHSON METHOD

Roots of a complex function can be found by Newton Raphson method. Complex variables are not defined in standart java libraries, therefore should be user defined. A complex class defined by author will be listed here and used for definition of Newton-Raphson method and other further complex methods. Before giving the complex code let us define basic mathematics of the complex numbers

$z = a + bi$ where $i = \sqrt{-1}$

$z_1 = a_1 + b_1 i \qquad z_2 = a_2 + b_2 i$

$z_1 + z_2 = (a_1 + a_2) + (b_1 + b_2)i$

$z_1 - z_2 = (a_1 - a_2) + (b_1 - b_2)i$

Cojugate$(z) = a - bi$

$z_1 * z_2 = (a_1 * a_2 - b_1 * b_2) + (a_1 * b_2 + b_1 * a_2)i$

$$\frac{z_1}{z_2} = \frac{a_1 + b_1 i}{a_2 + b_2 i} = \frac{(a_1 + b_1 i)(a_2 - b_2 i)}{(a_2 + b_2 i)(a_2 - b_2 i)} = \frac{(a_1 * a_2 + b_1 * b_2) + (b_1 * a_2 - a_1 * b_2)i}{a_2^2 + b_2^2}$$

$z = a + bi = r(\cos(\theta) + i\sin(\theta))$

$r = \sqrt{a^2 + b^2} \quad \theta = \text{atan}\left(\frac{b}{a}\right)$

$z^n = (a + bi)^n = r^n(\cos(n\theta) + i\sin(n\theta))$
$\sqrt{z} = (a + bi)^{0.5} = \sqrt{r}(\cos\left(\frac{\theta}{2}\right) + i\sin\left(\frac{\theta}{2}\right))$

Program 2.10-1 complex class

```
 import java.io.*;
import java.text.*;
import java.util.Locale;
import java.util.*;

import javax.swing.*;

// class c
//   c number abstraction

class c {
   // constructors
public c()
{nr=0;ni=0;}
public c(double nre,double nim)
{nr=nre;ni=nim;}
public static c toComplex(double nre,double nim)
{c c=new c(nre,nim);return c;}
public c(int nre,int nim)
{nr=(double)nre;ni=(double)nim;}
public static c toComplex(int nre,int nim)
{c c=new c(nre,nim);return c;}
public c(double numer)
{nr=numer;ni=0;}

public static c toComplex(double nre)
{c c=new c(nre,0.0);return c;}

public c(c value)
{nr=value.real();ni=value.imaginary();}

public static c p(String s,c x1)
{ c z=new c();
if(s.equals("sin")) {z=sin(x1);}
else if(s.equals("cos")) {z=cos(x1);}
else if(s.equals("tan")) {z=tan(x1);;}
if(s.equals("asin")) {z=asin(x1);}
else if(s.equals("acos")) {z=acos(x1);}
else if(s.equals("atan")) {z=atan(x1);;}
else if(s.equals("exp")) {z=exp(x1);}
else if(s.equals("sqrt")) {z=sqrt(x1);}
else if(s.equals("cbrt")) {z=cbrt(x1);}
else if(s.equals("log")) {z=log(x1);}
return z;
}
public static c p(String s,double x1,double x2)
{ c z=new c();
if(s.equals("+")) {z=(new c((x1+x2)));}
else if(s.equals("-")) {z=(new c((x1-x2)));}
```

```
else if(s.equals("*")) {z=(new c((x1*x2)));}
else if(s.equals("/")) {z=(new c((x1/x2)));}
else if(s.equals("pow")) {z=(new c(Math.pow(x1,x2)));}
return z;
}
public static c p(double x1,String s,double x2)
{ return p(s,x1,x2);}

public static c p(String s,c x1,double x2)
{c z=new c();
if(s.equals("+")) {z=c.add(x1,new c(x2));}
else if(s.equals("-")) {z=c.substract(x1,new c(x2));}
else if(s.equals("*")) {z=c.multiply(x1,new c(x2));}
else if(s.equals("/")) {z=c.divide(x1,new c(x2));}
return z;
}
public static c p(c x1,String s,double x2)
{ return p(s,x1,x2);}
public static c p(String s,double x1,c x2)
{ c z=new c();
if(s.equals("+")) {z=c.add(new c(x1),x2);}
else if(s.equals("-")) {z=c.substract(new c(x1),x2);}
else if(s.equals("*")) {z=c.multiply(new c(x1),x2);}
else if(s.equals("/")) {z=c.divide(new c(x1),x2);}
return z;
}
public static c p(double x1,String s,c x2)
{ return p(s,x1,x2);}
public static c p(String s,c x1,c x2)
{ c z=new c();
if(s.equals("+")) {z=c.add(x1,x2);}
else if(s.equals("-")) {z=c.substract(x1,x2);}
else if(s.equals("*")) {z=c.multiply(x1,x2);}
else if(s.equals("/")) {z=c.divide(x1,x2);}
return z;
}
public static c p(c x1,String s,c x2)
{ return p(s,x1,x2);}
// accessor functions
public double real()
{//return real part of the c number
return nr;}

public double imaginary()
{//return imaginary part of the c number
return ni;}
public double R()
{// Return radius of polar coordinate equivalent of c number
return Math.sqrt(nr*nr+ni*ni);}
public double theta()
{// Return angle of polar coordinate equivalent of c number in radian
return Math.atan2(ni,nr);}
public double dtheta()
{// Return angle of polar coordinate equivalent of c number in degree
return Math.atan2(ni,nr)*180.0/Math.PI;}
   // assignments
public void assign(c right)
```

```
{nr=right.real();ni=right.imaginary();}
public void add(c  right)
{ni = ni + right.imaginary();nr = nr + right.real();}
public void add(double  nr1)
{nr = nr+nr1;}
public static int sgn(c left)
{int i=(int)(left.real()/Math.abs(left.real()));
 return i;}
public void add(double  nr1,double ni1)
{nr = nr+nr1;ni = ni + ni1;}
public void substract(c right)
{ni = ni - right.imaginary();nr = nr - right.real();}
public void multiply(c right )
{nr = nr*right.real() - ni*right.imaginary();
ni = nr*right.imaginary() +  ni*right.real();}
public void divide(c right )
{double a=nr*nr+ni*ni;
nr = ( nr*right.real() +  ni*right.imaginary())/a;
ni = (-nr*right.imaginary() +  ni*right.real())/a;}
public static c add(c left, c right)
{ // return sum of two c numbers
 double r1=(left.real() + right.real());
 double i1=(left.imaginary() + right.imaginary());
 c result;
 result=new c(r1,i1);
 return result;}
public static c add(c left, double right)
{ // return sum of two c numbers
 double r1=(left.real() + right);
 double i1=left.imaginary();
 c result;
 result=new c(r1,i1);
 return result;
}
public static c conjugate(c z)
{c z1=new c(z.real(),-z.imaginary());return z1;}
public static c substract(c left, c right)
{ // return substraction of two c numbers
 c result;
 result=new c((left.real() - right.real()),
              (left.imaginary() - right.imaginary()));
 return result;
}
public static c multiply(c left, c right)
{ // return multiplication of two c numbers
 c result;
 result=new c
 ((left.real()*right.real() - left.imaginary()*right.imaginary()),
  (left.real()*right.imaginary() + left.imaginary()*right.real()));
  //System.out.println("left="+left.toString()+"right="+right.toString()+"result="+result.toString());
return result;}
public static c multiply(c left, double  right)
{c result;
 result=new c((left.real()*right),(left.imaginary()*right));
 return result;}
public static c multiply(double  right,c left )
{c result;
```

```
 result=new c((left.real()*right),(left.imaginary()*right));
 return result;}
public static c divide(c left, c  right)
{ // return division of two c numbers
 double a=right.real()*right.real()+right.imaginary()*right.imaginary();
 c result;
 result=new c
 ((left.real()*right.real()  + left.imaginary()*right.imaginary())/a,
  (-left.real()*right.imaginary() + left.imaginary()*right.real())/a);
 return result;}
public static c divide(c left, double  right)
{ // return division of a c number to a real number
 c result;
 result=new c((left.real()/right),(left.imaginary()/right));
 return result;
}
public static c pow(c left, double right)
{ // return sum of two c numbers
 double Rad,th;
 Rad=Math.pow(left.R(),right);
 th=right*left.theta();
 c result=new c((Rad*Math.cos(th)),(Rad*Math.sin(th)));
 return result;
}

public static c pow(double left, c right)
{ double p1=Math.log(left);
  c p2=c.multiply(right,p1);
  c p3=exp(p2);
  return p3;
}

public static c pow(c left, c right)
{ c p1=log(left);
  c p2=c.multiply(right,p1);
  c p3=exp(p2);
  return p3;
}

public static c exp(c left)
{ // exp(x+i*y)
 c result;
 result =new c((Math.exp(left.real())*Math.cos(left.imaginary()) ),
                (Math.exp(left.real())*Math.sin(left.imaginary()) ));
 return result;
}

public static c exp(double left)
{ // exp(i*left) imaginary exponent
 c result;
 result =new c(Math.cos(left),
                Math.sin(left) );
 return result;}
public static c sqrt(c z)
{return pow(z,0.5);}
public static c cbrt(c z)
{return pow(z,(1.0/3.0));}
```

```
public static double abs(c z)
{return z.R();}
public static c log(c z)
{ c z1;
  double x=Math.log(Math.sqrt(z.nr*z.nr+z.ni*z.ni));
  double y=Math.atan2(z.ni,z.nr);
  z1=new c(x,y);
  return z1;
}
public static c sin(c z)
{ c z1,z2,z3;
  z1=c.multiply(z,new c(0.0,1.0));
  z2=c.multiply(z,new c(0.0,-1.0));
  z3=c.substract(exp(z1),exp(z2));
  z3=c.divide(z,new c(0.0,2.0));
  return z3;
}
public static c asin(c z)
{ c z1,z2,z3,z4,z5,z6;
  z1=c.multiply(z,new c(0.0,1.0));//iz
  z2=c.substract(new c(1.0),c.multiply(z,z));//(1-z^2)
  z3=c.sqrt(z2);//(1-z*z)^.5
  z4=c.add(z1,z3);
  z5=log(z4);
  z6=c.multiply(z5,new c(0.0,-1.0));
  return z6;
}
public static c sinh(c left)
{ c z,z1,z2;
  z1=c.multiply(left,1.0);
  z2=c.multiply(left,1.0);
  z=c.substract(exp(z1),exp(z2));
  z=c.divide(z,2.0);
  return z;
}
public static c cos(c left)
{ c z,z1,z2;
  z1=c.multiply(left,new c(0.0,1.0));
  z2=c.multiply(left,new c(0.0,-1.0));
  z=c.add(exp(z1),exp(z2));
  z=c.divide(z,new c(2.0,0.0));
  return z;
}
public static c acos(c z)
{ c z1,z2,z3,z4,z5,z6,z7;
  z1=c.multiply(z,new c(0.0,1.0));//iz
  z2=c.substract(new c(1.0),c.multiply(z,z));//(1-z^2)
  z3=c.sqrt(z2);//(1-z*z)^.5
  z4=c.add(z1,z3);
  z5=log(z4);
  z6=c.multiply(z5,new c(0.0,1.0));
  z7=c.add(z6,Math.PI/2.0);
  return z7;
}

public static c cosh(c z)
{ c z1,z2,z3;
```

```
  z1=c.multiply(z,1.0);
  z2=c.multiply(z,-1.0);
  z3=c.add(exp(z1),exp(z2));
  z3=c.divide(z,2.0);
  return z3;
}

public static c tan(c z)
{return c.divide(sin(z),cos(z));}

public static c atan(c z)
{ c z1,z2,z3,z4,z5,z6;
  z1=c.multiply(z,new c(0.0,-1.0));
  z2=c.substract(new c(1.0),z1);
  z3=c.multiply(z,new c(0.0,1.0));
  z4=c.substract(new c(1.0),z3);
  z5=c.substract(log(z2),log(z4));
  z6=c.multiply(z5,new c(0.0,0.5));
  return z6;
}
public static c tanh(c left)
{return c.divide(sinh(left),cosh(left));}
public boolean smaller(c left,c right)
{return (left.R() < right.R());}
public boolean smaller_equal(c left,c right)
{return (left.R() <= right.R());}
public boolean greater(c left,c right)
{// greater then comparison of two c numbers
return left.R() > right.R();}
public boolean greater_equal(c left,c  right)
{// greater then and equal comparison of two c numbers
return left.R() >= right.R();}
public boolean equal(c left,c right)
{// equal comparison of two c numbers
return left.R() == right.R();}
public boolean not_equal(c left,c  right)
{// not equal comparison of two c numbers
return left.R() != right.R();}
public static String toString(double left, int w, int d)
// converts a double to a string with given width and decimals.
{   NumberFormat df=NumberFormat.getInstance(Locale.US);
            df.setMaximumFractionDigits(d);
   df.setMinimumFractionDigits(d);
   df.setGroupingUsed(false);
   String s = df.format(left);
   while (s.length() < w)
     s = " " + s;
   if (s.length() > w)
   {s = "";for (int i=0; i<w; i++) s = s + "-";
   }
   return s;
}

public static String toString(double left)
{// converts a double to a string with a constant width and constant decimals.
  return toString(left,15,10);}
public static String toString(c  value)
```

```
{ return value.toString();}

public String toString()
{ return toString("%10.5f");}

public String toString(String f)
{String b="";
 String s1="("+f+" + "+f+"*i ) ";
 String s2="("+f+" - "+f+"*i ) ";
 StringBuilder sb = new StringBuilder();
 Formatter fo = new Formatter(sb, Locale.US);
if(Math.abs(imaginary())!=1)
  {
  if(imaginary() >= 0)
    b=String.format(Locale.US,s1,real(),imaginary());
  else
    b=String.format(Locale.US,s2,real(),-imaginary());
  }
return b;
}

public double toDouble()
{ if(        imaginary()==0) return nr;
  else               return R();
}
public double[] toDoubleArray()
{
double a[]={nr,ni};
return a;
}

public static double[] toDoubleArray(complex value)
{ double a[]={value.nr,value.ni};
  return a;
}

public static double toDouble(c w)
{
if(w.imaginary()==0) return w.real();
else               return w.R();
}

    public static c toComplex(String s)
    {
            //bu metod compleks sayıyı ekrandan okur.
            //StringTokanizer kütüphane sınıfı bir stringi cümlelere ayırır
    String s1=JOptionPane.showInputDialog(s);
    StringTokenizer token=new StringTokenizer(s1);
    int n=token.countTokens()-1;
    int m=n+1;
    double a[]=new double[m];
    int j=0;
    while(token.hasMoreTokens())
    {
    Double ax=new Double(token.nextToken());
    a[j++]=ax.doubleValue();
    }
```

```
    c b=new c(a[0],a[1]);
    return b;
    }
    public static c toComplex()
    {return toComplex("input c number:");}
   // data areas
   public double   nr;
   public double   ni;
   //Matrix an vector input system for complex variable
   public static int countcharacters(char a,String s)
{
int n=s.length();
int number=0;
for(int i=0;i<n;i++)
{if(s.charAt(i)==a) number++;}
return number;
}

public static String[] word(char a,String s)
{
int nn=countcharacters(a,s);
int n=s.length();
String s1[]=new String[nn+1];
String s2="";
int j=0;
for(int i=0;i<n;i++)
{if(s.charAt(i)==a) {s1[j]=s2;s2="";j++;}
 else              {s2+=s.charAt(i);}
}
s1[nn]=s2;
return s1;
}
public static String[] word(char a1,char a2,char a3,String s)
{
int nn1=countcharacters(a1,s);
int nn2=countcharacters(a2,s);
if(nn1!=nn2) System.out.println("char couple is not match");
int n=s.length();
String s1[]=new String[nn1+1];
String s2="";
int j=0;
for(int i=0;i<n;i++)
{if(s.charAt(i)==a1) {s1[j]=s2;s2="";}
 else if(s.charAt(i)!=a2) {s2+=s.charAt(i);}
 else if(s.charAt(i)==a2) {j++;}
}
s1[nn1]=s2;
String s3[]=new String[nn1];
int n1=0;
for(int i=0;i<nn1;i++)
{n1=s1[i+1].length();
 if(s1[i+1].charAt(n1-1)==a3) s1[i+1]=s1[i+1].substring(0,n1-1);
 s3[i]=s1[i+1];
}
return s3;
}
```

```
public static String[] word(String s)
{
char a=' ';
return word(a,s);
}

public static String toString(String s[])
{int n=s.length;
 String ss="";
 for(int i=0;i<n;i++)
 {ss+="i = "+i+" "+s[i]+"\n";}
  return ss;
}
public static String toString(String s[][])
{int n=s.length;
 int m=s[0].length;
 String ss="";
 for(int i=0;i<n;i++)
 {
 for(int j=0;j<m;j++)
 {ss+="  "+s[i][j].toString();}
 ss+="\n";
 }
 return ss;
}

public static String toString(c s[][])
{int n=s.length;
 int m=s[0].length;
 String ss="";
 for(int i=0;i<n;i++)
 {
 for(int j=0;j<m;j++)
 {ss+="  "+s[i][j].toString();}
 ss+="\n";
 }
 return ss;
}

public static String toString(c s[])
{int n=s.length;
 String ss="";
 for(int i=0;i<n;i++)
 {ss+="  "+s[i];}
 return ss;
}

public static c[][] toComplexMatrix(String s)
{String s1[]=word('{','}',',',s);
int n=s1.length;
String s2[][]=new String[n][];
for(int i=0;i<n;i++)
  s2[i]=word('(',')',',',s1[i]);
int m=s2[0].length;
String s3[][][]=new String[n][m][];
c z[][]=new c[n][m];
for(int i=0;i<n;i++)
```

```
 {for(int j=0;j<m;j++)
  {s3[i][j]=word(',',s2[i][j]);}
 }
int k=s3[0][0].length;
double d[][][]=new double[n][m][k];
for(int i=0;i<n;i++)
 {for(int j=0;j<m;j++)
  {for(int l=0;l<k;l++){d[i][j][l]=Double.parseDouble(s3[i][j][l]);}}
 }
for(int i=0;i<n;i++)
 {for(int j=0;j<m;j++)
  {z[i][j]=toComplex(d[i][j][0],d[i][j][1]);}
 }
return z;
}

public static c[] toComplexVector(String s)
{
c[][] z1=toComplexMatrix(s);
return z1[0];
}
   public static void main(String arg[])
   {c a=c.toComplex();
    c b=toComplex();
    c c1=p("+",a,b);
    System.out.println("a="+a+"\nb="+b+"\nc="+c1);
   }}
```

Program 2.10-2 abstract class cf_cx general function and derivatives definitions of a complex variable

```
abstract class cf_cx
{
  abstract c func(c x);
 //complex derivative
  public c dfunc(c x)
  { // c numerical derivative
   double hh=0.001;
   c h=new c(hh,hh);
   c he=new c(-hh,-hh);
   c h2=new c(2.0*hh,2.0*hh);
   c he2=new c(-2.0*hh,-2.0*hh);
   c xah=c.p(x,"+",h);
   c xa2h=c.p(x,"+",h2);
   c xeh=c.p(x,"+",he);
   c xe2h=c.p(x,"+",he2);
   c fe2h=c.p(func(xe2h),"*",(1.0/12.0));
   c feh=c.p(func(xeh),"*",(-8.0/12.0));
   c fah=c.p(func(xah),"*",(8.0/12.0));
   c fa2h=c.p(func(xa2h),"*",(-1.0/12.0));
   c df=c.p(fe2h,"+",c.p(feh,"+",c.p(fah,"+",fa2h)));
   df=c.p(df,"/",h);
   return df;
  }
//complex second derivative
  public c d2func(c x)
  { // c sayısal ikinci türev fonksiyonu
   double hh=0.001563465314;
   c h=new c(hh,hh);
   c h_2=c.multiply(h,h);
   c he=new c(-hh,-hh);
   c h2=new c(2.0*hh,2.0*hh);
   c he2=new c(-2.0*hh,-2.0*hh);
   c xah=c.p(x,"+",h);
   c xa2h=c.p(x,"+",h2);
```

```
   c xeh=c.p(x,"+",he);
   c xe2h=c.p(x,"+",he2);
   c fe2h=c.p(func(xe2h),"*",(-1.0/12.0));
   c feh=c.p(func(xeh),"*",(16.0/12.0));
   c f=c.p(func(x),"*",(-30.0/12.0));
   c fah=c.p(func(xah),"*",(16.0/12.0));
   c fa2h=c.p(func(xa2h),"*",(-1.0/12.0));
   c df=c.p(fe2h,"+",c.p(feh,"+",c.p(f,"+",c.p(fah,"+",fa2h))));
   df=c.p(df,"/",h_2);
   return df;
  }
}
```

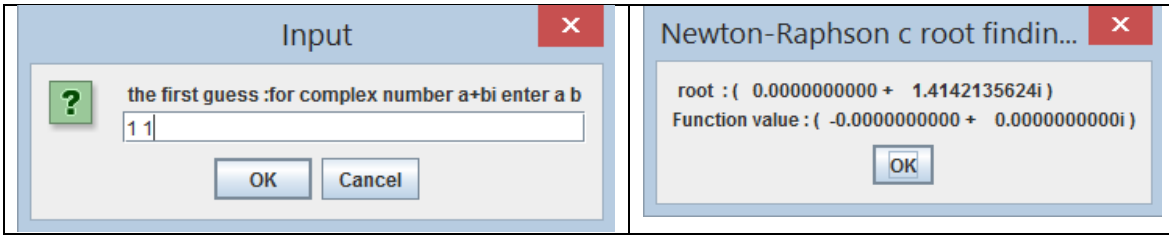
Now a complex definition of Newton-Raphson method can be given. Our example function is $f(x)=x^2+2$

Program 2.10-3 NA5A1 complex Newton-Raphson method

```
 // complex number newton-raphson root finding
import java.util.*;
import java.awt.*;
import java.applet.Applet;
import java.awt.event.*;
import javax.swing.*;
class fa extends cf_cx
{  //one function, one independent variable
  // example function f=x*x-2
c func(c x)
{ c f=c.p(c.p(x,"*",x),"+",2.0);
  return f;
}}

public class NA5A1
{

public static c newton(cf_cx f,c x)
{
int nmax=50;
double tolerance=1.0e-8;
c fx,dfx;
for(int i=0;i<nmax;i++)
{
  fx=f.func(x);
  dfx=f.dfunc(x);
  //x=c.add(x,c.multiply(c.divide(fx,dfx),-1.0));
  x=c.p(x,"-",c.p(fx,"/",dfx));
  if(c.abs(fx)<tolerance) { return x;}
}
JOptionPane.showMessageDialog(null,"Maximum number of iteration is exceeded \n"+
   " results may not be valid","MAXIMUM ITERATION WARNING",JOptionPane.WARNING_MESSAGE);
return x;
}
    public static void main (String args[])
    { fa f=new fa();
    double x0[]=NA5A.enterdata(" the first guess :for complex number a+bi enter a b ");
    c x1=new c(x0[0],x0[1]);
    c r= newton(f,x1);
    c ff=f.func(r);
    JOptionPane.showMessageDialog(null," root  : "+r+"\nFunction value : "+ff.toString(),
    "Newton-Raphson c root finding method : ",JOptionPane.PLAIN_MESSAGE);
    System.exit(0);
    }
}
```



In order to make a lambda expression version of the program, an interface version of the abstract function cf_cx is created as icf_x

Program 2.10-4 icf_x interface class

```
 import static java.lang.Math.*;
import java.io.*;
import java.util.*;
import javax.swing.*;
import javax.swing.table.*;
import java.awt.*;
  // single function single independent variable
  // example f=x*x
  // includes full set of derivatives
  // Reference :"Generation of Finite Difference Formulas on Arbitrary Spaced Grids",
  // Bength Fornberg, Mathematics of Computation, Volume 51, Number 184, October 1988
  // pages 699-706
  //interface version
@FunctionalInterface
interface icf_x
{  public c func(c x);
// nth order derivative
default c func(double x)
  {c x1=new c(x,0.0);
   return func(x1);
  }
  default c dfunc(c x)
  { // c sayısal türev fonksiyonu
   double hh=0.001563465314;
   c h=new c(hh,hh);
   c he=new c(-hh,-hh);
   c h2=new c(2.0*hh,2.0*hh);
   c he2=new c(-2.0*hh,-2.0*hh);
   c xah=c.add(x,h);
   c xa2h=c.add(x,h2);
   c xeh=c.add(x,he);
   c xe2h=c.add(x,he2);
   c fe2h=c.multiply(func(xe2h),(1.0/12.0));
   c feh=c.multiply(func(xeh),(-8.0/12.0));
   c fah=c.multiply(func(xah),(8.0/12.0));
   c fa2h=c.multiply(func(xa2h),(-1.0/12.0));
   c df=c.add(fe2h,c.add(feh,c.add(fah,fa2h)));
   df=c.divide(df,h);
   return df;
  }
  default c dfunc(double x)
  { c x1=new c(x,0.0);
   return dfunc(x1);
  }
  default c d2func(c x)
  { // c sayısal ikinci türev fonksiyonu
   double hh=0.001563465314;
   c h=new c(hh,hh);
   c h_2=c.multiply(h,h);
   c he=new c(-hh,-hh);
   c h2=new c(2.0*hh,2.0*hh);
   c he2=new c(-2.0*hh,-2.0*hh);
   c xah=c.add(x,h);
   c xa2h=c.add(x,h2);
   c xeh=c.add(x,he);
   c xe2h=c.add(x,he2);
   c fe2h=c.multiply(func(xe2h),(-1.0/12.0));
   c feh=c.multiply(func(xeh),(16.0/12.0));
   c f=c.multiply(func(x),(-30.0/12.0));
   c fah=c.multiply(func(xah),(16.0/12.0));
   c fa2h=c.multiply(func(xa2h),(-1.0/12.0));
   c df=c.add(fe2h,c.add(feh,c.add(f,c.add(fah,fa2h))));
   df=c.divide(df,h_2);
   return df;
}
default c d2func(double x)
{ c x1=new c(x,0.0);
```

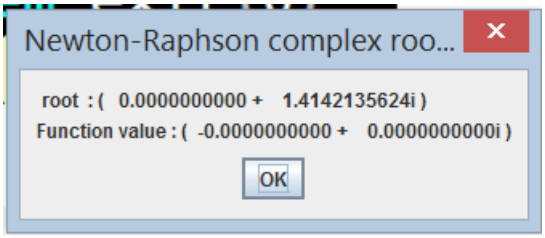
```
 return d2func(x1);
}}
```

Now we can use an lambda expression version of complex Newton-Raphson root finding method.

Program 2.10-5 complex Newton-Raphson method (lambda expression version) NA5A2.java

```
 import java.applet.Applet;
import java.awt.event.*;
import javax.swing.*;

public class NA5A2
{
public static c newton(icf_x f,c x)
{
int nmax=500;
double tolerance=1.0e-10;
c fx,dfx;
for(int i=0;i<nmax;i++)
{
  fx=f.func(x);
  dfx=f.dfunc(x);
  x=c.p(x,"-",c.p(fx,"/",dfx));
  if(c.abs(fx)<tolerance) { return x;}
}
JOptionPane.showMessageDialog(null,"Maximum number of iteration is exceeded \n"+
   " results may not be valid","MAXIMUM ITERATION WARNING",JOptionPane.WARNING_MESSAGE);
return x;
}
public static double[] enterdata(String s)
    {
    String s1=JOptionPane.showInputDialog(s);
    StringTokenizer token=new StringTokenizer(s1);
    int n=token.countTokens()-1;
    int m=n+1;
    double a[]=new double[m];
    int j=0;
    while(token.hasMoreTokens())
    {
    Double ax=new Double(token.nextToken());
    a[j++]=ax.doubleValue();
    }
    return a;
    }
    public static void main (String args[])
    { icf_x f=(c x)->c.p(c.p(x,"*",x),"+",2.0);
    double x0[]=enterdata(" the first guess for the function : ");
    c x1;
    if(x0.length==1) x1=new c(x0[0],0.0);
    else
                x1=new c(x0[0],x0[1]);
    c r= newton(f,x1);
    c ff=f.func(r);
    JOptionPane.showMessageDialog(null," root  : "+r+"\nFunction value : "+ff,
    "Newton-Raphson c root finding method : ",JOptionPane.PLAIN_MESSAGE);
    System.exit(0);
    }
}
```



In C++ in most of the compilers complex class is existed so we can use it directly in our programs similar to double variable

Program 2.10-6 complex Newton-Raphson method C++ version cnewton.cpp

```
 #include <iostream>
#include <math.h>
#include <complex>
#include <iomanip>
using namespace std;
typedef complex<double> cdouble;

class cf_x{
public: virtual cdouble func(cdouble x)=0;
public: cdouble dfunc(cdouble x)
{ cdouble h=0.0001;
  cdouble hh=1.0/h;
  cdouble df=(3.0*func(x-4.0*h)-32.0*func(x-3.0*h)+168.0*func(x-2.0*h)-672.0*func(x-h)+672.0*func(x+h)-168.0*func(x+2.0*h)
+32.0*func(x+3.0*h) - 3.0*func(x+4.0*h))/840.0*hh;
  return df;
}
};

class f1:public cf_x
{ public: cdouble func(cdouble x)
  { cdouble ff=x*x+2.0;return ff;}};

cdouble newton(cf_x &f,cdouble x)
{
int nmax=100;
double tolerance=1.0e-8;
cdouble fx,dfx;
for(int i=0;i<nmax;i++)
{ fx=f.func(x);
  dfx=f.dfunc(x);
  x-=fx/dfx;
  cout <<"i="<<i<<"x="<<x<<"fx="<<fx<<"dfx="<<dfx<<"\n";
  if(abs(fx)<tolerance) { return x;}
}
cout <<("Maximum number of iteration is exceeded \n result might not be valid");
return x;
}
//cnewton.cpp
int main()
{  f1 f;
    cdouble xa,r;
   cout << " first guess for the root value : ";
   cin >>xa;
   cout<<setprecision(20);
   r=newton(f,xa);
   cout<<"root value : ="<<r;
   return 0;
}
```

```
 ---------- Capture Output ----------
> "D:\okul\SCO1\run.bat" cnewton

D:\okul\SCO1>cnewton
 first guess for the root value : (1,1)
i=0x=(-1.7097434579227411e-013,0.99999999999857003)fx=(2,2)dfx=(2.000000000002518,1.9999999999967979)
i=1x=(-1.7182023000249511e-013,1.500000000000715)fx=(1.0000000000028599,-3.4194869158405925e-
013)dfx=(6.872809200604933e-013,1.9999999999971401)
i=2x=(2.8783559897908822e-014,1.4166666666667063)fx=(-0.25000000000214495,-5.1546069000773101e-
013)dfx=(1.0362081563168128e-012,3.0000000000014291)
i=3x=(0,1.4142156862745099)fx=(-0.0069444444445565523,8.1553419710743948e-014)dfx=(0,2.8333333333334125)
i=4x=(0,1.4142135623746899)fx=(-6.0073048828712672e-006,0)dfx=(0,2.8284313725490202)
i=5x=(0,1.4142135623730951)fx=(-4.510614104447086e-012,0)dfx=(0,2.8284271247493797)
root value : =(0,1.4142135623730951)
> Terminated with exit code 0.
```

In Python language, in order to make a complex variable code you only change the input variable. Function will remain exactly the same.

Program 2.10-6 complex Secant method in Python (lambda version) secant.py

```
 from math import *

#def df(f,x):
#  h=0.001
#   return (f(x+dx)-f(x-dx))/(2.0*h)
def df(f,x):
  h=0.001
  return (-f(x+2.0*h)+8.0*f(x+h)-8.0*f(x-h)+f(x-2.0*h))/(12.0*h);

def secant(f,x):
  # secant root finding method
  nmax=100
  tolerance=1.0e-10
  for i in range(0,nmax):
     fx= f(x);
     dfx=df(f,x)
     x=x-fx/dfx
     print("fx",fx,"dfx",dfx,"x=",x)
     if abs(fx)<tolerance: return x
  return x

f1 = lambda x: x*x+2
a=complex(1,1);
r=secant(f1,a)
print(" ROOT  : ",r,"\nFUNCTION VALUE : ",f1(r))
```

```
 ---------- Capture Output ----------
> "D:\co\anaconda\pythonw.exe" secant.py
fx (2+2j) dfx (2.0000000000000018+1.9999999999998908j) x= (-2.6867397195928788e-14+0.9999999999999722j)
fx (1.0000000000000555-5.373479439185609e-14j) dfx (-1.4802973661668753e-13+1.9999999999999445j) x= (3.700743415417599e-
14+1.5000000000000138j)
fx (-0.2500000000000413+1.11022302462529e-13j) dfx (-3.700743415417188e-14+3.0000000000000275j) x= (-1.0279842820604793e-
15+1.4166666666666674j)
fx (-0.006944444444446418-2.912622132504693e-15j) dfx 2.833333333333335j x= 1.4142156862745099j
fx (-6.007304882871267e-06+0j) dfx 2.8284313725490198j x= 1.4142135623746899j
fx (-4.510614104447086e-12+0j) dfx 2.82842712474938j x= 1.4142135623730951j
 ROOT  : 1.4142135623730951j
FUNCTION VALUE :  (-4.440892098500626e-16+0j)

> Terminated with exit code 0.
```

If functions defined by using abstract class f_x, code will be

Program 2.10-7 complex Secant method in Python (abstract class f_x derived class function version)

```
rom math import *
from f_x import *;

class f1(f_x):
  def func(self,x):
     y=x*x+2;
     return y;

def secant(f,x):
   # secant root finding method
  nmax=100
  tolerance=1.0e-10
  for i in range(0,nmax):
     fx= f.func(x);
     dfx=f.dfunc(x,1);
     x=x-fx/dfx
     print("fx",fx,"dfx",dfx,"x=",x)
     if abs(fx)<tolerance: return x
  return x

f=f1();
a=complex(1,1);
#a=complex(input("enter initial guess : "))
r=secant(f,a);
print(" ROOT  : ",r,"\nFUNCTION VALUE : ",f.func(r));
```

Similar to the Python , you do not have to specify code as complex in Octave(matlab) code to calculate complex roots

Program 2.10-8 complex Secant method in Octave(Matlab) secant.m

```
function p=secant(f,x,tolerance,nmax)
% secant method for root finding
if nargin<4 nmax=100;end
if nargin<3 tolerance=1e-10;end
 dx=0.0001;
i=0;
fx=f(x);
dfx=df(f,x);
while (abs(fx) > tolerance) && i<nmax
   fx=f(x);
   dfx=df(f,x);
   x=x-fx/dfx;
   i=i+1;
end
p=x;
if i>=nmax fprintf('Maximum number of iterations is exceeded the result may not be valid\n');end
endfunction
```

```
>> f=@(x)x*x+2;
>> x=0+1j;
>> secant(f,x)
ans =  0.000000000000000 + 1.414213562373095i
```

Program 2.10-9 complex Secant method in Fortran 90

```
program secant2
IMPLICIT NONE
COMPLEX(KIND = 8) a
a=cmplx(1,1,KIND=8)
print *,'secant(',a,') = ',secant(a)
```

```
CONTAINS
  FUNCTION func(x)
  COMPLEX(KIND = 8) func,x
  func=x*x+2.0
  END FUNCTION func

  FUNCTION dfunc(x)
  COMPLEX(KIND = 8) dfunc,x,h,df,f1,f2,f3,f4,f5,f6,f7,f8
  h=dcmplx(1.12314e-4,1.12314e-4)
  f1=3.0*func(x-4.0*h)
  f2=-32.0*func(x-3.0*h)
  f3=168.0*func(x-2.0*h)
  f4=-672.0*func(x-h)
  f5=672.0*func(x+h)
  f6=-168.0*func(x+2.0*h)
  f7=32.0*func(x+3.0*h)
  f8=-3.0*func(x+4.0*h)
  dfunc=(f1+f2+f3+f4+f5+f6+f7+f8)/(840.0*h)
  END FUNCTION dfunc

  FUNCTION secant(a)
  COMPLEX(KIND = 8) secant,a,x,fx,dfx
  REAL(KIND=8) eps
  INTEGER nmax,i
  eps=1.12314e-15
  nmax=100;
  x=a
  DO i=1,nmax
    fx=func(x)
    dfx=dfunc(x)
    x=x-fx/dfx
    IF (abs(fx).LT.eps) EXIT
  END DO
  secant=x
  END FUNCTION secant
END PROGRAM secant2
```

```
---------- Capture Output ----------
> "E:\okul\SCO1\run.bat" secant2

E:\okul\SCO1>secant2
 secant( (  1.0000000000000000     ,  1.0000000000000000     ) ) = (  3.2208930371605715E-033,  1.4142135623730951     )

> Terminated with exit code 0.
```

2.11 HIGHER DERIVATIVE ORDER METHODS

Consider the Taylor Formula, but this time assume that two term is remaining

$$f(x_{n+1}) = 0 = f(x_n) + \frac{f'(x_n)}{1!}(x_{n+1} - x_n) + \frac{f"(x_n)}{2!}(x_{n+1} - x_n)^2 + \frac{f^{(3)}(x_n)}{3!}(x_{n+1} - x_n)^3 + \cdots \qquad (2.11.1)$$

When second term of the Taylor Formula is also taken into acount the Newton-Raphson Formula takes the form:

$$x_{n+1} = x_n - \frac{2f(x_n)f'(x_n)}{2f'(x_n)f'(x_n) - f(x_n)f"(x_n)} \qquad (2.11.2)$$

This method is called **Halley's method**.



Figure 2.11.2 Edmond Halley, British Mathematician and Astronomer

The iterative methods with higher-order convergence are presented in some literature . **Olver's method** is also cubically convergent second order method. It is faster than the Newton-Raphson method.

$$x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)} - \frac{1}{2}\frac{f(x_n)f(x_n)f"(x_n)}{f'(x_n)f'(x_n)f'(x_n)} \quad (2.11.3)$$

In [48], Abbasbandy gives a third order iterative method, called **Abbasbandy's method**(AM) provisionally, which is expressed as



Figure 2.11.2 Daeid Abbasbandy, Iranian Mathematician

$$x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)} - \frac{1}{2}\frac{f(x_n)f(x_n)f"(x_n)}{f'(x_n)f'(x_n)f'(x_n)} - \frac{1}{6}\frac{f(x_n)f(x_n)f(x_n)f'''(x_n)}{f'(x_n)f'(x_n)f'(x_n)f'(x_n)} \quad (2.11.4)$$

Li and Wang[49] suggested two fourth order methods and called them **Thiele Method 1** (TM1) and **Thiele Method 2** (TM2) due to fact that equations derived using Thiele's continued fraction method. Equations has a general form of

$$x_{n+1} = x_n - \frac{P_1(x_n)}{Q_1(x_n)} \quad (2.11.5)$$

$$P_1(x) = 4f(x)f'(x)[3f'(x)f'(x)f"(x)\text{-}3f(x)f"(x)f"(x) + f(x)f'(x)f'''(x)] \quad (2.11.6)$$

$$Q_1(x) = 12f'(x)f'(x)f'(x)f'(x)f(x)\text{-}18f(x)f'(x)f'(x)f"(x)f"(x) + 3f(x)f(x)\,f"(x)f"(x)f"(x) + 4f(x)f'(x)f'(x)f'(x)f'''(x) \quad (2.11.7)$$

$$x_{n+1} = x_n - \frac{P_2(x_n)}{Q_2(x_n)} \quad (2.11.8)$$

$$P_2(x) = f(x)[6f'(x)f'(x)\text{-}3f(x)f"(x)f"(x) + 2f(x)f'(x)f'''(x)] \quad (2.11.9)$$

$$Q_2(x) = 2f'(x)\left[3f'(x)f'(x)f"(x) - 3f(x)f"(x)f"(x) + f(x)f'(x)f'''(x)\right] \quad (2.11.10)$$

Program 2.11-1 Higher order methods, Newton, Halley, Abbasbandy Thiele1 and Thiele2

```
 /  High order methods Halley, Abbasbandy hiele1 and Thiele2
import java.util.*;
import java.awt.*;
import java.applet.Applet;
import java.awt.event.*;
import javax.swing.*;
//============= function to be defined ================

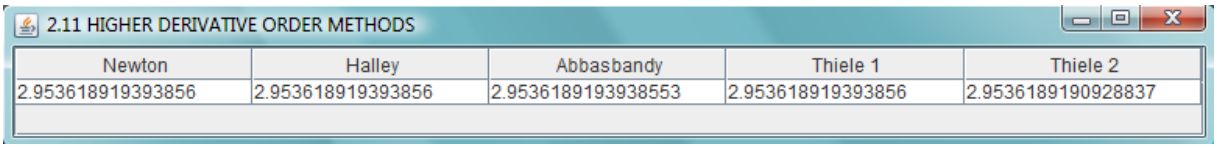

class fb extends f_x
```

```
{
  public double func (double x)
  { return x*x*x+3.6*x-36.4;}
}
//============================================================
public class NA5B
{
public static double newton(f_x f,double x)
{
int nmax=500;
double tolerance=1.0e-8;
double fx,dfx;
for(int i=0;i<nmax;i++)
{  fx=f.func(x);
  dfx=f.dfunc(x);
  x-=fx/dfx;
 if(Math.abs(fx)<tolerance) { return x;}
}
JOptionPane.showMessageDialog(null,"Maximum number of iteration is exceeded in Newton Method\n"+
  " results may not be valid","MAXIMUM ITERATION WARNING",JOptionPane.WARNING_MESSAGE);
return x;
}
public static double halley(f_x f,double x)
{
int nmax=500;
double tolerance=1.0e-8;
double fx,dfx,fxm,dfxm,d2fx;
for(int i=0;i<nmax;i++)
{  fx=f.func(x);
  dfx=f.dfunc(x);
  d2fx=f.dfunc(x,2);
  x-=2.0*fx*dfx/(2.0*dfx*dfx-fx*d2fx);
 if(Math.abs(fx)<tolerance) { return x;}
}
JOptionPane.showMessageDialog(null,"Maximum number of iteration is exceeded in Halley method\n"+
  " results may not be valid","MAXIMUM ITERATION WARNING",JOptionPane.WARNING_MESSAGE);
return x;
}
public static double abbasbandy(f_x f,double x)
{
int nmax=500;
double tolerance=1.0e-8;
double fx,dfx,d2fx,d3fx;
for(int i=0;i<nmax;i++)
{  fx=f.func(x);
  dfx=f.dfunc(x);
  d2fx=f.dfunc(x,2);
  d3fx=f.dfunc(x,3);
  x-=(fx/dfx+fx*fx*d2fx/(2.0*dfx*dfx*dfx)+fx*fx*fx*d3fx/(6.0*dfx*dfx*dfx*dfx));
 if(Math.abs(fx)<tolerance) { return x;}
}
 JOptionPane.showMessageDialog(null,"Maximum number of iteration is exceeded in Abbasbandy\n"+
  " results may not be valid","MAXIMUM ITERATION WARNING",JOptionPane.WARNING_MESSAGE);
  return x;
}

public static double TM1(f_x f,double x)
```

```
{ //Thiele 1 method
int nmax=500;
double tolerance=1.0e-8;
double fx,dfx,d2fx,d3fx;
double P1,Q1;
for(int i=0;i<nmax;i++)
{  fx=f.func(x);
   dfx=f.dfunc(x);
   d2fx=f.dfunc(x,2);
   d3fx=f.dfunc(x,3);
   P1=4.0*fx*dfx*(3.0*dfx*dfx*d2fx-3.0*fx*d2fx*d2fx+fx*dfx*d3fx);
   Q1=12.0*dfx*dfx*dfx*dfx*d2fx-18.0*fx*dfx*dfx*d2fx*d2fx+3.0*fx*fx*d2fx*d2fx*d2fx+4.0*fx*dfx*dfx*dfx*d3fx;
   x-=P1/Q1;
  if(Math.abs(fx)<tolerance) { return x;}
}
JOptionPane.showMessageDialog(null,"Maximum number of iteration is exceeded in Thiele 1\n"+
   " results may not be valid","MAXIMUM ITERATION WARNING",JOptionPane.WARNING_MESSAGE);
return x;
}

public static double TM2(f_x f,double x)
{ //Thiele 2 method
int nmax=500;
double tolerance=1.0e-8;
double fx,dfx,d2fx,d3fx;
double P2,Q2;
for(int i=0;i<nmax;i++)
{  fx=f.func(x);
   dfx=f.dfunc(x);
   d2fx=f.dfunc(x,2);
   d3fx=f.dfunc(x,3);
   P2=fx*(6.0*dfx*dfx*d2fx-3.0*fx*d2fx*d2fx+2*fx*dfx*d3fx);
   Q2=2.0*dfx*(3.0*dfx*dfx*d2fx-3.0*fx*d2fx*d2fx+fx*dfx*d3fx);
   x-=P2/Q2;
  if(Math.abs(fx)<tolerance) { return x;}
}
JOptionPane.showMessageDialog(null,"Maximum number of iteration is exceeded in Thiele 2 method\n"+
   " results may not be valid","MAXIMUM ITERATION WARNING",JOptionPane.WARNING_MESSAGE);
return x;
}
    public static void main (String args[])
    {       fb f=new fb();
            dfb df=new dfb();
    double x0;
    x0=Double.parseDouble(JOptionPane.showInputDialog(" first guess for x : "));
    double r[]=new double[5];
    String s[]={"Newton","Halley","Abbasbandy","Thiele 1","Thiele 2"};
    r[0]=newton(f,x0);
    r[1]= halley(f,x0);
    r[2]= abbasbandy(f,x0);
    r[3]= TM1(f,x0);
    r[4]= TM2(f,x0);
    Text.print(r,s,"2.11 HIGHER DERIVATIVE ORDER METHODS");
    }
}
```



We will write the same program in C++ now

Program 2.11-2 Higher order methods, Newton, Halley, Abbasbandy Thiele1 and Thiele2 C++

```
 //  High order methods Halley, Abbasbandy Thiele1 and Thiele2
#include <stdio.h>
#include <iostream>
#include <math.h>
#include <iomanip>
using namespace std;

class f_x{
public: virtual double func(double x)=0;
public: double dfunc(double x)
{ double h=0.0001;
  double hh=1.0/h;
  double df=(3.0*func(x-4.0*h)-32.0*func(x-3.0*h)+168.0*func(x-2.0*h)-672.0*func(x-h)+672.0*func(x+h)-168.0*func(x+2.0*h)
+32.0*func(x+3.0*h) - 3.0*func(x+4.0*h))/840.0*hh;
  return df;
}
public: double dfunc(double x,int n,double dx)
{       double df=0;
       if(n==0) df=func(x);
       else if( n==1) df=(3.0*func(x-4.0*dx)-32.0*func(x-3.0*dx)+168.0*func(x-2.0*dx)-672.0*func(x-dx)+672.0*func(x+dx)-
168.0*func(x+2.0*dx)+32.0*func(x+3.0*dx)-3.0*func(x+4.0*dx))/840.0/dx;
       else if( n==2) df=(-14350.0*func(x)-9.0*func(x-4.0*dx)+128*func(x-3.0*dx)-1008*func(x-2.0*dx)+8064*func(x-
dx)+8064.0*func(x+dx)-1008.0*func(x+2.0*dx)+128.0*func(x+3.0*dx)-9.0*func(x+4.0*dx))/5040.0/dx/dx;
       else if( n==3) df=(-7.0*func(x-4.0*dx)+72.0*func(x-3.0*dx)-338.0*func(x-2.0*dx)+488.0*func(x-dx)-
488.0*func(x+dx)+338.0*func(x+2.0*dx)-72.0*func(x+3.0*dx)+7.0*func(x+4.0*dx))/240.0/dx/dx/dx;
       else if( n==4) df=(2730.0*func(x)+7.0*func(x-4.0*dx)-96.0*func(x-3.0*dx)+676.0*func(x-2*dx)-1952.0*func(x-dx)-
1952.0*func(x+dx)+676.0*func(x+2.0*dx)-96.0*func(x+3.0*dx)+7.0*func(x+4.0*dx))/240.0/dx/dx/dx/dx;
       else if( n==5) df=(func(x-4.0*dx)-9.0*func(x-3.0*dx)+26.0*func(x-2.0*dx)-29.0*func(x-dx)+29.0*func(x+dx)-
26.0*func(x+2.0*dx)+9.0*func(x+3.0*dx)-func(x+4.0*dx))/6.0/dx/dx/dx/dx/dx;
       else if( n==6) df=(-150.0*func(x)-func(x-4.0*dx)+12.0*func(x-3.0*dx)-52.0*func(x-2.0*dx)+116.0*func(x-dx)+116.0*func(x+dx)-
52.0*func(x+2.0*dx)+12.0*func(x+3.0*dx)-func(x+4.0*dx))/4.0/dx/dx/dx/dx/dx/dx;
       else if( n==7) df=(-func(x-4.0*dx)+6.0*func(x-3.0*dx)-14.0*func(x-2.0*dx)+14.0*func(x-dx)-14.0*func(x+dx)+14.0*func(x+2.0*dx)-
6.0*func(x+3.0*dx)+func(x+4.0*dx))/2.0/dx/dx/dx/dx/dx/dx/dx;
       else if( n==8) df=(70.0*func(x)+func(x-4.0*dx)-8.0*func(x-3.0*dx)+28.0*func(x-2.0*dx)-56.0*func(x-dx)-
56.0*func(x+dx)+28.0*func(x+2.0*dx)-8.0*func(x+3.0*dx)+func(x+4.0*dx))/dx/dx/dx/dx/dx/dx/dx/dx;
       else df=0;
       return df;
}
public: double dfunc(double x,int n)
{      double dx=0.0001;
      return dfunc(x,n,dx);
}
};

class fb:public f_x
{ public: double func(double x)
  { double ff=x*x*x+3.6*x-36.4;
              return ff;
  }
};

double newton(f_x &f,double x)
{
int nmax=500;
double tolerance=1.0e-8;
double fx,dfx;
for(int i=0;i<nmax;i++)
```

```
{  fx=f.func(x);
   dfx=f.dfunc(x);
   x-=fx/dfx;
  if(fabs(fx)<tolerance) { return x;}
}
cout<<"Maximum number of iteration is exceeded in Newton Method\n";
return x;
}
double halley(f_x &f,double x)
{
int nmax=500;
double tolerance=1.0e-8;
double fx,dfx,fxm,dfxm,d2fx;
for(int i=0;i<nmax;i++)
{  fx=f.func(x);
   dfx=f.dfunc(x);
   d2fx=f.dfunc(x,2);
   x-=2.0*fx*dfx/(2.0*dfx*dfx-fx*d2fx);
  if(fabs(fx)<tolerance) { return x;}
}
cout<<"Maximum number of iteration is exceeded in Newton Method\n";return x;
}
double abbasbandy(f_x &f,double x)
{
int nmax=500;
double tolerance=1.0e-8;
double fx,dfx,d2fx,d3fx;
for(int i=0;i<nmax;i++)
{  fx=f.func(x);
   dfx=f.dfunc(x);
   d2fx=f.dfunc(x,2);
   d3fx=f.dfunc(x,3);
   x-=(fx/dfx+fx*fx*d2fx/(2.0*dfx*dfx*dfx)+fx*fx*fx*d3fx/(6.0*dfx*dfx*dfx*dfx));
  if(fabs(fx)<tolerance) { return x;}
}
cout<<"Maximum number of iteration is exceeded in Newton Method\n";
return x;
}

double TM1(f_x &f,double x)
{ //Thiele 1 method
int nmax=500;
double tolerance=1.0e-8;
double fx,dfx,d2fx,d3fx;
double P1,Q1;
for(int i=0;i<nmax;i++)
{  fx=f.func(x);
   dfx=f.dfunc(x);
   d2fx=f.dfunc(x,2);
   d3fx=f.dfunc(x,3);
   P1=4.0*fx*dfx*(3.0*dfx*dfx*d2fx-3.0*fx*d2fx*d2fx+fx*dfx*d3fx);
   Q1=12.0*dfx*dfx*dfx*dfx*d2fx-18.0*fx*dfx*dfx*d2fx*d2fx+3.0*fx*fx*d2fx*d2fx*d2fx+4.0*fx*dfx*dfx*dfx*d3fx;
   x-=P1/Q1;
  if(fabs(fx)<tolerance) { return x;}
}
cout<<"Maximum number of iteration is exceeded in Newton Method\n";
return x;
}

double TM2(f_x &f,double x)
{ //Thiele 2 method
int nmax=500;
double tolerance=1.0e-8;
double fx,dfx,d2fx,d3fx;
```

```
double P2,Q2;
for(int i=0;i<nmax;i++)
{  fx=f.func(x);
   dfx=f.dfunc(x);
   d2fx=f.dfunc(x,2);
   d3fx=f.dfunc(x,3);
   P2=fx*(6.0*dfx*dfx*d2fx-3.0*fx*d2fx*d2fx+2*fx*dfx*d3fx);
   Q2=2.0*dfx*(3.0*dfx*dfx*d2fx-3.0*fx*d2fx*d2fx+fx*dfx*d3fx);
   x-=P2/Q2;
  if(fabs(fx)<tolerance) { return x;}
}
cout<<"Maximum number of iteration is exceeded in Newton Method\n";
return x;
}

    int main ()
    {
              fb f;
              double x0;
              cout<<setprecision(20);
              cout << " first guess for x ";
    cin >>x0;
    double r[5];
    string s[]={"Newton    ","Halley    ","Abbasbandy","Thiele 1  ","Thiele 2  "};
    r[0]=newton(f,x0);
    r[1]= halley(f,x0);
    r[2]= abbasbandy(f,x0);
    r[3]= TM1(f,x0);
    r[4]= TM2(f,x0);
    cout<<"HIGHER DERIVATIVE ORDER METHODS\n";
    for(int i=0;i<5;i++)
      {cout<<s[i]<<" = "<<r[i]<<endl;}}
```

```
 ---------- Capture Output ----------
> "D:\okul\SCO1\run.bat" NA5B

D:\okul\SCO1>NA5B
 first guess for x 1
HIGHER DERIVATIVE ORDER METHODS
Newton     = 2.9536189193938553
Halley     = 2.9536189193938553
Abbasbandy = 2.9536189193938553
Thiele 1   = 2.9536189193938553
Thiele 2   = 2.9536189193938553

> Terminated with exit code 0.
```

Program 2.11-3 Higher order methods, Halley, Abbasbandy in Fortran 90

```
 program halley1
IMPLICIT NONE
REAL(KIND = SELECTED_REAL_KIND(10,34)) a
a=1.0
print *,'halley(',a,') = ',halley(a)
a=1.0
print *,'olver(',a,') = ',olver(a)
a=1.0
print *,'abbasbandy(',a,') = ',abbasbandy(a)
a=1.0
print *,'TM1(',a,') = ',TM1(a)
a=1.0
print *,'TM2(',a,') = ',TM2(a)

CONTAINS
```

```
FUNCTION func(x)
REAL(KIND = SELECTED_REAL_KIND(10,34)) func,x
func=x*x*x+3.6*x-36.4
END FUNCTION func

FUNCTION dfunc(x,n)
REAL(KIND = SELECTED_REAL_KIND(10,34)) dfunc,x,a,h,df,f1,f2,f3,f4,f5,f6,f7,f8,f9
INTEGER n
h=1.12314e-4
if(n==1) then
  f1=3.0*func(x-4.0*h)
  f2=-32.0*func(x-3.0*h)
  f3=168.0*func(x-2.0*h)
  f4=-672.0*func(x-h)
  f5=672.0*func(x+h)
  f6=-168.0*func(x+2.0*h)
  f7=32.0*func(x+3.0*h)
  f8=-3.0*func(x+4.0*h)
  dfunc=(f1+f2+f3+f4+f5+f6+f7+f8)/(840.0*h)
else if(n==2) then
  f1=-14350.0*func(x)
  f2=-9.0*func(x-4.0*h)
  f3=128.0*func(x-3.0*h)
  f4=-1008.0*func(x-2.0*h)
  f5=8064.0*func(x-h)
  f6=8064.0*func(x+h)
  f7=-1008.0*func(x+2.0*h)
  f8=128.0*func(x+3.0*h)
  f9=-9.0*func(x+4.0*h)
  dfunc=(f1+f2+f3+f4+f5+f6+f7+f8+f9)/(5040.0*h*h)
else if(n==3) then
  f1=-7.0*func(x-4.0*h)
  f2=72.0*func(x-3.0*h)
  f3=-338.0*func(x-2.0*h)
  f4=488.0*func(x-h)
  f5=-488.0*func(x+h)
  f6=338.0*func(x+2.0*h)
  f7=-72.0*func(x+3.0*h)
  f8=7.0*func(x+4.0*h)
  dfunc=(f1+f2+f3+f4+f5+f6+f7+f8)/(240.0*h*h*h);
end if
END FUNCTION dfunc

FUNCTION olver(x)
REAL(KIND = SELECTED_REAL_KIND(10,34)) olver,x,eps,fx,dfx,d2fx
INTEGER nmax,i
eps=1.0e-8;
nmax=100;
DO i=1,nmax
  fx=func(x)
  dfx=dfunc(x,1)
  d2fx=dfunc(x,2)
  x=x-fx/dfx-1.0/2.0*fx*fx*d2fx/(dfx*dfx*dfx)
  IF (abs(fx)<eps) EXIT
END DO
olver=x
END FUNCTION olver
  FUNCTION halley(x)
REAL(KIND = SELECTED_REAL_KIND(10,34)) halley,x,eps,fx,dfx,d2fx
INTEGER nmax,i
eps=1.0e-8;
nmax=100;
DO i=1,nmax
  fx=func(x)
```

```
    dfx=dfunc(x,1)
    d2fx=dfunc(x,2)
    x=x-2.0*fx*dfx/(2.0*dfx*dfx-fx*d2fx)
    IF (abs(fx)<eps) EXIT
  END DO
  halley=x
  END FUNCTION halley
  FUNCTION abbasbandy(x)
  REAL(KIND = SELECTED_REAL_KIND(10,34)) abbasbandy,x,eps,fx,dfx,d2fx,d3fx
  INTEGER nmax,i
  eps=1.0e-8;
  nmax=100;
  x=a
  DO i=1,nmax
    fx=func(x)
    dfx=dfunc(x,1)
    d2fx=dfunc(x,2)
    d3fx=dfunc(x,3)
    x=x-(fx/dfx+fx*fx*d2fx/(2.0*dfx*dfx*dfx)+fx*fx*fx*d3fx/(6.0*dfx*dfx*dfx*dfx));
    IF (abs(fx)<eps) EXIT
  END DO
  abbasbandy=x
  END FUNCTION abbasbandy
  FUNCTION TM1(x)
  REAL(KIND = SELECTED_REAL_KIND(10,34)) TM1,x,eps,fx,dfx,d2fx,d3fx,P1,Q1
  INTEGER nmax,i
  eps=1.0e-8;
  nmax=100;
  x=a
  DO i=1,nmax
    fx=func(x)
    dfx=dfunc(x,1)
    d2fx=dfunc(x,2)
    d3fx=dfunc(x,3)
    P1=4.0*fx*dfx*(3.0*dfx*dfx*d2fx-3.0*fx*d2fx*d2fx+fx*dfx*d3fx)
    Q1=12.0*dfx*dfx*dfx*dfx*d2fx-18.0*fx*dfx*dfx*d2fx*d2fx+3.0*fx*fx*d2fx*d2fx*d2fx+4.0*fx*dfx*dfx*dfx*d3fx
    x=x-P1/Q1;
    IF (abs(fx)<eps) EXIT
  END DO
  TM1=x
  END FUNCTION TM1

  FUNCTION TM2(x)
  REAL(KIND = SELECTED_REAL_KIND(10,34)) TM2,x,eps,fx,dfx,d2fx,d3fx,P2,Q2
  INTEGER nmax,i
  eps=1.0e-8;
  nmax=100;
  x=a
  DO i=1,nmax
    fx=func(x)
    dfx=dfunc(x,1)
    d2fx=dfunc(x,2)
    d3fx=dfunc(x,3)
    P2=fx*(6.0*dfx*dfx*d2fx-3.0*fx*d2fx*d2fx+2*fx*dfx*d3fx)
    Q2=2.0*dfx*(3.0*dfx*dfx*d2fx-3.0*fx*d2fx*d2fx+fx*dfx*d3fx)
    x=x-P2/Q2;
    IF (abs(fx)<eps) EXIT
  END DO
  TM2=x
  END FUNCTION TM2
END PROGRAM halley1
```

```
 ---------- Capture Output ----------
> "E:\okul\SCO1\run.bat" halley1
```

```
E:\okul\SCO1>halley1
 halley(   1.0000000000000000      ) =    2.9536189801080357
 olver(   1.0000000000000000      ) =    2.9536189801080357
 abbasbandy(   1.0000000000000000      ) =    2.9536189801080357
 TM1(   1.0000000000000000      ) =    2.9536189801080357
 TM2(   1.0000000000000000      ) =    2.9536189801080357
> Terminated with exit code 0.
```

In this stage, it will be interesting to compare iteration efficiencies of the higher order methods. In order to achive that a version of the program with 3 output values namely root x, function value at the root f(x) and number of iteration is developed.

Program 2.11-4 Higher order methods, Newton, Halley, Abbasbandy Thiele1 and Thiele2 with 3 output values root x, function f(x) and number of iteration n

```
 //  High order methods Halley, Abbasbandy hiele1 and Thiele2
import java.util.*;
import java.awt.*;
import java.applet.Applet;
import java.awt.event.*;
import javax.swing.*;
//============= function to be defined ================
class fb extends f_x
{  public double func (double x)
   { return x*x*x+3.6*x-36.4;}
}
public class NA5B1
{public static double[] newton(f_x f,double x)
{
double a[]=new double[3];
int nmax=500;
double tolerance=1.0e-10;
double fx=0,dfx;
int i;
for(i=0;i<nmax;i++)
{  fx=f.func(x);
   dfx=f.dfunc(x);
   x-=fx/dfx;
  if(Math.abs(fx)<tolerance) { a[0]=x;a[1]=fx;a[2]=i;return a;}
}
JOptionPane.showMessageDialog(null,"Maximum number of iteration is exceeded in Newton Method\n"+
   " results may not be valid","MAXIMUM ITERATION WARNING",JOptionPane.WARNING_MESSAGE);
a[0]=x;a[1]=fx;a[2]=i;return a;
}

public static double[] halley(f_x f,double x)
{
              double a[]=new double[3];
int nmax=500;
double tolerance=1.0e-10;
double fx=0,dfx,fxm,dfxm,d2fx;
int i;
for(i=0;i<nmax;i++)
{  fx=f.func(x);
   dfx=f.dfunc(x);
   d2fx=f.dfunc(x,2);
   x-=2.0*fx*dfx/(2.0*dfx*dfx-fx*d2fx);
  if(Math.abs(fx)<tolerance) { a[0]=x;a[1]=fx;a[2]=i;return a;}
}
JOptionPane.showMessageDialog(null,"Maximum number of iteration is exceeded in Halley method\n"+
   " results may not be valid","MAXIMUM ITERATION WARNING",JOptionPane.WARNING_MESSAGE);
a[0]=x;a[1]=fx;a[2]=i;return a;
```

```
}

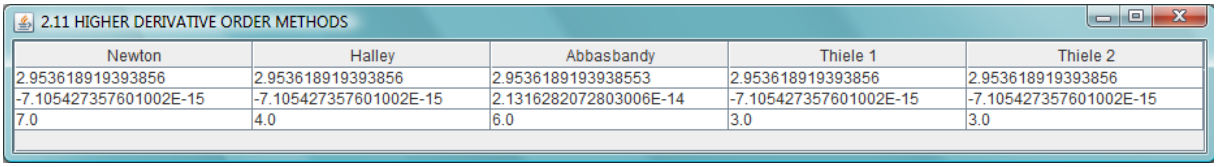
public static double[] abbasbandy(f_x f,double x)
{
            double a[]=new double[3];
int nmax=500;
double tolerance=1.0e-10;
double fx=0,dfx,d2fx,d3fx;
int i;
for(i=0;i<nmax;i++)
{  fx=f.func(x);
   dfx=f.dfunc(x);
   d2fx=f.dfunc(x,2);
   d3fx=f.dfunc(x,3);
   x-=(fx/dfx+fx*fx*d2fx/(2.0*dfx*dfx*dfx)+fx*fx*fx*d3fx/(6.0*dfx*dfx*dfx*dfx));
  if(Math.abs(fx)<tolerance) { a[0]=x;a[1]=fx;a[2]=i;return a;}
}
  JOptionPane.showMessageDialog(null,"Maximum number of iteration is exceeded in Abbasbandy\n"+
    " results may not be valid","MAXIMUM ITERATION WARNING",JOptionPane.WARNING_MESSAGE);
  a[0]=x;a[1]=fx;a[2]=i;return a;
}

public static double[] TM1(f_x f,double x)
{ //Thiele 1 method
double a[]=new double[3];
int nmax=500;
double tolerance=1.0e-10;
double fx=0,dfx,d2fx,d3fx;
double P1,Q1;
int i;
for(i=0;i<nmax;i++)
{  fx=f.func(x);
   dfx=f.dfunc(x);
   d2fx=f.dfunc(x,2);
   d3fx=f.dfunc(x,3);
   P1=4.0*fx*dfx*(3.0*dfx*dfx*d2fx-3.0*fx*d2fx*d2fx+fx*dfx*d3fx);
   Q1=12.0*dfx*dfx*dfx*dfx*d2fx-18.0*fx*dfx*dfx*d2fx*d2fx+3.0*fx*fx*d2fx*d2fx*d2fx+4.0*fx*dfx*dfx*dfx*d3fx;
   x-=P1/Q1;
  if(Math.abs(fx)<tolerance) { a[0]=x;a[1]=fx;a[2]=i;return a;}
}
JOptionPane.showMessageDialog(null,"Maximum number of iteration is exceeded in Thiele 1\n"+
   " results may not be valid","MAXIMUM ITERATION WARNING",JOptionPane.WARNING_MESSAGE);
{ a[0]=x;a[1]=fx;a[2]=i;return a;}
}

public static double[] TM2(f_x f,double x)
{ //Thiele 2 method
double a[]=new double[3];
int nmax=500;
double tolerance=1.0e-10;
double fx=0,dfx,d2fx,d3fx;
double P2,Q2;
int i;
for(i=0;i<nmax;i++)
{  fx=f.func(x);
   dfx=f.dfunc(x);
   d2fx=f.dfunc(x,2);
   d3fx=f.dfunc(x,3);
   P2=fx*(6.0*dfx*dfx*d2fx-3.0*fx*d2fx*d2fx+2*fx*dfx*d3fx);
   Q2=2.0*dfx*(3.0*dfx*dfx*d2fx-3.0*fx*d2fx*d2fx+fx*dfx*d3fx);
   x-=P2/Q2;
  if(Math.abs(fx)<tolerance) { a[0]=x;a[1]=fx;a[2]=i;return a;}
}
JOptionPane.showMessageDialog(null,"Maximum number of iteration is exceeded in Thiele 2 method\n"+
```

```
   " results may not be valid","MAXIMUM ITERATION WARNING",JOptionPane.WARNING_MESSAGE);
 a[0]=x;a[1]=fx;a[2]=i;return a;
}

    public static void main (String args[])
    {
                fb f=new fb();
                dfb df=new dfb();
    double x0;
    x0=Double.parseDouble(JOptionPane.showInputDialog(" first guess for x : "));
    double r[][]=new double[5][3];
    String s[]={"Newton","Halley","Abbasbandy","Thiele 1","Thiele 2"};
    r[0]=newton(f,x0);
    r[1]= halley(f,x0);
    r[2]= abbasbandy(f,x0);
    r[3]= TM1(f,x0);
    r[4]= TM2(f,x0);
    Text.printT(r,s,"2.11 HIGHER DERIVATIVE ORDER METHODS");
    }
}
```



2.11 HIGHER DERIVATIVE ORDER METHODS

Newton	Halley	Abbasbandy	Thiele 1	Thiele 2
2.953618919393856	2.953618919393856	2.9536189193938553	2.953618919393856	2.953618919393856
-7.105427357601002E-15	-7.105427357601002E-15	2.1316282072803006E-14	-7.105427357601002E-15	-7.105427357601002E-15
7.0	4.0	6.0	3.0	3.0

Program 2.11-5 Higher order numerical derivatives in Matlab(Octave) code

```
function df=dnf(f,x,n,dx)
% degree nth derivative of a function f
if nargin<4
   dx=0.001;
end
if nargin<3
   dx=0.001;n=1;
end
dxh=1.0/dx;
if n==0 df=f(x);
elseif n==1
df=(3.0*f(x-4.0*dx)-32.0*f(x-3.0*dx)+168.0*f(x-2.0*dx)-672.0*f(x-dx)+672.0*f(x+dx)-168.0*f(x+2.0*dx)+32.0*f(x+3.0*dx)-
3.0*f(x+4.0*dx))/840.0*dxh;
elseif n==2
df=(-14350.0*f(x)-9.0*f(x-4.0*dx)+128*f(x-3.0*dx)-1008*f(x-2.0*dx)+8064*f(x-dx)+8064.0*f(x+dx)-
1008.0*f(x+2.0*dx)+128.0*f(x+3.0*dx)-9.0*f(x+4.0*dx))/5040.0*dxh*dxh;
elseif n==3
df=(-7.0*f(x-4.0*dx)+72.0*f(x-3.0*dx)-338.0*f(x-2.0*dx)+488.0*f(x-dx)-488.0*f(x+dx)+338.0*f(x+2.0*dx)-
72.0*f(x+3.0*dx)+7.0*f(x+4.0*dx))/240.0*dxh*dxh*dxh;
elseif n==4
df=(2730.0*f(x)+7.0*f(x-4.0*dx)-96.0*f(x-3.0*dx)+676.0*f(x-2*dx)-1952.0*f(x-dx)-1952.0*f(x+dx)+676.0*f(x+2.0*dx)-
96.0*f(x+3.0*dx)+7.0*f(x+4.0*dx))/240.0*dxh*dxh*dxh*dxh;
elseif n==5
df=(f(x-4.0*dx)-9.0*f(x-3.0*dx)+26.0*f(x-2.0*dx)-29.0*f(x-dx)+29.0*f(x+dx)-26.0*f(x+2.0*dx)+9.0*f(x+3.0*dx)-
f(x+4.0*dx))/6.0*dxh*dxh*dxh*dxh*dxh;
elseif n==6
df=(-150.0*f(x)-f(x-4.0*dx)+12.0*f(x-3.0*dx)-52.0*f(x-2.0*dx)+116.0*f(x-dx)+116.0*f(x+dx)-52.0*f(x+2.0*dx)+12.0*f(x+3.0*dx)-
f(x+4.0*dx))/4.0*dxh*dxh*dxh*dxh*dxh*dxh;
elseif n==7
df=(-f(x-4.0*dx)+6.0*f(x-3.0*dx)-14.0*f(x-2.0*dx)+14.0*f(x-dx)-14.0*f(x+dx)+14.0*f(x+2.0*dx)-
6.0*f(x+3.0*dx)+f(x+4.0*dx))/2.0*dxh*dxh*dxh*dxh*dxh*dxh*dxh;
elseif n==8
df=(70.0*f(x)+f(x-4.0*dx)-8.0*f(x-3.0*dx)+28.0*f(x-2.0*dx)-56.0*f(x-dx)-56.0*f(x+dx)+28.0*f(x+2.0*dx)-
8.0*f(x+3.0*dx)+f(x+4.0*dx))*dxh*dxh*dxh*dxh*dxh*dxh*dxh*dxh;

else df=0;end
end
```

Program 2.11-6 Thiele2 method in Matlab(Octave) code

```
function xx = TM2(f,x)
%Thiele 2 Method
nmax=500;
tolerance=1.0e-8;
i=0;
f_x=1;
while   abs(f_x)>=tolerance && i<=nmax
   f_x=f(x);
   df_x=dnf(f,x,1);
   d2f_x=dnf(f,x,2);
   d3f_x=dnf(f,x,3);
   P2=f_x*(6.0*df_x*df_x*d2f_x-3.0*f_x*d2f_x*d2f_x+2*f_x*df_x*d3f_x);
   Q2=2.0*df_x*(3.0*df_x*df_x*d2f_x-3.0*f_x*d2f_x*d2f_x+f_x*df_x*d3f_x);
   x=x-P2/Q2;
   i=i+1;
  end
if(i>=nmax) fprintf('Maximum number of iteration is exceeded in Thiele 2 method\nresults may not be valid');end
xx=x;
end
```

```
>> f1=@(x)x*x-2;
>> TM2(f1,1)
ans =  1.41421356237310
>>
```

Program 2.11-7 Thiele1 method in python code

```
from math import *

def df(f,x):
    h=0.001
    return (-f(x+2.0*h)+8.0*f(x+h)-8.0*f(x-h)+f(x-2.0*h))/(12.0*h);

# second derivative
def d2f(f,x):
    dx=0.001
    return (-1.0/12.0*f(x-2.0*dx)+4.0/3.0*f(x-dx)-5.0/2.0*f(x)+4.0/3.0*f(x+dx)-1.0/12.0*f(x+2.0*dx))/dx

# nth derivative
def dnf(f,x,n):
    dx=0.001
    dxh=1.0/dx
    df=0.0;
    if n==0:  df=f(x);
    elif n==1: df=(3.0*f(x-4.0*dx)-32.0*f(x-3.0*dx)+168.0*f(x-2.0*dx)-672.0*f(x-dx)+672.0*f(x+dx)-
168.0*f(x+2.0*dx)+32.0*f(x+3.0*dx)-3.0*f(x+4.0*dx))/840.0*dxh;
    elif n==2: df=(-14350.0*f(x)-9.0*f(x-4.0*dx)+128*f(x-3.0*dx)-1008*f(x-2.0*dx)+8064*f(x-dx)+8064.0*f(x+dx)-
1008.0*f(x+2.0*dx)+128.0*f(x+3.0*dx)-9.0*f(x+4.0*dx))/5040.0*dxh*dxh;
    elif n==3:
      df=(-7.0*f(x-4.0*dx)+72.0*f(x-3.0*dx)-338.0*f(x-2.0*dx)+488.0*f(x-dx)-488.0*f(x+dx)+338.0*f(x+2.0*dx)-
72.0*f(x+3.0*dx)+7.0*f(x+4.0*dx))/240.0*dxh*dxh*dxh;
    elif n==4:
      df=(2730.0*f(x)+7.0*f(x-4.0*dx)-96.0*f(x-3.0*dx)+676.0*f(x-2*dx)-1952.0*f(x-dx)-1952.0*f(x+dx)+676.0*f(x+2.0*dx)-
96.0*f(x+3.0*dx)+7.0*f(x+4.0*dx))/240.0*dxh*dxh*dxh*dxh;
    elif n==5:
      df=(f(x-4.0*dx)-9.0*f(x-3.0*dx)+26.0*f(x-2.0*dx)-29.0*f(x-dx)+29.0*f(x+dx)-26.0*f(x+2.0*dx)+9.0*f(x+3.0*dx)-
f(x+4.0*dx))/6.0*dxh*dxh*dxh*dxh*dxh;
    return df;

def TM1(f,x):
    #Thiele 1 method
    a=[0.0 for i in range(3)];
    nmax=500;
    tolerance=1.0e-10;
    fx=0;
```

```
    for i in range(0,nmax):
      fx=f(x);
      dfx=df(f,x);
      d2fx=d2f(f,x);
      d3fx=dnf(f,x,3);
      P1=4.0*fx*dfx*(3.0*dfx*dfx*d2fx-3.0*fx*d2fx*d2fx+fx*dfx*d3fx);
      Q1=12.0*dfx*dfx*dfx*dfx*d2fx-18.0*fx*dfx*dfx*d2fx*d2fx+3.0*fx*fx*d2fx*d2fx*d2fx+4.0*fx*dfx*dfx*dfx*d3fx;
      x-=P1/Q1;
      if abs(fx)<tolerance: a[0]=x;a[1]=fx;a[2]=i;return a;
    print("Maximum number of iteration is exceeded in Thiele 1\n");
    a[0]=x;a[1]=fx;a[2]=i;
    return a;

f1 = lambda x: x*x-2
x=float(input("enter estimation : "))
print(x)
r=Mathd.TM1(f1,x);
print(r)
```

```
runfile('D:/okul/SCO1/TM1_2.py', wdir='D:/okul/SCO1')
enter estimation : 1
1.0
[1.4142135623730951, 4.429789868254375e-12, 4]
```

Program 2.11-8 Thiele1 method in python code abstract f_x class based class function func is used

```
from math import *
from f_x import *;

class f1(f_x):
  def func(self,x):
    y=x*x+2;
    return y;

def TM1(f,x):
    #Thiele 1 method
    a=[0.0 for i in range(3)];
    nmax=500;
    tolerance=1.0e-10;
    fx=0;
    for i in range(0,nmax):
      fx=f.func(x);
      dfx=f.dfunc(x,1);
      d2fx=f.dfunc(x,2);
      d3fx=f.dfunc(x,3);
      P1=4.0*fx*dfx*(3.0*dfx*dfx*d2fx-3.0*fx*d2fx*d2fx+fx*dfx*d3fx);
      Q1=12.0*dfx*dfx*dfx*dfx*d2fx-18.0*fx*dfx*dfx*d2fx*d2fx+3.0*fx*fx*d2fx*d2fx*d2fx+4.0*fx*dfx*dfx*dfx*d3fx;
      x-=P1/Q1;
      if abs(fx)<tolerance: a[0]=x;a[1]=fx;a[2]=i;return a;
    print("Maximum number of iteration is exceeded in Thiele 1\n");
    a[0]=x;a[1]=fx;a[2]=i;
    return a;

f =  f1();
x=complex(input("enter estimation example: 1+2j : "));
r=TM1(f,x);
print(r)
```

```
---------- Capture Output ----------
> "D:\co\anaconda\pythonw.exe" TM1_2.py
enter estimation example: 1+2j : 1+1j
[(9.341989763213027e-46+1.414213562373095j), (-4.440892098500626e-16+0j), 3]
```

```
> Terminated with exit code 0.
```

A complex version of Halley's method is given here:

Program 2.11-9 Haley's method complex lambda version (java)

```
// compleks Halley's method to find roots of function
// xn+1=xn-2(f(xn)*f(xn)*f'(xn))/(2f'(xn)*f'(xn)-f(xn)*f''(xn))
import java.util.*;
import java.awt.*;
import java.applet.Applet;
import java.awt.event.*;
import javax.swing.*;

public class Halley1
{
public static c Halley(icf_cx f,c x)
{
double eps=1.0e-15;
int m=100;
int k=0;
c f1,df,d2f;
c b1,b2,b3,b4,b5;
do{
  f1=f.func(x);
  df=f.dfunc(x);
  d2f=f.d2func(x);
  b1=c.p(c.p(f1,"*",df),"*",2.0);
  b2=c.p(c.p(df,"*",df),"*",2.0);
  b3=c.p(f1,"*",d2f);
  b4=c.p(b2,"-",b3);
  b5=c.p(b1,"/",b4);
  x=c.p(x,"-",b5);
  if(c.abs(f1)<eps) return x;
  k++;
}while(c.abs(f1)>eps && k<m);
JOptionPane.showMessageDialog(null,"Warning : Maximum number of iteration is exceeded \n"+
   " result may not be valid","MAXIMUM ITERATION NUMBER WARNING",JOptionPane.WARNING_MESSAGE);
return x;
}

public static c Halley(cf_cx f,c x)
{
double eps=1.0e-15;
int m=100;
int k=0;
c f1,df,d2f;
c b1,b2,b3,b4,b5;
do{
  f1=f.func(x);
  df=f.dfunc(x);
  d2f=f.d2func(x);
  b1=c.p(c.p(f1,"*",df),"*",2.0);
  b2=c.p(c.p(df,"*",df),"*",2.0);
  b3=c.p(f1,"*",d2f);
  b4=c.p(b2,"-",b3);
  b5=c.p(b1,"/",b4);
  x=c.p(x,"-",b5);
  if(c.abs(f1)<eps) return x;
  k++;
}while(c.abs(f1)>eps && k<m);
JOptionPane.showMessageDialog(null,"Warning : Maximum number of iteration is exceeded \n"+
   " result may not be valid","MAXIMUM ITERATION NUMBER WARNING",JOptionPane.WARNING_MESSAGE);
return x;
```

```
}

public static String toString(c x[])
{int n1=x.length;
 String s="";
 for(int i=n1-1;i>=0;i--) {s+="x["+i+"] = "+x[i]+"\n";}
 return s;
}
    public static double[] input(String s)
    {
    String s1=JOptionPane.showInputDialog(s);
    StringTokenizer token=new StringTokenizer(s1);
    int n=token.countTokens()-1;
    int m=n+1;
    double a[]=new double[m];
    int j=0;
    while(token.hasMoreTokens())
    {
    Double ax=new Double(token.nextToken());
    a[j++]=ax.doubleValue();
    }
    return a;
    }

    public static void main (String args[])
    { icf_cx f=(c x)->c.p(c.p(x,"*",x),"+",2.0);
      c x1=new c(1.0,1.0);
      c y=Halley(f,x1);
      System.out.println("root = "+y);
    }
}
```

```
---------- Capture Output ----------
> "C:\java\bin\java.exe" Halley1
root = (   0.0000000000 +    1.4142135624i )

> Terminated with exit code 0.
```

Program 2.11-10 Halley's method complex C++ code

```
#include <iostream>
#include <math.h>
#include <complex>
#include <iomanip>
using namespace std;
typedef complex<double> cdouble;

class cf_x{
public: virtual cdouble func(cdouble x)=0;
public: cdouble dfunc(cdouble x)
{ cdouble h=0.0001;
  cdouble hh=1.0/h;
  cdouble df=(3.0*func(x-4.0*h)-32.0*func(x-3.0*h)+168.0*func(x-2.0*h)-672.0*func(x-h)+672.0*func(x+h)-168.0*func(x+2.0*h)
+32.0*func(x+3.0*h) - 3.0*func(x+4.0*h))/840.0*hh;
  return df;
}
public: cdouble dfunc(cdouble x,int n,cdouble dx)
{       cdouble df=0;
        if(n==0) df=func(x);
        else if( n==1) df=(3.0*func(x-4.0*dx)-32.0*func(x-3.0*dx)+168.0*func(x-2.0*dx)-672.0*func(x-dx)+672.0*func(x+dx)-
168.0*func(x+2.0*dx)+32.0*func(x+3.0*dx)-3.0*func(x+4.0*dx))/840.0/dx;
        else if( n==2) df=(-14350.0*func(x)-9.0*func(x-4.0*dx)+128.0*func(x-3.0*dx)-1008.0*func(x-2.0*dx)+8064.0*func(x-
dx)+8064.0*func(x+dx)-1008.0*func(x+2.0*dx)+128.0*func(x+3.0*dx)-9.0*func(x+4.0*dx))/5040.0/dx/dx;
        else if( n==3) df=(-7.0*func(x-4.0*dx)+72.0*func(x-3.0*dx)-338.0*func(x-2.0*dx)+488.0*func(x-dx)-
```

```
488.0*func(x+dx)+338.0*func(x+2.0*dx)-72.0*func(x+3.0*dx)+7.0*func(x+4.0*dx))/240.0/dx/dx/dx;
      else if( n==4) df=(2730.0*func(x)+7.0*func(x-4.0*dx)-96.0*func(x-3.0*dx)+676.0*func(x-2.0*dx)-1952.0*func(x-dx)-
1952.0*func(x+dx)+676.0*func(x+2.0*dx)-96.0*func(x+3.0*dx)+7.0*func(x+4.0*dx))/240.0/dx/dx/dx/dx;
      else if( n==5) df=(func(x-4.0*dx)-9.0*func(x-3.0*dx)+26.0*func(x-2.0*dx)-29.0*func(x-dx)+29.0*func(x+dx)-
26.0*func(x+2.0*dx)+9.0*func(x+3.0*dx)-func(x+4.0*dx))/6.0/dx/dx/dx/dx/dx;
      else if( n==6) df=(-150.0*func(x)-func(x-4.0*dx)+12.0*func(x-3.0*dx)-52.0*func(x-2.0*dx)+116.0*func(x-dx)+116.0*func(x+dx)-
52.0*func(x+2.0*dx)+12.0*func(x+3.0*dx)-func(x+4.0*dx))/4.0/dx/dx/dx/dx/dx/dx;
      else if( n==7) df=(-func(x-4.0*dx)+6.0*func(x-3.0*dx)-14.0*func(x-2.0*dx)+14.0*func(x-dx)-14.0*func(x+dx)+14.0*func(x+2.0*dx)-
6.0*func(x+3.0*dx)+func(x+4.0*dx))/2.0/dx/dx/dx/dx/dx/dx/dx;
      else if( n==8) df=(70.0*func(x)+func(x-4.0*dx)-8.0*func(x-3.0*dx)+28.0*func(x-2.0*dx)-56.0*func(x-dx)-
56.0*func(x+dx)+28.0*func(x+2.0*dx)-8.0*func(x+3.0*dx)+func(x+4.0*dx))/dx/dx/dx/dx/dx/dx/dx/dx;
      else df=0;
      return df;
}
public: cdouble dfunc(cdouble x,int n)
{     cdouble dx=0.0001;
     return dfunc(x,n,dx);
}};

class f1:public cf_x
{ public: cdouble func(cdouble x)
  { cdouble ff=x*x+2.0;return ff;}};
cdouble halley(cf_x &f,cdouble x)
{
int nmax=500;
double tolerance=1.0e-8;
cdouble fx,dfx,fxm,dfxm,d2fx;
for(int i=0;i<nmax;i++)
{  fx=f.func(x);
   dfx=f.dfunc(x);
   d2fx=f.dfunc(x,2);
   x-=2.0*fx*dfx/(2.0*dfx*dfx-fx*d2fx);
  if(fabs(fx)<tolerance) { return x;}
}
cout<<"Maximum number of iteration is exceeded in halley Method\n";return x;
}
cdouble newton(cf_x &f,cdouble x)
{
int nmax=100;
double tolerance=1.0e-8;
cdouble fx,dfx;
for(int i=0;i<nmax;i++)
{ fx=f.func(x);
  dfx=f.dfunc(x);
  x-=fx/dfx;
  cout <<"i="<<i<<"x="<<x<<"fx="<<fx<<"dfx="<<dfx<<"\n";
  if(abs(fx)<tolerance) { return x;}
}
cout <<("Maximum number of iteration is exceeded \n result might not be valid");
return x;
}

int main()
{  f1 f;
              cdouble xa,r;
   cout << " first guess for the root value : ";
   cin >>xa;
   cout<<setprecision(20);
   r=halley(f,xa);
   cout<<"root value : ="<<r;
   return 0;
}
```

```
---------- Capture Output ----------
```

```
> "D:\okul\SCO1\run.bat" challey
D:\okul\SCO1>challey
 first guess for the root value : (1,1)
root value : =(-2.5849394142282115e-026,1.4142135623730949)
> Terminated with exit code 0.
```

Program 2.11-11 Halley's and Abasbandy methods complex Fortran 90 code

```
program halley2
IMPLICIT NONE
COMPLEX(KIND = 8) a
a=cmplx(1,1,KIND=8)
print *,'halley(',a,') = ',halley(a)
a=cmplx(1,1,KIND=8)
print *,'abbasbandy(',a,') = ',abbasbandy(a)

CONTAINS
 FUNCTION func(x)
 COMPLEX(KIND = 8) func,x
 func=x*x+2
 END FUNCTION func

 FUNCTION dfunc(x,n)
 COMPLEX(KIND = 8) dfunc,x,a,h,df,f1,f2,f3,f4,f5,f6,f7,f8,f9
 INTEGER n
 h=cmplx(1.12314e-4,1.12314e-4,KIND=8)
 if(n==1) then
  f1=3.0*func(x-4.0*h)
  f2=-32.0*func(x-3.0*h)
  f3=168.0*func(x-2.0*h)
  f4=-672.0*func(x-h)
  f5=672.0*func(x+h)
  f6=-168.0*func(x+2.0*h)
  f7=32.0*func(x+3.0*h)
  f8=-3.0*func(x+4.0*h)
  dfunc=(f1+f2+f3+f4+f5+f6+f7+f8)/(840.0*h)
 else if(n==2) then
  f1=-14350.0*func(x)
  f2=-9.0*func(x-4.0*h)
  f3=128.0*func(x-3.0*h)
  f4=-1008.0*func(x-2.0*h)
  f5=8064.0*func(x-h)
  f6=8064.0*func(x+h)
  f7=-1008.0*func(x+2.0*h)
  f8=128.0*func(x+3.0*h)
  f9=-9.0*func(x+4.0*h)
  dfunc=(f1+f2+f3+f4+f5+f6+f7+f8+f9)/(5040.0*h*h)
 else if(n==3) then
  f1=-7.0*func(x-4.0*h)
  f2=72.0*func(x-3.0*h)
  f3=-338.0*func(x-2.0*h)
  f4=488.0*func(x-h)
  f5=-488.0*func(x+h)
  f6=338.0*func(x+2.0*h)
  f7=-72.0*func(x+3.0*h)
  f8=7.0*func(x+4.0*h)
  dfunc=(f1+f2+f3+f4+f5+f6+f7+f8)/(240.0*h*h*h);
 end if
 END FUNCTION dfunc

 FUNCTION halley(x)
 COMPLEX(KIND = 8) halley,x,fx,dfx,d2fx
 REAL(KIND=8) eps
 INTEGER nmax,i
```

```
  eps=1.0e-8;
  nmax=100;
  DO i=1,nmax
    fx=func(x)
    dfx=dfunc(x,1)
    d2fx=dfunc(x,2)
    x=x-2.0*fx*dfx/(2.0*dfx*dfx-fx*d2fx)
    IF (abs(fx)<eps) EXIT
  END DO
  halley=x
  END FUNCTION halley

  FUNCTION abbasbandy(x)
  COMPLEX(KIND = 8) abbasbandy,x,fx,dfx,d2fx,d3fx
  REAL(KIND=8) eps
  INTEGER nmax,i
  eps=1.0e-8;
  nmax=100;
  x=a
  DO i=1,nmax
    fx=func(x)
    dfx=dfunc(x,1)
    d2fx=dfunc(x,2)
    d3fx=dfunc(x,3)
    x=x-(fx/dfx+fx*fx*d2fx/(2.0*dfx*dfx*dfx)+fx*fx*fx*d3fx/(6.0*dfx*dfx*dfx*dfx));
    IF (abs(fx)<eps) EXIT
  END DO
  abbasbandy=x
  END FUNCTION abbasbandy
END PROGRAM halley2
```

```
---------- Capture Output ----------
> "E:\okul\SCO1\run.bat" halley2

E:\okul\SCO1>halley2
 halley( (  1.0000000000000000     ,  1.0000000000000000     ) ) =  ( -3.6628389551222020E-026,  1.4142135623730951     )
 abbasbandy( (  1.0000000000000000     ,  1.0000000000000000     ) ) =  (  3.2208545212530678E-033,  1.4142135623730951     )

> Terminated with exit code 0.
```

Halley Method in Python language: Not that a graphic input window is used to input variables in this code.

Program 2.11-10 Haley's method complex Python code

```
 #halley.py
from math import *
from graphics import *
from f_x import *;

class f1(f_x):
  def func(self,x):
      y=x*x+2;
      return y;

def halley(f,x):
   # halley root finding method
  nmax=100
  tolerance=1.0e-10
  for i in range(0,nmax):
      fx= f.func(x);
      dfx=f.dfunc(x,1);
      d2fx=f.dfunc(x,2);
      x=x-2.0*fx*dfx/(2.0*dfx*dfx-fx*d2fx)
```

```
    print("fx",fx,"dfx",dfx,"x=",x)
    if abs(fx)<tolerance: return x
  return x

def read(s1,s2):
   # reads x1 and x2 values as float then
   # converts to a float array and returned
   s="read data "+s1+" and "+s2+" as float push mouse to exit";
   win = GraphWin(s, 450, 100)
   win.setCoords(0.0, 0.0, 3.0, 4.0)
   # Draw the interface
   Text(Point(1,2), s1).draw(win)
   Text(Point(1,1),s2).draw(win)
   i1 = Entry(Point(2,2), 20)
   i1.setText("")
   i1.draw(win);
   i2 = Entry(Point(2,1), 20)
   i2.setText("")
   i2.draw(win);
   # wait for a mouse click
   win.getMouse()
   # convert input
   x1 = float(i1.getText());
   x2 = float(i2.getText());
   x=[x1,x2];
   return x;
def main():
   x1=read("x1:","x2:");
   x=complex(x1[0],x1[1]);
   f=f1();
   r=halley(f,x);
   print(" ROOT  : ",r,"\nFUNCTION VALUE : ",f.func(r));
main();
```

```
---------- Capture Output ----------
> "D:\co\anaconda\pythonw.exe" halley.py
fx (2+2j) dfx (2.0000000000002363+1.9999999999999245j) x= (-0.199999999825631+1.3999999997473398j)
fx (0.08000000063770107-0.5599999994107027j) dfx (-0.39999999965142174+2.7999999994947347j) x=
(0.0010152284274246581+1.4142131979720642j)
fx (2.0613703868299638e-06+0.002871498882040751j) dfx (0.002030456854918 6653+2.8284263959441294j) x= (-
1.3079811626154347e-10+1.414213562373095j)
fx (4.440892098500626e-16-3.699529398998553e-10j) dfx (-2.615318015063481e-10+2.8284271247461894j) x= 1.4142135623730951j
fx (-4.440892098500626e-16+0j) dfx 2.8284271247461903j x= (4.4398356754174374e-46+1.414213562373095j)
 ROOT  :  (4.4398356754174374e-46+1.414213562373095j)
FUNCTION VALUE :  (4.440892098500626e-16+1.25577516537665e-45j)

> Terminated with exit code 0.
```

Hasan[50] proposed several higher order methods that is relatively more economical to use (relatively less function calculations) Some of them are listed here.

$$x_{n+1} = x_n - \frac{f(x_n)}{f'\left(x_n - \frac{f(x_n)}{2f'(x_n)}\right)} \qquad (2.11.11)$$

$$x_{n+1} = x_n - \frac{2f(x_n)}{f'(x_n) + f'\left(x_n - \frac{f(x_n)}{2f'(x_n)}\right)} \qquad (211.12)$$

$$x_{n+1} = x_n - \frac{2f(x_n)}{f'\left(x_n - \alpha\frac{f(x_n)}{f'(x_n)}\right) + f'\left(x_n - \beta\frac{f(x_n)}{f'(x_n)}\right)} \quad where\ \alpha + \beta = 1 \qquad (2.11.13)$$

$$x_{n+1} = x_n - \frac{2f(x_n)}{f'(x_n) + \frac{1}{r}f'(x_n)log\left(\frac{f'(x_n)f'(x_n) - f(x_n)f''(x_n)}{f'(x_n)f'(x_n)}\right)} \qquad (2.11.14)$$

Program 2.11-11 Higher order methods, proposed by Hasan[50] java code

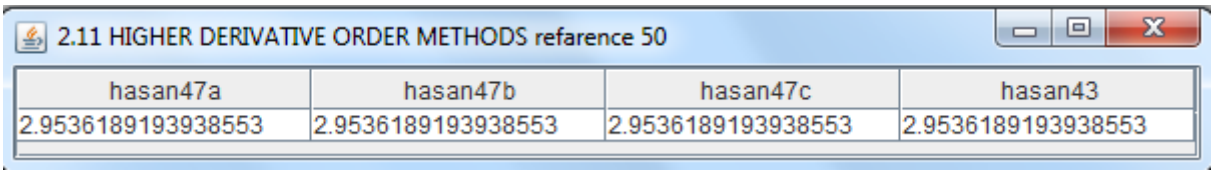
```
//  High order methods proposed by Hasan
// 50. Hasan, Mohammed A., On the Derivation of Higher Order Root-finding Methods,
//  Proceedings of the 2007 American Control Conference, ThA07.3,
//  New York City, USA, July 11-13 2007
import java.util.*;
import java.awt.*;
import java.applet.Applet;
import java.awt.event.*;
import javax.swing.*;
//============= function to be defined ================

class fb extends f_x
{
  public double func (double x)
  { return x*x*x+3.6*x-36.4;}
}
//============================================================
public class NA5B2
{
public static double hasan47a(f_x f,double x)
{ // third order
int nmax=500;
double tolerance=1.0e-8;
double fx,dfx;
double x1=0,dfx1=0;
for(int i=0;i<nmax;i++)
{  fx=f.func(x);
  dfx=f.dfunc(x);
  x1=x-fx/(2.0*dfx);
  dfx1=f.dfunc(x1);
  x-=fx/dfx1;
 if(Math.abs(fx)<tolerance) { return x;}
}
JOptionPane.showMessageDialog(null,"Maximum number of iteration is exceeded in hasan47a Method\n"+
  " results may not be valid","MAXIMUM ITERATION WARNING",JOptionPane.WARNING_MESSAGE);
return x;
}
public static double hasan47b(f_x f,double x)
{ //third order
int nmax=500;
double tolerance=1.0e-8;
double fx,dfx;
double x1=0,dfx1=0;
for(int i=0;i<nmax;i++)
{  fx=f.func(x);
  dfx=f.dfunc(x);
  x1=x-fx/dfx;
  dfx1=f.dfunc(x1);
  x-=2.0*fx/(dfx+dfx1);
 if(Math.abs(fx)<tolerance) { return x;}
}
JOptionPane.showMessageDialog(null,"Maximum number of iteration is exceeded in hasan47b Method\n"+
  " results may not be valid","MAXIMUM ITERATION WARNING",JOptionPane.WARNING_MESSAGE);
return x;
}
public static double hasan47c(f_x f,double x,double alfa)
```

```
{ //third order
int nmax=500;
double beta=1-alfa;
double tolerance=1.0e-8;
double fx,dfx;
double x1=0,dfx1=0;
double x2=0,dfx2=0;
for(int i=0;i<nmax;i++)
{  fx=f.func(x);
   dfx=f.dfunc(x);
   x1=x-alfa*fx/dfx;
   dfx1=f.dfunc(x1);
   x2=x-beta*fx/dfx;
   dfx2=f.dfunc(x2);
   x-=2.0*fx/(dfx1+dfx2);
  if(Math.abs(fx)<tolerance) { return x;}
}

JOptionPane.showMessageDialog(null,"Maximum number of iteration is exceeded in hasan47c Method\n"+
   " results may not be valid","MAXIMUM ITERATION WARNING",JOptionPane.WARNING_MESSAGE);
return x;
}

public static double hasan43(f_x f,double x)
{ //fourth order
int r=3;
int nmax=500;
double tolerance=1.0e-8;
double fx,dfx,d2fx;
for(int i=0;i<nmax;i++)
{  fx=f.func(x);
   dfx=f.dfunc(x);
   d2fx=f.dfunc(x,2);
   x-=fx/(dfx+1.0/r*dfx*Math.log((dfx*dfx-fx*d2fx)/(dfx*dfx)));
  if(Math.abs(fx)<tolerance) { return x;}
}
JOptionPane.showMessageDialog(null,"Maximum number of iteration is exceeded in hasan43 Method\n"+
   " results may not be valid","MAXIMUM ITERATION WARNING",JOptionPane.WARNING_MESSAGE);
return x;
}
    public static void main (String args[])
    {
            fb f=new fb();
            dfb df=new dfb();
    double x0;
    x0=Double.parseDouble(JOptionPane.showInputDialog(" first guess for x : "));
    double r[]=new double[4];
    String s[]={"hasan47a","hasan47b","hasan47c","hasan43"};
    r[0]=hasan47a(f,x0);
    r[1]= hasan47b(f,x0);
    r[2]= hasan47c(f,x0,0.5);
    r[3]= hasan43(f,x0);
    Text.print(r,s,"2.11 HIGHER DERIVATIVE ORDER METHODS referance 50");
    }
}
```



Program 2.11-12 Higher order methods, proposed by Hasan[50] python code

```
 #halley.py
from math import *
from graphics import *
from f_x import *;

class f1(f_x):
  def func(self,x):
      y=x*x-2;
      return y;
def hasan(f,x):
       # finding real roots by using hasan method fourth order
       nmax=500;
       r=3;
       tolerance=1.0e-15;
       for iter in range(0,nmax):
          fx=f.func(x);
          dfx=f.dfunc(x,1);
          d2fx=f.dfunc(x,2);
          x=x-fx/(dfx+1.0/r*dfx*log((dfx*dfx-fx*d2fx)/(dfx*dfx)));
          if abs(fx)<tolerance: return x;
       return x;
f=f1();
x=float(input("enter initial estimation : "))
r=hasan(f,x);
print(" hasan1 method ROOT  : ",r,"\nFUNCTION VALUE : ",f.func(r));
```

```
> "D:\co\anaconda\pythonw.exe" hasan2.py
enter initial estimation : 1
 hasan1 method ROOT  :  1.4142135623730951
FUNCTION VALUE :  4.440892098500626e-16

> Terminated with exit code 0.
```

A complex version of Hasan formula is also given here:

Program 2.11-13 Complex Hasan method[50] (java version) Hasan1.java

```
// compleks Hasan's method to find roots of the polynomials
// xn+1=xn-f(xn)/(f'(xn)+1/3*f'(xn)*log((f'(xn)*f'(xn)*-f(xn)*f''(xn))/(f'(xn)*f'(xn))
import java.util.*;
import java.awt.*;
import java.applet.Applet;
import java.awt.event.*;
import javax.swing.*;

public class Hasan1
{
public static c Hasan(icf_cx f,c x)
{
double eps=1.0e-15;
int m=100;
int k=0;
c f1,df,d2f;
c b1,b2,b3,b4,b5,b6,b7;
double r=3.0;
do{
```

```
  f1=f.func(x);
  df=f.dfunc(x);
  d2f=f.dfunc(x);
  b1=c.p(c.p(df,"*",df),"-",c.p(f1,"*",d2f));
  b2=c.p(df,"*",df);
  b3=c.p(b1,"/",b2);
  b4=c.p(df,"*",c.log(b3));
  b5=c.p(b4,"/",r);
  b6=c.p(df,"+",b5);
  b7=c.p(f1,"/",b6);
  x=c.p(x,"-",b7);
  if(c.abs(f1)<eps) return x;
  k++;
}while(c.abs(f1)>eps && k<m);
JOptionPane.showMessageDialog(null,"Warning : Maximum number of iteration is exceeded \n"+
   " result may not be valid","MAXIMUM ITERATION NUMBER WARNING",JOptionPane.WARNING_MESSAGE);
return x;
}
public static String toString(c x[])
{ int n=x.length;
  String s="";
  for(int i=n-1;i>=0;i--) {s+="x["+i+"] = "+x[i]+"\n";}
  return s;
}
public static c Hasan(cf_cx f,c x)
{
double eps=1.0e-15;
int m=100;
int k=0;
c f1,df,d2f;
c b1,b2,b3,b4,b5,b6,b7;
double r=3.0;
do{
  f1=f.func(x);
  df=f.dfunc(x);
  d2f=f.dfunc(x);
  b1=c.p(c.p(df,"*",df),"-",c.p(f1,"*",d2f));
  b2=c.p(df,"*",df);
  b3=c.p(b1,"/",b2);
  b4=c.p(df,"*",c.log(b3));
  b5=c.p(b4,"/",r);
  b6=c.p(df,"+",b5);
  b7=c.p(f1,"/",b6);
  x=c.p(x,"-",b7);
  if(c.abs(f1)<eps) return x;
  k++;
}while(c.abs(f1)>eps && k<m);
JOptionPane.showMessageDialog(null,"Warning : Maximum number of iteration is exceeded \n"+
   " result may not be valid","MAXIMUM ITERATION NUMBER WARNING",JOptionPane.WARNING_MESSAGE);
return x;
}
    public static double[] input(String s)
    {
    String s1=JOptionPane.showInputDialog(s);
    StringTokenizer token=new StringTokenizer(s1);
    int n=token.countTokens()-1;
    int m=n+1;
    double a[]=new double[m];
    int j=0;
    while(token.hasMoreTokens())
    {
    Double ax=new Double(token.nextToken());
    a[j++]=ax.doubleValue();
    }
```

```
    return a;
    }

    public static void main (String args[])
    { icf_cx f=(c x)->c.p(c.p(x,"*",x),"+",2.0);
      c x1=new c(1.0,1.0);
      c y=Hasan(f,x1);
      System.out.println("root = "+y);
}}
```

```
---------- Capture Output ----------
> "C:\java\bin\java.exe" Hasan1
root = (   0.0000000000 -    1.4142135624i
> Terminated with exit code 0.
```

In order to change python code to complex version, changing input variable to complex and adding the complex math class (log function should be defined as complex) is enough

Program 2.11-14 Higher order methods, proposed by Hasan[50] complex python code hasan2.py

```
#halley.py
from math import *
from cmath import *
from graphics import *
from f_x import *;

class f1(f_x):
  def func(self,x):
     y=x*x+2;
     return y;
def hasan(f,x):
      # finding real roots by using hasan method fourth order
      nmax=500;
      r=3;
      tolerance=1.0e-15;
      for iter in range(0,nmax):
         fx=f.func(x);
         dfx=f.dfunc(x,1);
         d2fx=f.dfunc(x,2);
         x=x-fx/(dfx+1.0/r*dfx*log((dfx*dfx-fx*d2fx)/(dfx*dfx)));
         if abs(fx)<tolerance: return x;
      return x;
f=f1();
x=complex(input("enter initial estimation : "))
r=hasan(f,x);
print(" hasan1 method ROOT  : ",r,"\nFUNCTION VALUE : ",f.func(r));
```

```
---------- Capture Output ----------
> "D:\co\anaconda\pythonw.exe" hasan2.py
enter initial estimation : 1+1j
 hasan1 method ROOT  :  1.414213562373095j
FUNCTION VALUE :  (4.440892098500626e-16+0j)
> Terminated with exit code 0.
```

2.12 FINDING THE ROOTS LIMITS

In a one variable function, if a root to be find and if only one root existed in the given region boundaries of the function should carry inverse sign. To further process this, if given region is divided up to smaller region and function values at the limits are cheched, the root area can be narrowed down. region can be enlarged to the side that it is estimated to have a root to ensure that at least one root is existed in a given region.

Program 2.12-1 Root finding NA13B

```
//
// root findings in a given region
// Dr. M. Turhan Çoban

import java.util.*;
import java.awt.*;
import java.applet.Applet;
import java.awt.event.*;
import javax.swing.*;

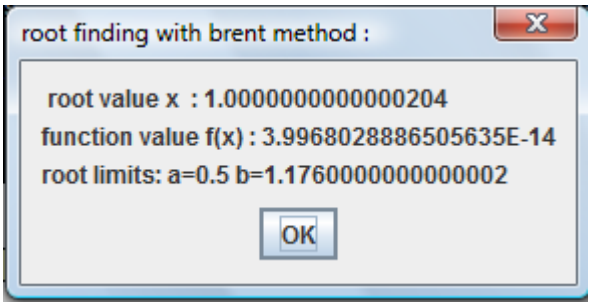
class fx extends f_x
{
public double func(double x)
{return x*x*x-6.0*x*x+11.0*x-6.0;}
}

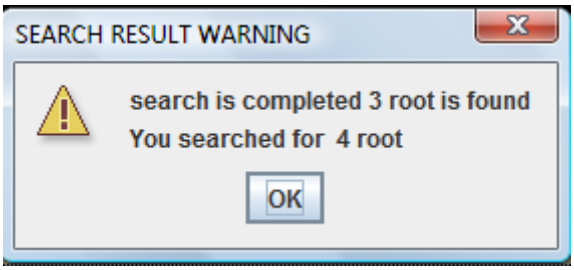
public class NA13B
{
public static double[] enlarge(f_x f,double x1)
{return enlarge(f,x1,(x1+0.1));}

public static double[] enlarge(f_x f,double x1,double x2)
{
	//kök olan bir bölge bulana kadar verilen siniri genişletir
	int NTRY=50;
	double a[]=new double[2];
	double FACTOR=1.6;
	int j;
	double f1,f2;
	if (x1 == x2) System.out.println("Bad initial range in ZBRAC");
	f1=f.func(x1);
	f2=f.func(x2);
	for (j=1;j<=NTRY;j++) {
		if (f1*f2 < 0.0) {a[0]=x1;a[1]=x2;return a;}
		if (Math.abs(f1) < Math.abs(f2))
			f1=f.func(x1 += FACTOR*(x1-x2));
		else
			f2=f.func(x2 += FACTOR*(x2-x1));
	}
	{a[0]=x1;a[1]=x2;return a;}
}

public static double[][] rootlimits(f_x f,double x1,double x2,int n,int nbb)
{
	// find root limits
	// n : number of subdivisions
	// x1,x2 : limit values of the root
	// nbb = number of roots in the serched region
	//
	int nb;
	int i;
	double x,fp,fc,dx;
	double xb[][]=new double[2][nbb];
	nb=0;
	dx=(x2-x1)/n;
	x=x1;
	fp=f.func(x1);
```

```
            for (i=1;i<=n;i++)
            {
                        x+=dx;
                        fc=f.func(x);
                        // eğer kök olan bölge bulunduysa.....
                        if (fc*fp < 0.0 || fp==0) {
                                    xb[0][nb]=x-dx;
                                    xb[1][nb]=x;
                                    nb++;
                        }
                        fp=fc;
                        if (nbb == nb) return xb;
            }
if( nb == 0)
            JOptionPane.showMessageDialog(null,"search is completed no root is found"
   ,"SEARCH RESULT WARNING",JOptionPane.WARNING_MESSAGE);
else if(nb<nbb)
  {         JOptionPane.showMessageDialog(null,"search is completed "+nb+" root is found \n"+
    "You searched for  "+nbb+" root","SEARCH RESULT WARNING",JOptionPane.WARNING_MESSAGE);
   double xc[][]=new double[2][nb];
   for (i=0;i<nb;i++) {xc[0][i]=xb[0][i];xc[1][i]=xb[1][i];}
   return xc;
  }
return xb;
}
    public static void main (String args[])
    {
    double a[],x0,r;
    double dx=0.1;
    fx f=new fx();
    a=enlarge(f,Double.parseDouble(JOptionPane.showInputDialog(" first guess x0 : ")));
    System.out.println(a[0]+" "+a[1]);
    r= NA13.brent(f,a[0],a[1]);
    JOptionPane.showMessageDialog(null," root value x  : "+r+"\nfunction value f(x) : "+f.func(r)+"\nroot limits:
a="+a[0]+" b="+a[1],
    "root finding with brent method : ",JOptionPane.PLAIN_MESSAGE);
    double b[][]=rootlimits(f,0.0,10.0,10,4);
    System.out.println(Matrix.toString(b));
    b=rootlimits(f,b[0][0],b[1][0],10,4);
    System.out.println(Matrix.toString(b));
    System.exit(0);
    }
}
```






```
---------- Capture Output ----------
> "C:\turhan\java\bin\javaw.exe" NA13B
0.5 1.1760000000000002
      1.000000000000000      2.000000000000000      3.000000000000000
      2.000000000000000      3.000000000000000      4.000000000000000

      1.000000000000000
      1.100000000000000

> Terminated with exit code 0.
```

2.13 RIDDER METHOD

False position (Ragula Falsi) method was previously investigated. Ridder method is a modified form False position method. If a root existed between x1 and x2 Ridder method first evaluate the midpoint

$$x_3 = \frac{x_1+x_2}{2} \qquad (2.13.1)$$

and than solve a special conversion conversion factor e^Q. This convertion linearize the equation. It is basically find the root of the quadratic function

$$f(x_1) - 2\,f(x_3)e^Q + f(x_1) - 2\,f(x_2)e^Q = 0 \qquad (2.13.2)$$

The root of this function can be calculated as
of the quadratic function

$$e^Q = \frac{f(x_3)+sign[f(x_2)]\sqrt{f(x_3)^2-f(x_1)f(x_2)}}{f(x_2)} \qquad (2.13.3)$$

Function sign(x) will be given the sign of x. If x is smaller than 0 it will be -1, if x is bigger than 0 it will be +1. Now the false position method is applied, not to the values f(x1), f(x3), f(x2), but to the values $f(x_1)$, $f(x_3)e^Q$, $f(x_2)e^{2Q}$, yielding a new guess for the root, x_4. The overall formula is:

$$x_4 = x_3 + (x_3 - x_1)\frac{f(x_3)+sign[f(x_1)-f(x_2)]f(x_3)}{\sqrt{f(x_3)^2-f(x_1)f(x_2)}} \qquad (2.13.4)$$

Because the equation is quadratic, it supplies a quadratic approximation. In addition the root x_4 should be in between x_1 and x_2.

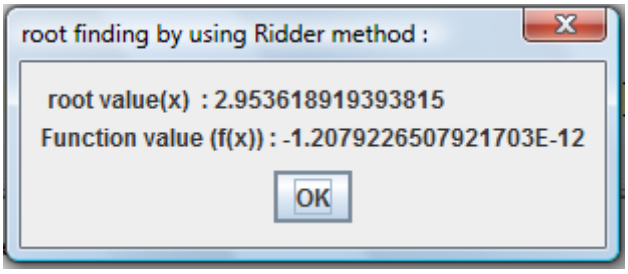
Program 2.13-1 Root finding NA13C Ridder Method

```
// Ridder method
import java.util.*;
import java.awt.*;
import java.applet.Applet;
import java.awt.event.*;
import javax.swing.*;
//==============Function to be defined ================
class fb extends f_x
{
  public double func (double x)
  { return x*x-2;}
```

```
}
public class NA13C
{
// Ridder method for solving roots of non-linear equations

public static double SIGN(double a,double b)
{
return Math.abs(a)*b/Math.abs(b);
}
public static double ridder(f_x f, double x1, double x2)
{double xacc=1.0e-9;
 return ridder(f, x1,x2,xacc);
}
public static double ridder(f_x f, double x1, double x2, double xacc)
{
        int NR;
        int MAXITER=200;
        double ABC=-1.0e30;
        int j;
        double answer,f_high,f_low,f_mid,fnew,s,x_high,x_low,x_mid,xnew;
        f_low=f.func(x1);
        f_high=f.func(x2);
        if ((f_low > 0.0 && f_high < 0.0) || (f_low < 0.0 && f_high > 0.0)) {
                x_low=x1;
                x_high=x2;
                answer=ABC;
                for (j=0;j<MAXITER;j++) {
                        x_mid=0.5*(x_low+x_high);
                        f_mid=f.func(x_mid);
                        s=Math.sqrt(f_mid*f_mid-f_low*f_high);
                        if (s == 0.0) return answer;
                        xnew=x_mid+(x_mid-x_low)*((f_low >= f_high ? 1.0 : -1.0)*f_mid/s);
                        if (Math.abs(xnew-answer) <= xacc) return answer;
                        answer=xnew;
                        fnew=f.func(answer);
                        if (fnew == 0.0) return answer;
                        if (SIGN(f_mid,fnew) != f_mid) {
                                x_low=x_mid;
                                f_low=f_mid;
                                x_high=answer;
                                f_high=fnew;
                        } else if (SIGN(f_low,fnew) != f_low) {
                                x_high=answer;
                                f_high=fnew;
                        } else if (SIGN(f_high,fnew) != f_high) {
                                x_low=answer;
                                f_low=fnew;
                        } else System.out.println("we should not reach to this point");
                        if (Math.abs(x_high-x_low) <= xacc) return answer;
                }
                JOptionPane.showMessageDialog(null,"Warning : Maximum number of iteration is exceeded \n"+
    " result may not be valid","MAXİMUM ITERATION NUMBER WARNING",JOptionPane.WARNING_MESSAGE);
        }
        else {
                if (f_low == 0.0) return x1;
                if (f_high == 0.0) return x2;
                JOptionPane.showMessageDialog(null,"Warning : No roots existed between the given limits ","NO ROOTS
WARNING",JOptionPane.WARNING_MESSAGE);
        }
        return 0.0;
}
   public static void main (String args[])
   {
         fb f=new fb();
```

```
    double a,b;
    a=Double.parseDouble(JOptionPane.showInputDialog(" Enter lower limit of the function a : "));
    b=Double.parseDouble(JOptionPane.showInputDialog(" Enter upper limit of the function b : "));
    double r;
    r= ridder(f,a,b);
    JOptionPane.showMessageDialog(null," root value(x)  : "+r+"\nFunction value (f(x)) : "+f.func(r),
    "root finding by using Ridder method : ",JOptionPane.PLAIN_MESSAGE);
    System.exit(0);
    }
}
```



Program 2.13-2 Root finding Ridder Method C++ version

```
//Ridder method C++
#include <stdio.h>
#include <iostream>
#include <math.h>
#include <iomanip>
using namespace std;

class f_x{
public: virtual double func(double x)=0;
public: double dfunc(double x)
{ double h=0.0001;
  double hh=1.0/h;
  double df=(3.0*func(x-4.0*h)-32.0*func(x-3.0*h)+168.0*func(x-2.0*h)-672.0*func(x-h)+672.0*func(x+h)-168.0*func(x+2.0*h)
+32.0*func(x+3.0*h) - 3.0*func(x+4.0*h))/840.0*hh;
  return df;
}
};

class f1:public f_x
{ public: double func(double x)
  { double ff=x*x-2.0;
              return ff;
  }
};
double SIGN(double a,double b)
{
return fabs(a)*b/fabs(b);
}
double ridder(f_x &f, double x1, double x2, double xacc)
{   int NR;
            int MAXITER=200;
            double ABC=-1.0e30;
            int j;
            double answer,f_high,f_low,f_mid,fnew,s,x_high,x_low,x_mid,xnew;
            f_low=f.func(x1);
            f_high=f.func(x2);
            if ((f_low > 0.0 && f_high < 0.0) || (f_low < 0.0 && f_high > 0.0)) {
                        x_low=x1;
                        x_high=x2;
                        answer=ABC;
                        for (j=0;j<MAXITER;j++) {
```

```
                                x_mid=0.5*(x_low+x_high);
                                f_mid=f.func(x_mid);
                                s=sqrt(f_mid*f_mid-f_low*f_high);
                                if (s == 0.0) return answer;
                                xnew=x_mid+(x_mid-x_low)*((f_low >= f_high ? 1.0 : -1.0)*f_mid/s);
                                if (fabs(xnew-answer) <= xacc) return answer;
                                answer=xnew;
                                fnew=f.func(answer);
                                if (fnew == 0.0) return answer;
                                if (SIGN(f_mid,fnew) != f_mid) {
                                        x_low=x_mid;
                                        f_low=f_mid;
                                        x_high=answer;
                                        f_high=fnew;
                                } else if (SIGN(f_low,fnew) != f_low) {
                                        x_high=answer;
                                        f_high=fnew;
                                } else if (SIGN(f_high,fnew) != f_high) {
                                        x_low=answer;
                                        f_low=fnew;
                                } else cout<<"we should not reach to this point";
                                if (fabs(x_high-x_low) <= xacc) return answer;
                        }
                cout<<"Warning : Maximum number of iteration is exceeded \n";
                }
                else {
                        if (f_low == 0.0) return x1;
                        if (f_high == 0.0) return x2;
                        cout<<"Warning : No roots existed between the given limits ";
                }
                return 0.0;
}
    int main ()
    {
    double x1,x2,x3;
    cout << "root estimation x1 : ";
    cin >>x1;
    cout << "root estimation x2 : ";
    cin >>x2;
    double r;
    f1 f;
    cout<<setprecision(20);
    r= ridder(f,x1,x2,1.0e-9);
    cout<<" root value  : "<<r<<"\nFunction value : "<<f.func(r);
    }
```

```
---------- Capture Output ----------
> "D:\okul\SCO1\run.bat" ridder

D:\okul\SCO1>ridder
root estimation x1 : 1
root estimation x2 : 3
 root value  : 1.4142135621695027
Function value : -5.7584620860758051e-010
> Terminated with exit code 0.
```

Program 2.13-3 Root finding Ridder Method python version

```
from math import *

def SIGN(a,b):
     return abs(a)*b/abs(b);

def ridder(f,x1,x2):
```

```
    xacc=1.0e-9;
    MAXITER=200;ABC=-1.0e30;
    f_low=f(x1);
    f_high=f(x2);
    b1=f_low > 0.0 and f_high < 0.0;
    b2=f_low < 0.0 and f_high > 0.0;
    b3=b1 or b2;
    if b3 :
      x_low=x1;
      x_high=x2;
      answer=ABC;
      for j in range(0,MAXITER):
        x_mid=0.5*(x_low+x_high);
        f_mid=f(x_mid);
        s=sqrt(f_mid*f_mid-f_low*f_high);
        if s == 0.0: return answer;
        if f_low>=f_high: xx=1;
        else: xx=-1;
        xnew=x_mid+(x_mid-x_low)*xx*f_mid/s;
        if abs(xnew-answer) <= xacc: return answer;
        answer=xnew;
        fnew=f(answer);
        if fnew == 0.0: return answer;
        if SIGN(f_mid,fnew) != f_mid:x_low=x_mid;f_low=f_mid;x_high=answer;f_high=fnew;
        elif SIGN(f_low,fnew) != f_low:x_high=answer;f_high=fnew;
        elif SIGN(f_high,fnew) != f_high:x_low=answer;f_low=fnew;
        else: print("we should not reach to this point");
      if abs(x_high-x_low) <= xacc: return answer;
      print("Warning : Maximum number of iteration is exceeded \n");
    else:
       if f_low == 0.0: return x1;
       if f_high == 0.0: return x2;
       print("Warning : No roots existed between the given limits ");
    return 0.0;

f1 = lambda x: x*x-2
a=float(input("enter lower limit : "))
b=float(input("enter upper limit : "))
r=ridder(f1,a,b);
print(" ROOT  : ",r,"\nFUNCTION VALUE : ",f1(r))
```

```
runfile('D:/okul/SCO1/ridder2.py', wdir='D:/okul/SCO1')
Reloaded modules: Mathd

enter lower limit : 1

enter upper limit : 3
 ROOT  :  1.4142135621695027
FUNCTION VALUE :  -5.758462595650826e-10
```

2.14 ROOTS OF THE SECOND AND THIRD DEGREE POLYNOMIALS

The roots of quadratic polynomials are well known.

$$f(x) = ax^2 + bx + c = 0 \qquad (2.14.1)$$

$$\Delta = b^2 - 4ac \qquad (2.14.2)$$

$$x_0 = \frac{-b}{2a} + \frac{\sqrt{\Delta}}{2a} \qquad (2.14.3)$$

$$x_1 = \frac{-b}{2a} - \frac{\sqrt{\Delta}}{2a} \qquad (2.14.4)$$

Of course if $\Delta = b^2 - 4ac$ is negative, roots will be complex numbers.

Program 2.14-1 Root of quadratic polynomial (analytical formula)

```
public class square_roots
{
public static double[][] square_roots(double d[])
{
   double x[][]=new double[2][2];
   double a=d[2];
   double b=d[1];
   double c=d[0];
   double delta=b*b-4*a*c;
   if(delta>=0)
   {
   x[0][0]=(-b+Math.sqrt(delta))/(2*a);
   x[1][0]=(-b-Math.sqrt(delta))/(2*a);
   }
   else
   {x[0][0]=-b/(2*a);
    x[1][0]=-b/(2*a);
    x[0][1]=Math.sqrt(-delta)/(2*a);
    x[1][1]=Math.sqrt(-delta)/(2*a);
   }
   return x;
}
public static void main(String arg[])
{
double d[]={1,-2,1};
double x[][]=square_roots(d);
String s="";
for(int i=0;i<x.length;i++)
{ s+="x["+i+"] = "+x[i][0]+" + "+x[i][1]+"*i\n";}
System.out.println(s);
}
}
```

```
---------- Capture Output ----------
> "D:\co\java\bin\javaw.exe" square_roots
x[0] = 1.0 + 0.0*i
x[1] = 1.0 + 0.0*i


> Terminated with exit code 0.
```

Program 2.14-2 Root of quadratic polynomial (analytical formula) python version

```
from math import *

def square_roots(d):
   x= [0 for i in range(2)];
   a=d[2];
   b=d[1];
   c=d[0];
   delta=b*b-4*a*c;
   x[0]=(-b+sqrt(delta))/(2*a);
   x[1]=(-b-sqrt(delta))/(2*a);
   return x;

a=[1.0,-2.0,1.0]  # y(x)=x^2-2*x++1
print("Roots of the polynomial : ",square_roots(a))
```

```
---------- Capture Output ----------
> "D:\co\python\pythonw.exe" square_roots.py
Roots of the polynomial :  [1.0, 1.0]
```

```
program quadratic1
IMPLICIT NONE
REAL(KIND=8) a,b,c
COMPLEX(KIND=8), DIMENSION(0:1) :: x
print*,'ax^2+bx+c=0'
print*,'a'
read*, a
print*,'b'
read*, b
```

```
print*,'c'
read*, c
call quadratic(a,b,c,x)
print *,'x(0) = ',x(0)
print *,'x(1) = ',x(1)

CONTAINS
  SUBROUTINE quadratic(a,b,c,x)
  REAL(KIND=8) a,b,c,determinant
  COMPLEX(KIND=8), DIMENSION(0:1) :: x
  determinant=b*b-4.0*a*c
  x(0)=(-b+sqrt(determinant))/(2.0*a)
  x(1)=(-b-sqrt(determinant))/(2.0*a)
  END SUBROUTINE quadratic
end program quadratic1
```

```
---------- Capture Output ----------
> "E:\okul\SCO1\run.bat" quadratic1
E:\okul\SCO1>quadratic1
 ax^2+bx+c=0
 a
1
 b
-2
 c
1
 x(0) =  ( 1.0000000000000000    , 0.0000000000000000    )
 x(1) =  ( 1.0000000000000000    , 0.0000000000000000    )
> Terminated with exit code 0.
```

Similar to quadratic polynomials, third degree polynomials can also be solved by using analytical formulations. The basic difference is that the roots should be calculated as complex variables. Assuming a third degree polynomial of



Figure 2.141 NiccoloTartaglia

$$f(x) = d_3x^3 + d_2x^2 + d_1x + d_0 = 0 \quad (2.14.1)$$

$$a = \frac{d_2}{d_3} \quad b = \frac{d_1}{d_3} \quad c = \frac{d_0}{d_3} \quad (2.14.2)$$

$$f(x) = x^3 + ax^2 + bx + c = 0 \quad (2.14.3)$$

$Q = \frac{a^2-3b}{9}$ (2.14.4) $\quad y_1 = 2a^3$ (2.14.5) $y_2 = -9ab$ (2.14.6) $y_3 = 27c$ (2.14.7)

$y_4 = y_1 + y_2 + y_3$ (2.14.8)

$R = \frac{y_4}{54} = \frac{2a^3-9ab+27c}{54} \quad \theta = cos^{-1}\left(\frac{R}{\sqrt{Q^3}}\right)$ (2.14.9)

If $R^2 < Q^3$ three real roots are existed

$$x_0 = \left(-2\sqrt{Q}\right)cos\left(\frac{\theta}{3}\right) - \frac{a}{3} \quad (2.14.10)$$

$$x_1 = \left(-2\sqrt{Q}\right)cos\left(\frac{\theta-2\pi}{3}\right) - \frac{a}{3} \quad (2.14.11)$$

$$x_2 = \left(-2\sqrt{Q}\right)cos\left(\frac{\theta+2\pi}{3}\right) - \frac{a}{3} \quad (2.14.12)$$

(This equation first appears in Chapter VI of François Viete's treatise "De emendatione," published in 1615!, Original equation created by Niccolo Tartaglia in 1539)

If $R^2 \geq Q^3$

$$A = (R + \sqrt{R^2 - Q^3})^{1/3} \quad (2.14.13)$$

If A=0 ⇒B=0 (2.14.14)

If A< >0 ⇒ $B = \frac{Q}{A}$ (2.14.15)

$$x_0 = (A + B) - \frac{a}{3} \quad (2.14.16) \quad \text{real root}$$

$$x_1 = \left[-\frac{(A+B)}{2} - \frac{a}{3}\right] + \left[-\frac{\sqrt{3}(A-B)}{2}\right] i \quad (2.14.17) \quad \text{complex root}$$

$$x_2 = \left[-\frac{(A+B)}{2} - \frac{a}{3}\right] - \left[-\frac{\sqrt{3}(A-B)}{2}\right] i \quad (2.14.18) \quad \text{complex root}$$

Program 2.14-3 Root of third degree polynomial (analytical formula-java version)

```
 import java.util.*;
import java.awt.*;
import java.applet.Applet;
import java.awt.event.*;
import javax.swing.*;
import java.io.*;
import java.text.*;
import java.util.Locale;
//import Numeric;

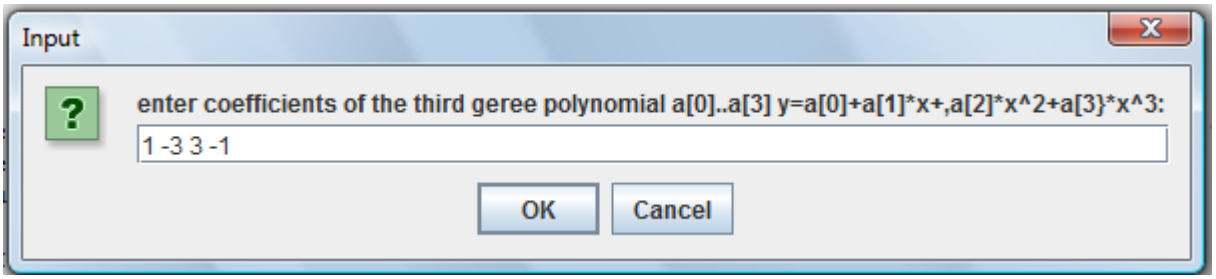
class NA13D
{
public static double[][] cubic_roots(double d[])
{
// roots of a degree 3 polynomial
// P(x)=a[3]*x^3+a[2]*x^2+a[1]*x+a[0]=0;
// assuming all a is all real;
// if more than 4 values are entered the remaining ones will be ignored
double x[][]=new double[2][3];
double a,b,c;
a=d[2]/d[3];
b=d[1]/d[3];
c=d[0]/d[3];
double Q,R,theta;
Q=(a*a-3.0*b)/9.0;
double y1,y2,y3,y4;
y1=2.0*a*a*a;
y2=-9*a*b;
y3=27*c;
y4=y1+y2+y3;
R=y4/54.0;
double Q3=Q*Q*Q;
double R2=R*R;
double qq;
double A,B;
if(R2<Q3)
{
qq=-2.0*Math.sqrt(Q);
theta=Math.acos(R/Math.sqrt(Q3));
x[0][0]=qq*Math.cos(theta/3.0)-a/3.0;
x[0][1]=qq*Math.cos((theta-2.0*Math.PI)/3.0)-a/3.0;
x[0][2]=qq*Math.cos((theta+2.0*Math.PI)/3.0)-a/3.0;
}
else
{
A=-Math.pow((R+Math.sqrt(R2-Q3)),(1.0/3.0));
if(A==0) B=0;
else     B=Q/A;
```

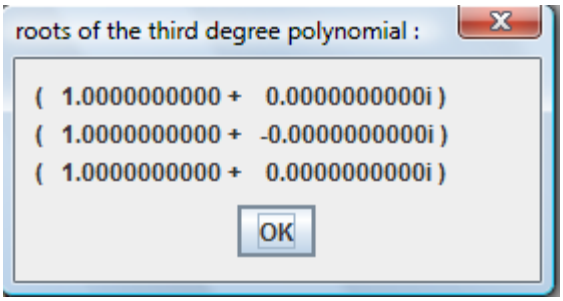
```
x[0][0]=(A+B)-a/3.0;
x[0][1]=-0.5*(A+B)-a/3;
x[1][1]=Math.sqrt(3.0)/2.0*(A-B);
x[0][2]=-0.5*(A+B)-a/3;
x[1][2]=-Math.sqrt(3.0)/2.0*(A-B);
}
return x;
}

public static c[] cubic_rootsC(double c[])
{
// roots of a degree 3 polynomial
//return 3 complex roots
double a[][]=new double[2][3];
a=cubic_roots(c);
c e[]=new c[3];
for(int i=0;i<3;i++)
  e[i]=new c(a[0][i],a[1][i]);
return e;
}

    public static double[] enterdata(String s)
    {
    String s1=JOptionPane.showInputDialog(s);
    StringTokenizer token=new StringTokenizer(s1);
    int n=token.countTokens()-1;
    int m=n+1;
    double a[]=new double[m];
    int j=0;
    while(token.hasMoreTokens())
    {
    Double ax=new Double(token.nextToken());
    a[j++]=ax.doubleValue();
    }
    return a;
    }

  public static void main(String args[]) throws IOException
  {
  String s="enter coefficients of the third degree polynomial a[0]..a[3] y=a[0]+a[1]*x+,a[2]*x^2+a[3}*x^3: ";
  double [] x0=enterdata(s);
  c z[]=cubic_rootsC(x0);
  String s1="";
  for(int i=0;i<3;i++)
    {s1+=c.toString(z[i])+"\n";}
  JOptionPane.showMessageDialog(null,s1,
  "roots of the third degree polynomial : ",JOptionPane.PLAIN_MESSAGE);
  }
}
```





Excel version of the program

Program 2.14-4 Root of third degree polynomial (analytical formula) excel version

Root of third degree polynomial (analytical formula) Excel version

$f(x)=d_3x^3+d_2x^2+d_1x+d_0=0$ $f(x)=x^3+ax^2+bx+c=0$

Enter poynomial coefficients

d3	d2	d1	d0
1	3	3	-1

	a	b	c
1	3	3	-1

Q	0
y1	54
y2	-81
y3	-27
y4	-54
R	-1
θ	#SAYI/0!

true/false
0

true if R^2<Q^3

x0	x1	x2
0	0	0

true if R^2>=Q^3

A	0	
B	0	

true/false
1

xo real	xo imaginary
-1	0

x1 real	x1 imaginary
-1	0

x2 real	x2 imaginary
-1	0

Python version of the program:

Program 2.14-5 Root of third degree polynomial (analytical formula) Python version

```
from math import *

def cubic_roots(d):
  x=[[0 for i in range(3)] for j in range(2)]
  a=d[2]/d[3]
  b=d[1]/d[3]
  c=d[0]/d[3]
  Q=(a*a-3.0*b)/9.0
  y1=2.0*a*a*a
  y2=-9*a*b
  y3=27*c
  y4=y1+y2+y3
  R=y4/54.0
  Q3=Q*Q*Q
  R2=R*R
  if R2<Q3:
    qq=-2.0*sqrt(Q);
    theta=acos(R/Math.sqrt(Q3));
```

```
    x[0][0]=qq*cos(theta/3.0)-a/3.0;
    x[0][1]=qq*cos((theta-2.0*pi)/3.0)-a/3.0;
    x[0][2]=qq*cos((theta+2.0*pi)/3.0)-a/3.0;
  else:
    A=-pow((R+sqrt(R2-Q3)),(1.0/3.0));
    if A==0:  B=0
    else   :  B=Q/A
    x[0][0]=(A+B)-a/3.0
    x[0][1]=-0.5*(A+B)-a/3
    x[1][1]=sqrt(3.0)/2.0*(A-B)
    x[0][2]=-0.5*(A+B)-a/3
    x[1][2]=-sqrt(3.0)/2.0*(A-B)
  return x

a=[1.0,3.0,3.0,1.0]
print("Roots of the polynomial : ",cubic_roots(a))
```

```
---------- Capture Output ----------
> "D:\co\python\pythonw.exe" NA13D.py
Roots of the polynomial :  [[-1.0, -1.0, -1.0], [0, -0.0, 0.0]]
```

Program 2.14-6 Root of third degree polynomial (analytical formula) fortran version cubic1.f90

```
program cubic1
IMPLICIT NONE
REAL(KIND=8), DIMENSION(0:3) :: d
COMPLEX(KIND=8), DIMENSION(0:2) :: x
d=(/1,-3,3,-1/)
call cubic(d,x)
print *,'x(0) = ',x(0)
print *,'x(1) = ',x(1)
print *,'x(2) = ',x(2)

CONTAINS
  SUBROUTINE cubic(d,x)
  REAL(KIND=8), DIMENSION(0:3) :: d
  COMPLEX(KIND=8), DIMENSION(0:2) :: x
  REAL(KIND=8) a,b,c,Q,R,tetha,A1,B1,xr,xi,PI
  PI=4.0*atan(1.0)
  a=d(2)/d(3)
  b=d(1)/d(3)
  c=d(0)/d(3)
  Q=(a*a-3.0*b)
  R=(2.0*a*a*a-9.0*a*b+27.0*c)/54
  tetha=acos(R/sqrt(Q*Q*Q))
  if(R*R<Q*Q*Q) then
    x(0)=(-2.0*sqrt(Q))*cos(tetha/3.0)-a/3.0
    x(1)=(-2.0*sqrt(Q))*cos((tetha-2.0*PI)/3.0)-a/3.0
    x(2)=(-2.0*sqrt(Q))*cos((tetha+2.0*PI)/3.0)-a/3.0
  else
    A1=(R+sqrt(R*R-Q*Q*Q))**(1.0/3.0)
    if(A1==0) THEN
      B1=0
    ELSE
      B1=Q/A
    END IF
    x(0)=(A1+B1)-a/3.0
    xr=-(A1+B1)/2.0-a/3
    xi=-sqrt(3.0)*(A1-B1)/2.0
    x(1)=cmplx(xr,xi)
    x(2)=cmplx(xr,-xi)
  end if
  END SUBROUTINE cubic
```

```
end program cubic1
```

```
- ---------- Capture Output ----------
> "E:\okul\SCO1\run.bat" cubic1

E:\okul\SCO1>cubic1
 x(0) =  (  1.0000000000000000    ,  0.0000000000000000    )
 x(1) =  (  1.0000000000000000    , -0.0000000000000000    )
 x(2) =  (  1.0000000000000000    ,  0.0000000000000000    )

> Terminated with exit code 0.
```

Tartaglia's Poem

Italian:	English Translation:
Quando chel cubo con le cose appresso *Se agguaglia à qualche numero discreto* *Trouan dui altri differenti in esso.* *Dapoi terrai questo per consueto* *Che'llor produtto sempre sia eguale* *Alterzo cubo delle cose neto,* *El residuo poi suo generale* *Delli lor lati cubi ben sottratti* *Varra la tua cosa principale.* *In el secondo de cotestiatti* *Quando che'l cubo restasse lui solo* *Tu osseruarai quest'altri contratti,* *Del numer farai due tal part'à uolo* *Che l'una in l'altra si produca schietto* *El terzo cubo delle cose in stolo* *Delle qual poi, per communprecetto* *Torrai li lati cubi insieme gionti* *Et cotal somma sara il tuo concetto.* *El terzo poi de questi nostri conti* *Se solue col secondo se ben guardi* *Che per natura son quasi congionti.* *Questi trouai, & non con paβi tardi* *Nel mille cinquecentè, quatroe trenta* *Con fondamenti ben sald'è gagliardi* *Nella citta dal mar'intorno centa.*	01) *When the cube with the cose beside it <x^3+px >* 02) *Equates itself to some other whole number, <=q>* 03) *Find two other, of which it is the difference. <u–v=q>* 04) *Hereafter you will consider this customarily* 05) *That their product always will be equal <uv=>* 06) *To the third of the cube of the cose net.* 3/3, instead of (*p*/3)^3> 07) *Its general remainder then* 08) *Of their cube sides , well subtracted,* <u−−√3−v√3u3−v3> 09) *Will be the value of your principal unknown. <=x>* 10) *In the second of these acts,* 11) *When the cube remains solo , <$x^3=px+q$>* 12) *You will observe these other arrangements:* 13) *Of the number <q> you will quickly make two such parts, <q=u+v>* 14) *That the one times the other will produce straightforward <uv=>* 15) *The third of the cube of the cose in a multitude,* <p^3/3, instead of (*p*/3)^3> 16) *Of which then, per common precept,* 17) *You will take the cube sides joined together.* <u−−√3+v√3u3+v3 18) *And this sum will be your concept. <=x>* 19) *The third then of these our calculations <$x^3+q=px$>* 20) *Solves itself with the second, if you look well after,* 21) *That by nature they are quasi conjoined.* 22) *I found these, & not with slow steps,* 23) *In thousand five hundred, four and thirty* 24) *With very firm and strong foundations* 25) *In the city girded around by the sea.*

2.15 COMPLEX ROOTS OF THE POLYNOMIALS BY UTILISING SYNTHETIC DIVISION PROCESS

Horners method utilises synthetic division process to solve roots of polynomials. Synthetic division process use recursive process to obtain a division for a polynomial. If

$P(x) = f_n(x) = a_0 + a_1x + a_1x^2 + \cdots + a_nx^n$ (2.15.1)

and if this polynomial divides with $(x - x_0)$ to form polynomial

$Q(x) = f_{n-1}(x) = b_1 + b_2x + b_3x^2 + \cdots + b_{n-1}x^{n-1}$ (2.15.2)

Where $P(x) = f_n(x) = b_0 + (x - x_0)f_{n-1}(x)$ or $P(x) = b_0 + (x - x_0)Q(x)$ (2.15.3)

$R = b_0$ is the remainder of the synthetic division process. Of course if the remainder is equal to zero x_0 would be the root of this polynomial. Synthetic division can be calculated as following iterative equation:

$b_n = a_n$ (2.15.4)

$b_i = a_i + b_{i+1}x_0$ (2.15.5)

For example if a polynomial

$f_n(x) = 2x^4 + 0x^3 - 3x^2 + 3x - 4$ is given, and x_0 is -2 {divide by (x+2)} synthetic division is applied as

x_0	a_4	a_3	a_2	a_1	a_0
-2	**2**	**0**	**-3**	**3**	**-4**
	b_4	b_3	b_2	b_1	b_0
	2	**0+2*(-2)=4**	**5**	**-7**	**10**

$Q(x) = f_{n-1}(x) = 2x^3 - 4x^2 + 5x - 7$

$P(x) = f_n(x) = 10 + (x + 2)Q(x)$

Differentiating of $f_n(x)$with respect to x gives

$f'_n(x) = f_{n-1}(x) + (x - x_0)f'_{n-1}(x)$ (2.15.6)

and at $x=x_0$

$f_n(x) = b_0$

$f'_n(x) = f_{n-1}(x) = Q(x)$ (2.15.7)

Therefore Newton - Raphson iterative method can be applied to find the roots by using iterative process

$x_n = x_{n-1} - \frac{f_n(x_{n-1})}{f'_n(x_{n-1})} = x_{n-1} - \frac{b_0}{f'_n(x_{n-1})}$ (2.15.8)

$x_n = x_{n-1} - \frac{b_0}{Q(x_{n-1})}$ (2.15.9)

xo	a4	a3	a2	a1	a0
-2	2	0	-3	3	-4
	2	-4	5	-7	10

Q	-49
x1	-1.7959184

x1	a4	a3	a2	a1	a0
-1.7959184	2	0	-3	3	-4
	2	-3.5918367	3.4506456	-3.1970777	1.7416906

Q	-32.563821
x2	-1.7424329

x2	a4	a3	a2	a1	a0
-1.7424329	2	0	-3	3	-4
	2	-3.4848658	3.0721449	-2.3530065	0.0999559

Q	-28.866623
x3	-1.7389702

x3	a4	a3	a2	a1	a0
-1.7389702	2	0	-3	3	-4
	2	-3.4779405	3.048035	-2.3004421	0.0004003

Q	-28.63559

x4	-1.7389563

x4	a4	a3	a2	a1	a0
-1.7389563	2	0	-3	3	-4
	2	-3.4779125	3.0479377	-2.3002304	6.505E-09

Q	-28.634659
x5	-1.7389563

x5	a4	a3	a2	a1	a0
-1.7389563	2	0	-3	3	-4
	2	-3.4779125	3.0479377	-2.3002304	0

Q	-28.634659
x5	-1.7389563

Note that roots of a polynomial can be complex numbers, therefore it is better to calculate the roots as complex numbers. It should be also note that after reducing polynomial to a third, second or first degree remaining roots can be calculated by using standard root finding equations. An example program for Horner's method is listed below (complex class in java language was defined previously) :

Program 2.15-1 Complex Horner method example to find the roots of a polynomial horner.java

```
 import javax.swing.*;
import java.util.*;
public class horner
{
c CA[];//complex polynomial coefficient
public static c f_poly(c c1[],c x)
{
// this function calculates the value of
// c least square curve fitting function
int n=c1.length;
c ff;
if(n!=0.0)
  { ff=new c(c1[n-1]);
  for(int i=n-2;i>=0;i--)
   { ff=c.p(c.p(ff,"*",x),"+",c1[i]); }
  }
else
  ff=new c(0,0);
return ff;
}
public double[] toDouble()
{int n=CA.length;
 double d[]=new double[n];
 for(int i=0;i<n;i++)
    {   d[i]=c.toDouble(CA[i]);}
    return d;}
public  c horner(double a[], c x)
{   int n=a.length;
                c c1[]=new c[n];
    for(int i=0;i<n;i++) c1[i]=new c(a[i]);
    return horner(c1,x);
}
public  c horner(c c3[], c x)
{   int n=c3.length;
    int iter=0;
    int nmax=200;
    boolean b=true;
    int j=0;
```

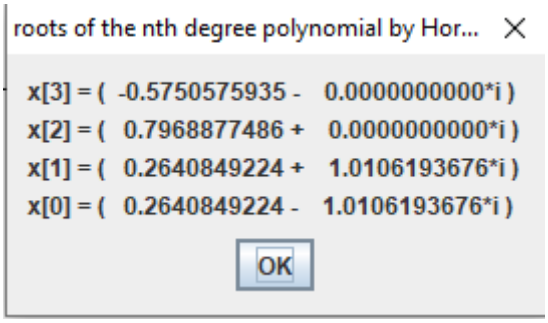
```
            c c1[]=new c[n];
            int n1=n-1;
            c cp[]=new c[n1];
            c1[n-1]=new c(c3[n-1]);
   for(int i=n-2;i>=0;i--)
   {   c c2=new c(c3[i]);
               c1[i]=c.p(c2,"+",c.p(x,"*",c1[i+1]));
   }
            c P=c1[0];
            for(int i=n1-1;i>=0;i--)
            {cp[i]=c1[i+1];}
            CA=cp;
            c Q=f_poly(cp,x);
            //Newton's method
            x=c.p(x,"-",c.p(P,"/",Q));//x=x-P/Q
            iter++;
            if(c.abs(P)<1.0e-25 || c.abs(P)==0 ||iter>nmax)
            {return x;}
            else
            {
            while(b)
            {
            c1[n-1]=new c(c3[n-1]);
   for(int i=n-2;i>=0;i--)
   {   c c2=new c(c3[i]);
               c1[i]=c.p(c2,"+",c.p(x,"*",c1[i+1]));
            }
            P=c1[0];
            for(int i=n1-1;i>=0;i--)
            {cp[i]=c1[i+1];}
            CA=cp;
   Q=f_poly(cp,x);
            if(c.abs(P)<1.0e-25||iter>nmax)
            {b=false;
             return x;
            }
            //Newton's method
            x=c.p(x,"-",c.p(P,"/",Q));
            iter++;
  }
}
  return x;
}

public String toString()
{int n=CA.length;
 String s="";
 for(int i=n-1;i>=0;i--) {s+=CA[i]+" ";}
 return s;
}
public static String toString(c x[])
{int n1=x.length;
 String s="";
 for(int i=n1-1;i>=0;i--) {s+="x["+i+"] = "+x[i]+"\n";}
 return s;
}
public c[] root(double a[])
{   int n=a.length;
   c y[]=new c[n-1];
            c x=new c(1.0,1.0);
            c x1=horner(a, x);
            int i=n-2;
   y[i]=x1;
   i--;
```

```
    int nn=CA.length;
    while(i>=0)
    {  if(nn!=CA.length)
       {c D[]=new c[nn];
       for(int j=0;j<nn;j++) {D[j]=CA[j];}
       CA=D;
       }
     {c z=horner(CA,x);
      y[i]=z;
     i--;
     nn--;
     }
     }
    return y;
}
     public static double[] enterdata(String s)
     {
     String s1=JOptionPane.showInputDialog(s);
     StringTokenizer token=new StringTokenizer(s1);
     int n=token.countTokens()-1;
     int m=n+1;
     double a[]=new double[m];
     int j=0;
     while(token.hasMoreTokens())
     {
     Double ax=new Double(token.nextToken());
     a[j++]=ax.doubleValue();
     }
     return a;
     }
public static void calculate_Horner()
{
                String s="enter coefficients of the n degree polynomial a[0] a[1]..a[n] y=a[0]+a[1]*x+a[2]*x^2+a[3]*x^3+..+a[n]*x^n ";
    double [] x0=enterdata(s);
    c x=new c(1.0,1.0);
    horner h=new horner();
    int n=x0.length;
    String s1[]=new String[n];
    String s2=" coefficients of the n degree polynomial y=a[0]+a[1]*x+a[2]*x^2+a[3]*x^3+..+a[n]*x^n ";
    for(int i=0;i<n;i++)
    {s1[i]="a["+i+"]";}
    JOptionPane.showMessageDialog(null,toString(h.root(x0)),
    "roots of the nth degree polynomial by Horner's synthetic division & Newton-Raphson method : ",JOptionPane.PLAIN_MESSAGE);

}
public static void main(String arg[])
{   calculate_Horner();}
}
```

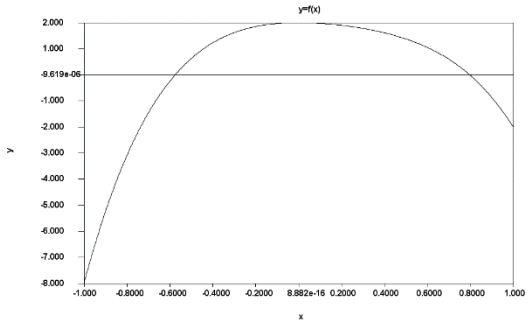
$$f(x) = -4x^4 + 3x^3 - 3x^2 + 2$$




```
 public class gr1
{
```

```
public static void main(String arg[])
{double a[]={2,0,-3,3,-4};
 if_poly f=new if_poly(a);
 f.plot(-1,1);
 System.out.println(f.print_roots());
}}
```

```
 ---------- Capture Output ----------
> "E:\co\java\bin\java.exe" gr1
 coefficients of the n degree polynomial y=a[0]+a[1]*x+a[2]*x^2+a[3]*x^3+..+a[n]*x^n
x[0](   0.2640849224 -    1.0106193676*i )
x[1](   0.2640849224 +    1.0106193676*i )
x[2](   0.7968877486 +    0.0000000000*i )
x[3](  -0.5750575935 -    0.0000000000*i )
```



Horner Method in python language:

Program 2.15-2 Complex Horner method example to find the roots of a polynomial (python language)

```
from math import *

def f_poly(c1,x):
    # this function calculates the value of
    # c least square curve fitting function
    n=len(c1);
    if n!=0.0:
      ff=c1[n-1];
      for i in range(n-2,-1,-1):
        ff=ff*x+c1[i];
    else:
      ff=0,0;
    return ff;

def horner(a):
  # roots of a polynomial by using synthetic division process
    n=len(a);
    x1=complex(1.56435,0.1);
    n1=n;
    nmax=150;
    tolerance=1.0e-10;
    n1=n+1;
    ref=3.1;
    y=[complex(0.0,0.0) for i in range(n1-2)];
    n2=n1-2;
    for k in range(0,n2):
          n=len(a);
          x=x1;
          xold=3.4878;
          ref=abs(xold-x);
          b=[complex(0.0,0) for i in range(n)];
          c=[complex(0.0,0.0) for i in range(n-1)];
          b[n-1]=a[n-1];
```

```
            ii=0;
            while ref>tolerance and ii<nmax:
              for i in range(n-2,-1,-1):
                 b[i]=a[i]+b[i+1]*x;
              for i in range(n-2,-1,-1):  c[i]=b[i+1];
              Q=f_poly(c,x);
              P=b[0];
              x=x-P/Q;
              ref=abs(xold-x);
              ii=ii+1;
              xold=x;
            a=c;
            n=len(a);
            y[k]=x;
     return y;

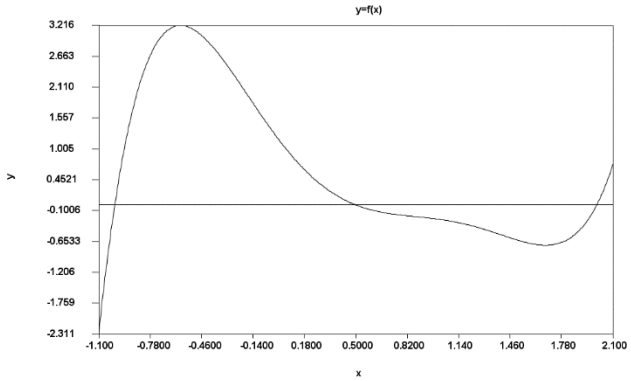
a=[-4,3,-3,0,2];
print(horner(a));
```

```
runfile('D:/okul/SCO1/Mathd.py', wdir='D:/okul/SCO1')
[(1.254881884834291+1.34106353887572e-29j),
 (0.24203718580880437-0.9262454872675325j),
(0.24203718584251482+0.9262454872691118j),
 (-1.738956256536873-9.031220216115798e-12j)]
```

$$f(x) = x^5 - 3.5x^4 + 2.75x^3 + 2.125x^2 - 3.875x + 1.25$$

```
public class gr1
{
public static void main(String arg[])
{double a[]={1.25,-3.875,2.125,2.75,-3.5,1};
 if_poly f=new if_poly(a);
 f.plot(-1.1,2.1);
 System.out.println(f.print_roots());
}}
```

```
---------- Capture Output ----------
> "E:\co\java\bin\java.exe" gr1
 coefficients of the n degree polynomial y=a[0]+a[1]*x+a[2]*x^2+a[3]*x^3+..+a[n]*x^n
x[0](  -1.0000000000 +    0.0000000000*i )
x[1](   2.0000000000 +    0.0000000000*i )
x[2](   1.0000000000 -    0.5000000000*i )
x[3](   0.5000000000 -    0.0000000000*i )
x[4](   1.0000000000 +    0.5000000000*i )
```



Same function results in python code:

```
a=[1.25,-3.875,2.125,2.75,-3.5,1];
print(horner(a));
```

```
runfile('D:/okul/SCO1/horner1.py', wdir='D:/okul/SCO1')
[(1.0000000000000002-0.5j), (0.5000000000000011+1.0217019574503208e-15j), (1.9999999999895546-4.1185274253612687e-13j),
(1.0000000000176792+0.49999999997300376j), (-0.9999999999955536-3.5171310308612647e-12j)]
```

Now let us find roots of $f(x) = x^3 + 9813.18x^2 + 8571.08x + 0.781736 = 0$ by using horner method

```
public static void main(String arg[])
{   double c1[]={0.781736,8571.08,9813.18,1};
   horner h=new horner();
            System.out.println(h.toString(h.root(c1)));
}
```

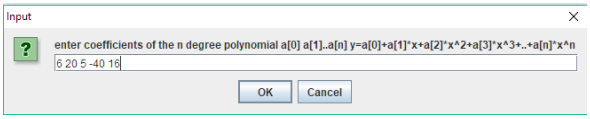
```
---------- Capture Output ----------
> "C:\java\bin\java.exe" horner
x[2] = (-9.121576846347157E-5)
x[1] = (-0.8734118615166153)
x[0] = (-9812.306496922716)
```

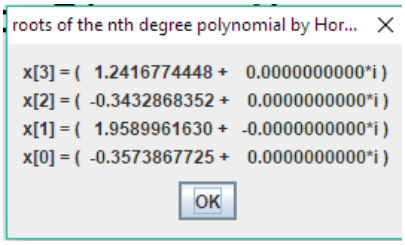
Same function results in python code:

```
a=[0.781736,8571.08,9813.18,1];
print(horner(a));
```

```
runfile('D:/okul/SCO1/horner1.py', wdir='D:/okul/SCO1')
[(-9.121576846347157e-05+4.9325894681659426e-26j), (-0.8734118615171431-4.424838606974589e-13j), (-9812.306496922753-
6.1173554997356664e-12j)]
```

Now let us find roots of $f(x) = 16x^4 - 40x^3 + 5x^2 + 20x + 6 = 0$ by using horner method






```
a=[6,20,5,-40,16];
print(Mathd.horner(a));
```

```
runfile('D:/okul/SCO1/Mathd.py', wdir='D:/okul/SCO1')
[(1.2416774447647838+0j), (1.9704460787298799-4.672866064934229e-20j), (-0.35606176190945776+0.16275838255939531j), (-
0.3560617615461311-0.16275838258481318j)]
```

Le Guerre method is also specificaly developed to find roots of a polynomial. Method formulations is as follows:

$$g(x) = [f'(x)]^2 - \frac{nf(x)f''(x)}{n-1} \qquad (2.15.10)$$

$$x_{k+1} = x_k - \frac{nf(x_k)}{f'(x_k) \pm (n-1)\sqrt{g(x_k)}} \qquad (2.15.11)$$

In this definition, n is the degree of polynomial. The term $\sqrt{g(x_k)}$ in the equation can be complex (roots of polynomials can be complex numbers). Therefore the method will be written as complex variable method. By combining LeGuerre root finding method with synthetic division process, the remaining roots can also be found.

Program 2.15-2 Complex LeGuerre with synthetic division example to find the roots of a polynomial

```
 //
//
// compleks Lequerre method to find roots of the polynomials
//
```

```
import java.util.*;
import java.awt.*;
import java.applet.Applet;
import java.awt.event.*;
import javax.swing.*;

public class Leguerre
{ public static c func(double ai[],c x)
  {   int n=ai.length;
                c a[]=new c[n];
                for(int i=0;i<n;i++) a[i]=new c(ai[i]);
                return func(a,x);
  }
  public static c func(c a[],c x)
  {  // c polynomial
  int n=a.length;
  c y=new c(a[n-1]);
  for(int i=n-2;i>=0;i--)
  {y=c.p(c.p(y,"*",x),"+",a[i]);}
  return y;
  }
  public static c dfunc(double ai[],c x)
  {   int n=ai.length;
                c a[]=new c[n];
                for(int i=0;i<n;i++) a[i]=new c(ai[i]);
                return dfunc(a,x);
  }
  static c dfunc(c a[],c x)
  {//c numeric derivative of a polynomial
   double hh=0.001;
   c h=new c(hh,0.0);
   c he=new c(-hh,0.0);
   c h2=new c(2.0*hh,0.0);
   c he2=new c(-2.0*hh,0.0);
   c xah=c.p(x,"+",h);
   c xa2h=c.p(x,"+",h2);
   c xeh=c.p(x,"+",he);
   c xe2h=c.p(x,"+",he2);
   c fe2h=c.p(func(a,xe2h),"*",(1.0/12.0));
   c feh=c.p(func(a,xeh),"*",(-8.0/12.0));
   c fah=c.p(func(a,xah),"*",(8.0/12.0));
   c fa2h=c.p(func(a,xa2h),"*",(-1.0/12.0));
   c df=c.p(fe2h,"+",c.p(feh,"+",c.p(fah,"+",fa2h)));
   df=c.p(df,"/",h);
   return df;
  }
  public static c d2func(double ai[],c x)
  {   int n=ai.length;
                c a[]=new c[n];
                for(int i=0;i<n;i++) a[i]=new c(ai[i]);
                return d2func(a,x);
  }
  static c d2func(c a[],c x)
  {//c second numeric derivative of a polynomial
   double hh=0.001;
   c h=new c(hh,hh);
   c hkare=c.pow(h,2);
   c he=new c(-hh,-hh);
   c h2=new c(2.0*hh,2.0*hh);
   c he2=new c(-2.0*hh,-2.0*hh);
   c xah=c.p(x,"+",h);
   c xa2h=c.p(x,"+",h2);
   c xeh=c.p(x,"+",he);
   c xe2h=c.p(x,"+",he2);
   c fe2h=c.p(func(a,xe2h),"*",(-1.0/12.0));
   c feh=c.p(func(a,xeh),"*",(4.0/3.0));
   c ff=c.p(func(a,x),"*",(-5.0/2.0));
   c fah=c.p(func(a,xah),"*",(4.0/3.0));
   c fa2h=c.p(func(a,xa2h),"*",(-1.0/12.0));
   c d2f=c.p(fe2h,"+",c.p(feh,"+",c.p(ff,"+",c.p(fah,"+",fa2h))));
   d2f=c.p(d2f,"/",hkare);
   return d2f;
  }
```

```
public static c Leguerre(double ai[],c x)
{int n=ai.length;
                c a[]=new c[n];
                for(int i=0;i<n;i++) a[i]=new c(ai[i]);
                return Leguerre(a,x);
}

public static c Leguerre(c a[],c x)
{
int n=a.length-1;
double eps=1.0e-15;
int m=50;
int k=0;
c f0=func(a,x);
c f1=new c();
c f2=new c();
c g=new c();
c d1=new c();
c d2=new c();
c x0=x;
c x1=c.p(a[0],"/",a[1]);
x1=c.p(x1,"*",-1.0);
do{
  f1=dfunc(a,x0);
  f2=d2func(a,x0);
  g=c.p(c.p(f1,"*",f1),"+",c.p(c.p(f0,"*",f2),"*",(-(double)n/(double)(n-1))));
  d1=c.p(f1,"+",c.p(c.pow(g,0.5),"*",(double)(n-1)));
  d2=c.p(f1,"+",c.p(c.pow(g,0.5),"*",(double)(1-n)));
  if(c.abs(d1)>=c.abs(d2)) x1=c.p(x0,"+",c.p(c.p(f0,"/",d1),"*",(double)(-n)));
  else x1=c.p(x0,"+",c.p(c.p(f0,"/",d2),"*",(double)(-n)));
  x0=c.p(x1,"+",0.0);
  f0=func(a,x0);
  k++;
}while(c.abs(f0)>eps && k<m);
if(c.abs(f0)<eps) { return x1;}
JOptionPane.showMessageDialog(null,"Warning : Maximum number of iteration is exceeded \n"+
" result may not be valid","MAXIMUM ITERATION NUMBER WARNING",JOptionPane.WARNING_MESSAGE);
return x1;
}

public static c[] synthetic_division(double ci[],c x)
{
  int n=ci.length;
  c c[]=new c[n];
  for(int i=0;i<n;i++) c[i]=new c(ci[i]);
  return synthetic_division(c,x);
}
public static c[] synthetic_division(c c4[],c x)
{ int n=c4.length;
  c c1[]=new c[n];
  c1[n-1]=new c(c4[n-1]);
    for(int i=n-2;i>=0;i--)
    {   c c2=new c(c4[i]);
                  c1[i]=c.p(c2,"+",c.p(x,"*",c1[i+1]));
                }
  c c3[]=new c[n-1];
  for(int i=n-1;i>=1;i--)
{   c c2=new c(c1[i]);
              c3[i-1]=c1[i];
              }
  return c3;
}

public static c[] quadratic(c C[])
{ c z[]=new c[2];
     c z1=new c(C[0]);
    c ac4=c.p(z1,"*",C[2]);
    ac4=c.p(ac4,"*",4.0);
    z1=new c(C[1]);
    c b2=c.p(z1,"*",z1);
    c DD=c.sqrt(c.p(b2,"-",ac4));
    z1=new c(C[1]);
    z[0]=c.p(DD,"-",z1);
    z[0]=c.p(z[0],"/",2.0);
```

```
    z[0]=c.p(z[0],"/",C[2]);
    z[1]=c.p(DD,"+",C[1]);
    z[1]=c.p(z[1],"/",2.0);
    z[1]=c.p(z[1],"/",C[2]);
    z[1]=c.p(z[1],"*",-1.0);
    return z;
}
public static c[] root(double C1[])
{ int n=C1.length;
c C[]=new c[n];
for(int i=0;i<n;i++)
{C[i]=new c(C1[i]);}
return root(C);
}

public static c[] root(c C[])
{   int n=C.length;
   c y[]=new c[n-1];
              c x0=new c(1.0,0.0);
              int i=n-2;
              int nn=C.length;
   while(i>=0)
   {
    if(nn>3)
    { c z=Leguerre(C,x0);
      y[i]=z;
      C=synthetic_division(C,z);
      nn=C.length;
      i--;
    }
    else if(nn==3)
    { c z[]=quadratic(C);
      y[i]=z[0];
                y[i-1]=z[1];
      i-=2;
      nn-=2;
    }
    else if(nn==2)
    {c z;
    c z1=new c(C[0]);
              z=c.p(z1,"*",-1.0);
              z=c.p(z,"/",C[1]);
              y[i]=z;
              i-=2;
    nn-=2;
              }
    }
   return y;
}
public static String toString(c x[])
{int n1=x.length;
 String s="";
 for(int i=n1-1;i>=0;i--) {s+="x["+i+"] = "+x[i]+"\n";}
 return s;
}

    public static double[] input(String s)
    {
    String s1=JOptionPane.showInputDialog(s);
    StringTokenizer token=new StringTokenizer(s1);
    int n=token.countTokens()-1;
    int m=n+1;
    double a[]=new double[m];
    int j=0;
    while(token.hasMoreTokens())
    {
    Double ax=new Double(token.nextToken());
    a[j++]=ax.doubleValue();
    }
    return a;
    }

    public static void main (String args[])
    { double c[]={-4,3,-3,0,2};
```

```
      c y[]=root(c);
      System.out.println(toString(y));
    }
}
```

```
---------- Capture Output ----------
> "C:\java\bin\java.exe" Leguerre
x[3] = (   1.2548818848 +    0.0000000000i )
x[2] = (   0.2420371858 -    0.9262454873i )
x[1] = (   0.2420371858 +    0.9262454873i )
x[0] = (  -1.7389562565)
> Terminated with exit code 0.
```

Of course method can be used to find roots of any general function as well, the next program will utilise LeGuerre method for general function with lambda expressions and general interface defination icf_cx defined previously.

Program 2.15-4 Complex Lequerre method example NA13E1 general complex function defined with lambda expressions

```
 //
//  compleks Lequerre method to find roots of the polynomials
//  Lambda expression version for a general function

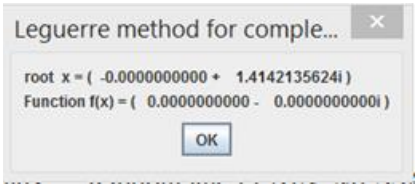
import java.util.*;
import java.awt.*;
import java.applet.Applet;
import java.awt.event.*;
import javax.swing.*;

public class NA13E1
{
public static c LeguerreC(icf_cx f,c x)
{
int n=10; //degree of polynomial
double eps=1.0e-15;
int m=50;
int k=0;
c f0=f.func(x);
c f1=new c();
c f2=new c();
c g=new c();
c d1=new c();
c d2=new c();
c x0=x;
c x1=new c();
do{
  f1=f.dfunc(x0);
  f2=f.d2func(x0);
  g=c.p(c.p(f1,"*",f1),"+",c.p(c.p(f0,"*",f2),"*",(-(double)n/(double)(n-1))));
  d1=c.p(f1,"+",c.p(c.pow(g,0.5),"*",(double)(n-1)));
  d2=c.p(f1,"+",c.p(c.pow(g,0.5),"*",(double)(1-n)));
if(c.abs(d1)>=c.abs(d2)) x1=c.p(x0,"+",c.p(c.p(f0,"/",d1),"*",(double)(-n)));
else                     x1=c.p(x0,"+",c.p(c.p(f0,"/",d2),"*",(double)(-n)));
  x0=c.p(x1,"+",0.0);
  f0=f.func(x0);
  k++;
}while(c.abs(f0)>eps && k<m);

if(c.abs(f0)<eps) { return x1;}
JOptionPane.showMessageDialog(null,"Warning : Maximum number of iteration is exceeded \n"+
   " result may not be valid","MAXIMUM ITERATION NUMBER WARNING",JOptionPane.WARNING_MESSAGE);
return x1;
}
```

```
    public static double[] verigir(String s)
    {
    String s1=JOptionPane.showInputDialog(s);
    StringTokenizer token=new StringTokenizer(s1);
    int n=token.countTokens()-1;
    int m=n+1;
    double a[]=new double[m];
    int j=0;
    while(token.hasMoreTokens())
    {
    Double ax=new Double(token.nextToken());
    a[j++]=ax.doubleValue();
    }
    return a;
    }

    public static void main (String args[])
    {
    double x0[]=verigir(" first guess for the root give c number with a space in between 1.0 0.5 ");
    c x1=new c(x0[0],x0[1]);
    icf_cx f=(c x)->c.p(c.p(x,"*",x),"+",2.0);
    c r=LeguerreC(f,x1);
    String s="root  x = "+r.toString()+"\n"+"Function f(x) = "+f.func(r).toString();
    JOptionPane.showMessageDialog(null,s,
    "Leguerre method for c root of polynomials : ",JOptionPane.PLAIN_MESSAGE);
    System.exit(0);
    }
}
```



Muller's method was also previously investigated as a general real function root finding method. In Horners's method the basic method used to find the roots was Newton's method. In Muller's method a quadratic polynomial root finding is actually used to determine the roots. As Horner's method, synthetic division process can also be utilised to find the remaing roots also.

Program 2.15-5 Complex Muller's method example to find the roots of a polynomial muller.java

```
 import javax.swing.*;
public class muller
{ //This version of the program is written for
  //polynomials only
c C[];
public static c f_poly(c c1[],c x)
{
// this function calculates the value of
// c least square curve fitting function
int n=c1.length;
c ff;
if(n!=0.0)
  { ff=new c(c1[n-1]);
  for(int i=n-2;i>=0;i--)
   { ff=c.p(c.p(ff,"*",x),"+",c1[i]); }
  }
else
  ff=new c(0,0);
```

```
return ff;
}
public static c muller(c c1[],double p0i,double p1i,double p2i)
{
  c p0=new c(p0i);
  c p1=new c(p1i);
  c p2=new c(p2i);
  return muller(c1,p0,p1,p2);
}
public static c muller(double ci[],double p0i,double p1i,double p2i)
{ int n=ci.length;
  c c1[]=new c[n];
  for(int i=0;i<n;i++) c1[i]=new c(ci[i]);
  c p0=new c(p0i);
  c p1=new c(p1i);
  c p2=new c(p2i);
  return muller(c1,p0,p1,p2);
}

public static c muller(c Ci[],c p0,c p1,c p2)
{
int nmax=300;
double tolerance=1.0e-20;
c fp0,fp1,fp2,fp3;
c p3=new c(0.0);
c a,b,ca;
c b1,b2,b3,b4,b5,b6,b7,b8,b9,b10,b11,b12;
c a1,a2,a3;
c c1,c2,c3,c4,c5,c6,c7,c8,c9,c10,c11;
fp0=f_poly(Ci,p0);
fp1=f_poly(Ci,p1);
fp2=f_poly(Ci,p2);
for(int i=0;i<nmax;i++)
{               ca=fp2;
   b1=c.p(p0,"-",p2);
   b2=c.p(p1,"-",p2);
   b3=c.p(p0,"-",p1);
   b4=c.p(b1,"*",b1);
   b5=c.p(b2,"*",b2);
   b6=c.p(fp1,"-",fp2);
   b7=c.p(fp0,"-",fp2);
   b8=c.p(b1,"*",b2);
   b8=c.p(b8,"*",b3);
   b9=c.p(b8,"*",b3);
   b10=c.p(b4,"*",b6);
   b11=c.p(b5,"*",b7);
   b12=c.p(b10,"-",b11);
   b=c.p(b12,"/",b8);
   a1=c.p(b2,"*",b7);
   a2=c.p(b1,"*",b6);
   a3=c.p(a1,"-",a2);
   a=c.p(a3,"/",b8);
   c1=c.p(ca,"*",-2.0);
   c2=c.p(b,"*",b);
   c3=c.p(a,"*",ca);
   c4=c.p(-4.0,"*",c3);
   c5=c.p(c2,"-",c4);
   c6=c.sqrt(c5);
   c7=c.p(c.sgn(b),"*",c6);
   c8=c.p(b,"+",c7);
   c9=c.p(c1,"/",c8);
   p3=c.p(p2,"+",c9);
   fp3=f_poly(Ci,p3);
   if(c.abs(fp3)<tolerance) {return p3;}
```

```
    else{p0=p1;p1=p2;p2=p3;fp0=fp1;fp1=fp2;fp2=fp3;}
}
JOptionPane.showMessageDialog(null,"Maximum number of iteration is exceeded \n"+
   " results may not be valid","MAXIMUM ITERATION WARNING",JOptionPane.WARNING_MESSAGE);
return p3;
}

public double[] toDouble()
{int n=C.length;
 double d[]=new double[n];
 for(int i=0;i<n;i++)
    {   d[i]=c.toDouble(C[i]);}
    return d;}


public static c[] synthetic_division(double ci[],c x)
{
  int n=ci.length;
  c c1[]=new c[n];
  for(int i=0;i<n;i++) c1[i]=new c(ci[i]);
  return synthetic_division(c1,x);
}
public static c[] synthetic_division(c ci[],c x)
{ int n=ci.length;
  c c1[]=new c[n];
  c1[n-1]=new c(ci[n-1]);
    for(int i=n-2;i>=0;i--)
    {   c c2=new c(ci[i]);
                  c1[i]=c.p(c2,"+",c.p(x,"*",c1[i+1]));
               }
  c c3[]=new c[n-1];
  for(int i=n-1;i>=1;i--)
{   c c2=new c(c1[i]);
               c3[i-1]=c1[i];
               }
  return c3;
}

public static c[] quadratic(c Ci[])
{ c z[]=new c[2];
      c z1=new c(Ci[0]);
     c ac4=c.p(z1,"*",Ci[2]);
     ac4=c.p(ac4,"*",4.0);
     z1=new c(Ci[1]);
     c b2=c.p(z1,"*",z1);
     c DD=c.sqrt(c.substract(b2,ac4));
     z1=new c(Ci[1]);
     z[0]=c.p(DD,"-",z1);
     z[0]=c.p(z[0],"/",2.0);
     z[0]=c.p(z[0],"/",Ci[2]);
     z[1]=c.p(DD,"+",Ci[1]);
     z[1]=c.p(z[1],"/",2.0);
     z[1]=c.p(z[1],"/",Ci[2]);
     z[1]=c.p(z[1],"*",-1.0);
     return z;
}
public static c[] root(double C1[])
{ int n=C1.length;
c Ci[]=new c[n];
for(int i=0;i<n;i++)
{Ci[i]=new c(C1[i]);}
return root(Ci);
}
```

```
public static c[] root(c Ci[])
{   int n=Ci.length;
   c y[]=new c[n-1];
              c x0=new c(0.0,0.0);
              c x1=new c(1.0,0.0);
              c x2=new c(2.0,0.0);
              int i=n-2;
              int nn=Ci.length;
   while(i>=0)
   {
    if(nn>3)
    { c z=muller(Ci,x0,x1,x2);
      y[i]=z;
      Ci=synthetic_division(Ci,z);
      nn=Ci.length;
      i--;
    }
    else if(nn==3)
    { c z[]=quadratic(Ci);
      y[i]=z[0];
                 y[i-1]=z[1];
      i-=2;
      nn-=2;
    }
    else if(nn==2)
    {c z;
    c z1=new c(Ci[0]);
              z=c.p(z1,"*",-1.0);
              z=c.p(z,"/",Ci[1]);
              y[i]=z;
              i-=2;
    nn-=2;
              }
    }
   return y;
}
public static String toString(c x[])
{int n1=x.length;
 String s="";
 for(int i=n1-1;i>=0;i--) {s+="x["+i+"] = "+x[i]+"\n";}
 return s;
}
public static void main(String arg[])
{   double c1[]={-4,3,-3,0,2};
   c y[]=root(c1);
   System.out.println(toString(y));
}}
```

```
---------- Capture Output ----------
> "C:\java\bin\java.exe" muller
x[3] = (   1.2548818848)
x[2] = (  -1.7389562565 +    0.0000000000i )
x[1] = (   0.2420371858 +    0.9262454873i )
x[0] = (   0.2420371858 -    0.9262454873i )
> Terminated with exit code 0.
```

Inverse quadratic Lagrange interpolation formula is a quadratic formula relatively similar to Mullers formula. The detailed formulations are previously given for general non-linear function root solving. In this subchapter the method will be utilised to find roots of polynomial.

$$x = \frac{f(x_1)f(x_2)x_3}{[f(x_3)-f(x_1)][f(x_3)-f(x_2)]} + \frac{f(x_2)f(x_3)x_1}{[f(x_1)-f(x_2)][f(x_1)-f(x_3)]} + \frac{f(x_3)f(x_1)x_2}{[f(x_2)-f(x_3)][f(x_2)-f(x_1)]} \qquad (2.8.2)$$

Program 2.15-6 Complex Inverse Lagrange interpolation method example to find the

roots of a polynomial inverse_quadratic.java

```
 import javax.swing.*;
//Inverse quadratic interpolation formula for c polynomial
public class inverse_quadratic
{ //This version of the program is written for
  //polynomials only
c C[];
public static c f_poly(c c1[],c x)
{
// this function calculates the value of
// c least square curve fitting function
int n=c1.length;
c ff;
if(n!=0.0)
  { ff=new c(c1[n-1]);
  for(int i=n-2;i>=0;i--)
   { ff=c.p(c.p(ff,"*",x),"+",c1[i]); }
  }
else
  ff=new c(0,0);
return ff;
}
public static c inverse_quadratic(c c1[],double p0i,double p1i,double p2i)
{
  c p0=new c(p0i);
  c p1=new c(p1i);
  c p2=new c(p2i);
  return inverse_quadratic(c1,p0,p1,p2);
}
public static c inverse_quadratic(double ci[],double p0i,double p1i,double p2i)
{ int n=ci.length;
  c c1[]=new c[n];
  for(int i=0;i<n;i++) c1[i]=new c(ci[i]);
  c p0=new c(p0i);
  c p1=new c(p1i);
  c p2=new c(p2i);
  return inverse_quadratic(c1,p0,p1,p2);
}

public static c inverse_quadratic(c Ci[],c p0,c p1,c p2)
{
int nmax=300;
double tolerance=1.0e-20;
c fp0,fp1,fp2,fp3;
c R,S,T,P,Q;
c p3=new c(0.0);
c a,b,ca;
c b1,b2,b3,b4,b5,b6,b7,b8,b9,b10,b11,b12;
c a1,a2,a3;
c c1,c2,c3,c4,c5,c6,c7,c8,c9,c10,c11;
fp0=f_poly(Ci,p0);
fp1=f_poly(Ci,p1);
fp2=f_poly(Ci,p2);
for(int i=0;i<nmax;i++)
{               ca=fp2;
   R=c.p(fp1,"/",fp2);
   S=c.p(fp1,"/",fp0);
   T=c.p(fp0,"/",fp2);
   b1=c.p(R,"-",T);//R-T
   b2=c.p(p2,"-",p1);//x3-x2
   b3=c.p(R,"-",1.0);//R-1
   b4=c.p(p1,"-",p0);//x2-x1
   b5=c.p(b4,"*",b3);//(R-1)(x2-x1)
   b6=c.p(b1,"*",T);//T(R-T)
```

```
    b7=c.p(b6,"*",b2);//T(R-T)(x3-x2)
    b8=c.p(b7,"+",b5);//T(R-T)(x3-x2)+(R-1)(x2-x1)
    P=c.p(S,"*",b8);//P=S*[T(R-T)(x3-x2)+(R-1)(x2-x1)]
    b9=c.p(T,"-",1.0);
    b10=c.p(S,"-",1.0);;
    b11=c.p(b3,"*",b9);
    Q=c.p(b11,"*",b10);//Q=(T-1)(S-1)(R-1)
    p3=c.p(p1,"+",c.p(P,"/",Q));//x3=x1+P/Q
    fp3=f_poly(Ci,p3);
    if(c.abs(fp3)<tolerance) {return p3;}
    else{p0=p1;p1=p2;p2=p3;fp0=fp1;fp1=fp2;fp2=fp3;}
}
JOptionPane.showMessageDialog(null,"Maximum number of iteration is exceeded \n"+
   " results may not be valid","MAXIMUM ITERATION WARNING",JOptionPane.WARNING_MESSAGE);
return p3;
}
public double[] toDouble()
{int n=C.length;
 double d[]=new double[n];
 for(int i=0;i<n;i++)
    {   d[i]=c.toDouble(C[i]);}
    return d;}
public static c[] synthetic_division(double ci[],c x)
{
  int n=ci.length;
  c c1[]=new c[n];
  for(int i=0;i<n;i++) c1[i]=new c(ci[i]);
  return synthetic_division(c1,x);
}
public static c[] synthetic_division(c ci[],c x)
{ int n=ci.length;
  c c1[]=new c[n];
  c1[n-1]=new c(ci[n-1]);
    for(int i=n-2;i>=0;i--)
    {   c c2=new c(ci[i]);
                c1[i]=c.add(c2,c.multiply(x,c1[i+1]));
              }
  c c3[]=new c[n-1];
  for(int i=n-1;i>=1;i--)
{   c c2=new c(c1[i]);
              c3[i-1]=c1[i];
              }
  return c3;
}

public static c[] quadratic(c Ci[])
{ c z[]=new c[2];
     c z1=new c(Ci[0]);
    c ac4=c.multiply(z1,Ci[2]);
    ac4=c.multiply(ac4,4.0);
    z1=new c(Ci[1]);
    c b2=c.multiply(z1,z1);
    c DD=c.sqrt(c.substract(b2,ac4));
    z1=new c(Ci[1]);
    z[0]=c.substract(DD,z1);
    z[0]=c.divide(z[0],2.0);
    z[0]=c.divide(z[0],Ci[2]);
    z[1]=c.add(DD,Ci[1]);
    z[1]=c.divide(z[1],2.0);
    z[1]=c.divide(z[1],Ci[2]);
    z[1]=c.multiply(z[1],-1.0);
    return z;
}
public static c[] root(double C1[])
```

```
{ int n=C1.length;
c Ci[]=new c[n];
for(int i=0;i<n;i++)
{Ci[i]=new c(C1[i]);}
return root(Ci);
}

public static c[] root(c Ci[])
{   int n=Ci.length;
   c y[]=new c[n-1];
              c x0=new c(0.5,0.0);
              c x1=new c(1.0,0.0);
              c x2=new c(2.0,0.0);
              int i=n-2;
              int nn=Ci.length;
   while(i>=0)
   {
    if(nn>3)
    { c z=inverse_quadratic(Ci,x0,x1,x2);
      y[i]=z;
      Ci=synthetic_division(Ci,z);
      nn=Ci.length;
      i--;
    }
    else if(nn==3)
    { c z[]=quadratic(Ci);
      y[i]=z[0];
                 y[i-1]=z[1];
      i-=2;
      nn-=2;
    }
    else if(nn==2)
    {c z;
    c z1=new c(Ci[0]);
              z=c.multiply(z1,-1.0);
              z=c.divide(z,Ci[1]);
              y[i]=z;
              i-=2;
    nn-=2;
              }
    }
   return y;
}
public static String toString(c x[])
{int n1=x.length;
 String s="";
 for(int i=n1-1;i>=0;i--) {s+="x["+i+"] = "+x[i]+"\n";}
 return s;
}
public static void main(String arg[])
{   double c1[]={-4,3,-3,0,2};
   c y[]=root(c1);
   System.out.println(toString(y));
}}
```

```
---------- Capture Output ----------
> "C:\java\bin\java.exe" inverse_quadratic
x[3] = (   1.2548818848)
x[2] = (  -1.7389562565)
x[1] = (   0.2420371858 -    0.9262454873i )
x[0] = (   0.2420371858 +    0.9262454873i )


> Terminated with exit code 0.
```

Halley Method of root finding is a second order Newton type method that previously investigated to find roots of real functions. A complex polynomial version of the method with sysnthetic division to find all the roots are given next

$$x_{n+1} = x_n - \frac{2f(x_n)f'(x_n)}{2f'(x_n)f'(x_n) - f(x_n)f''(x_n)} \qquad (2.11.2)$$

Program 2.15-7 Complex Halley's method example to find the roots of a polynomial

```
 // compleks Halley's method to find roots of the polynomials
// xn+1=xn-2(f(xn)*f(xn)*f'(xn))/(2f'(xn)*f'(xn)-f(xn)*f''(xn))
import java.util.*;
import java.awt.*;
import java.applet.Applet;
import java.awt.event.*;
import javax.swing.*;

public class Halley
{ public static c func(double ai[],c x)
  {   int n=ai.length;
                c a[]=new c[n];
                for(int i=0;i<n;i++) a[i]=new c(ai[i]);
                return func(a,x);
  }
  public static c func(c a[],c x)
  {  // c polynomial
  int n=a.length;
  c y=new c(a[n-1]);
  for(int i=n-2;i>=0;i--)
  {y=c.p(c.p(y,"*",x),"+",a[i]);}
  return y;
  }
  public static c dfunc(double ai[],c x)
  {   int n=ai.length;
                c a[]=new c[n];
                for(int i=0;i<n;i++) a[i]=new c(ai[i]);
                return dfunc(a,x);
  }
  static c dfunc(c a[],c x)
  {//c numeric derivative of a polynomial
   double hh=0.001;
   c h=new c(hh,0.0);
   c he=new c(-hh,0.0);
   c h2=new c(2.0*hh,0.0);
   c he2=new c(-2.0*hh,0.0);
   c xah=c.add(x,h);
   c xa2h=c.p(x,"+",h2);
   c xeh=c.p(x,"+",he);
   c xe2h=c.p(x,"+",he2);
   c fe2h=c.p(func(a,xe2h),"*",(1.0/12.0));
   c feh=c.p(func(a,xeh),"*",(-8.0/12.0));
   c fah=c.p(func(a,xah),"*",(8.0/12.0));
   c fa2h=c.p(func(a,xa2h),"*",(-1.0/12.0));
   c df=c.p(fe2h,"+",c.p(feh,"+",c.p(fah,"+",fa2h)));
   df=c.p(df,"/",h);
   return df;
  }
  public static c d2func(double ai[],c x)
  {   int n=ai.length;
                c a[]=new c[n];
                for(int i=0;i<n;i++) a[i]=new c(ai[i]);
                return d2func(a,x);
  }
  static c d2func(c a[],c x)
```

```
  {//c second numeric derivative of a polynomial
   double hh=0.001;
   c h=new c(hh,hh);
   c hkare=c.pow(h,2);
   c he=new c(-hh,-hh);
   c h2=new c(2.0*hh,2.0*hh);
   c he2=new c(-2.0*hh,-2.0*hh);
   c xah=c.p(x,"+",h);
   c xa2h=c.p(x,"+",h2);
   c xeh=c.p(x,"+",he);
   c xe2h=c.p(x,"+",he2);
   c fe2h=c.p(func(a,xe2h),"*",(-1.0/12.0));
   c feh=c.p(func(a,xeh),"*",(4.0/3.0));
   c ff=c.p(func(a,x),"*",(-5.0/2.0));
   c fah=c.p(func(a,xah),"*",(4.0/3.0));
   c fa2h=c.p(func(a,xa2h),"*",(-1.0/12.0));
   c d2f=c.p(fe2h,"+",c.p(feh,"+",c.p(ff,"+",c.p(fah,"+",fa2h))));
   d2f=c.p(d2f,"/",hkare);
   return d2f;
  }

public static c Halley(double ai[],c x)
{int n=ai.length;
                c a[]=new c[n];
                for(int i=0;i<n;i++) a[i]=new c(ai[i]);
                return Halley(a,x);
}

public static c Halley(c a[],c x)
{
int n=a.length-1;
double eps=1.0e-15;
int m=100;
int k=0;
c f,df,d2f;
c b1,b2,b3,b4,b5;
do{
  f=func(a,x);
  df=dfunc(a,x);
  d2f=d2func(a,x);
  b1=c.p(c.p(f,"*",df),"*",2.0);
  b2=c.p(c.p(df,"*",df),"*",2.0);
  b3=c.p(f,"*",d2f);
  b4=c.p(b2,"-",b3);
  b5=c.p(b1,"/",b4);
  x=c.p(x,"-",b5);
  if(c.abs(f)<eps) return x;
  k++;
}while(c.abs(f)>eps && k<m);
JOptionPane.showMessageDialog(null,"Warning : Maximum number of iteration is exceeded \n"+
   " result may not be valid","MAXIMUM ITERATION NUMBER WARNING",JOptionPane.WARNING_MESSAGE);
return x;
}
public static c[] synthetic_division(double ci[],c x)
{
  int n=ci.length;
  c c1[]=new c[n];
  for(int i=0;i<n;i++) c1[i]=new c(ci[i]);
  return synthetic_division(c1,x);
}
public static c[] synthetic_division(c ci[],c x)
{ int n=ci.length;
  c c1[]=new c[n];
  c1[n-1]=new c(ci[n-1]);
```

```
    for(int i=n-2;i>=0;i--)
    {   c c2=new c(ci[i]);
                  c1[i]=c.p(c2,"+",c.p(x,"*",c1[i+1]));
                }
  c c3[]=new c[n-1];
  for(int i=n-1;i>=1;i--)
{   c c2=new c(c1[i]);
                c3[i-1]=c1[i];
                }
  return c3;
}

public static c[] quadratic(c Ci[])
{ c z[]=new c[2];
     c z1=new c(Ci[0]);
    c ac4=c.p(z1,"*",Ci[2]);
    ac4=c.p(ac4,"*",4.0);
    z1=new c(Ci[1]);
    c b2=c.p(z1,"*",z1);
    c DD=c.sqrt(c.p(b2,"-",ac4));
    z1=new c(Ci[1]);
    z[0]=c.p(DD,"-",z1);
    z[0]=c.p(z[0],"/",2.0);
    z[0]=c.p(z[0],"/",Ci[2]);
    z[1]=c.p(DD,"+",Ci[1]);
    z[1]=c.p(z[1],"/",2.0);
    z[1]=c.p(z[1],"/",Ci[2]);
    z[1]=c.p(z[1],"*",-1.0);
    return z;
}
public static c[] root(double C1[])
{ int n=C1.length;
c Ci[]=new c[n];
for(int i=0;i<n;i++)
{Ci[i]=new c(C1[i]);}
return root(Ci);
}

public static c[] root(c Ci[])
{   int n=Ci.length;
   c y[]=new c[n-1];
                c x0=new c(1.0,0.0);
                int i=n-2;
                int nn=Ci.length;
   while(i>=0)
   {
    if(nn>3)
    { c z=Halley(Ci,x0);
      y[i]=z;
      Ci=synthetic_division(Ci,z);
      nn=Ci.length;
      i--;
    }
    else if(nn==3)
    { c z[]=quadratic(Ci);
      y[i]=z[0];
                  y[i-1]=z[1];
      i-=2;
      nn-=2;
    }
    else if(nn==2)
    {c z;
    c z1=new c(Ci[0]);
                z=c.multiply(z1,-1.0);
```

```
                z=c.divide(z,Ci[1]);
                y[i]=z;
                i-=2;
    nn-=2;
                }
    }
   return y;
}
public static String toString(c x[])
{int n1=x.length;
 String s="";
 for(int i=n1-1;i>=0;i--) {s+="x["+i+"] = "+x[i].toString("%10.7f")+"\n";}
 return s;
}
    public static double[] input(String s)
    {
    String s1=JOptionPane.showInputDialog(s);
    StringTokenizer token=new StringTokenizer(s1);
    int n=token.countTokens()-1;
    int m=n+1;
    double a[]=new double[m];
    int j=0;
    while(token.hasMoreTokens())
    {
    Double ax=new Double(token.nextToken());
    a[j++]=ax.doubleValue();
    }
    return a;
    }

    public static void main (String args[])
    { double c1[]={-4,3,-3,0,2};
      c y[]=root(c1);
      System.out.println(toString(y));
    }
}
```

```
 ---------- Capture Output ----------
> "C:\java\bin\javaw.exe" Halley
x[3] = ( 1.2548819 +  0.0000000*i )
x[2] = (-1.7389563 +  0.0000000*i )
x[1] = ( 0.2420372 +  0.9262455*i )
x[0] = ( 0.2420372 -  0.9262455*i )
> Terminated with exit code 0.
```

Hasan Method of root finding is one of the most efficient method to find roots, a complex polynomial version of the method with sysnthetic division to find all the roots are given next

Program 2.15-8 Complex Hasan's method example to find the roots of a polynomial

```
 // compleks Hasan's method to find roots of the polynomials
// xn+1=xn-f(xn)/(f'(xn)+1/3*f'(xn)*log((f'(xn)*f'(xn)*-f(xn)*f''(xn))/(f'(xn)*f'(xn))
import java.util.*;
import java.awt.*;
import java.applet.Applet;
import java.awt.event.*;
import javax.swing.*;

public class Hasan
{ public static c func(double ai[],c x)
  {   int n=ai.length;
                c a[]=new c[n];
                for(int i=0;i<n;i++) a[i]=new c(ai[i]);
                return func(a,x);
  }
  public static c func(c a[],c x)
```

```
  {  // c polynomial
  int n=a.length;
  c y=new c(a[n-1]);
  for(int i=n-2;i>=0;i--)
  {y=c.p(c.p(y,"*",x),"+",a[i]);}
  return y;
  }
  public static c dfunc(double ai[],c x)
  {   int n=ai.length;
                c a[]=new c[n];
                for(int i=0;i<n;i++) a[i]=new c(ai[i]);
                return dfunc(a,x);
  }
  static c dfunc(c a[],c x)
  {//c numeric derivative of a polynomial
   double hh=0.001;
   c h=new c(hh,0.0);
   c he=new c(-hh,0.0);
   c h2=new c(2.0*hh,0.0);
   c he2=new c(-2.0*hh,0.0);
   c xah=c.add(x,h);
   c xa2h=c.p(x,"+",h2);
   c xeh=c.p(x,"+",he);
   c xe2h=c.p(x,"+",he2);
   c fe2h=c.p(func(a,xe2h),"*",(1.0/12.0));
   c feh=c.p(func(a,xeh),"*",(-8.0/12.0));
   c fah=c.p(func(a,xah),"*",(8.0/12.0));
   c fa2h=c.p(func(a,xa2h),"*",(-1.0/12.0));
   c df=c.p(fe2h,"+",c.p(feh,"+",c.p(fah,"+",fa2h)));
   df=c.p(df,"/",h);
   return df;
  }
  public static c d2func(double ai[],c x)
  {   int n=ai.length;
                c a[]=new c[n];
                for(int i=0;i<n;i++) a[i]=new c(ai[i]);
                return d2func(a,x);
  }
  static c d2func(c a[],c x)
  {//c second numeric derivative of a polynomial
   double hh=0.001;
   c h=new c(hh,hh);
   c hkare=c.pow(h,2);
   c he=new c(-hh,-hh);
   c h2=new c(2.0*hh,2.0*hh);
   c he2=new c(-2.0*hh,-2.0*hh);
   c xah=c.p(x,"+",h);
   c xa2h=c.p(x,"+",h2);
   c xeh=c.p(x,"+",he);
   c xe2h=c.p(x,"+",he2);
   c fe2h=c.p(func(a,xe2h),"*",(-1.0/12.0));
   c feh=c.p(func(a,xeh),"*",(4.0/3.0));
   c ff=c.p(func(a,x),"*",(-5.0/2.0));
   c fah=c.p(func(a,xah),"*",(4.0/3.0));
   c fa2h=c.p(func(a,xa2h),"*",(-1.0/12.0));
   c d2f=c.p(fe2h,"+",c.p(feh,"+",c.p(ff,"+",c.p(fah,"+",fa2h))));
   d2f=c.p(d2f,"/",hkare);
   return d2f;
  }

public static c Hasan(double ai[],c x)
{int n=ai.length;
                c a[]=new c[n];
                for(int i=0;i<n;i++) a[i]=new c(ai[i]);
```

```
                return Hasan(a,x);
}

public static c Hasan(c a[],c x)
{
int n=a.length-1;
double eps=1.0e-15;
int m=100;
int k=0;
c f,df,d2f;
c b1,b2,b3,b4,b5,b6,b7;
double r=3.0;
do{
  f=func(a,x);
  df=dfunc(a,x);
  d2f=dfunc(a,x);
  b1=c.p(c.p(df,"*",df),"-",c.p(f,"*",d2f));
  b2=c.p(df,"*",df);
  b3=c.p(b1,"/",b2);
  b4=c.p(df,"*",c.log(b3));
  b5=c.p(b4,"/",r);
  b6=c.p(df,"+",b5);
  b7=c.p(f,"/",b6);
  x=c.p(x,"-",b7);
  if(c.abs(f)<eps) return x;
  k++;
}while(c.abs(f)>eps && k<m);
JOptionPane.showMessageDialog(null,"Warning : Maximum number of iteration is exceeded \n"+
   " result may not be valid","MAXIMUM ITERATION NUMBER WARNING",JOptionPane.WARNING_MESSAGE);
return x;
}


public static c[] synthetic_division(double ci[],c x)
{
  int n=ci.length;
  c c1[]=new c[n];
  for(int i=0;i<n;i++) c1[i]=new c(ci[i]);
  return synthetic_division(c1,x);
}
public static c[] synthetic_division(c ci[],c x)
{ int n=ci.length;
  c c1[]=new c[n];
  c1[n-1]=new c(ci[n-1]);
    for(int i=n-2;i>=0;i--)
    {   c c2=new c(ci[i]);
                  c1[i]=c.p(c2,"+",c.p(x,"*",c1[i+1]));
                }
  c c3[]=new c[n-1];
  for(int i=n-1;i>=1;i--)
{   c c2=new c(c1[i]);
                c3[i-1]=c1[i];
                }
  return c3;
}

public static c[] quadratic(c Ci[])
{ c z[]=new c[2];
     c z1=new c(Ci[0]);
    c ac4=c.p(z1,"*",Ci[2]);
    ac4=c.p(ac4,"*",4.0);
    z1=new c(Ci[1]);
    c b2=c.p(z1,"*",z1);
    c DD=c.sqrt(c.p(b2,"-",ac4));
```

```
    z1=new c(Ci[1]);
    z[0]=c.p(DD,"-",z1);
    z[0]=c.p(z[0],"/",2.0);
    z[0]=c.p(z[0],"/",Ci[2]);
    z[1]=c.p(DD,"+",Ci[1]);
    z[1]=c.p(z[1],"/",2.0);
    z[1]=c.p(z[1],"/",Ci[2]);
    z[1]=c.p(z[1],"*",-1.0);
    return z;
}
public static c[] root(double C1[])
{ int n=C1.length;
c Ci[]=new c[n];
for(int i=0;i<n;i++)
{Ci[i]=new c(C1[i]);}
return root(Ci);
}

public static c[] root(c Ci[])
{   int n=Ci.length;
   c y[]=new c[n-1];
              c x0=new c(1.0,0.0);
              int i=n-2;
              int nn=Ci.length;
   while(i>=0)
   {
    if(nn>3)
    { c z=Hasan(Ci,x0);
      y[i]=z;
      Ci=synthetic_division(Ci,z);
      nn=Ci.length;
      i--;
    }
    else if(nn==3)
    { c z[]=quadratic(Ci);
      y[i]=z[0];
                y[i-1]=z[1];
      i-=2;
      nn-=2;
    }
    else if(nn==2)
    {c z;
    c z1=new c(Ci[0]);
              z=c.multiply(z1,-1.0);
              z=c.divide(z,Ci[1]);
              y[i]=z;
              i-=2;
    nn-=2;
              }
    }
   return y;
}
public static String toString(c x[])
{int n1=x.length;
 String s="";
 for(int i=n1-1;i>=0;i--) {s+="x["+i+"] = "+x[i]+"\n";}
 return s;
}
    public static double[] input(String s)
    {
    String s1=JOptionPane.showInputDialog(s);
    StringTokenizer token=new StringTokenizer(s1);
    int n=token.countTokens()-1;
    int m=n+1;
```

```
    double a[]=new double[m];
    int j=0;
    while(token.hasMoreTokens())
    {
    Double ax=new Double(token.nextToken());
    a[j++]=ax.doubleValue();
    }
    return a;
    }

    public static void main (String args[])
    { double c1[]={-4,3,-3,0,2};
      c y[]=root(c1);
      System.out.println(toString(y));
}}
```

```
---------- Capture Output ----------
> "C:\java\bin\java.exe" Hasan
x[3] = (   1.2548818848)
x[2] = (   0.2420371858 +    0.9262454873i )
x[1] = (   0.2420371858 -    0.9262454873i )
x[0] = (  -1.7389562565 +    0.0000000000i )


> Terminated with exit code 0.
```

2.16 COMPLEX ROOTS OF THE POLYNOMIALS BY BAIRSTOW'S METHOD

Bairstow's method is an iterative approach, like Horner's method it also uses synthetic division as basic tool. But instead of a single step division, a two step division is taken and utilised. The basic reason for that is to get a better utilisation of complex roots conjugate effect. Remember from the previous section that sythetic division recursion calculations has the form:

$f_n(x) = a_0 + a_1x + a_1x^2 + \cdots + a_nx^n$ (2.16.1)

and if this polynomial divides with $(x - x_0)$ to form polynomial

$f_{n-1}(x) = b_1 + b_2x + b_3x^2 + \cdots + b_{n-1}x^{n-1}$ (2.16.2)

Where $f_n(x) = b_0 + (x - x_0)f_{n-1}(x)$ (2.16.3) and $R = b_0$ is the remainder of the synthetic division process. Of course if the remainder is equal to zero x_0 would be the root of this polynomial. Synthetic division can be calculated as following iterative equation:

$b_n = a_n$ (2.16.4)

$b_i = a_i + b_{i-1}x_0$ (2.16.5)

Now if a second degree synthetic division is desired, the formulation will change a little. If second degree division equation is $x^2 - rx - s$ the result is a new polynomial in the form of

$f_{n-2}(x) = b_2 + b_3x + b_4x^2 + \cdots + b_nx^{n-2}$ (2.16.6) with the remainder

$R = b_1(x - r) + b_0$ (2.16.7) As with the normal synthetic division a simple recurance relation can be used as: $b_n = a_n$ (2.16.8)

$b_{n-1} = a_{n-1} + rb_n$ (2.16.9)

$b_i = a_i + rb_{i+1} + sb_{i+2}$ for i = n-2 to 0 (2.16.10)

As seen from the equation we have two roots (r and s) to solve simultaneously in one step calculation now. Consider that b coefficients are function of r and s, First degree Taylor series expansion of these coefficients can be written as

$b_1^n = b_1^{n-1} + \frac{\partial b_1^{n-1}}{\partial r}\Delta r + \frac{\partial b_1^{n-1}}{\partial s}\Delta s$ (2.16.11)

$b_0^n = b_0^{n-1} + \frac{\partial b_0^{n-1}}{\partial r}\Delta r + \frac{\partial b_0^{n-1}}{\partial s}\Delta s$ (2.16.12) Higher order terms of the Taylor series are ignored. The roots are defined as remainder terms of sysnthetic division as zero (b_0 and b_1 should be equal to zero) so if we assume the left hand side of the equation is zero, this equation becomes

$\frac{\partial b_1}{\partial r}\Delta r + \frac{\partial b_1}{\partial s}\Delta s = -b_1$ (2.16.13)

$\frac{\partial b_0}{\partial r}\Delta r + \frac{\partial b_0}{\partial s}\Delta s = -b_0$ (2.16.14) If partial derivatives of b is determined this set of equation can be easily solved. Bairstow showed that the partial derivatives can be obtained by synthetic division of b values. Then the partial derivatives can be substituted in above equation

$c_n = b_n$ (2.16.15)

$c_{n-1} = b_{n-1} + rc_n$ (2.16.16)

$c_i = b_i + rc_{i+1} + sc_{i+2}$ for i = n-2 to 1 (2.16.17)

So the equation becomes:

$c_2\Delta r + c_3\Delta s = -b_1$ (2.16.18)

$c_1\Delta r + c_2\Delta s = -b_0$ (2.16.19)

Solution of this equation by Gauss elimination process will give the following solution:

$W = \frac{c_1}{c_2}$ (2.16.20)

$\Delta s = \frac{b_0 - Wb_1}{c_2 - Wc_3}$ (2.16.21)

$\Delta r = \frac{-b_1 - c_3\Delta s}{c_2}$ (2.16.22)

$r^{n+1} = r^n + \Delta r^{n+1}$ (2.16.23)

$s^{n+1} = s^n + \Delta s^{n+1}$ (2.16.24)

This equation can be solved iteratively until the error term shrink to a minimum value

$|\varepsilon| = \left|\frac{\Delta r}{r}\right| + \left|\frac{\Delta s}{s}\right| < \varepsilon_s$ (2.16.25) when the error level drops under a specified stopping criterion, iteration stops, and roots can be calculated by using quadratic equation

$x = \frac{r \pm \sqrt{r^2 + 4s}}{2}$ (2.16.26)

Note that all these procedure can be calculated with complex numbers. It should be also note that after reducing polynomial to a third, second or first degree remaining roots can be calculated by using standard root finding equations. An example program is listed below:

$f(x) = x^5 - 3.5x^4 + 2.75x^3 + 2.125x^2 - 3.875x + 1.25$ with r=s=-1

Program 2.17-1 Complex Bairstow's method example

```
 public class bairstow
{
c CA[];
public double[] toDouble()
{int n=CA.length;
 double d[]=new double[n];
 for(int i=0;i<n;i++)
    {   d[i]=c.toDouble(CA[i]);}
   return d;
}
public double[] toDouble(c w[])
{int n=w.length;
 double d[]=new double[n];
 for(int i=0;i<n;i++)
    {   d[i]=c.toDouble(w[i]);}
   return d;
}

public c[] bairstow(double a[], c r,c s)
{   int n=a.length;
```

```
    boolean bb=true;
    int j=0;
                c b[]=new c[n];
                int n1=n-2;
                c cp[]=new c[n1];
                c ci[]=new c[n];
                c ds,dr;
                while(bb)
                {
                b[n-1]=new c(a[n-1]);
                b[n-2]=c.p(c.p(r,"*",b[n-1]),"+",a[n-2]);
    for(int i=n1-1;i>=0;i--)
    {   c c2=new c(a[i]);
                   b[i]=c.p(c2,"+",c.p(r,"*",b[i+1]));
                   b[i]=c.p(b[i],"+",c.p(s,"*",b[i+2]));
    }
    ci[n-1]=b[n-1];
    ci[n-2]=c.p(b[n-2],"+",c.p(r,"*",ci[n-1]));
    for(int i=n1-1;i>=0;i--)
    {   ci[i]=c.p(b[i],"+",c.p(r,"*",ci[i+1]));
                   ci[i]=c.p(ci[i],"+",c.p(s,"*",ci[i+2]));
    }
                for(int i=n1-1;i>=0;i--)
                {cp[i]=b[i+2];}
                CA=cp;
                // Gauss elimination formula to solve system of equation
                c CC=c.p(ci[1],"/",ci[2]);
                c b0=c.p(CC,"*",b[1]);
                b0=c.p(b0,"-",b[0]);
                c c0=c.p(CC,"*",ci[3]);
                c0=c.p(ci[2],"-",c0);
                ds=c.p(b0,"/",c0);
                c d0=c.p(ds,"*",ci[3]);
                d0=c.p(d0,"+",b[1]);
                d0=c.p(d0,"*",-1.0);
                dr=c.p(d0,"/",ci[2]);
                r=c.p(r,"+",dr);
                s=c.p(s,"+",ds);
                double de=c.abs(c.p(dr,"/",r))+c.abs(c.p(ds,"/",s));
                if(de<1e-15) bb=false;
    }
    c x[]=new c[2];
    c DD=c.sqrt(c.p(c.p(r,"*",r),"+",c.p(s,"*",4.0)));
    x[0]=c.p(r,"+",DD);
    x[0]=c.p(x[0],"/",2.0);
    x[1]=c.p(r,"-",DD);
    x[1]=c.p(x[1],"/",2.0);
   return x;
}
public c[] root(double a[])
{int n=a.length;
 c x[]=new c[n-1];
 c r=new c(1.0,0.0);
 c s=new c(1.0,0.0);
 c y[]=bairstow(a,r,s);
 int i=n-2;
 x[i]=y[0];
 x[i-1]=y[1];
 i-=2;
 int nn=CA.length;
 while(i>=0)
 {  if(nn!=CA.length)
    {c D[]=new c[nn];
     for(int j=0;j<nn;j++) {D[j]=CA[j];}
```

```
     CA=D;
     }
              y=bairstow(CA,r,s);
    if(nn==2) x[i]=y[0];
    else {x[i]=y[0];x[i-1]=y[1];}
    nn-=2;
    i-=2;
  }
return x;
}

public c[] bairstow(c a[], c r,c s)
{   int n=a.length;
    c x[];
    if(n>3)
    {
              x=new c[2];
    boolean bb=true;
    int j=0;
              c b[]=new c[n];
              int n1=n-1;
              c cp[]=new c[n1];
              c ci[]=new c[n];
              c ds,dr;
              while(bb)
              {
              b[n-1]=a[n-1];
              c z1=c.p(r,"*",b[n-1]);
              b[n-2]=c.add(z1,a[n-2]);
    for(int i=n-3;i>=0;i--)
    {   c c2=new c(a[i]);
                b[i]=c.p(c2,"+",c.p(r,"*",b[i+1]));
                b[i]=c.p(b[i],"+",c.p(s,"*",b[i+2]));
    }
    ci[n-1]=b[n-1];
    ci[n-2]=c.p(b[n-2],"+",c.p(r,"*",ci[n-1]));
    for(int i=n-3;i>=0;i--)
    {   ci[i]=c.p(b[i],"+",c.p(r,"*",ci[i+1]));
                ci[i]=c.p(ci[i],"+",c.p(s,"*",ci[i+2]));
    }
              for(int i=n1-2;i>=0;i--)
              {cp[i]=b[i+2];}
              CA=cp;
              c CC=c.p(ci[1],"/",ci[2]);
              c b0=c.p(CC,"*",b[1]);
              b0=c.p(b0,"-",b[0]);
              c c0=c.p(CC,"*",ci[3]);
              c0=c.p(ci[2],"-",c0);
              ds=c.p(b0,"/",c0);
              c d0=c.p(ds,"*",ci[3]);
              d0=c.p(d0,"+",b[1]);
              d0=c.p(d0,"*",-1.0);
              dr=c.p(d0,"/",ci[2]);
              r=c.p(r,"+",dr);
              s=c.p(s,"+",ds);
              double de=c.abs(c.p(dr,"/",r))+c.abs(c.p(ds,"/",s));
              if(de<1e-15) bb=false;
    }

    c DD=c.sqrt(c.p(c.p(r,"*",r),"+",c.p(s,"*",4.0)));
    x[0]=c.p(r,"+",DD);
    x[0]=c.p(x[0],"/",2.0);
    x[1]=c.p(r,"-",DD);
    x[1]=c.p(x[1],"/",2.0);
```

```
    }
    else if(n==3)
    { x=new c[2];
     c ac4=c.p(a[0],"*",a[2]);
     ac4=c.p(ac4,"*",4.0);
     c b2=c.p(a[1],"*",a[1]);
     c DD=c.sqrt(c.p(b2,"-",ac4));
     x[0]=c.p(DD,"-",a[1]);
     x[0]=c.p(x[0],"/",2.0);
     x[0]=c.p(x[0],"/",a[2]);
     x[1]=c.p(DD,"+",a[1]);
     x[1]=c.p(x[1],"/",2.0);
     x[1]=c.p(x[1],"/",a[2]);
     x[1]=c.p(x[1],"*",-1.0);
     }
     else //(n==2)
     { x=new c[1];
                x[0]=c.p(a[0],"*",-1.0);
                x[0]=c.p(x[0],"/",a[1]);
                }
   return x;
}

public String toString()
{int n1=CA.length;
 String s="";
 for(int i=n1-1;i>=0;i--) {s+=CA[i]+" ";}
 return s;
}
public String toString(c x[])
{int n1=x.length;
 String s="";
 for(int i=n1-1;i>=0;i--) {s+="x["+i+"] = "+x[i]+"\n";}
 return s;
}

public static void main(String arg[])
{   double c1[]={-4,3,-3,0,2};
                bairstow h=new bairstow();
                c x[]=h.root(c1);
                System.out.println(h.toString(x));
}
}
```

```
 ---------- Capture Output ----------
> "C:\java\bin\java.exe" bairstow
x[3] = (   1.2548818848)
x[2] = (  -1.7389562565)
x[1] = (   0.2420371858 -    0.9262454873i )
x[0] = (   0.2420371858 +    0.9262454873i )
```

For: **double a[]={24,-50,35,-10,1};** $f(x) = x^4 - 10x^3 + 35x^2 - 50x + 24$

```
 ---------- Capture Output ----------
> "C:\java\bin\java.exe" bairstow
x[3] = (   4.0000000000)
x[2] = (   1.0000000000)
x[1] = (   3.0000000000)
x[0] = (   2.0000000000)
```

For: **double a[]={-1 3 -3 1};** $f(x) = x^3 - 3x^2 + 3x - 1$

```
 ---------- Capture Output ----------
> "C:\java\bin\java.exe" bairstow
x[2] = (   1.0000003728 +    0.0000006458i )
```

```
x[1] = (   1.0000003728 -    0.0000006458i )
x[0] = (   0.9999992543)

> Terminated with exit code 0.
```

2.17 ROOT FINDING BY USING SOFTWARE PACKAGES

In order to find roots, some built in functions already existed in package math programs. In Matlab-Octave function **fzero** is existed to find roots of a function and **roots** to find roots of a polynomial

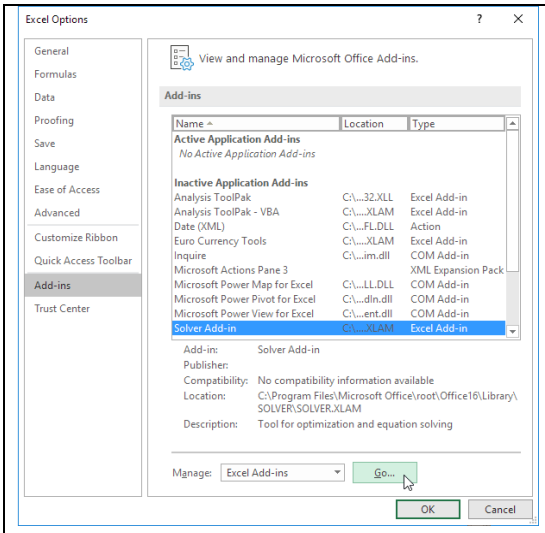
```
>> format long
>> x=2;
>> y=@(x)x*x-2;
>> fzero(y,x)
ans =  1.41421356237310
>>
```

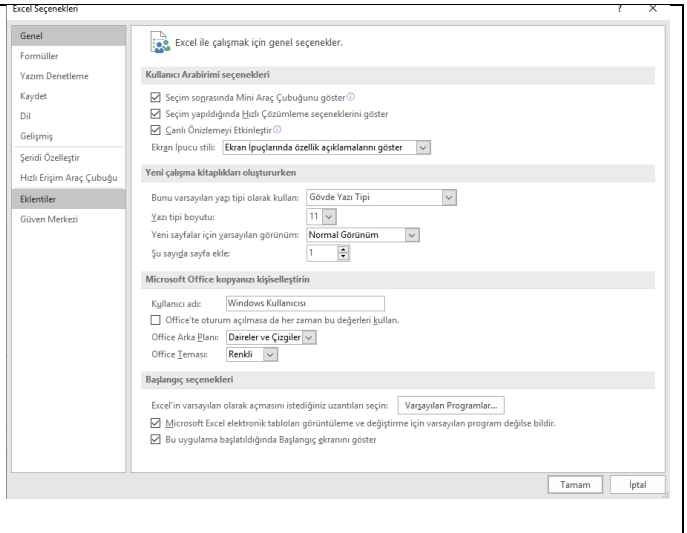
```
>> x=1;
>> y=@(x)x^5-3.5*x^4+2.75*x^3+2.125*x^2-3.875*x+1.25;
>> fzero(y,x)
ans =  0.500000000000001
```

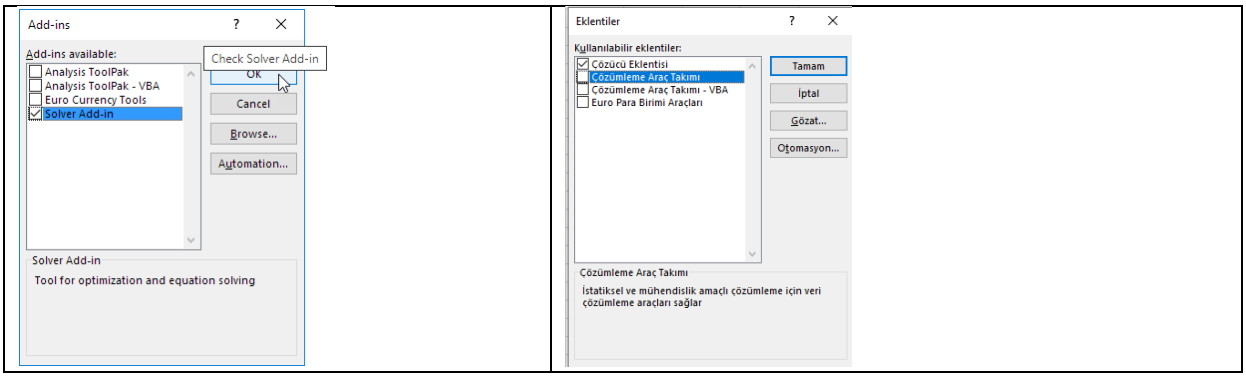
```
>> a=[1 -3.5 2.75 2.125 -3.875 1.25];
>> roots(a)
ans =
   2.000000000000003 + 0.000000000000000i
  -1.000000000000002 + 0.000000000000000i
   1.000000000000000 + 0.500000000000000i
   1.000000000000000 - 0.500000000000000i
   0.499999999999999 + 0.000000000000000i
```

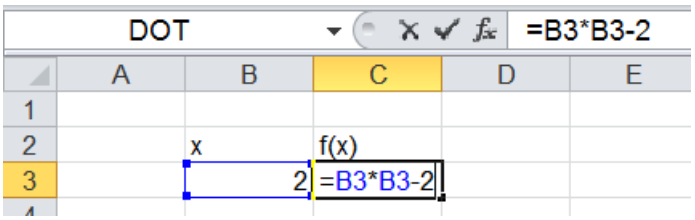
```
>> y=@(x)x*x+2;
>> x=1+1i;
>> fsolve(y,x)
ans =
 -0.000000000000016 + 1.414213562374690i
```

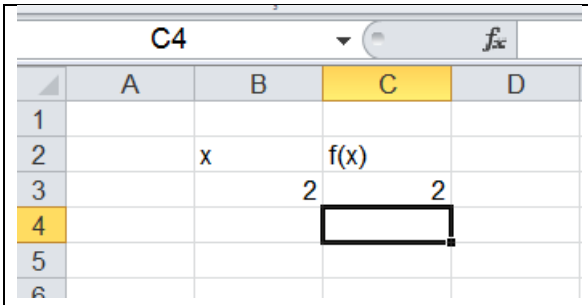
In excel, solver is existed to find real roots: We can use solver add in. In order to open solver:
On the File tab, click Options. (in turkish version dosya, seçenekler)
Under Add-ins(in turkish version eklentiler), select Solver Add-in (in turkish version Çözücü eklentisi), and click on the Go(Git..) button. You can find the Solver on the Data tab(Veri), in the Analyze group.

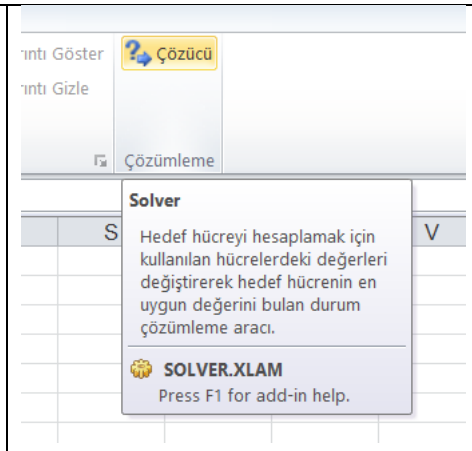


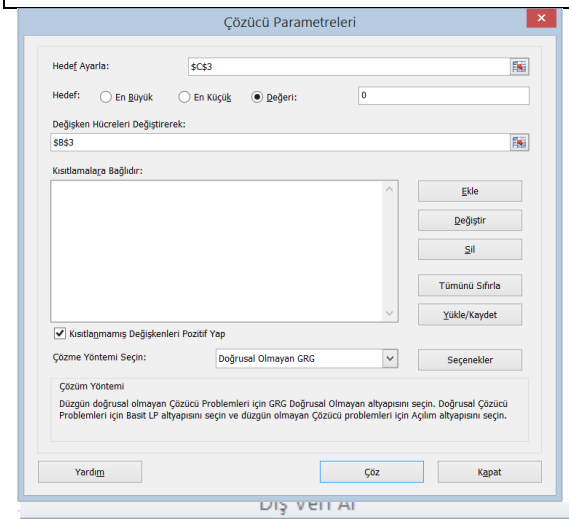


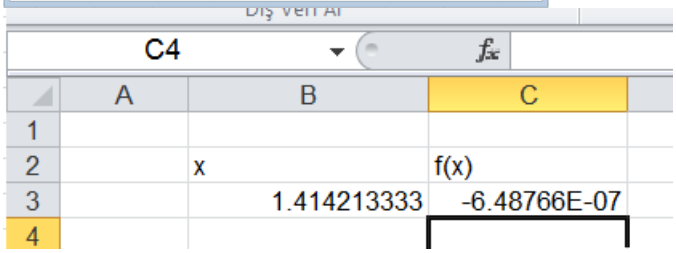

Add-ins
Add-ins available:
Analysis ToolPak
Analysis ToolPak - VBA
Euro Currency Tools
Solver Add-in
Check Solver Add-in
OK
Cancel
Browse...
Automation...
Solver Add-in
Tool for optimization and equation solving
Eklentiler
Kullanılabilir eklentiler:
Çözücü Eklentisi
Çözümleme Araç Takımı
Çözümleme Araç Takımı - VBA
Euro Para Birimi Araçları
Tamam
İptal
Gözat...
Otomasyon...
Çözümleme Araç Takımı
İstatiksel ve mühendislik amaçlı çözümleme için veri çözümleme araçları sağlar


DOT
=B3*B3-2
A
B
C
D
E
1
2
x
f(x)
3
2
=B3*B3-2


C4
A
B
C
D
1
2
x
f(x)
3
2
2
4
5


ıntı Göster
ıntı Gizle
Çözücü
Çözümleme
Solver
Hedef hücreyi hesaplamak için kullanılan hücrelerdeki değerleri değiştirerek hedef hücrenin en uygun değerini bulan durum çözümleme aracı.
SOLVER.XLAM
Press F1 for add-in help.


Çözücü Parametreleri
Hedef Ayarla:
C3
Hedef:
En Büyük
En Küçük
Değeri:
0
Değişken Hücreleri Değiştirerek:
B3
Kısıtlamalara Bağlıdır:
Ekle
Değiştir
Sil
Tümünü Sıfırla
Yükle/Kaydet
Kısıtlanmamış Değişkenleri Pozitif Yap
Çözme Yöntemi Seçin:
Doğrusal Olmayan GRG
Seçenekler
Çözüm Yöntemi
Düzgün doğrusal olmayan Çözücü Problemleri için GRG Doğrusal Olmayan altyapısını seçin. Doğrusal Çözücü Problemleri için Basit LP altyapısını seçin ve düzgün olmayan Çözücü problemleri için Açılım altyapısını seçin.
Yardım
Çöz
Kapat

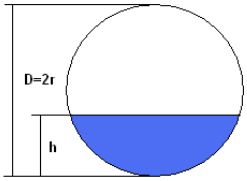

C4
A
B
C
1
2
x
f(x)
3
1.414213333
-6.48766E-07
4

2.18 PROBLEMS

PROBLEM 1 Find the roots of $\boldsymbol{f(x) = x^2 - 0.9x - 8.5}$ by using bisection method

PROBLEM 2 Calculate $\sqrt{7}$ by using one of the root finding method

PROBLEM 3



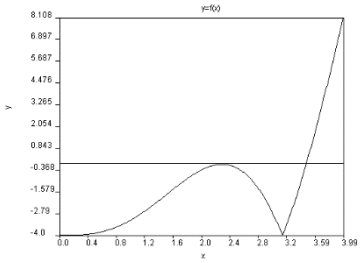
A spherical tank of radius r is filled with a liquid of height h. The Volume of liquid (V) in the tank is given by he following eqaution

$\boldsymbol{V = \pi h^2(3r - h)/3}$

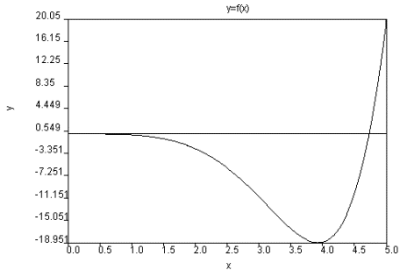
If r = 1 m and V= 0.5 m^3 find the liquid height

Note : due to geometry of a sphere h can not be less than 0 and can not be bigger than D=2r. The maximum amount of liquid ths spherical container can be take is 4.18879 m^3 (you do not need this data to solve problem)

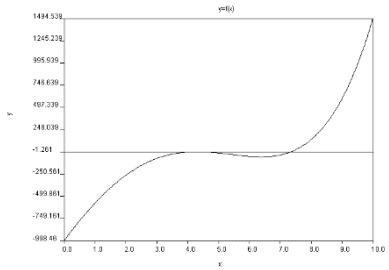
PROBLEM 4 Find the roots of $f(x) = x^2|sin(x)| - 4.$ by using one of the root finding method



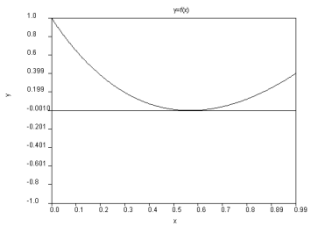
PROBLEM 5 Find the roots of $f(x) = \cos(x)\cosh(x) - 1$ by using one of the root finding method



PROBLEM 6 Find the roots of $f(x) = x^4 - 8.6x^3 - 35.5x^2 + 464.4x - 998.46$ by using one of the root finding method Try x_0=4 and $\boldsymbol{7 \leq x \leq 8}$

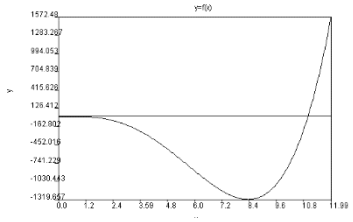


PROBLEM 7 Function $\boldsymbol{f(x) = x^2 - 2e^{-x} + e^{-2x}}$ has only one real root. Find the root bu using Newton-Raphson method. Use as starting value x_0=1

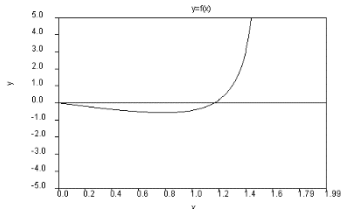


PROBLEM 8

Find all real roots of function $f(\boldsymbol{x}) = x^4 - 7.223x^3 + 13.4475x^2 - 0.672x - 10.223$



PROBLEM 9 Find roots of functionf $(\boldsymbol{x}) = \tan(\mathrm{x}) - 2x$.



PROBLEM 10 For Turbulent flow inside of a pipe (Re>2300) friction coefficient inside of a pipe can be calculated by using Colebrook equation.

$$\frac{\mathbf{1}}{\sqrt{\boldsymbol{f}}} = -2.0log_{10}\left[\frac{\varepsilon/D}{3.7} + \frac{2.51}{Re\sqrt{f}}\right]$$

Re is Reynolds number, ε is surface rougness. Write a program to calculate total pressure drop in the pipe for a given Re number, pipe diameter, pipe length and local pressure drop coefficient.

f : friction coefficient

D pipe diameter

Hint : Change the colebrook equation into the form

$$F(x) = X + 2.0log_{10}\left[\frac{\varepsilon/D}{3.7} + \frac{2.51X}{Re}\right] = 0$$

Then find the root of the equation where $X = \frac{1}{\sqrt{f}}$

Use Haaland equation to obtaing the first estimate of the solution which has the form of

$$\frac{\mathbf{1}}{\sqrt{\boldsymbol{f}}} = -1.8log_{10}\left[\frac{(\varepsilon/D)^{1.11}}{3.7} + \frac{6.9}{Re}\right]$$

```
import java.io.*;
import javax.swing.*;

class f1 extends f_x
{ public double eod;
  public double Re;
  public f1(double eod,double Re)
  {this.eod=eod;
   this.Re=Re;
  }
  public double func(double X)
  {double F=X+2.0*Math.log10(eod/3.7+2.51*X/Re);
        return F;
  }
  public double Haaland()
  {double X=-1.8*Math.log10(Math.pow(eod,1.11)/3.7+6.9/Re);
   return X;
  }
}
class colebrook1
{
public static double secant(f_x f,double x)
```

```
{
int nmax=500;
double tolerance=1.0e-10;
double fx,dfx;
for(int i=0;i<nmax;i++)
{
fx=f.func(x);
dfx=f.dfunc(x);
x-=fx/dfx;
if(Math.abs(fx)<tolerance) { return x;}
}
JOptionPane.showMessageDialog(null,"Warning : Maximum number of iteration is exceeded \n"+
   "result might not be valid","MAXİMUM ITERATION NUMBER WARNING",JOptionPane.WARNING_MESSAGE);
return x;
}
public static void main (String args[]) throws java.io.IOException
{  f1 f=new f1(0.01,2500);
   double a=f.Haaland();
   double r=secant(f,a);
   double f1=1.0/(r*r);
   System.out.println("f colebrook = "+f1);
}}
```

```
---------- Capture Output ----------
> "D:\co\java\bin\java.exe" colebrook1
f colebrook = 0.053919466488887825

> Terminated with exit code 0.
```

```
from math import *
from f_x import *;

class f1(f_x):
    def __init__(self,eod,Re):
        self.name = 'mathd',
        self.eod=eod;
        self.Re=Re;
    def func(self,X):
        F=X+2.0*log10(self.eod/3.7+2.51*X/self.Re);
        return F;
    def Haaland(self):
        X=-1.8*log10(pow(self.eod,1.11)/3.7+6.9/self.Re);
        return X;

def secant(f,x):
   # secant root finding method
  nmax=100
  tolerance=1.0e-10
  for i in range(0,nmax):
      fx= f.func(x);
      dfx=f.dfunc(x,1);
      x=x-fx/dfx
      if abs(fx)<tolerance: return x
  return x


f=f1(0.01,2500);
a=f.Haaland();
r=secant(f,a);
f1=1.0/(r*r);
print(" f colebrook  : ",f1);
```

```
runfile('D:/okul/SCO1/secant1.py', wdir='D:/okul/SCO1')
```

```
Reloaded modules: f_x
 f colebrook  :  0.053919466488887825
```

PROBLEM 11

In round smooth tubes for turbulent flow region Fanning friction factor is given as:

$$\frac{\mathbf{1}}{\sqrt{\boldsymbol{f}}} = 1.737\ln\left[Re\sqrt{f}\right] - 0.4$$

This equation is valid for $4 * 10^3 < Re < 3 * 10^6$ By using one of the root finding methods, calculate friction factor for Re=5000. You can utilize any root finding method you learned so far.

Note : You can assume that f is changing between 0,001 and 0.01

PROBLEM 12

van der Waals equation of state for the gases :

$P = \frac{RT}{v-b} - \frac{a}{v^2}$ In this equation

$$a = \frac{27}{64}\frac{R^2T_c^2}{P_c}$$

$$b = \frac{RT_c}{8P_c}$$

Redlich ve Kwong equation of states

$P = \frac{RT}{v-b} - \frac{a}{v(v+b)T^{1/2}}$ In this equation

$$a = 0.42748\frac{R^2T_c^{2.5}}{P_c}$$

$$b = 0.08664\frac{RT_c}{P_c}$$

O2, for oxygen P_c=5.043 MPa T_c=154.58 K M=31.999 kg/kmol.

N2, for nitrogen P_c=3.39 MPa T_c=126.2 K M=28.013 kg/kmol.

R=8314.5 J/kmolK

Benedict -Webb-Rubin (BWR) equation of state

$$P = \frac{RT}{v} + \frac{RTB_0 - A_0 - C_0/T^2}{v^2} + \frac{RTb - a}{v^3} + \frac{a\alpha}{v^6} + \frac{c}{v^3}\left(1 + \frac{\gamma}{v^2}\right)e^{-\frac{\gamma}{v^2}}$$

Gas	A_0	B_0	$C_0.10^{-6}$	A	B	$c.10^{-6}$	$\alpha.10^3$	$\gamma.10^3$
O_2	1.49880	0.046524	0.0038617	-0.040507	-0.00027963	-0.00020376	0.008641	0.359
N_2	1.1925	0.0458	0.0058891	0.01490	0.00198154	0.000548064	0.291545	0.75

Develop a program to calculate v when T and P is given. By using

A) van der Waals equation of state : gas : oxygen and nitrogen
B) Redlich Kwong equation of state : gas : oxygen and nitrogen
C) Benedict-webb rubin equation of state : gas : oxygen
D) Benedict-webb equation of state : gas : nitrogen
E) Compare the result with ideal gas equation $P = \frac{RT}{v}$

```
 import java.io.*;
import javax.swing.*;

class f2 extends f_x
{ public String name;
  public double R;
  public double a,b;
  public double T,P;
  public f2(String n,double Pc,double Tc,double M,double Pi,double Ti)
  {   name=n;
        R=8314.5/M; //J/kmolK
        a=27.0/64.0*R*R*Tc*Tc/Pc;
        b=R*Tc/(8.0*Pc);
        P=Pi;
        T=Ti;
```

```
        }
        public double func(double v)
        { double x=R*T/(v-b)-a/(v*v)-P;
          return x;
        }
        public void setT(double Ti)
        {T=Ti;}
        public void setP(double Pi)
        {P=Pi;}

      }
      class vanderWaals1
      {
      public static double secant(f_x f,double x)
      {
      int nmax=500;
      double tolerance=0.01;
      double fx,dfx;
      double dx[];
      for(int i=0;i<nmax;i++)
      {
      fx=f.func(x);
      dfx=f.dfunc(x);
      x-=fx/dfx;
      if(Math.abs(fx)<tolerance) { return x;}
      }
      JOptionPane.showMessageDialog(null,"Warning : Maximum number of iteration is exceeded \n"+
        "result might not be valid","MAXİMUM ITERATION NUMBER WARNING",JOptionPane.WARNING_MESSAGE);
      return x;
      }
      public static void main (String args[]) throws java.io.IOException
      {  double T=300.0;  //deg K
        double P=0.2e6;  //Pa
        f2 f=new f2("oxygen",5.043e6,154.58,31.999,P,T);
        double v1=f.R*T/P; //ideal gas;
        double P1=f.func(v1);
        double v=secant(f,v1);
        System.out.println("ideal gas v = "+v1+" m^3/kg");
        System.out.println("van der Waals v = "+v+" m^3/kg");
      }
      }
```

```
---------- Capture Output ----------
> "D:\co\java\bin\java.exe" vanderWaals1
ideal gas v = 0.3897543673239789 m^3/kg
van der Waals v = 0.3890197822476924 m^3/kg
> Terminated with exit code 0.
```

```
from math import *
from f_x import *;

class f2(f_x):
    def __init__(self,name,Pc,Tc,M,P,T):
        self.name = name,
        self.T=T;
        self.P=P;
        self.R=8314.5/M;
        self.a=27.0/64.0*self.R*self.R*Tc*Tc/Pc;
        self.b=self.R*Tc/(8.0*Pc)

    def func(self,v):
        x=self.R*self.T/(v-self.b)-self.a/(v*v)-self.P
        return x;
    def setT(T):
        self.T=T;
    def setP(P):
        self.P=P;
```

```
    def Haaland(self):
        X=-1.8*log10(pow(self.eod,1.11)/3.7+6.9/self.Re);
        return X;

def secant(f,x):
  # secant root finding method
  nmax=100
  tolerance=1.0e-10
  for i in range(0,nmax):
      fx= f.func(x);
      dfx=f.dfunc(x,1);
      x-=fx/dfx
      if abs(fx)<tolerance: return x
  return x


T=300.0;  #deg K
P=0.2e6;  #Pa
f=f2("oxygen",5.043e6,154.58,31.999,P,T);
v1=f.R*T/P; #ideal gas;
v=secant(f,v1);
print("ideal gas v = ",v1," m^3/kg");
print("van der Waals v = ",v," m^3/kg");
```

```
runfile('D:/okul/SCO1/vanderWaals.py', wdir='D:/okul/SCO1')
Reloaded modules: f_x
ideal gas v =  0.3897543673239789  m^3/kg
van der Waals v =  0.3890197822476924  m^3/kg
```

PROBLEM 13

Find the root of function f(x)=x $e^{-x/2}$ + x^3 + 1 with the limits of x_1=-1, x_2=0, x_3=1 ny using muller method.

PROBLEM 14

Find the root of function f(x)=x^4-20x^3-25x^2+100x+130 by using Leguerre method.

PROBLEM 15

Find the root of fuction f(x)=x^2+1.

PROBLEM 16

For a paralel flow heat exchanger Number of transfer unit(NTU) and effectiveness (ε) equation given as follows

$$\varepsilon = \frac{1 - exp[-NTU(1 + C^*)]}{(1 + C^*)}$$

After the experiments for a specific heat exchanger NTU=1.989 and ε = 0.574 is found from the experiment

a) Determine the range C^* . For this start C^*=0.1 and increase 0.1 in each step
b) Calculate C^* by using false position method.

PROBLEM 17

Two end of 5 cm long plate with crossection of 0.1cm x 20 cm plate is held at 80 °C ve 60 °C. The environmental temperature is T_∞ = 10 °C, convective heat transfer coefficient h=15 W/m^2 K. Bar cross sectional area A=2×10^{-4} m^2, and perimeter P=0.402 m.In the experiment mid point(x=0.025 m) temperature is found as T=67.5 °C. In this conditions, temperature distribution in the bar is given as

$$T - T_\infty = 68.7e^{-0.025m} + 11.3e^{0.025m}$$

In this equation m is defined as: $m = (\frac{hP}{kA})^{0.5}$

a) Find the range of m, stating from m=5, and increasing 5 at each step.
b) Solve m by using Newton-Raphson method
c) Determine thermal conductivity coefficient k from this value.

PROBLEM 18

The cost of the heat exchanger(M), is defined as a function of the length of heat excahger, L, diameter of the heat exchanger, D as**:** $\boldsymbol{M=900 + 1100\ D^{2,5}\ L + 320\ D\ L}$ In the optimization study the length of the heat exchanger is found as L=1.3 m and the cost as M=1900 TL. Find the diameter of the heat exchanger.

PROBLEM 19

An instrument 20000 TL worth is purcahased with no first payment and 4000 TL annual payment for 6 years. Calculate the interest rate applied by using secant method.

$$A = P\frac{i(i+1)^n}{(i+1)^n - 1}$$

In the equation P is the present worth, A is the annual payment, i is the interestrate, n is the number of payment years.

PROBLEM 20

In order to heat a substance to a required temperature the energy equation is given as

$Q = \int mC_p dT$ eşitliği ile bulunmaktadir.

A substance with an initial temperature of 30 °C, with a mass m=0.2 kg, is heated to some temperature and total energy spend is calculated as 3069.7 kJ . The specific heat of the substance is given with the equation C_p = 41.87+ 1.6748* T + 0.007537* T^2 (kJ/kg K). Find the final temperature by using bi-section method.

PROBLEM 21

Find the roots of f(x) = x^3-6x^2+11x-6.1 by using Legurre method

PROBLEM 22 Find the root(s) of the following equation in the range of [-3.5,3.5]

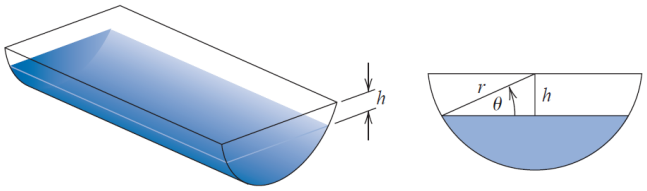
$$f(x) = \frac{1}{\sqrt{2\pi}}e^{-(\frac{1}{2})x^2} + \frac{1}{10}\sin(\pi x)$$

PROBLEM 23 Find the root(s) of the following equation

$$f(x) = \left[10x\left(1 - e^{-\frac{0.0004}{2000x}}\right) - 0.00001\right]$$

PROBLEM 24 In celestial mechanics, **Kepler's equation** is important. It reads $\boldsymbol{x = y - \varepsilon \sin y}$, in which *x* is a planet's mean anomaly, *y* its eccentric anomaly, and ε the eccentricity of its orbit. Taking ε = 0.9, construct a table of *y* for 30 equally spaced values of *x* in the interval 0 to π. Use Newton's method to obtain each value of *y*. The *y* corresponding to an *x* can be used as the starting point for the iteration when *x* is changed slightly.

PROBLEM 25 A trough of length *L* has a cross section in the shape of a semicircle with radius *r*. When filled with water to within a distance *h* of the top, the volume *V* of water is



$$V = L\left[0.5\pi r^2 - r^2 \arcsin\left(\frac{h}{r}\right) - h(r^2 - h^2)^{1/2}\right]$$

Suppose L=1 m, r=0.25 m and V=0.05 m^3 Find the depth of water**,** h

PROBLEM 26 Following functions and root values are given. Investigate number of iteration they are taken when Newton, Halley, Abbasbandy, Thiele 1 and Thiele 2 methods are used.

$$f(x) = x^3 + 4x^2 - 25x^* = 2.035268481182$$
$$f(x) = x^3 - e^x - 3x + 2x^* = 0.257530285439$$
$$f(x) = x^3 - 10x^* = 2.154434690032$$
$$f(x) = cos(x) - xx^* = 0.739085133215$$
$$f(x) = sin^2(x) - x^2 + 1x^* = 1.404491648215$$
$$f(x) = x^2 + sin(x/5) - 1/4x^* = 0.409992017989$$

PROBLEM 27 (a) Use bisection to find the smallest positive zero of the function $f(x) = cot(3x) - \frac{x^2-1}{2x}$

```
from math import *

def bisection(f,xl,xu):
  # bisection root finding method
  maxit=100
  iter=0
  es=0.0000001
  ea=1.1*es
  s=""
  fxl= f(xl)
  fxu= f(xu)
  xr=xl
  while ea>es and iter<maxit:
    xold=xr
    xr=(xl+xu)/2.0;
    fxr= f(xr);
    iter=iter+1;
    if xr!=0:
      ea=abs((xr-xold)/xr)*100
    test= fxl*fxr;
    if test==0.0 :      ea=0;
    elif test<0.0: xu=xr;fxu=fxr;
    elif test>0:    xl=xr;fxl=fxr;
    else :          ea=0;
    if iter>=maxit: print("Maximum number of iteration is exceeded \n result might not be valid")
  return xr

f1 = lambda x: 1.0/tan(3*x)-(x*x-1)/(2*x);
a=float(input("enter lower limit : "))
b=float(input("enter upper limit : "))
r=bisection(f1,a,b);
print(" ROOT  : ",r,"\nFUNCTION VALUE : ",f1(r))
```

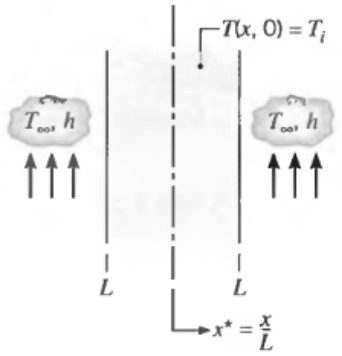
```
---------- Capture Output ----------
> "D:\co\anaconda\pythonw.exe" bisection.py
enter lower limit : .1
enter upper limit : 3
 ROOT  :  0.6588271550484934
FUNCTION VALUE :  -5.676126790810088e-10

> Terminated with exit code 0.
```

(b) Did you remember to plot the function before using bisection?

c) repeat using the root by Newton-Raphson, Illinois, Halley, Abbasbandy , Thiele1 and Thiele 2 methods. Find the number of iteration it takes in each case.

PROBLEM 28 In time dependent heat transfer, heat transfer of a Wall where both sides are exposed to convective heat transfer with convective heat transfer coefficient h and temperature T_∞ is shown in the figure (reference Fundamentals of heat and mass transfer Incropera, DeWitt, Bergman**)**



Time dependent temperature difference for such a wall is given by the equation

$$\theta_0^* = \frac{T_0 - T_\infty}{T_i - T_\infty} = \sum_{n=1}^{\infty} C_n \, exp(-\xi_n^2 Fo) \, cos(\xi_n x^*)$$

Where $Fo = \frac{\alpha t}{L^2}$is Fourier number t is time(second), $\alpha = \frac{k}{\rho C_p}$ is thermal diffisuvity coefficient, L is the half length of the Wall. In order to solve coefficients ξ_n (eigenvalues) the following equation should be solved.

$tan\,\xi_n - \frac{Bi}{\xi_n} = 0$where $Bi = \frac{hL}{k}$is Biot number, h is thermal covection coefficient and k is thermal conductivity of the Wall material. Assume Bi number is given and developed an algorithm to solve first 4 roots of the ξ_n. In order to help you The first 4 roots of the equation for various Bi number are given in the following table.

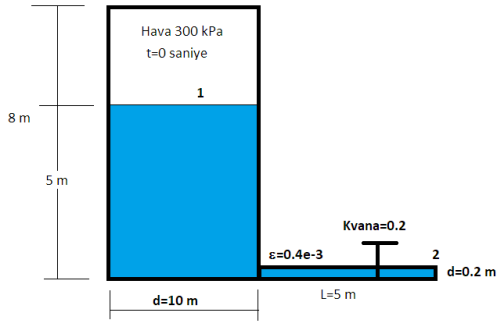
$Bi = \frac{hL}{k}$	ξ_1	ξ_2	ξ_3	ξ_4
0	0	3.1416	6.2832	9.4248
0.001	0.0316	3.1419	6.2833	9.4249
0.002	0.0447	3.1422	6.2835	9.4250
0.004	0.0632	3.1429	6.2838	9.4252
0.006	0.0774	3.1435	6.2841	9.4254
0.008	0.0893	3.1441	6.2845	9.4256
0.01	0.0998	3.1448	6.2848	9.4258
0.02	0.1410	3.1479	6.2864	9.4269
0.04	0.1987	3.1543	6.2895	9.4290
0.06	0.2425	3.1606	6.2927	9.4311
0.08	0.2791	3.1668	6.2959	9.4333
0.1	0.3111	3.1731	6.2991	9.4354
0.2	0.4328	3.2039	6.3148	9.4459
0.3	0.5218	3.2341	6.3305	9.4565
0.4	0.5932	3.2636	6.3461	9.4670
0.5	0.6533	3.2923	6.3616	9.4775
0.6	0.7051	3.3204	6.3770	9.4879
0.7	0.7506	3.3477	6.3923	9.4983
0.8	0.7910	3.3744	6.4074	9.5087
0.9	0.8274	3.4003	6.4224	9.5190
1.0	0.8603	3.4256	6.4373	9.5293
1.5	0.9882	3.5422	6.5097	9.5801
2.0	1.0769	3.6436	6.5783	9.6296
3.0	1.1925	3.8088	6.7040	9.7240
4.0	1.2646	3.9352	6.8140	9.8119
5.0	1.3138	4.0336	6.9096	9.8928

PROBLEM 29 Solve equation $3^x x^3 = 1$

PROBLEM 30 Calculate all roots of the following polynomials

$$f(x) = x^3 - x^2 + 2x - 2$$
$$f(x) = 3.704x^3 + 16.3x^2 - 21.97x + 9.34$$

PROBLEM 31A water container filled to H=5 m level. Total height of the container is HT=8 meters. At the top of water air with 300 kPa and 27 C existed. The top of the storage tank is closed. Diameter of the container is 10 m. Water is discharged through a pipe of d=0.2 m in diameter and L=5 m in length. There is a globe valve at this pipe (KL=0.2). Calculate velocity of the discharge water as a function of time




```
import javax.swing.JOptionPane;
import java.util.Locale;
import java.util.*;
import java.awt.*;
import java.applet.Applet;
import java.awt.event.*;
import javax.swing.*;

public class sudeposu_boru_akis
{  public liquid_CS su;   // soğuk sıvı (boru içi)
   public Gas hava;       // deponun tepesindeki su
   public double T;       // sistem sıcaklığı
   public double Phava[]; // depo hava basıncı zamana bağlı
   public double mhava;   // depoda bulunan hava miktarı kg
   public double Patm;    // atmosfer basıncı
   public double Vhava[]; // depo hava hacmi zamana bağlı m^3
   public double eps;     // yüzey pürüzlülüğü
   public double d;       // boru iç çapı
   public double Do;      // depo çapı
   double L;       // boşaltma borusu boyu
   double eod;     // eps/di
   double H;       // depo yüksekliği
   double h[];     // zamana bağlı depo su seviyesi
   double V[];     // zamana bağlı hız
   double V2[];    // zamana bağlı depo içi su hızı
   double KL;      // yerel basınç düşümü
   int n_finite;   // sonlu farklar zaman step sayısı
   double Ahava;   // depo kesit alanı
   double A;       // boru kesit alanı
   double Aratio;  // A/Ahava
   double rohava,rosu;
   double g;  //yerçekimi sabiri 9.81 m/s^2
   double gammasu; //rosu*g
   public if_x ff;
   public double dt; //zaman adımı
 public sudeposu_boru_akis(double Ti,double Pi,double Patmi,double epsi,double di,double Doi, double Li,double Hi,double H1i,int
n_finitei,double KLi)
 {    n_finite=n_finitei;
              su=new liquid_CS("su");
   hava=new Gas("air");
   T=Ti;
   double rohava=hava.M/hava.v((T+273.15),Pi*1e-2);
   double rosu=su.ro(T);
   eps=epsi;
   d=di;
   Do=Doi;
   eod=eps/d;
   L=Li;
   H=Hi;
   h=new double[n_finite+1];
   h[0]=H1i;
   Vhava=new double[n_finite+1];

   V=new double[n_finite+1];
   V2=new double[n_finite+1];
   Phava=new double[n_finite+1];
   Phava[0]=Pi;
   Patm=Patmi;
   KL=KLi;
```

```
   Ahava=Math.PI*Do*Do/4.0;
   A=Math.PI*di*di/4.0;
   Aratio=A/Ahava;
   Vhava[0]=Ahava*(H-h[0]);
   mhava=Vhava[0]*rohava;
   System.out.println("mhava = "+mhava+" kg");
   System.out.println("rohava = "+rohava+" kg/m^3");
   System.out.println("Vhava = "+Vhava[0]+" m^3");
   g=9.81; //m/s^2
   gammasu=rosu*g;
   dt=1;
  }
  // boru içi
  public double f(double Re)
  {
  //Goudar equation
  double a=2/Math.log(10);
  double b=eod/3.7;
  double d=Math.log(10.0)/5.02*Re;
  double s=b*d+Math.log(d);
  double q=Math.pow(s,(s/(s+1)));
  double g=b*d+Math.log(d/q);
  double z=Math.log(q/g);
  double dLA=(g/(g+1))*z;
  double dCFA=dLA*(1+z/2.0/((g+1)*(g+1)+(z/3)*(2.0*g-1)));
  double f1=a*(Math.log(d/q)+dCFA);
  f1=1.0/(f1*f1);
  return f1;
  }
 // boru dışı ısı transferi
 public double Reout(double t,double v)
 {
 //======= Reynold sayısı boru dışı ==========
 //x : % karışım miktarı
 //t sıvı sıcaklığı
 //boru geçiş alanı
 double Dh=d;
 double ro=su.ro(t);
 double vis=su.vis(t);
 //System.out.println("vis="+vis/ro);
 double res=v*ro*Dh/vis;
 return res;
 }
  public static double secant(if_x f,double x0)
  { int nmax=500;
    double tolerance=1.0e-10;
    double fx1,dfx1;
    double x=x0;
    for(int i=0;i<nmax;i++)
    { fx1=f.func(x);
      dfx1=f.dfunc(x);
      x=x-fx1/dfx1;
      if(Math.abs(fx1)<tolerance) { return x;}
    }
    JOptionPane.showMessageDialog(null,"Warning : MAximum number of iteration is exceeded \n","MAXİMUM ITERATION
NUMBER WARNING",JOptionPane.WARNING_MESSAGE);
  return x;
  }
  public void calculate()
  {
                for(int i=0;i<n_finite;i++)
     {double pp=Phava[i]*1e3;
      double vv=10.0;
      double hh=h[i];

      ff=v->pp/gammasu+Aratio*v*Aratio*v/(2.0*g)+hh-(Patm*1e3/gammasu+v*v/(2.0*g)+(f(Reout(T,v))*L/d+KL)*v*v/(2.0*g));
      V[i]=secant(ff,vv);
      V2[i]=Aratio*V[i];
      double Re=Reout(T,V[i]);
      double ff=f(Re);
      System.out.println("t="+i*dt+"s V="+V[i]+"m/s h = "+h[i]+" m f = "+ff+" Re = "+Re);
      Vhava[i+1]=Vhava[i]+dt*V2[i]*Ahava;
      rohava=mhava/Vhava[i+1];
      Phava[i+1]=mhava*(8314.5/28.97)*(T+273.15)/Vhava[i+1]*1e-3;
```

```
      h[i+1]=h[i]-dt*V2[i];
    }
 }
 public static void main(String arg[])
 { sudeposu_boru_akis sdba=new sudeposu_boru_akis(20.0,300.0,100.0,0.4e-3,0.2,10.0,5.0,8.0,5.0,120,0.2);
   sdba.calculate();
 } }
```

```
---------- Capture Output ----------
> "D:\co\java\bin\javaw.exe" sudeposu_boru_akis
mhava = 839.9778757312198 kg
rohava = 3.5649768269454247 kg/m^3
Vhava = 235.61944901923448 m^3
t=0.0s V=16.708041102062847m/s h = 5.0 m f = 0.023476798315720825 Re = 3342389.2009554342
t=1.0s V=16.681425413399687m/s h = 4.993316783559175 m f = 0.023476887797448397 Re = 3337064.8191311196
t=2.0s V=16.656929405513416m/s h = 4.986644213393815 m f = 0.02347697040454885 Re = 3332164.472541986
t=3.0s V=16.63253191594394m/s h = 4.979981441631609 m f = 0.023477052920333044 Re = 3327283.8342212336
t=4.0s V=16.608232150184605m/s h = 4.973328428865232 m f = 0.023477135345636377 Re = 3322422.745230393
t=5.0s V=16.58402932285497m/s h = 4.966685136005158 m f = 0.02347721768128694 Re = 3317581.0484566656
t=6.0s V=16.559922657564837m/s h = 4.960051524276016 m f = 0.02347729992810565 Re = 3312758.5885857213
t=7.0s V=16.53591138678078m/s h = 4.953427555212991 m f = 0.023477382086906338 Re = 3307955.2120749974
t=8.0s V=16.51199475169512m/s h = 4.946813190658278 m f = 0.023477464158495845 Re = 3303170.767127491
t=9.0s V=16.48817200209729m/s h = 4.9402083927576 m f = 0.023477546143674078 Re = 3298405.10366602
t=10.0s V=16.464442396247446m/s h = 4.933613123956761 m f = 0.02347762804323416 Re = 3293658.0733079454
t=11.0s V=16.440805200752475m/s h = 4.927027346998262 m f = 0.02347770985796245 Re = 3288929.529340364
t=12.0s V=16.417259690444155m/s h = 4.920451024917961 m f = 0.023477791588638663 Re = 3284219.326695734
t=13.0s V=16.393805148259506m/s h = 4.913884121041783 m f = 0.02347787323603596 Re = 3279527.3219279437
t=14.0s V=16.370440865123307m/s h = 4.907326598982479 m f = 0.02347795480092101 Re = 3274853.3731888034
t=15.0s V=16.347166139832648m/s h = 4.90077842263643 m f = 0.023478036284054097 Re = 3270197.3402049546
t=16.0s V=16.323980278943615m/s h = 4.894239556180497 m f = 0.023478117686189164 Re = 3265559.0842552017
t=17.0s V=16.300882596659886m/s h = 4.88770996406892 m f = 0.023478199008073942 Re = 3260938.468148224
t=18.0s V=16.27787241472335m/s h = 4.881189611030256 m f = 0.023478280250449957 Re = 3256335.3562006964
t=19.0s V=16.254949062306622m/s h = 4.874678462064367 m f = 0.02347836141405269 Re = 3251749.614215784
t=20.0s V=16.232111875907446m/s h = 4.868176482439444 m f = 0.023478442499611594 Re = 3247181.1094620223
t=21.0s V=16.209360199244927m/s h = 4.861683637689081 m f = 0.02347852350785017 Re = 3242629.7106525563
t=22.0s V=16.18669338315761m/s h = 4.855199893609384 m f = 0.023478604439486083 Re = 3238095.287924752
t=23.0s V=16.16411078550325m/s h = 4.8487252162561205 m f = 0.023478685295231182 Re = 3233577.712820144
t=24.0s V=16.141611771060393m/s h = 4.842259571941919 m f = 0.023478766075791614 Re = 3229076.858264749
t=25.0s V=16.119195711431583m/s h = 4.835802927233495 m f = 0.023478846781867867 Re = 3224592.598549703
t=26.0s V=16.09686198494829m/s h = 4.8295552489489215 m f = 0.023478927414154842 Re = 3220124.809312242
t=27.0s V=16.074609976577428m/s h = 4.822916504154942 m f = 0.023479007973341934 Re = 3215673.3675170005
t=28.0s V=16.052439077829458m/s h = 4.816486660164311 m f = 0.023479088460113064 Re = 3211238.151437627
t=29.0s V=16.0303486866681m/s h = 4.810065684533179 m f = 0.023479168875146828 Re = 3206819.040638726
t=30.0s V=16.008338207421517m/s h = 4.803653545058512 m f = 0.023479249219116427 Re = 3202415.915958087
t=31.0s V=15.98640705069504m/s h = 4.797250209775544 m f = 0.023479329492689875 Re = 3198028.6594892256
t=32.0s V=15.964554633285358m/s h = 4.790855646955266 m f = 0.023479409696529972 Re = 3193657.154564218
t=33.0s V=15.942780378096115m/s h = 4.784469825101952 m f = 0.02347948983129437 Re = 3189301.285736819
t=34.0s V=15.921083714054966m/s h = 4.778092712950714 m f = 0.02347956989763568 Re = 3184960.938765866
t=35.0s V=15.899464076031977m/s h = 4.771724279465092 m f = 0.02347964989620149 Re = 3180636.000598957
t=36.0s V=15.877920904759419m/s h = 4.765364493834679 m f = 0.02347972982763444 Re = 3176326.3593564034
t=37.0s V=15.856453646752842m/s h = 4.759013325472775 m f = 0.023479809692572292 Re = 3172031.9043154437
t=38.0s V=15.835061754233509m/s h = 4.7526707440140745 m f = 0.02347988949164796 Re = 3167752.5258947266
t=39.0s V=15.813744685052056m/s h = 4.7463367193123815 m f = 0.02347996922548958 Re = 3163488.115639038
t=40.0s V=15.792501902613447m/s h = 4.740011221438361 m f = 0.023480048894720596 Re = 3159238.566204288
t=41.0s V=15.771332875803136m/s h = 4.733694220677315 m f = 0.02348012849995975 Re = 3155003.771342742
t=42.0s V=15.75023707891443m/s h = 4.727385687526994 m f = 0.02348020804182119 Re = 3150783.625888488
t=43.0s V=15.72921399157705m/s h = 4.721085592695428 m f = 0.023480287520914515 Re = 3146578.025743147
t=44.0s V=15.708263098686835m/s h = 4.714793907098797 m f = 0.023480366937844808 Re = 3142386.8678618087
t=45.0s V=15.687383890336601m/s h = 4.708510601859323 m f = 0.023480446293212703 Re = 3138210.0502392026
t=46.0s V=15.666575861748093m/s h = 4.702235648303188 m f = 0.023480525587614428 Re = 3134047.4718960826
t=47.0s V=15.645838513205042m/s h = 4.695969017958489 m f = 0.02348060482164186 Re = 3129899.0328658377
t=48.0s V=15.6251713499873m/s h = 4.689710682553207 m f = 0.023480683995882597 Re = 3125764.634181312
t=49.0s V=15.604573882306024m/s h = 4.683460614013212 m f = 0.02348076311091994 Re = 3121644.177861844
t=50.0s V=15.584045625239874m/s h = 4.6772187844602895 m f = 0.023480842167333035 Re = 3117537.566900499
t=51.0s V=15.56358609867225m/s h = 4.670985166210193 m f = 0.02348092165696847 Re = 3113444.705251515
t=52.0s V=15.543194827229526m/s h = 4.664759731770724 m f = 0.023481000106582238 Re = 3109365.4978179457
t=53.0s V=15.522871340220219m/s h = 4.658542453839832 m f = 0.023481078990556 Re = 3105299.8504394935
t=54.0s V=15.502615171575158m/s h = 4.652333305303744 m f = 0.02348115781818092 Re = 3101247.66988054
t=55.0s V=15.482425859788583m/s h = 4.646132259235114 m f = 0.02348123659001581 Re = 3097208.8638183596
t=56.0s V=15.462302947860138m/s h = 4.639939288891198 m f = 0.023481315306615565 Re = 3093183.34083152
t=57.0s V=15.442245983237811m/s h = 4.633754367712054 m f = 0.02348139396853117 Re = 3089171.0103884656
t=58.0s V=15.422254517761722m/s h = 4.6275774693187595 m f = 0.023481472576309805 Re = 3085171.782836272
```

t=59.0s V=15.402328107608822m/s h = 4.621408567511655 m f = 0.023481551130494834 Re = 3081185.5693895835
t=60.0s V=15.382466313238403m/s h = 4.615247636268611 m f = 0.023481629631625867 Re = 3077212.282119714
t=61.0s V=15.362668699338498m/s h = 4.609094649743316 m f = 0.023481708080238806 Re = 3073251.8339439216
t=62.0s V=15.342934834773047m/s h = 4.60294958226358 m f = 0.023481786476865895 Re = 3069304.138614842
t=63.0s V=15.323264292529943m/s h = 4.596812408329671 m f = 0.023481864822035705 Re = 3065369.1107100947
t=64.0s V=15.303656649669799m/s h = 4.590683102612659 m f = 0.023481943116273267 Re = 3061446.665622032
t=65.0s V=15.284111487275554m/s h = 4.584561639952791 m f = 0.02348202136010004 Re = 3057536.719547661
t=66.0s V=15.264628390402816m/s h = 4.578447995357881 m f = 0.023482099554033957 Re = 3053639.189478707
t=67.0s V=15.245206948030962m/s h = 4.57234214400172 m f = 0.0234821776985895 Re = 3049753.9931918336
t=68.0s V=15.225846753014972m/s h = 4.566244061222507 m f = 0.023482255794277686 Re = 3045881.0492390054
t=69.0s V=15.206547402038018m/s h = 4.560153722521301 m f = 0.02348233384160617 Re = 3042020.276938005
t=70.0s V=15.187308495564722m/s h = 4.554071103560486 m f = 0.023482411841079222 Re = 3038171.596363081
t=71.0s V=15.168129637795147m/s h = 4.54799618016226 m f = 0.023482489793197784 Re = 3034334.9283357463
t=72.0s V=15.149010436619458m/s h = 4.541928928307142 m f = 0.023482567698459515 Re = 3030510.194415704
t=73.0s V=15.129950503573275m/s h = 4.535869324132494 m f = 0.02348264555735883 Re = 3026697.3168919208
t=74.0s V=15.11094945379366m/s h = 4.5298173439310645 m f = 0.02348272337038691 Re = 3022896.2187738204
t=75.0s V=15.092006905975806m/s h = 4.523772964149547 m f = 0.023482801138031754 Re = 3019106.8237826168
t=76.0s V=15.073122482330307m/s h = 4.517736161387156 m f = 0.023482878860778215 Re = 3015329.056342771
t=77.0s V=15.054295808541113m/s h = 4.511706912394224 m f = 0.023482956539108035 Re = 3011562.841573576
t=78.0s V=15.035526513724049m/s h = 4.505685194070808 m f = 0.023483034173499853 Re = 3007808.1052808594
t=79.0s V=15.016814230385984m/s h = 4.499670983465318 m f = 0.02348311176442928 Re = 3004064.7739488194
t=80.0s V=14.998158594384602m/s h = 4.493664257773164 m f = 0.023483189312368896 Re = 3000332.774731971
t=81.0s V=14.979559244888693m/s h = 4.48766499433541 m f = 0.023483266817788295 Re = 2996612.035447205
t=82.0s V=14.961015824339116m/s h = 4.481673170637455 m f = 0.023483344281154124 Re = 2992902.484565981
t=83.0s V=14.942527978410245m/s h = 4.475688764307719 m f = 0.0234834217029301 Re = 2989204.051206612
t=84.0s V=14.924095355972028m/s h = 4.469711753116355 m f = 0.023483499083577046 Re = 2985516.6651266734
t=85.0s V=14.90571760905257m/s h = 4.463742114973966 m f = 0.023483576423552935 Re = 2981840.256715522
t=86.0s V=14.887394392801237m/s h = 4.457779827930345 m f = 0.02348365372331289 Re = 2978174.7569869114
t=87.0s V=14.869125365452346m/s h = 4.451824870173224 m f = 0.02348373098330924 Re = 2974520.0975717297
t=88.0s V=14.850910188289294m/s h = 4.445877220027043 m f = 0.02348380820399155 Re = 2970876.2107108245
t=89.0s V=14.832748525609274m/s h = 4.439936855951728 m f = 0.023483885385806626 Re = 2967243.029247942
t=90.0s V=14.814640044688442m/s h = 4.434003756541484 m f = 0.023483962529198564 Re = 2963620.486622761
t=91.0s V=14.796584415747608m/s h = 4.428077900523609 m f = 0.02348403963460877 Re = 2960008.516864029
t=92.0s V=14.778581311918385m/s h = 4.422159266757309 m f = 0.023484116702476004 Re = 2956407.05458279
t=93.0s V=14.760630409209856m/s h = 4.416247834232542 m f = 0.023484193733236368 Re = 2952816.034965716
t=94.0s V=14.742731386475665m/s h = 4.410343582068858 m f = 0.02348427072732338 Re = 2949235.3937685243
t=95.0s V=14.724883925381597m/s h = 4.404446489514267 m f = 0.023484347685167967 Re = 2945665.0673094923
t=96.0s V=14.70708771037361m/s h = 4.398556535944115 m f = 0.023484424607198528 Re = 2942104.992463062
t=97.0s V=14.689342428646306m/s h = 4.392673700859966 m f = 0.023484501493840907 Re = 2938555.1066535325
t=98.0s V=14.671647770111836m/s h = 4.386797963888507 m f = 0.02348457834551846 Re = 2935015.347848839
t=99.0s V=14.654003427369265m/s h = 4.380929304780462 m f = 0.023484655162652066 Re = 2931485.6545544257
t=100.0s V=14.636409095674322m/s h = 4.375067703409514 m f = 0.023484731945660167 Re = 2927965.965807195
t=101.0s V=14.618864472909596m/s h = 4.369213139771245 m f = 0.023484808694958774 Re = 2924456.2211695416
t=102.0s V=14.601369259555144m/s h = 4.36336559398208 m f = 0.02348488541096149 Re = 2920956.360723477
t=103.0s V=14.583923158659463m/s h = 4.357525046278258 m f = 0.023484962094079558 Re = 2917466.32506482
t=104.0s V=14.566525875810932m/s h = 4.3516914770147945 m f = 0.02348503874472185 Re = 2913986.055297485
t=105.0s V=14.549177119109572m/s h = 4.345864866664471 m f = 0.023485115363294944 Re = 2910515.4930278324
t=106.0s V=14.531876599139231m/s h = 4.340045195816827 m f = 0.02348519195020308 5 Re = 2907054.580359103
t=107.0s V=14.514624028940146m/s h = 4.334232445177171 m f = 0.02348526850584823 Re = 2903603.259885931
t=108.0s V=14.497419123981853m/s h = 4.3284265955655945 m f = 0.023485345030630125 Re = 2900161.474688924
t=109.0s V=14.48026160213651m/s h = 4.322627627916002 m f = 0.02348542152494624 Re = 2896729.1683293264
t=110.0s V=14.463151183652515m/s h = 4.316835523275147 m f = 0.023485497989191834 Re = 2893306.284843741
t=111.0s V=14.446087591128554m/s h = 4.311050262801686 m f = 0.02348557442376 Re = 2889892.768738939
t=112.0s V=14.429070549487918m/s h = 4.305271827765234 m f = 0.023485650829041648 Re = 2886488.5649867193
t=113.0s V=14.412099785953238m/s h = 4.299500199545439 m f = 0.02348572720542554 Re = 2883093.61901886
t=114.0s V=14.395175030021525m/s h = 4.293735359631057 m f = 0.023485803553298323 Re = 2879707.876722118
t=115.0s V=14.378296013439503m/s h = 4.2879772896190484 m f = 0.023485879873044513 Re = 2876331.2844332987
t=116.0s V=14.36146247017937m/s h = 4.2822259712136725 m f = 0.023485956165046593 Re = 2872963.7889344078
t=117.0s V=14.344674136414769m/s h = 4.2764813862256 m f = 0.023486032429684923 Re = 2869605.337447841
t=118.0s V=14.327930750497151m/s h = 4.270743516571034 m f = 0.023486108667337857 Re = 2866255.8776316587
t=119.0s V=14.311232052932434m/s h = 4.265012344270835 m f = 0.02348618487838174 Re = 2862915.3575749155

> Terminated with exit code 0.

3 LINEAR SYSTEM OF EQUATIONS

3.1 LINEAR SYSTEM OF EQUATIONS GENERAL INFORMATIONS

Linear system of equations is a collection of linear equations of the same set of variables. For example

3x+2y-z=1
2x-2y+4z=-2
-x+0.5y-z=0

Is system of three equation in 3 variables x,y and z. A solution to a linear system is an assignment of numbers to the variables such that all the equations are simultaneously satisfied. A solution to the system above is given by

x=1,y=-2,z=-2

This system of equations can be written in matrix multiplication form as:

$$\begin{bmatrix} 3 & 2 & -1 \\ 2 & -2 & 4 \\ -1 & 0.5 & -1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \end{bmatrix} = \begin{bmatrix} 1 \\ -2 \\ 0 \end{bmatrix}$$

In more general form linear system of equations are in the following form:

$\mathbf{a_{11} X_1 + a_{12} X_2 + a_{13} X_3 + \ldots + a_{1n} X_n = b_1}$
$\mathbf{a_{21} X_1 + a_{22} X_2 + a_{23} X_3 + \ldots + a_{2n} X_n = b_2}$
$\mathbf{a_{31} X_1 + a_{32} X_2 + a_{33} X_3 + \ldots + a_{3n} X_n = b_3}$
.....
$\mathbf{a_{n1} X_1 + a_{n2} X_2 + a_{n3} X_3 + \ldots + a_{nn} X_n = b_n}$

The same set of equation can be given in matrix notation

$[A]\{X\}=\{B\}$ or

$\vec{A}\vec{x}=\vec{B}$

If matrix notation is written in open form

$$\begin{bmatrix} a_{11} & a_{12} & \ldots & a_{1n} \\ a_{21} & a_{22} & \ldots & a_{2n} \\ \ldots & \ldots & \ldots & \ldots \\ a_{n1} & a_{n2} & \ldots & a_{nn} \end{bmatrix} \begin{bmatrix} X_1 \\ X_2 \\ \ldots \\ X_n \end{bmatrix} = \begin{bmatrix} b_1 \\ b_2 \\ \ldots \\ b_n \end{bmatrix}$$

In some sources the same system of equation can also be given as:

$$\left[\begin{array}{cccc|c} a_{11} & a_{12} & \ldots & a_{1n} & b_1 \\ a_{21} & a_{22} & \ldots & a_{2n} & b_2 \\ \ldots & \ldots & \ldots & \ldots & \ldots \\ a_{n1} & a_{n2} & \ldots & a_{nn} & b_n \end{array}\right] \text{ or } \begin{bmatrix} a_{11} & a_{12} & \ldots & a_{1n} & b_1 \\ a_{21} & a_{22} & \ldots & a_{2n} & b_2 \\ \ldots & \ldots & \ldots & \ldots & \ldots \\ a_{n1} & a_{n2} & \ldots & a_{nn} & b_n \end{bmatrix}$$

In this set A and B is known values and X is unknown vector. Finding of X values are colled solution of system of equations.

3.2 GAUSS ELIMINATION METHOD

Gauss elimination method based on the properties of the linear system of equations. Changing the matrix rows, adding one row to another, or multiplying one row with a constant will not change the solution values of the system of equations. Therefore a process such as

(row i)- constant * (row j) can be easily applied without changing the solution set of the linear system of equations. The purpose of this manipulations is to change the A matrix to a upper diagonal matrix. For example if n=5, system of equation will take the following upper diagonal form

$$\begin{bmatrix} a'_{11} & a'_{12} & a'_{13} & a'_{14} & a'_{14} \\ 0 & a'_{22} & a'_{23} & a'_{24} & a'_{25} \\ 0 & 0 & a'_{33} & a'_{34} & a'_{35} \\ 0 & 0 & 0 & a'_{44} & a'_{45} \\ 0 & 0 & 0 & 0 & a'_{55} \end{bmatrix} \begin{bmatrix} X_1 \\ X_1 \\ X_1 \\ X_1 \\ X_1 \end{bmatrix} = \begin{bmatrix} b'_1 \\ b'_2 \\ b'_3 \\ b'_4 \\ b'_5 \end{bmatrix}$$

A systematic elimination processs is applied for this purpose. For example in order to change element a_{21} to 0 in the equation

$a'_{21} = 0 = a_{21} - C_1 * a_{11}$

a constant with the value $C_1=(a_{21}/a_{11})$ is required. Then each member of the second row will be multiply with the same constant and take a new value.

$a'_{22} = a_{22} - C_1*a_{12}$

$a'_{23} = a_{23} - C_1*a_{13}$

……..

$a'_{25} = a_{25} - C_1*a_{15}$

$b'_2=b_2 - C_1*b_1$

At the end of this process a'_{21} become zero. The same process repeat first repeat the whole rows under element a_{11} and all first column rows except the first one become zero. Then the same process repeat for the rows under the diagonal element a_{22} . A for structure doing this process will look like as follows

```
for(k=0;k<(n-1);k++)
 {   for(i=k+1;i<n;i++)
       {  C[k]=a[i][k]/a[k][k];
          a[i][k]=0;
          for(j=k+1;j<n;j++)  {   a[i][j]-= C[k]*a[k][j]; }
          b[i]    =b[i]  - C[k]*b[k];
      }}
```

After the linear system of equation converted into upper diagonal form, unnowns are solved by using back-substitution method. The set of the back-substitution for the set given above(n=5) will be:

$X_5 = b'_5 / a'_{55}$

$X_4 = (b'_4 - a'_{45}*X_5)/a'_{44}$

$X_3 = (b'_3 - a'_{34}*X_4 - a'_{35}*X_5)/a'_{33}$

$X_2 = (b'_2 - a'_{23}*X_3 - a'_{24}*X_3 - a'_{25}*X_5)/a'_{22}$

$X_1 = (b'_1 - a'_{12}*X_2 - a'_{13}*X_3 - a'_{14}*X_4 - a'_{15}*X_5)/a'_{11}$

Java code equivalent for an n by n matrix

```
 x[n-1]=b[n-1]/a[n-1][n-1];
 for(i=n-2;i>=0;i--)
 {
   toplam=0;
   for(j=i+1;j<n;j++)
   {  toplam+=a[i][j]*x[j];}
   x[i]=(b[i]-toplam)/a[i][i];
 }
```

A matrix output class is also created in here to get easy outputs from the matrices.

PROGRAM 3.2-1 matris output program matrixoutput.java to format the system of equation

```
import java.io.*;
import java.text.*;
import java.util.Locale;

public class matrixoutput
{
public static String toString(double left, int w, int d)
// convert double into a w digit string with d digits in real part
{
            NumberFormat df=NumberFormat.getInstance(Locale.US);
            df.setMaximumFractionDigits(d);
   df.setMinimumFractionDigits(d);
   df.setGroupingUsed(false);
   String s = df.format(left);
```

```
    while (s.length() < w)
      s = " " + s;
    if (s.length() > w)
    {
       s = "";
       for (int i=0; i<w; i++)
         s = s + "-";
    }
    return s;
}

public static String toString(double left)
{return toString(left,25,15);}

public static String toString(double[][] left,int w,int d)
{
//string equivalent of a double matris
int n,m;
String b;
b="";
n=left.length;
m=left[0].length;
for(int i=0;i<n;i++)
  {
  for(int j=0;j<m;j++)
    {
    b=b+toString(left[i][j],w,d);
    }
    b=b+"\n";
  }
return b;
}

public static String toString(double[][] a,double b[],int w,int d)
{//string equivalent of a lineear system of equation
int n,m;
String s;
s="";
n=a.length;
m=a[0].length;
for(int i=0;i<n;i++)
  {
  s+=" |";
  for(int j=0;j<m;j++)
    {
    s+=toString(a[i][j],w,d);
    }
    s+="| "+toString(b[i],w,d)+"| \n";
  }
return s;
}

public static String toString(double[][] a,double b[])
{//string equivalent of a lineear system of equation
return toString(a,b,25,15);
}

public static String toStringT(double[][] left,int w,int d)
{ return toString(Matrix.Transpose(left),w,d);}

public static String toStringT(double[][] left)
{ return toString(Matrix.Transpose(left),15,10);}
```

```
public static String toStringT(double[] left)
{
// returns a vertical string representation
// of a double vector
int n,m;
String b;
b="";
n=left.length;
for(int i=0;i<n;i++)
  {
    b=b+toString(left[i])+"\n";
  }
return b;
}

public static String toString(double[] left)
{
// returns a horizontal string representation
// of a double vector
int n,m;
String b;
b="";
n=left.length;
for(int i=0;i<n;i++)
  {
    b=b+toString(left[i])+"\t ";
  }
  b=b+"\n";
return b;
}}
```

Now A complete Gauss elimination code can be developed

PROGRAM 3.2-2 Gauss elimination

```
import java.io.*;

class NA14
{
  public static double[] gauss(double a[][],double b[])
  {
  int n=b.length;
  double x[]=new double[n];
  double carpan=0;
  double toplam=0;
  //gauss eleme
  int i,j,k;
  System.out.println("Original matrix :\n"+Matrix.toString(a));
  for(k=0;k<(n-1);k++)
  {
    for(i=k+1;i<n;i++)
    {  carpan=a[i][k]/a[k][k];
       a[i][k]=0;
       for(j=k+1;j<n;j++)
       {   a[i][j]-=carpan*a[k][j]; }
           b[i]   =b[i]   -carpan*b[k];
                }
  }
  System.out.println("Eliminated matrix :\n"+Matrix.toString(a));
  //back substituting
  x[n-1]=b[n-1]/a[n-1][n-1];
  for(i=n-2;i>=0;i--)
  {
    toplam=0;
```

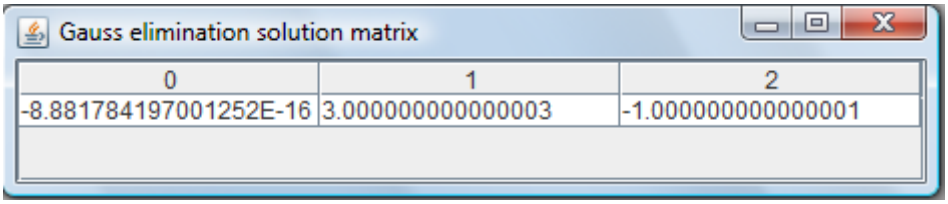
```
    for(j=i+1;j<n;j++)
    {  toplam+=a[i][j]*x[j];}
    x[i]=(b[i]-toplam)/a[i][i];
  }
  return x;
  }

  public static void main(String args[]) throws IOException
  {
  double a[][]={{1,2,5},{2,1,2},{3,1,1}};
  double b[]={1,1,2};

  double x[]=gauss(a,b);
  Text.print(x,"Gauss elimination solution matrix");
  }
}
```

output 3.2-1 Gauss elimination process

```
---------- Capture Output ----------
> "C:\turhan\java\bin\javaw.exe" NA14
Original matrix :
      1.000000000000000      2.000000000000000      5.000000000000000
      2.000000000000000      1.000000000000000      2.000000000000000
      3.000000000000000      1.000000000000000      1.000000000000000
Eliminated matrix :
      1.000000000000000      2.000000000000000      5.000000000000000
      0.000000000000000     -3.000000000000000     -8.000000000000000
      0.000000000000000      0.000000000000000     -0.666666666666666
      0.000000000000000      0.000000000000000     -0.666666666666666
> Terminated with exit code 0.
```



In the program following set of system of equation is taken :

$$\begin{bmatrix}1 & 2 & 5\\ 2 & 1 & 2\\ 3 & 1 & 1\end{bmatrix}\begin{bmatrix}X_0\\ X_1\\ X_2\end{bmatrix}=\begin{bmatrix}1\\ 1\\ 2\end{bmatrix}$$

The solution set of this set of equation is 0,3 ve -1. As it is seen from the results, a small error is remained in the solution set. If the same elinination process is carried out by hand:

column 0(column indices starts from 0), row 1(raw indices starts from 0) multiplier $C_1=a_{10}/a_{00} = 2/1=2$

$a_{10} = a_{10} - C_1 * a_{00} = a_{10} - (a_{10}/a_{00})* a_{00} = 2 - 2*1 = 0$

$a_{11} = a_{11} - C_1 * a_{01} = a_{11} - (a_{10}/a_{00})* a_{01} = 1 - 2*2 = -3$

$a_{12} = a_{12} - C_1 * a_{02} = a_{12} - (a_{10}/a_{00})* a_{02} = 2 - 2*5 = -8$

$b_2 = b_2 - C_1 * b_1 = b_2 - (a_{10}/a_{00})* b_1 = 1 - 2*1 = -1$

column0 raw 1 elimination is completed

colun0 raw 2 elimination:

$C_1=a_{20}/a_{00} = 3/1=3$

$A_{20} = a_{20} - C_2 * a_{00} = a_{20} - (a_{20}/a_{00})* a_{00} = 3 - 3*1 = 0$

$a_{21} = a_{21} - C_2 * a_{01} = a_{21} - (a_{20}/a_{00})* a_{01} = 1 - 3*2 = -5$

$a_{22} = a_{22} - C_2 * a_{02} = a_{22} - (a_{20}/a_{00})* a_{02} = 2 - 3*5 = -14$

$b_2 = b_2 - C_2 * b_1 = b_2 - (a_{10}/a_{00})* b_1 = 1 - 3*1 = -1$

column 1, row 2 multiplier C

$C_3=a_{21}/a_{11}$ = -5/-3=(5/3)

a_{21} = $a_{21} – C_3 * a_{11} = a_{21}$ - $(a_{21}/a_{11})* a_{11}$ = -5 – (5/3)*(-3) = 0

a_{22} = a_{22} - $C_3 * a_{12} = a_{11}$ - $(a_{21}/a_{11})* a_{01}$ = -14 – (5/3)*(-8) = -0.6666666666666

b_2 = b_2 - $C_3 * b_1 = b_2$ - $(a_{21}/a_{11})* b_1$ = -1 – (5/3)*(-1) = 0.6666666666666

system of equation becomes:

$$\begin{bmatrix}1 & 2 & 5\\ 0 & -3 & -8\\ 0 & 0 & -0.66666\end{bmatrix}\begin{bmatrix}X_0\\ X_1\\ X_2\end{bmatrix}=\begin{bmatrix}1\\ -1\\ 0.66666\end{bmatrix}$$

Back-substitution process :

$X_2 = b_2/a_{22}$ = 0.6666666666/(-0.6666666666) = -1

$X_1 = (b_1 - a_{12}*X_2)/a_{11}$ = (-1 - (-8)*(-1))/(-3) =(-9)/(-3)=3

$X_0 = (b_0 - a_{01}*X_1 - a_{02}*X_2)/a_{00}$ = (1 – 2*3-5*(-1))/1=0

$$\begin{bmatrix}X_0\\ X_1\\ X_2\end{bmatrix}=\begin{bmatrix}0\\ 3\\ -1\end{bmatrix}$$

Another example for Gauss elimination with detailed steps :

$$\begin{bmatrix}7 & 1 & 2\\ -1 & 4 & -1\\ 3 & 15 & 20\end{bmatrix}\begin{bmatrix}X_0\\ X_1\\ X_2\end{bmatrix}=\begin{bmatrix}47\\ 19\\ 87\end{bmatrix}$$

	A			B
	7	1	2	47
	-1	4	-1	19
	3	15	20	87

c				
-0.14286	7	1	2	47
0.428571	0	4.142857	-0.71429	25.71429
	0	14.57143	19.14286	66.85714

c				
3.517241	7	1	2	47
	0	4.142857	-0.71429	25.71429
	0	0	21.65517	-23.5862

X_0	**6.165605**
X_1	**6.019108**
X_2	**-1.08917**

In excel or Octave gauss elimination can also be carried out by using MMULTIPLY and MINVERSE functions (DİZEY_TERS and DÇARP in turkish version of excel)

= MINVERSE (B3:D5)	ctrl-shift-enter	=DİZEY_TERS(B3:D5)	ctrl-shift-enter
= MMULTIPLY(B7:D9;E3:E5)	ctrl-shift-enter	=DÇARP(B7:D9;E3:E5)	ctrl-shift-enter

$$\begin{bmatrix}-8 & 1 & 2\\ 5 & 7 & -3\\ 2 & 1 & -2\end{bmatrix}\begin{bmatrix}X_0\\ X_1\\ X_2\end{bmatrix}=\begin{bmatrix}0\\ 1\\ -2\end{bmatrix}$$

Input Matrix

-8.000E+00	1.000E+00	2.000E+00	0.000E+00
5.000E+00	7.000E+00	-3.000E+00	1.000E+01
2.000E+00	1.000E+00	-2.000E+00	-2.000E+00

Inverse Matrix — turkish fx — english fx

-1.486E-01	5.405E-02	-2.297E-01	DİZEY_TERS	MINVERSE	ctrl-shift-enter
5.405E-02	1.622E-01	-1.892E-01			
-1.216E-01	1.351E-01	-8.243E-01			

Solution

1.000E+00	DÇARP	MMULTIPLY	ctrl-shift-enter
2.000E+00			
3.000E+00			

The same program can be given as Matlab(Octave) code as:

PROGRAM 3.2-3 Gauss elimination in Octave(Matlab)

```
function x=gauss(a,b)
% Gauss elimination
n=length(b);
carpan=0;
toplam=0;
 %gauss eleme
 for k=1:n
   for i=k+1:n
     carpan=a(i,k)/a(k,k);
      a(i,k)=0;
      for j=k+1:n
        a(i,j)=a(i,j)-carpan*a(k,j);
      end
         b(i)   =b(i)   -carpan*b(k);
   end
 end
 %back substituting
 x(n)=b(n)/a(n,n);
 for i=n-1:-1:1
   toplam=0;
   for j=i+1:n
     toplam=toplam+a(i,j)*x(j);end
   x(i)=(b(i)-toplam)/a(i,i);
 end
end
```

```
>> a=[7 1 2;-1 4 -1;3 15 20]
a =
    7    1    2
   -1    4   -1
    3   15   20
>> b=[47 19 87]'
b =
   47
   19
   87
>> x=gauss(a,b)
x =
   6.1656   6.0191  -1.0892

>> xx=a\b
xx =
   6.1656
   6.0191
  -1.0892
>>
```

The same program can be given as Python code as:

PROGRAM 3.2-4 Gauss Elimination process in Python language:

```
from math import *

def gauss(a,b):
  n=len(a)
  x=[]
  print("origial matrix :\na =",a,"\nb =",b)
  for i in range(n):
    x.append(0.0)
  for k in range(len(a)):
    for i in range(k+1,n):
      carpan=a[i][k]/a[k][k]
```

```
      a[i][k]=0;
      for j in range(k+1,n):
        a[i][j]=a[i][j]-carpan*a[k][j]
      b[i]   =b[i]-carpan*b[k]
  print("eliminated matrix :\na =",a,"\nb =",b)
  x[n-1]=b[n-1]/a[n-1][n-1]
  for i in range(n-1,0,-1):
    toplam=0;
    for j in range(i+1,n):
      toplam=toplam+a[i][j]*x[j]
     x[i]=(b[i]-toplam)/a[i][i]
  return x;

a=[[1,2,5],[2,1,2],[3,1,1]]
b=[1,1,2]
x=gauss(a,b);
print("Gauss elimination solution matrix :\nx = ",x)
```

```
---------- Capture Output ----------
> "D:\co\python\pythonw.exe" NA14.py
origial matrix :
a = [[1, 2, 5], [2, 1, 2], [3, 1, 1]]
b = [1, 1, 2]
eliminated matrix :
a = [[1, 2, 5], [0, -3.0, -8.0], [0, 0, -0.6666666666666661]]
b = [1, -1.0, 0.6666666666666667]
Gauss elimination solution matrix :
x =  [0.0, 3.000000000000003, -1.000000000000001]

> Terminated with exit code 0.
```

PROGRAM 3.2.5 Gauss elimination in C++ using pointers to pass matrix and vector to method

```
#include <iostream>
#include <cmath>
#include <vector>

using namespace std;

void print(double *a,double *b,int n,int m)
{
   //int n = a.size();
   for (int i=0; i<n; i++) {
//       for (int j=0; j<m; j++) {cout << *(a+i*n+j) << "\t";}
       for (int j=0; j<m; j++) {cout << *(a+i*n+j) << "\t";}
       cout << "| "<<*(b+i)<<"\n";
   }
   cout<<"\n";
}

void print(double *b,int m) {
   for (int j=0; j<m; j++) {
       cout <<*(b+j)<< "\t";
   }
   cout<<"\n";
}
double* pivotlugaus(double *a,double *b,int n)
{ //gauss elimination with partial pivoting
  double x[n];
  double carpan=0;
  double toplam=0;
  double buyuk;
  double dummy;
  //gauss eleme
```

```
  int i,j,k,p,ii,jj;
  for(k=0;k<(n-1);k++)
  { //Solving gauss elimination
    for(i=k+1;i<n;i++)
    {  carpan=(*(a+i*n+k))/(*(a+k*n+k));
       *(a+i*n+k)=0;
       for(j=k+1;j<n;j++)
       {   *(a+n*i+j)-=carpan**(a+n*k+j); }
           *(b+i)   -=carpan**(b+k) ;
    }
  }
  //back substitution
  cout<<"elimination matrix\n";
  print(a,b,n,n);
  *(x+n-1)=(*(b+n-1))/(*(a+n*(n-1)+(n-1)));
  cout<<"i="<<(n-1)<<"x="<<(*(x+n-1))<<endl;
  for(i=n-2;i>=0;i--)
  {
    toplam=0;
    for(j=i+1;j<n;j++)
    {  toplam+=(*(a+n*i+j))*(*(x+j));}
    *(x+i)=((*(b+i))-toplam)/(*(a+i*n+i));
    cout<<"i="<<i<<"x="<<(*(x+i))<<endl;
  }
  print(x,n);
  return x;
  }
int main()
{
              int n=3;//row
    int m=3;//column
    double a[n][m]={{1,2,5},{2,1,2},{3,1,1}};
    double b[n]={1,1,2};
    double *x=pivotlugaus(&a[0][0],&b[0],n);
    cout<<"x[0]="<<x[0]<<" x[1]= "<<x[1]<<endl;
}
```

```
---------- Capture Output ----------
> "D:\SCO1\run.bat" gauss

D:\co\SCO_CPP>gauss
elimination matrix
1            2            5            | 1
0            -3           -8           | -1
0            0            -0.666667 | 0.666667
i=2x=-1
i=1x=3
i=0x=0
> Terminated with exit code -1073741819.
```

Gauss elimination can be applied also to complex variables. A complex version of the Gauss elimination routine is given below:

PROGRAM 3.2-6 complex Gauss elimination

```
 import java.io.*;

class NA14C
{
  public static c[] c_gauss(c a[][],c b[])
  {
  int n=b.length;
  c x[]=new c[n];
  c carpan=new c(1.0,0.0);
  c toplam=new c(0.0,0.0);
```

```
  //gauss eleme
  int i,j,k;
  for(k=0;k<(n-1);k++)
  {
    for(i=k+1;i<n;i++)
    {  carpan=c.p(a[i][k],"/",a[k][k]);
       a[i][k]=new c(0.0,0.0);
       for(j=k+1;j<n;j++)
       {   a[i][j]=c.p(a[i][j],"-",c.p(carpan,"*",a[k][j])); }
           b[i]=c.p(b[i],"-",c.p(carpan,"*",b[k]));
                }
  System.out.println("k = "+k+" :\n"+c.toString(a));
  }
  //back substituting
  x[n-1]=c.p(b[n-1],"/",a[n-1][n-1]);
  for(i=n-2;i>=0;i--)
  {
    toplam=new c(0.0,0.0);
    for(j=i+1;j<n;j++)
    {  toplam=c.p(toplam,"+",c.p(a[i][j],"*",x[j]));}
    x[i]=c.p(c.p(b[i],"-",toplam),"/",a[i][i]);
  }
  return x;
  }


  public static void main(String args[]) throws IOException
  {
  String s1="{(1.2,2.3),(1.1,-1.543),(3.77,0)},{(-1,1),(3.25,0),(1,0)},{(-1.75,0),(1.25,0.5),(1,0.7)}";
  String s2="{(1.0,2.0),(1.1,0),(1.2,1.3)}";
  c a[][]=c.toComplexMatrix(s1);
  c b[]=c.toComplexVector(s2);
  System.out.println("Original matrix :\n"+c.toString(a));
  c x[]=c_gauss(a,b);
  System.out.println("x = \n"+c.toString(x));
  }
}
```

```
 ---------- Capture Output ----------
> "C:\java\bin\javaw.exe" NA14C
Original matrix :
 (  1.200000 +   2.300000*i )  (  1.100000 -   1.543000*i )  (  3.770000 +   0.000000*i )
 ( -1.000000 +   1.000000*i )  (  3.250000 +   0.000000*i )  (  1.000000 +   0.000000*i )
 ( -1.750000 +   0.000000*i )  (  1.250000 +   0.500000*i )  (  1.000000 +   0.700000*i )

k = 0 :
 (  1.200000 +   2.300000*i )  (  1.100000 -   1.543000*i )  (  3.770000 +   0.000000*i )
 (  0.000000 +   0.000000*i )  (  2.267756 -   0.319866*i )  (  0.383804 -   1.960624*i )
 (  0.000000 +   0.000000*i )  (  0.670420 -   0.639346*i )  (  2.176374 -   1.554718*i )

k = 1 :
 (  1.200000 +   2.300000*i )  (  1.100000 -   1.543000*i )  (  3.770000 +   0.000000*i )
 (  0.000000 +   0.000000*i )  (  2.267756 -   0.319866*i )  (  0.383804 -   1.960624*i )
 (  0.000000 +   0.000000*i )  (  0.000000 +   0.000000*i )  (  2.511972 -   0.819554*i )

x =
 ( -0.809176 -   0.009645*i )  ( -0.088646 -   0.064709*i )  (  0.569280 +   1.009835*i )

> Terminated with exit code 0.
```

3.3 GAUSS ELIMINATION WITH PARTIAL PIVOTING

As it is seen from the previous example some error existed in the solution set. It is called round off error. When gauss elimination process is used as it is defined above some error might be created by the process. In order to

minimize the round off error, absolute value of the multiplier terms should be as big as possible. In the calculation process of the multiplier if divisor is zero term will go to infinity, if divisior is relatively small it will be a very big number, and this will cause an extra error in digital computers. In order to prevent this in a degree, elimination raw is interchanged with the raw with maximum absolute value for the multiplier term (in the diagonal). This process is called partial pivoting. Partial pivoting is shown in the example

$$\begin{bmatrix}1 & 2 & 5\\2 & 1 & 2\\3 & 1 & 1\end{bmatrix}\begin{bmatrix}X_0\\X_1\\X_2\end{bmatrix}=\begin{bmatrix}1\\1\\2\end{bmatrix}$$

Before starting the gauss elimination the column 0 elements are compared and raw with the largest absolute value of this column is replace with elimination raw (raw 0)

$$\begin{bmatrix}3 & 1 & 1\\2 & 1 & 2\\1 & 2 & 5\end{bmatrix}\begin{bmatrix}X_0\\X_1\\X_2\end{bmatrix}=\begin{bmatrix}2\\1\\1\end{bmatrix}$$

If this process is converted to computer code

```
double buyuk, dummy;
for(k=0;k<(n-1);k++)
 { //pivoting
            p=k;
            buyuk=Math.abs(a[k][k]);
            for(ii=k+1;ii<n;ii++)
            {  dummy=Math.abs(a[ii][k]);
              if(dummy > buyuk) {buyuk=dummy;p=ii;}
            }
            if(p!=k)
            { for(jj=k;jj<n;jj++)
              { dummy=a[p][jj];
                a[p][jj]=a[k][jj];
                a[k][jj]=dummy;
              }
              dummy=b[p];
              b[p]=b[k];
              b[k]=dummy;
            }}
```

After pivoting gauss elimination process is carried out as above

PROGRAM 3.3-1 Gauss elimination with partial pivoting

```
import java.io.*;
class NA15
{
 public static double[] pivotlugauss(double a[][],double b[])
 { //gauss elimination with partial pivoting
 int n=b.length;
 double x[]=new double[n];
 double carpan=0;
 double toplam=0;
 double buyuk;
 double dummy=0;
 //gauss eleme
 int i,j,k,p,ii,jj;
 System.out.println("Original matrix :\n"+Matrix.toString(a));
 for(k=0;k<(n-1);k++)
 { //pivoting
            p=k;
            buyuk=Math.abs(a[k][k]);
            for(ii=k+1;ii<n;ii++)
            {  dummy=Math.abs(a[ii][k]);
              if(dummy > buyuk) {buyuk=dummy;p=ii;}
            }
            if(p!=k)
```

```
              { for(jj=k;jj<n;jj++)
                { dummy=a[p][jj];
                  a[p][jj]=a[k][jj];
                  a[k][jj]=dummy;
                }
                dummy=b[p];
                b[p]=b[k];
                b[k]=dummy;
              }
                System.out.println("pivoted matrix : k="+k+"\n"+Matrix.toString(a));
              //Solving gauss elimination
   for(i=k+1;i<n;i++)
   {  carpan=a[i][k]/a[k][k];
      a[i][k]=0;
      for(j=k+1;j<n;j++)
      {   a[i][j]-=carpan*a[k][j]; }
         b[i]   =b[i]   -carpan*b[k];
              }
 }
 System.out.println("final upper triangular matrix : \n"+Matrix.toString(a));
 //back substitution
 x[n-1]=b[n-1]/a[n-1][n-1];
 for(i=n-2;i>=0;i--)
 {
   toplam=0;
   for(j=i+1;j<n;j++)
   {  toplam+=a[i][j]*x[j];}
   x[i]=(b[i]-toplam)/a[i][i];
 }
 return x;
 }

 public static void main(String args[]) throws IOException
 {
 double a[][]={{1,2,5},{2,1,2},{3,1,1}};
 double b[]={1,1,2};
 //double a[][]={{9.915,2.205},{2.205,0.52}};
 //double b[]={1.046,-2.042};
 double x[]=pivotlugauss(a,b);
 Text.print(x,"Pivotlu Gauss eliminasyon metodu çözüm matrisi");
 }
}
```

Output 3.3-1 gauss elimination with partial pivoting

```
---------- Capture Output ---------- Capture Output ----------
> "C:\turhan\java\bin\javaw.exe" NA15
Original matrix :
      1.000000000000000      2.000000000000000      5.000000000000000
      2.000000000000000      1.000000000000000      2.000000000000000
      3.000000000000000      1.000000000000000      1.000000000000000
pivoted matrix : k=0
      3.000000000000000      1.000000000000000      1.000000000000000
      2.000000000000000      1.000000000000000      2.000000000000000
      1.000000000000000      2.000000000000000      5.000000000000000
pivoted matrix : k=1
      3.000000000000000      1.000000000000000      1.000000000000000
      0.000000000000000      1.666666666666667      4.666666666666667
      0.000000000000000      0.333333333333333      1.333333333333334
final upper triangular matrix :
      3.000000000000000      1.000000000000000      1.000000000000000
      0.000000000000000      1.666666666666667      4.666666666666667
      0.000000000000000      0.000000000000000      0.400000000000000
```

As it is seen error shown in the previous non-pivoted version is disapeared here. In solution of small matrices Gauss elimination is a good solution, but when the matrix size is big, due to enourmous calculation size required, it is not a practical method.

PROGRAM 3.3-2 Gauss elimination with partial pivoting in Matlab(Octave) code

```
function x=gauss1(a,b)
% Gauss elimination
n=length(b);
carpan=0;
toplam=0;
 for k=1:n
  %pivoting
  p=k;
  buyuk=abs(a(k,k));
  for ii=k+1:n
   dummy=abs(a(ii,k));
    if dummy > buyuk buyuk=dummy;p=ii;end
  end
  if p~=k
   for jj=k:n
    dummy=a(p,jj);
    a(p,jj)=a(k,jj);
    a(k,jj)=dummy;
    end
   dummy=b(p);
   b(p)=b(k);
   b(k)=dummy;
  end
  %gauss eleme
  for i=k+1:n
   carpan=a(i,k)/a(k,k);
    a(i,k)=0;
    for j=k+1:n
      a(i,j)=a(i,j)-carpan*a(k,j);
    end
       b(i)   =b(i)   -carpan*b(k);
  end
 end
 %back substituting
 x(n)=b(n)/a(n,n);
 for i=n-1:-1:1
  toplam=0;
  for j=i+1:n
   toplam=toplam+a(i,j)*x(j);end
  x(i)=(b(i)-toplam)/a(i,i);
 end
end
```

PROGRAM 3.3.3 Gauss elimination with partial pivoting in Python code

```
# Gauss Elimination with pivoting
# python version

from math import *

def gauss(a,b):
 n=len(a)
 x=[0 for i in range(n)]
 #print("origial matrix :\na =",a,"\nb =",b)
 for k in range(n):
  p=k
  buyuk=abs(a[k][k])
  for ii in range(k+1,n):
   dummy=abs(a[ii][k])
   if dummy>buyuk:
    buyuk=dummy
    p=ii
  if p!=k:
   for jj in range(k,n):
    dummy=a[p][jj]
    a[p][jj]=a[k][jj]
    a[k][jj]=dummy
```

```
      dummy=b[p]
      b[p]=b[k]
      b[k]=dummy
    #print("pivoted matrix :\na =",a,"\nb =",b)
    for i in range(k+1,n):
       carpan=a[i][k]/a[k][k]
       a[i][k]=0;
       for j in range(k+1,n):
          a[i][j]=a[i][j]-carpan*a[k][j]
       b[i]   =b[i]-carpan*b[k]
  #print("eliminated matrix :\na =",a,"\nb =",b)
  x[n-1]=b[n-1]/a[n-1][n-1]
  for i in range(n-1,-1,-1):
    toplam=0;
    for j in range(i+1,n):
      toplam=toplam+a[i][j]*x[j]
    x[i]=(b[i]-toplam)/a[i][i];
  return x;

a=[[1,2,5],[2,1,2],[3,1,1]]
b=[1,1,2]
x=gauss(a,b);
print("Gauss elimination solution vector :\nx = ",x)
```

```
Gauss elimination solution vector :
x =  [7.401486830834377e-17, 2.9999999999999996, -0.9999999999999998]
Reloaded modules: fi_xi, gauss
Gauss elimination solution vector :
x =  [7.401486830834377e-17, 2.9999999999999996, -0.9999999999999998]
```

PROGRAM 3.3.4 Pivoted Gauss elimination in C++ using vector class (c++ std 11 version is used to compile)

```
#include <iostream>
#include <cmath>
#include <vector>
using namespace std;

void print(vector< vector<double> > a,vector<double> b) {
    int n = a.size();
    for (int i=0; i<n; i++) {
        for (int j=0; j<n; j++) {cout << a[i][j] << "\t";}
        cout << "| "<<b[i]<<"\n";
    }
    cout<<"\n";
}
void print(vector<double> b) {
    int n = b.size();
    for (int i=0; i<n; i++) {
        cout <<b[i]<< "\t";
    }
    cout<<"\n";
}
vector<double> pivotlugauss(vector< vector<double> > a,vector<double> b)
{//gauss elimination with partial pivoting
  int n=a.size();
  int m=b.size();
  vector<double> x(n);
  double carpan=0;
  double toplam=0;
  double buyuk;
  double dummy=0;
  //gauss eleme
  int i,j,k,p,ii,jj;
  for(k=0;k<(n-1);k++)
  { //pivoting
              p=k;
              buyuk=abs(a[k][k]);
              for(ii=k+1;ii<n;ii++)
              {  dummy=abs(a[ii][k]);
                 if(dummy > buyuk) {buyuk=dummy;p=ii;}
              }
              if(p!=k)
```

```
                { for(jj=k;jj<n;jj++)
                  { dummy=a[p][jj];
                    a[p][jj]=a[k][jj];
                    a[k][jj]=dummy;
                  }
                  dummy=b[p];
                  b[p]=b[k];
                  b[k]=dummy;

                }
                //Solving gauss elimination
    for(i=k+1;i<n;i++)
    {  carpan=a[i][k]/a[k][k];
       a[i][k]=0;
       for(j=k+1;j<n;j++)
       {   a[i][j]-=carpan*a[k][j]; }
           b[i]   =b[i]-carpan*b[k];
    }
  }
  //back substitution
  cout<<"elimination matrix\n";
  print(a,b);
  x[n-1]=b[n-1]/a[n-1][n-1];
  for(i=n-2;i>=0;i--)
  {
    toplam=0;
    for(j=i+1;j<n;j++)
    {  toplam+=a[i][j]*x[j];}
    x[i]=(b[i]-toplam)/a[i][i];
  }
  return x;
  }

int main() {
    int n=3;
    vector< vector<double> > a{{1,2,5},{2,1,2},{3,1,1}};
    vector<double> b{1,1,2};
    vector<double> x(n);
    x = pivotlugauss(a,b);
    // Print result
    cout<<"x = \n";
    print(x);
}
```

```
---------- Capture Output ----------
> "D:\SCO1\run.bat" gauss1

D:\okul\SCO1>gauss1
elimination matrix
3           1           1           | 2
0           1.66667     4.66667     | 0.333333
0           0           0.4         | -0.4

x =
2.22045e-016           3           -1

> Terminated with exit code 0.
```

Gauss elimination can also be applied by using complex variables, we are given here a complex equivalent of the partial pivoted Gauss elimination code below

PROGRAM 3.3-5 Gauss elimination with partial pivoting in complex variables (java version)

```
 import java.io.*;

class NA15C
{ public static c[] c_gauss_pivot(c a[][],c b[])
  { //gauss elimination with partial pivoting
  int n=b.length;
  c x[]=new c[n];
  c carpan=new c();
  c toplam=new c();
  c buyuk;
```

```
c dummy=new c();
//gauss eleme
int i,j,k,p,ii,jj;
System.out.println("Original matrix :\n"+c.toString(a));
for(k=0;k<(n-1);k++)
{ //pivoting
              p=k;
              buyuk=a[k][k];
              for(ii=k+1;ii<n;ii++)
              {  dummy=a[ii][k];
                 if(dummy.R() > buyuk.R()) {buyuk=dummy;p=ii;}
              }
              if(p!=k)
              { for(jj=k;jj<n;jj++)
                { dummy=a[p][jj];
                  a[p][jj]=a[k][jj];
                  a[k][jj]=dummy;
                }
                dummy=b[p];
                b[p]=b[k];
                b[k]=dummy;
              }
}
//gauss eleme
System.out.println("Matrix after pivoting = \n"+c.toString(a));
for(k=0;k<(n-1);k++)
{
  for(i=k+1;i<n;i++)
  {  carpan=c.p(a[i][k],"/",a[k][k]);
     a[i][k]=new c(0.0,0.0);
     for(j=k+1;j<n;j++)
     {   a[i][j]=c.p(a[i][j],"-",c.p(carpan,"*",a[k][j])); }
        b[i]=c.p(b[i],"-",c.p(carpan,"*",b[k]));
              }
               System.out.println("k = "+k+" :\n"+c.toString(a));
}
//back substituting
x[n-1]=c.p(b[n-1],"/",a[n-1][n-1]);
for(i=n-2;i>=0;i--)
{
  toplam=new c(0.0,0.0);
  for(j=i+1;j<n;j++)
  {  toplam=c.p(toplam,"+",c.p(a[i][j],"*",x[j]));}
  x[i]=c.p(c.p(b[i],"-",toplam),"/",a[i][i]);
}
return x;
}

public static c[] c_gauss(c a[][],c b[])
{
int n=b.length;
c x[]=new c[n];
c carpan=new c(1.0,0.0);
c toplam=new c(0.0,0.0);
//gauss eleme
int i,j,k;
for(k=0;k<(n-1);k++)
{
  for(i=k+1;i<n;i++)
  {  carpan=c.p(a[i][k],"/",a[k][k]);
     a[i][k]=new c(0.0,0.0);
     for(j=k+1;j<n;j++)
     {   a[i][j]=c.p(a[i][j],"-",c.p(carpan,"*",a[k][j])); }
        b[i]=c.p(b[i],"-",c.p(carpan,"*",b[k]));
              }
}
//back substituting
x[n-1]=c.p(b[n-1],"/",a[n-1][n-1]);
for(i=n-2;i>=0;i--)
{
  toplam=new c(0.0,0.0);
  for(j=i+1;j<n;j++)
  {  toplam=c.p(toplam,"+",c.p(a[i][j],"*",x[j]));}
  x[i]=c.p(c.p(b[i],"-",toplam),"/",a[i][i]);
```

```
  }
  return x;
  }


  public static void main(String args[]) throws IOException
  {
  String s1="{(-1.75,0),(1.25,0.5),(1,0.7)},{(1.2,2.3),(1.1,-1.543),(3.77,0)},{(-1,1),(3.25,0),(1,0)}";
  String s2="{(1.0,2.0),(1.1,0),(1.2,1.3)}";
  c a[][]=c.toComplexMatrix(s1);
  c b[]=c.toComplexVector(s2);
  c x[]=c_gauss_pivot(a,b);
  System.out.println("x = \n"+c.toString(x));
  }
}
```

```
 ---------- Capture Output ----------
> "C:\java\bin\javaw.exe" NA15C
Original matrix :
  ( -1.750000 +   0.000000*i )   (  1.250000 +   0.500000*i )   (  1.000000 +   0.700000*i )
  (  1.200000 +   2.300000*i )   (  1.100000 -   1.543000*i )   (  3.770000 +   0.000000*i )
  ( -1.000000 +   1.000000*i )   (  3.250000 +   0.000000*i )   (  1.000000 +   0.000000*i )

Matrix after pivoting =
  (  1.200000 +   2.300000*i )   (  1.100000 -   1.543000*i )   (  3.770000 +   0.000000*i )
  ( -1.750000 +   0.000000*i )   (  3.250000 +   0.000000*i )   (  1.000000 +   0.000000*i )
  ( -1.000000 +   1.000000*i )   (  1.250000 +   0.500000*i )   (  1.000000 +   0.700000*i )

k = 0 :
  (  1.200000 +   2.300000*i )   (  1.100000 -   1.543000*i )   (  3.770000 +   0.000000*i )
  (  0.000000 +   0.000000*i )   (  2.670420 -   1.139346*i )   (  2.176374 -   2.254718*i )
  (  0.000000 +   0.000000*i )   (  0.267756 +   0.180134*i )   (  0.383804 -   1.260624*i )

k = 1 :
  (  1.200000 +   2.300000*i )   (  1.100000 -   1.543000*i )   (  3.770000 +   0.000000*i )
  (  0.000000 +   0.000000*i )   (  2.670420 -   1.139346*i )   (  2.176374 -   2.254718*i )
  (  0.000000 +   0.000000*i )   (  0.000000 +   0.000000*i )   (  0.041909 -   1.327228*i )

x =
  (  0.747087 -   1.487163*i )   (  1.054138 -   0.592099*i )   ( -0.918547 +   0.621788*i )

> Terminated with exit code 0.
```

3.4 GAUSS-JORDAN METHOD

Gauss Jordan elimination method is similar to Gauss elimination with exception of a normalisation process before elimination and application of the elimination for all raws (instead of only lower lows of diagonal reference row in gauss elimination) except the diagonal reference raw. The result is conversion of the right hand side into the solution without any back substitution process. Example program is given below

PROGRAM 3.4.1 Gauss_Jordan elimination method

```
import java.io.*; import java.io.*;

clasS NA16
{
//Gauss-Jordan elimination process
  public static double[] gaussjordan(double a[][],double b[])
  { //NO PIVOTİNG
  int n=b.length;
  int np1=n+1;
  double C[][]=new double[n][np1];
  double x[]=new double[n];
  double carpan=0;
  int i,j,k;

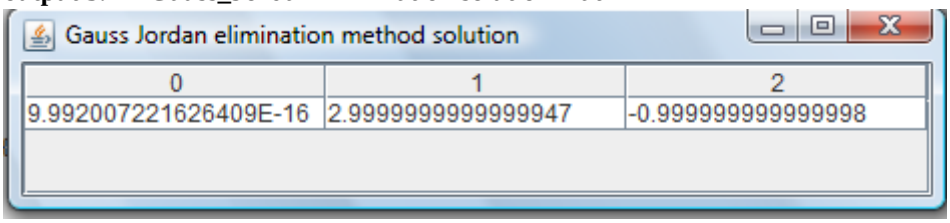
  for(i=0;i<n;i++)
  {for(j=0;j<n;j++){C[i][j]=a[i][j];};C[i][n]=b[i];}

  for(k=0;k<n;k++)
```

```
 {
            //gauss jordan elimination
            for(j=k+1;j<np1;j++)
            { C[k][j]/=C[k][k]; }
            for(i=0;i<n;i++)
  {  if(i!=k)
    {  carpan=C[i][k];
      C[i][k]=0;
      for(j=k+1;j<np1;j++)
      {   C[i][j]-=carpan*C[k][j]; }
    }
            }
 }
 for(i=0;i<n;i++)
 {x[i]=C[i][n];}
 return x;
 }

 public static void main(String args[]) throws IOException
 {
 double a[][]={{1,2,5},{2,1,2},{3,1,1}};
 double b[]={1,1,2};
 double x[]=gaussjordan(a,b);
 Text.print(x,"Gauss Jordan elimination method solution");
 }
}
```

output 3.4-1 Gauss_Jordan Elimination solution matrix



To enter a second example into the code :

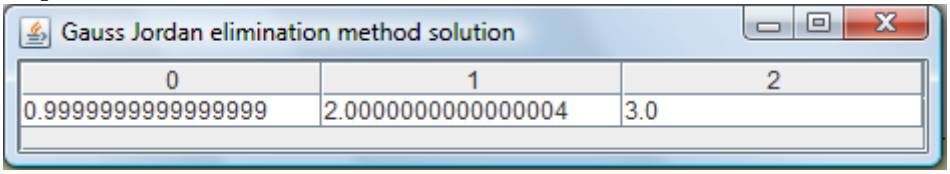
$$\begin{bmatrix} 3 & 1 & -1 \\ 1 & 4 & 1 \\ 2 & 1 & 2 \end{bmatrix} \begin{bmatrix} X_0 \\ X_1 \\ X_2 \end{bmatrix} = \begin{bmatrix} 2 \\ 12 \\ 10 \end{bmatrix}$$

Following change can be apply to the program code

```
double a[][]={{3,1,-1},{1,4,1},{2,1,2}};
double b[]={2,12,10};
```

solution set will be:

output 3.4-2 Gauss_Jordan Elimination solution matrix



Now if the same system of equation is solved by hand:

$$\begin{bmatrix} 1 & 0.3333 & -0.3333 \\ 1 & 4 & 1 \\ 2 & 1 & 2 \end{bmatrix} \begin{bmatrix} X_0 \\ X_1 \\ X_2 \end{bmatrix} = \begin{bmatrix} 0.6667 \\ 12 \\ 10 \end{bmatrix}$$

$$\begin{bmatrix} 1 & 0.3333 & -0.3333 \\ 0 & 3.6667 & 1.3333 \\ 2 & 1 & 2 \end{bmatrix} \begin{bmatrix} X_0 \\ X_1 \\ X_2 \end{bmatrix} = \begin{bmatrix} 0.6667 \\ 11.333 \\ 10 \end{bmatrix}$$

$$\begin{bmatrix}1 & 0.3333 & -0.3333\\0 & 3.6667 & 1.3333\\0 & 0.3333 & 2.6667\end{bmatrix}\begin{bmatrix}X_0\\X_1\\X_2\end{bmatrix}=\begin{bmatrix}0.6667\\11.333\\8.6667\end{bmatrix}$$

$$\begin{bmatrix}1 & 0.3333 & -0.3333\\0 & 1 & 0.3636\\0 & 0.3333 & 2.6667\end{bmatrix}\begin{bmatrix}X_0\\X_1\\X_2\end{bmatrix}=\begin{bmatrix}0.6667\\3.0909\\8.6667\end{bmatrix}$$

$$\begin{bmatrix}1 & 0 & -0.4545\\0 & 1 & 0.3636\\0 & 0.3333 & 2.6667\end{bmatrix}\begin{bmatrix}X_0\\X_1\\X_2\end{bmatrix}=\begin{bmatrix}-0.3636\\3.0909\\8.6667\end{bmatrix}$$

$$\begin{bmatrix}1 & 0 & -0.4545\\0 & 1 & 0.3636\\0 & 0 & 2.5455\end{bmatrix}\begin{bmatrix}X_0\\X_1\\X_2\end{bmatrix}=\begin{bmatrix}-0.3636\\3.0909\\7.6365\end{bmatrix}$$

$$\begin{bmatrix}1 & 0 & -0.4545\\0 & 1 & 0.3636\\0 & 0 & 1\end{bmatrix}\begin{bmatrix}X_0\\X_1\\X_2\end{bmatrix}=\begin{bmatrix}-0.3636\\3.0909\\3.0000\end{bmatrix}$$

$$\begin{bmatrix}1 & 0 & 0\\0 & 1 & 0.3636\\0 & 0 & 1\end{bmatrix}\begin{bmatrix}X_0\\X_1\\X_2\end{bmatrix}=\begin{bmatrix}1.0000\\3.0909\\3.0000\end{bmatrix}$$

$$\begin{bmatrix}1 & 0 & 0\\0 & 1 & 0\\0 & 0 & 1\end{bmatrix}\begin{bmatrix}X_0\\X_1\\X_2\end{bmatrix}=\begin{bmatrix}1.0000\\2.0001\\3.0000\end{bmatrix}$$

Solution vector

$$\begin{bmatrix}X_0\\X_1\\X_2\end{bmatrix}=\begin{bmatrix}1.0000\\2.0001\\3.0000\end{bmatrix}$$

Exact solution :

$$\begin{bmatrix}X_0\\X_1\\X_2\end{bmatrix}=\begin{bmatrix}1.0000\\2.0000\\3.0000\end{bmatrix}$$

Another example for Gauss-Jordan elimination with detailed steps :

$$\begin{bmatrix}7 & 1 & 2\\-1 & 4 & -1\\3 & 15 & 20\end{bmatrix}\begin{bmatrix}X_0\\X_1\\X_2\end{bmatrix}=\begin{bmatrix}47\\19\\87\end{bmatrix}$$

Gauss-Jordan

A				B
7	1	2	X_0	47
-1	4	-1	X_1	19
3	15	20	X_2	87
Normalisation				
1	0.142857	0.285714	X_0	6.714286
-1	4	-1	X_1	19
3	15	20	X_2	87
Elimination				
1	0.142857	0.285714	X_0	6.714286
0	4.142857	-0.71429	X_1	25.71429
0	14.57143	19.14286	X_2	66.85714
Normalisation				
1	0.142857	0.285714	X_0	6.714286
0	1	-0.17241	X_1	6.206897
0	14.57143	19.14286	X_2	66.85714
Elimination				
1	0	0.310345	X_0	5.827586

0	1	-0.17241	X_1	6.206897
0	0	21.65517	X_2	-23.5862

Normalisation

1	0	0.310345	X_0	5.827586
0	1	-0.17241	X_1	6.206897
0	0	1	X_2	-1.08917

Elimination

1	0	0	X_0	**6.165605**
0	1	0	X_1	**6.019108**
0	0	1	X_2	**-1.08917**

PROGRAM 3.4.1 Gauss_Jordan elimination in Matlab(Octave) code

```
function x=gaussjordan(a,b)
 n=length(b);
 np1=n+1;
 carpan=0;
 C=a;
 for i=1:n
    C(i,n+1)=b(i);
 end
 for k=1:n
  %gauss jordan elimination
  for j=k+1:np1
   C(k,j)=C(k,j)/C(k,k);
  end
  for i=1:n
   if i~=k
     carpan=C(i,k);
      C(i,k)=0;
      for j=k+1:np1
        C(i,j)=C(i,j)-carpan*C(k,j); end
    end
  end
 end
 for i=1:n
  C(i,i)=1;
  x(i)=C(i,np1);
 end
 end
```

PROGRAM 3.4.1 Gauss_Jordan elimination in Python code

```
from math import *

def gaussjordan(a,b):
 n=len(a)
 np1=n+1
 x=[]
 C=[[0 for x in range(np1)] for y in range(n)]
 print("origial matrix :\na =",a,"\nb =",b)
 for i in range(n):
  x.append(0.0)
 for i in range(n):
  for j in range(n):
   C[i][j]=a[i][j]
  C[i][n]=b[i]
 for k in range(n):
  for j in range(k+1,np1):
   C[k][j]=C[k][j]/C[k][k]
  for i in range(n):
   if i!=k:
    carpan=C[i][k]
    C[i][k]=0.0
    for j in range(k+1,np1):
     C[i][j]=C[i][j]-carpan*C[k][j]
 print("eliminated matrix :\nC =",C)
 for i in range(n):
  x[i]=C[i][n]
 return x;
```

```
a=[[1,2,5],[2,1,2],[3,1,1]]
b=[1,1,2]
x=gaussjordan(a,b);
print("Gauss - Jordan elimination solution matrix :\nx = ",x)
```

```
---------- Capture Output ----------
> "D:\co\python\pythonw.exe" NA16.py
origial matrix :
a = [[1, 2, 5], [2, 1, 2], [3, 1, 1]]
b = [1, 1, 2]
eliminated matrix :
a = [[1, 0.0, 0.0, 9.992007221626409e-16], [0.0, -3.0, 0.0, 2.9999999999999947], [0.0, 0.0, -0.6666666666666679, -0.9999999999999998]]
Gauss elimination solution matrix :
x =  [9.992007221626409e-16, 2.9999999999999947, -0.9999999999999998]

> Terminated with exit code 0.
```

3.5 GAUSS-JORDAN METHOD WITH PARTIAL PIVOTING

Pivoting process can be applied to Gauss-Jordan elimination similar to Gauss elimination

PROGRAM 3.5-1 Gauss_Jordan Elimination with partial pivoting

```
import java.io.*;

class NA17
{ public static double[][] C(double a[][],double b[])
  {int n=a.length;int m=a[0].length;double c[][]=new double[n][m+1];
  for(int i=0;i<n;i++){for(int j=0;j<m;j++){c[i][j]=a[i][j];};c[i][m]=b[i];}
  return c;
  }

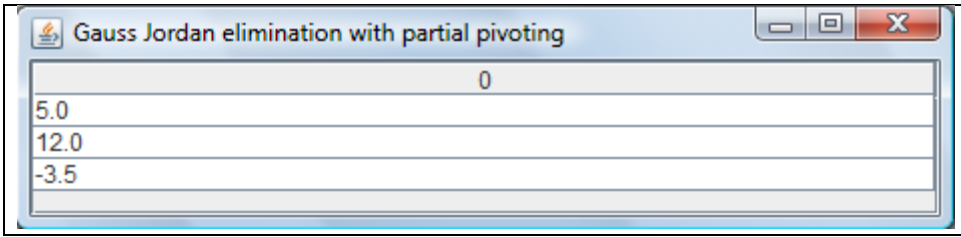
  public static double[][] C(double a[][],double b[][])
  {int n=a.length;int o=b[0].length;int m=n+o;double c[][]=new double[n][m];
  for(int i=0;i<n;i++)
  {for(int j=0;j<n;j++){c[i][j]=a[i][j];};
   for(int j=n;j<m;j++){c[i][j]=b[i][j];}
  }
  return c;
  }

  public static double[][] gaussjordanwithpivot(double a[][])
  { //pivotlu
  int n=a.length;
  int m=a[0].length;
  int o=m-n;
  double x[][]=new double[n][o];
  double carpan=0;
  double toplam=0;
  double buyuk;
  double dummy=0;
  int i,j,k,p,ii,jj;
  i=0;
  for(k=0;k<n;k++)
  { //pivoting
              p=k;
              buyuk=Math.abs(a[k][k]);
              for(ii=0;ii<n;ii++)
              {  if(i!=k)
                 {
                 dummy=Math.abs(a[ii][k]);
                 if(dummy > buyuk) {buyuk=dummy;p=ii;}
     }
              }
              if(p!=k)
```

```
            { for(jj=k;jj<m;jj++)
              { dummy=a[p][jj];
                a[p][jj]=a[k][jj];
                a[k][jj]=dummy;
              }
            }
            //gauss jordan elimination
            for(j=k+1;j<m;j++)
            { a[k][j]/=a[k][k]; }
            a[k][k]=1.0;
  for(i=0;i<n;i++)
  {  if(i!=k)
     {
                carpan=a[i][k];
       a[i][k]=0;
       for(j=k+1;j<m;j++)
         {a[i][j]-=carpan*a[k][j]; }
     }
            }
 }
 for(jj=0;jj<o;jj++)
 {for(i=0;i<n;i++) {x[i][jj]=a[i][n+jj];}}
 return x;
 }

 public static void main(String args[]) throws IOException
 {
 double a[][]={{3,0,2,8},{0,0,2,-7},{6,2,8,26}};
 //double a[][]=Text.readDouble_Table();
 double x[][]=gaussjordanwithpivot(a);
 Text.print(x,"Gauss Jordan elimination with partial pivoting");
 }
}
```

output 3.5-1Gauss_Jordan Elimination with partial pivoting



PROGRAM 3.5-2 Gauss_Jordan Elimination with partial pivoting in Matlab(Octave)

```
function x=gaussjordan1(a,b)
 n=length(b);
 np1=n+1;
 carpan=0;
 C=a;
 for i=1:n
    C(i,np1)=b(i);
 end
 for k=1:n
  %pivoting
  p=k;
  buyuk=abs(C(k,k));
  for ii=k+1:n
   dummy=abs(C(ii,k));
    if dummy > buyuk buyuk=dummy;p=ii;end
  end
  if p~=k
   for jj=k:np1
     dummy=C(p,jj);
```

```
      C(p,jj)=C(k,jj);
      C(k,jj)=dummy;
    end
  end
  %gauss jordan elimination
  for j=k+1:np1
   C(k,j)=C(k,j)/C(k,k);
  end
  for i=1:n
   if i~=k
     carpan=C(i,k);
      C(i,k)=0;
       for j=k+1:np1
         C(i,j)=C(i,j)-carpan*C(k,j); end
    end
  end
 end
 for i=1:n
  C(i,i)=1;
  x(i)=C(i,np1);
 end
 end
```

```
>> a=[3 0 2;0 0 2;6 2 8]
a =
    3    0    2
    0    0    2
    6    2    8
>> b=[8,-7,26]'
b =
    8
   -7
    26
>> gaussjordan1(a,b)
ans =
   5.0000   12.0000   -3.5000
```

3.6 LU-DECOMPOSITION- GAUSS (DOLİTTLE) METHOD

In Gauss or Gauss-Jordan elimination processes left hand matrix A and right hand vector b is eliminated together. Therefore if a new left hand side vector b1 will be considered to be used in as second set with the same A matrix, Matrix A is reprocessed. This will spend valuable calculation time for a repeated process. In order to avoid this repetation, Matrix A is decomposed into an upper triangle(U) and a lower triangle (L) matrices so that multiplication of this two matricess will give the original A matrix

For example for N=5

$$[L] = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 \\ l_{21} & 1 & 0 & 0 & 0 \\ l_{31} & l_{32} & 1 & 0 & 0 \\ l_{41} & l_{42} & l_{43} & 1 & 0 \\ l_{51} & l_{52} & l_{53} & l_{54} & 1 \end{bmatrix}$$

$$[U] = \begin{bmatrix} u_{11} & u_{12} & u_{13} & u_{14} & u_{15} \\ 0 & u_{22} & u_{23} & u_{24} & u_{25} \\ 0 & 0 & u_{33} & u_{34} & u_{35} \\ 0 & 0 & 0 & u_{44} & u_{45} \\ 0 & 0 & 0 & 0 & u_{55} \end{bmatrix}$$

To solve system of equation with a B vector following process applies

[A]=[L][U]

[A]{X}={B}

[L][U]{X}={B}

[L]{D}={B}

[U]{X}={D}

In Gauss(Doolittle) decomposition process matrix U is the same as Gauss elimination upper triangualar matrix. L matrix is consist of the multipliers calculated in Gauss elimination process. Due to diagonal element of matrix L is 1, LU matrix can be represented in one matrix if space saving is desired. Example code is given below

PROGRAM 3.6-1 Gauss (Dolittle) Elemimination & decomposition

```
import java.io.*;
import javax.swing.*;
class NA18
{
 public static double[][] gaussLU(double ai[][])
 {
 int n=ai.length;
 double x[]=new double[n];
 double a[][]=new double[n][n];
 double carpan=0;
 double toplam=0;
 //gauss eleme
 int i,j,k;
 for(i=0;i<n;i++)
    for(j=0;j<n;j++){a[i][j]=ai[i][j];};
 for(k=0;k<(n-1);k++)
 {
  for(i=k+1;i<n;i++)
  {  carpan=a[i][k]/a[k][k];
     a[i][k]=carpan;///L elemani
     for(j=k+1;j<n;j++)
     {   a[i][j]-=carpan*a[k][j]; }
            }
 }
 return a;
 }

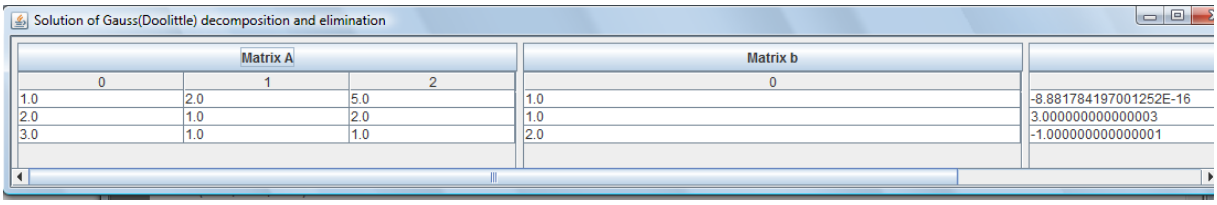

 public static double[] yerine_koyma(double ai[][],double bi[])
 { //gauss LU çözümüyle LU indirgenmesi yapilmiş Matris kullanarak linear
  //denklem sistemi kökleri
  int n=ai.length;
  double toplam;
  double x[]=new double[n];
   double a[][]=new double[n][n];
  double b[]=new double[n];
  int i,j;
 for(i=0;i<n;i++)
    {for(j=0;j<n;j++){a[i][j]=ai[i][j];};b[i]=bi[i];}
 System.out.println("a=\n "+Matrix.toString(a));
 System.out.println("b=\n "+Matrix.toStringT(b));
 for(i=1;i<n;i++)
    {toplam=b[i];for(j=0;j<i;j++){toplam-=a[i][j]*b[j];};b[i]=toplam;}
 System.out.println("d=\n "+Matrix.toStringT(b));
 x[n-1]=b[n-1]/a[n-1][n-1];
 for(i=(n-2);i>=0;i--)
    {toplam=0;
     for(j=i+1;j<n;j++)
      {toplam+=a[i][j]*x[j];};
      x[i]=(b[i]-toplam)/a[i][i];
    }
 System.out.println("d=\n "+Matrix.toStringT(x));
 return x;
 }
```

```
  public static double[][] yerine_koyma(double ai[][],double bi[][])
  {
    int n=ai.length;
    int m=bi[0].length;
    double toplam;
    double x[][]=new double[n][m];
    double a[][]=new double[n][n];
    double b[][]=new double[n][m];
    int i,j,k;
  for(i=0;i<n;i++)
     {for(j=0;j<n;j++){a[i][j]=ai[i][j];};for(k=0;k<m;k++) {b[i][k]=bi[i][k];}}
  System.out.println("a=\n "+Matrix.toString(a));
  System.out.println("b=\n "+Matrix.toStringT(b));
  for(k=0;k<m;k++)
  {
  for(i=1;i<n;i++)
     {toplam=b[i][k];for(j=0;j<i;j++){toplam-=a[i][j]*b[j][k];};b[i][k]=toplam;}
  System.out.println("d=\n "+Matrix.toStringT(b));
  x[n-1][k]=b[n-1][k]/a[n-1][n-1];
  for(i=(n-2);i>=0;i--)
     {toplam=0;
      for(j=i+1;j<n;j++)
       {toplam+=a[i][j]*x[j][k];};
       x[i][k]=(b[i][k]-toplam)/a[i][i];
     }
  }//kin sonu
  System.out.println("d=\n "+Matrix.toStringT(x));
  return x;
  }

  public static double [] AXB(double a[][],double b[])
  {
  //denklem sistemi çözümü b vektörü için
  int n=a.length;
  double c[][]=gaussLU(a);
  Matrix.toString(c);
  double s[]=yerine_koyma(c,b);
  return s;
  }

  public static double [][] AXB(double a[][],double b[][])
  {//denklem sistemi çözümü b matrisi için
  int n=a.length;
  double c[][]=gaussLU(a);
  Matrix.toString(c);
  double s[][]=yerine_koyma(c,b);
  return s;
  }

  public static void main(String args[]) throws IOException
  {
  double a[][]={{1,2,5},{2,1,2},{3,1,1}};
  double b[]={1,1,2};
  JPanel y=new JPanel();
  Text.add(y,a,"Matrix A");
  Text.addT(y,b,"Matrix b");
  double x[]=AXB(a,b);
  Text.addT(y,x,"Solution Matrix");
  Text.print(y,"Solution of Gauss(Doolittle) decomposition and elimination");
  }
 }
```



PROGRAM 3.6-2 Gauss (Dolittle) LU decomposition with partial pivoting

```
import java.io.*;
import javax.swing.*;
class NA19
{
 public static void pivot(double a[][],int o[],double s[],int k)
 {// Gauss(Dolittle) LU decomposition with partial pivoting
  int n=a.length;
  int dummy1=0;
  double buyuk;
  double dummy=0;
  //pivotlama
            int p=k;
            buyuk=Math.abs(a[o[k]][k]);
            for(int ii=k+1;ii<n;ii++)
            {  dummy=Math.abs(a[o[ii]][k]/s[o[ii]]);
              if(dummy > buyuk) {buyuk=dummy;p=ii;}
            }
            dummy1=o[p];
            o[p]=o[k];
            o[k]=dummy1;
 }

 public static int[] sirali_indeks(int n)
 {   // sirali rakamlari taşiyan bir referans vektörü oluşturur
             int o[]=new int[n];
             for(int i=0;i<n;i++) {o[i]=i;}
             return o;
 }

 public static double[][] gaussLU(double ai[][],int o[])
 { //kismi pivotlu gauss eleme yöntemi
 int n=ai.length;
 double a[][]=new double[n][n];
 double s[]=new double[n];
 double carpan=0;
 double toplam=0;
 double absaij;

 //gauss eleme
 int i,j,k,p,ii,jj;
 //System.out.println("Orijinal Matris :\n"+Matrix.toString(ai));
 for(i=0;i<n;i++)
 {for(j=0;j<n;j++){a[i][j]=ai[i][j];}}
 for(i=0;i<n;i++)
 { o[i]=i;s[i]=Math.abs(a[i][0]);
   for(j=1;j<n;j++)
     { absaij=Math.abs(a[i][j]);
       if(absaij>s[i]) {s[i]=absaij;}
     }
 }
 for(k=0;k<(n-1);k++)
 { pivot(a,o,s,k);
            //pivot referansini değiştir
            //System.out.println("pivotlu Matris : k="+k+"\n"+Matrix.toString(a));
```

```
            //gauss elemeyi çözme
  for(i=k+1;i<n;i++)
  {  carpan=a[o[i]][k]/a[o[k]][k];
     a[o[i]][k]=carpan;
     for(j=k+1;j<n;j++)
     {   a[o[i]][j]-=carpan*a[o[k]][j]; }
            }
}
return a;
}

public static double[] yerine_koyma(double ai[][],double bi[])
{   int n=ai.length;
            return yerine_koyma(ai,bi,sirali_indeks(n));
}

public static double[][] yerine_koyma(double ai[][],double bi[][])
{   int n=ai.length;
            return yerine_koyma(ai,bi,sirali_indeks(n));
}

public static double[] yerine_koyma(double ai[][],double bi[],int o[])
{ //gauss LU çözümüyle LU indirgenmesi yapilmiş Matris kullanarak linear
  //denklem sistemi kökleri
  int n=ai.length;
  double toplam;
  double x[]=new double[n];
   double a[][]=new double[n][n];
  double b[]=new double[n];
  int i,j;
for(i=0;i<n;i++)
    {for(j=0;j<n;j++){a[i][j]=ai[i][j];};b[i]=bi[i];}
//System.out.println("a=\n "+Matrix.toString(a));
//System.out.println("b=\n "+Matrix.toStringT(b));
for(i=1;i<n;i++)
    {toplam=b[o[i]];for(j=0;j<i;j++){toplam-=a[o[i]][j]*b[o[j]];};b[o[i]]=toplam;}
 //System.out.println("d=\n "+Matrix.toStringT(b));
x[n-1]=b[o[n-1]]/a[o[n-1]][n-1];
for(i=(n-2);i>=0;i--)
    {toplam=0;
     for(j=i+1;j<n;j++)
      {toplam+=a[o[i]][j]*x[j];};
      x[i]=(b[o[i]]-toplam)/a[o[i]][i];
    }
//System.out.println("d=\n "+Matrix.toStringT(x));
return x;
}

public static double[][] yerine_koyma(double ai[][],double bi[][],int o[])
{
  int n=ai.length;
  int m=bi[0].length;
  double toplam;
  double x[][]=new double[n][m];
  double a[][]=new double[n][n];
  double b[][]=new double[n][m];
  int i,j,k;
for(i=0;i<n;i++)
    {for(j=0;j<n;j++){a[i][j]=ai[i][j];};for(k=0;k<m;k++) {b[i][k]=bi[i][k];}}
//System.out.println("a=\n "+Matrix.toString(a));
//System.out.println("b=\n "+Matrix.toStringT(b));
for(k=0;k<m;k++)
{
for(i=1;i<n;i++)
```

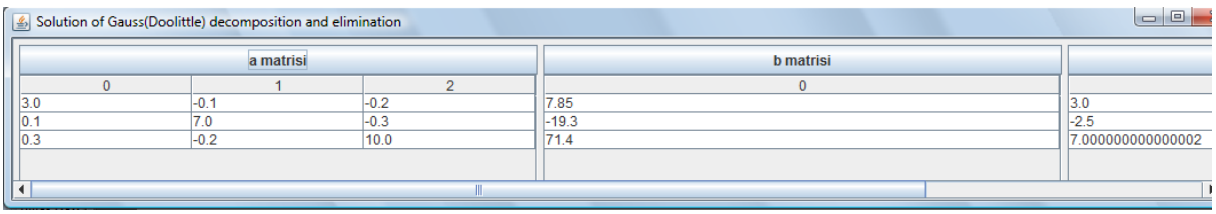
```
    {toplam=b[o[i]][k];for(j=0;j<i;j++){toplam-=a[o[i]][j]*b[o[j]][k];};b[o[i]][k]=toplam;}
 //System.out.println("d=\n "+Matrix.toStringT(b));
x[n-1][k]=b[o[n-1]][k]/a[o[n-1]][n-1];
for(i=(n-2);i>=0;i--)
   {toplam=0;
    for(j=i+1;j<n;j++)
     {toplam+=a[o[i]][j]*x[j][k];};
     x[i][k]=(b[o[i]][k]-toplam)/a[o[i]][i];
   }
}//kin sonu
//System.out.println("d=\n "+Matrix.toStringT(x));
return x;
}

public static double [] AXB(double a[][],double b[])
{
//denklem sistemi çözümü b vektörü için
int n=a.length;
int q[]=new int[n];
double c[][]=gaussLU(a,q);
double s[]=yerine_koyma(c,b,q);
return s;
}

public static double [][] AXB(double a[][],double b[][])
{//denklem sistemi çözümü b matrisi için
int n=a.length;
int q[]=new int[n];
double c[][]=gaussLU(a,q);
double s[][]=yerine_koyma(c,b,q);
return s;
}

public static void main(String args[]) throws IOException
{
double a[][]={{3,-0.1,-0.2},{0.1,7,-0.3},{0.3,-0.2,10}};
JPanel y=new JPanel();
Text.add(y,a,"a matrisi");
double b[]={7.85,-19.3,71.4};
Text.addT(y,b,"b matrisi");
double x[]=AXB(a,b);
Text.addT(y,x,"Solution Matrix");
Text.print(y,"Solution of Gauss(Doolittle) decomposition and elimination");
}
}
```

Output 3.6-1 Kismi Pivotlu LU Gauss (Dolittle) Eleme yöntemi



3.7 CRAUT LU DECOMPOSITION METHOD

In the Gauss (Dolittle) LU decomposition diagonal of matrix L was setted as 1, In the Craut LU decompositon diagonal of matrix U is given as 1. For example for a 5 by 5 system

$$[L] = \begin{bmatrix} l_{11} & 0 & 0 & 0 & 0 \\ l_{21} & l_{22} & 0 & 0 & 0 \\ l_{31} & l_{32} & l_{33} & 0 & 0 \\ l_{41} & l_{42} & l_{43} & l_{44} & 0 \\ l_{51} & l_{52} & l_{53} & l_{54} & l_{55} \end{bmatrix}$$

$$[U] = \begin{bmatrix} 1 & u_{12} & u_{13} & u_{14} & u_{15} \\ 0 & 1 & u_{23} & u_{24} & u_{25} \\ 0 & 0 & 1 & u_{34} & u_{35} \\ 0 & 0 & 0 & 1 & u_{45} \\ 0 & 0 & 0 & 0 & 1 \end{bmatrix}$$

Solution process is similar

[A]=[L][U]
[A]{X}={B}
[L][U]{X}={B}
[L]{D}={B}
[U]{X}={D}
Algorithm of Craut is different compare to Dolittle
$l_{i,1} = a_{i,1}$ loop i=1,2,….,n
$u_{1,j} = \frac{a_{1,j}}{l_{11}}$ loop i=2,3,….,n
loop j=2,3,….,n – 1
$l_{i,j} = a_{i,j} - \sum_{k=1}^{j-1} l_{i,k} u_{k,j}$ loop k=j,j+2,….,n
$u_{i,j} = \frac{a_{i,j} - \sum_{i=1}^{j-1} l_{j,i} u_{i,k}}{l_{j,j}}$ loop k=j+1,j+2,….,n
End of loop $l_{n,n} = a_{n,n} - \sum_{k=1}^{j-1} l_{n,k} u_{k,n}$

Algorithm is converted to programming code in the following example

PROGRAM 3.7-1 LU Craut Decomposition method

```
import java.io.*;

class NA20
{
  public static void pivot(double a[][],int o[],double s[],int k)
  {// Craut LU decomposition
    int n=a.length;
    int dummy1=0;
    double buyuk;
    double dummy=0;
    //pivoting
            int p=k;
            buyuk=Math.abs(a[o[k]][k]);
            for(int ii=k+1;ii<n;ii++)
            {  dummy=Math.abs(a[o[ii]][k]/s[o[ii]]);
              if(dummy > buyuk) {buyuk=dummy;p=ii;}
            }
            dummy1=o[p];
            o[p]=o[k];
            o[k]=dummy1;
  }

  public static int[] index(int n)
  {
```

```
            int o[]=new int[n];
            for(int i=0;i<n;i++) {o[i]=i;}
            return o;
}

public static double[] back_substitution(double ai[][],double bi[])
{   int n=ai.length;
            return back_substitution(ai,bi,index(n));
}

public static double[][] back_substitution(double ai[][],double bi[][])
{   int n=ai.length;
            return back_substitution(ai,bi,index(n));
}

public static double[] back_substitution(double ai[][],double bi[],int o[])
{
  int n=ai.length;
  double toplam;
  double x[]=new double[n];
   double a[][]=new double[n][n];
  double b[]=new double[n];
  int i,j;
for(i=0;i<n;i++)
   {for(j=0;j<n;j++){a[i][j]=ai[i][j];};b[i]=bi[i];}
for(i=1;i<n;i++)
   {toplam=b[o[i]];for(j=0;j<i;j++){toplam-=a[o[i]][j]*b[o[j]];};b[o[i]]=toplam;}
x[n-1]=b[o[n-1]]/a[o[n-1]][n-1];
for(i=(n-2);i>=0;i--)
   {toplam=0;
    for(j=i+1;j<n;j++)
     {toplam+=a[o[i]][j]*x[j];};
     x[i]=(b[o[i]]-toplam)/a[o[i]][i];
   }
return x;
}

public static double[][] back_substitution(double ai[][],double bi[][],int o[])
{
  int n=ai.length;
  int m=bi[0].length;
  double toplam;
  double x[][]=new double[n][m];
  double a[][]=new double[n][n];
  double b[][]=new double[n][m];
  int i,j,k;
for(i=0;i<n;i++)
   {for(j=0;j<n;j++){a[i][j]=ai[i][j];};for(k=0;k<m;k++) {b[i][k]=bi[i][k];}}
for(k=0;k<m;k++)
{
for(i=1;i<n;i++)
   {toplam=b[o[i]][k];for(j=0;j<i;j++){toplam-=a[o[i]][j]*b[o[j]][k];};b[o[i]][k]=toplam;}
x[n-1][k]=b[o[n-1]][k]/a[o[n-1]][n-1];
for(i=(n-2);i>=0;i--)
   {toplam=0;
    for(j=i+1;j<n;j++)
     {toplam+=a[o[i]][j]*x[j][k];};
```

```
        x[i][k]=(b[o[i]][k]-toplam)/a[o[i]][i];
      }
  }
  return x;
  }

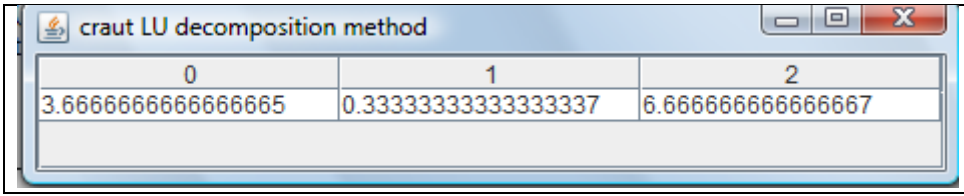
public static double[][] crautLU(double c[][])
{
//craut LU decomposition without pivoting

int n=c.length;
double sum;
int i,j,k;
double d[][]=new double[n][n];
for(i=0;i<n;i++)
{for(j=0;j<n;j++) d[i][j]=c[i][j];}

for(j=1;j<=n;j++)
{
  for(i=1;i<j;i++)
    {
    sum=d[i-1][j-1];
    for(k=1;k<i;k++)
              { sum-=d[i-1][k-1]*d[k-1][j-1];}
    d[i-1][j-1]=sum;
    }
  for(i=j;i<=n;i++)
    {
    sum=d[i-1][j-1];
    for(k=1;k<j;k++)
      {sum-=d[i-1][k-1]*d[k-1][j-1];}
    d[i-1][j-1]=sum;
    }
  if(j != n)
  { for(i=j+1;i<=n;i++)
      d[i-1][j-1]/=d[j-1][j-1];
  }
}
System.out.println(Matrix.toString(d));
return d;
}



  public static void main(String args[]) throws IOException
  {
  double a[][]={{2,-6,1},{-1,7,-1},{1,-3,2}};
  double b[]={12,-8,16};
  double c[][]= crautLU(a);
  double x[]=back_substitution(c,b);
  Text.print(x,"craut LU decomposition method");
  }
}
```



3.8 CHOLESKY SYMMETRIC MATRIX LU DECOMPOSITION METHOD

Cholesky method is similar to Dolittle(Gauss) LU method. This method is applied to symmetric matrix. Therefore, instead of LU, it decomposes to $[L][L]^T$ form.

$[A]=[L][L]^T$

For the decomposition process following steps are used

$l_{k,i} = \frac{a_{k,i}-\sum_{j=1}^{i-1} l_{i,j}l_{k,j}}{l_{i,i}}$ k=1,2,….,k-1

And

$$l_{k,k} = \sqrt{a_{k,k} - \sum_{j=0}^{n} l_{k,j}l_{k,j}}$$

Backsubstitution is similar to Dolittle method.

PROGRAM 3.8-1 Cholesky symmetric matrix decomposition method java version

```
import java.io.*;

class NA22
{
public static double[][] choleskyLU(double c[][])
{
//cholesky symmetric matrix LU
//symmetric matrices only
double factor=0;
int i,j,k,p,ii,jj;
int n=c.length;
double d[][]=new double[n][n];
double dummy;
double sum;
for(i=0;i<n;i++)
{for(j=0;j<n;j++) d[i][j]=c[i][j];}
for(k=1;k<=n;k++)
{
 for(i=1;i<=(k-1);i++)
   {sum=0;;
    for(j=1;j<=(i-1);j++)
    { sum+=d[i-1][j-1]*d[k-1][j-1]; }
      d[k-1][i-1]=(d[k-1][i-1]-sum)/d[i-1][i-1];

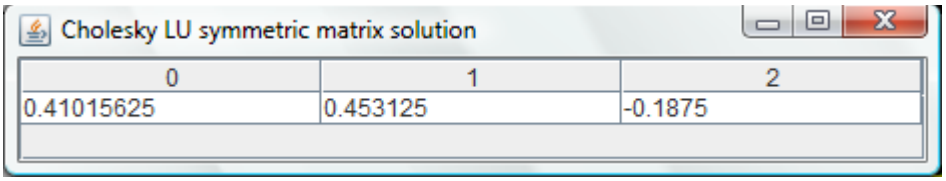
   }
 sum=0.0;
 for(j=1;j<=(k-1);j++)
 { sum+=d[k-1][j-1]*d[k-1][j-1]; }
 d[k-1][k-1]=Math.sqrt((d[k-1][k-1]-sum));
 for(j=k+1;j<=n;j++) d[k-1][j-1]=0.0;
```

```
}
return d;
}
  public static double[] substitutecholesky(double c[][],double b[])
  {
  //solving of system of equation by using cholesky decomposed matrix
  int i,j;
  double U[][]=T(c);
  double L[][]=c;
  int n=b.length;
  double sum=0;
  double x[]=new double[n];
  double d[]=new double[n];
  //forward substituting
  double toplam=0;
  for(i=0;i<n;i++)
  {toplam=0;
   for(j=0;j<i;j++)
   {toplam+=L[i][j]*d[j];}
   d[i]=(b[i]-toplam)/L[i][i];
  }
  //backward substitution
  x[n-1]=d[n-1]/U[n-1][n-1];
  for(i=n-2;i>=0;i--)
  {
    toplam=0;
    for(j=i+1;j<n;j++)
    {  toplam+=U[i][j]*x[j];}
    x[i]=(d[i]-toplam)/U[i][i];
  }
  return x;
  }
  public static double[][] T(double [][] left)
  {
  //transpose matrix (if A=a(i,j) T(A)=a(j,i)
  int i,j;
  int n=left.length;
  int m=left[0].length;
  double b[][];
  b=new double[m][n];
  for(i=0;i<n;i++)
  {
  for(j=0;j<m;j++)
    {
    b[j][i]=left[i][j];
    }
  }
  return b;
  }

  public static void main(String args[]) throws IOException
  {
  double a[][]={{4,-1,1},{-1,4.25,2.75},{1,2.75,3.5}};
  double b[]={1,1,1};
  double L[][]=choleskyLU(a);
  System.out.println("Matris L : \n"+Matrix.toString(L));
  double x[]=substitutecholesky(L,b);
```

```
  Text.print(x,"Cholesky LU symmetric matrix solution");
  }
}
```

```
---------- Capture Output ----------
> "C:\turhan\java\bin\javaw.exe" NA22
Matris L :
       2.000000000000000       0.000000000000000       0.000000000000000
      -0.500000000000000       2.000000000000000       0.000000000000000
       0.500000000000000       1.500000000000000       1.000000000000000
```



PROGRAM 3.8-2 Cholesky symmetric matrix decomposition method ptyhon version

```
 # -*- coding: utf-8 -*-
"""
Created on Thu Aug 23 09:46:37 2018

@author: Mustafa Turhan Çoban
"""
from math import *
def choleskyLU(c):
#cholesky symmetric matrix LU
#symmetric matrices only
    factor=0;
    n=len(c);
    d=[[0.0 for i in range(n)] for j in range(n)];
    for i in range(n):
        for j in range(n):
            d[i][j]=c[i][j];
    for k in range(1,n+1):
        for i in range(1,k):
            sum=0;
            for j in range(1,i):
                sum+=d[i-1][j-1]*d[k-1][j-1];
            d[k-1][i-1]=(d[k-1][i-1]-sum)/d[i-1][i-1];
        sum=0.0;
        for j in range(1,k):
            sum+=d[k-1][j-1]*d[k-1][j-1];
        d[k-1][k-1]=sqrt((d[k-1][k-1]-sum));
        for j in range(k+1,n+1):
            d[k-1][j-1]=0.0;
    return d;

def T(A):
    #transpose matrix
    n=len(A);
    m=len(A[0]);
    B = [[0 for x in range(m)] for y in range(n)];
    for i in range(n):
        for j in range(m):
            B[i][j]=A[j][i];
    return B;

def substitutecholesky(c,b):
```

```
#solving of system of equation by using cholesky decomposed matrix
U=T(c);
L=c;
n=len(b);
sum=0;
x=[0.0 for j in range(n)];
d=[0.0 for j in range(n)];
#forward substituting
toplam=0;
for i in range(n):
    toplam=0;
    for j in range(i):
        toplam+=L[i][j]*d[j];
    d[i]=(b[i]-toplam)/L[i][i];
#backward substitution
x[n-1]=d[n-1]/U[n-1][n-1];
for i in range(n-2,-1,-1):
    toplam=0;
    for j in range(i+1,n):
        toplam+=U[i][j]*x[j];
    x[i]=(d[i]-toplam)/U[i][i];
return x;

a=[[4.0,-1.0,1.0],[-1.0,4.25,2.75],[1.0,2.75,3.5]];
b=[1.0,1.0,1.0];
L=choleskyLU(a);
print("L=\n",L);
x=substitutecholesky(L,b);
print("x=\n",x);
```

```
runfile('D:/okul/SCO1/choleskyLU.py', wdir='D:/okul/SCO1')
L=
 [[2.0, 0.0, 0.0], [-0.5, 2.0, 0.0], [0.5, 1.5, 1.0]]
x=
 [0.41015625, 0.453125, -0.1875]
```

Let us solve the same problem step by step by hand:

$$\begin{bmatrix} 4 & -1 & 1 \\ -1 & 4.25 & 2.75 \\ 1 & 2.75 & 3.5 \end{bmatrix}$$

$$l_{11} = \sqrt{a_{11}} = \sqrt{4} = 2$$

$$l_{21} = \frac{a_{21}}{l_{11}} = \frac{-1}{2} = -0.5$$

$$l_{22} = \sqrt{a_{22} - l_{21}^2} = \sqrt{4.25 - (-0.5)^2} = 2$$

$$l_{32} = \frac{a_{32} - l_{21} - l_{31}}{l_{22}} = \frac{2.75 - (-0.5) * 0.5}{2} = 1.5$$

$$l_{33} = \sqrt{a_{33} - l_{31}^2 - l_{32}^2} = \sqrt{3.5 - (0.5)^2 - (1.5)^2} = 1$$

so cholesky decomposed L matrix :

$$[L] = \begin{bmatrix} 2 & 0 & 0 \\ -0.5 & 2 & 0 \\ 0.5 & 1.5 & 1 \end{bmatrix}$$

$$[U] = [L]^T = \begin{bmatrix} 2 & -0.5 & 0.5 \\ 0 & 2 & 1.5 \\ 0 & 0 & 1 \end{bmatrix}$$

$$[L]\{D\} = \{B\}$$

$$\begin{bmatrix} 2 & 0 & 0 \\ -0.5 & 2 & 0 \\ 0.5 & 1.5 & 1 \end{bmatrix} \begin{Bmatrix} d_1 \\ d_2 \\ d_3 \end{Bmatrix} = \begin{Bmatrix} 1 \\ 1 \\ 1 \end{Bmatrix}$$

d_1=1/2=0.5

d_2=(1-(-0.5)*0.5)/2=0.625

d_3=(1-0.5*0.5-1.5*0.625)/1=-0.1875

$$[U]\{x\} = \{D\}$$

$$\begin{bmatrix} 2 & -0.5 & 0.5 \\ 0 & 2 & 1.5 \\ 0 & 0 & 1 \end{bmatrix} \begin{Bmatrix} x_1 \\ x_2 \\ x_3 \end{Bmatrix} = \begin{Bmatrix} 0.5 \\ 0.625 \\ -0.1875 \end{Bmatrix}$$

x_3=-0.1875

x_2=(0.625-1.5*(-0.1875))/2=0.453125

x_1=(0.5-(-0.5)*0.453125-0.5*(-0.1875))/2=0.41015625

3.9 JACOBI GAUSS -SEIDEL AND SUCCESSIVE RELAXATION METHODS

In this section, iterative solution methods will be described. Iterative methods could be important in solution of big matrices. Gauss elimination requires $n^3/3$, Gauss-Jordan method requires $n^3/2$ process. When n is big, the total number of process will climb up. In this case the total number of processes in iterative methods will be much smaller. The simplest of the iterative methods are Jacobi method. In this method, a set of x solution vector is selected and substituted into the system of equation in the following form :

$$x_i^{k+1} = \frac{1}{A_{ij}}\left(b_i - \sum_{j \neq i} A_{ij} x_j^k\right) \quad 1 \leq i \leq n \qquad (3.9.1)$$

When a new set of x values is found, it will be substituted in the the equation as the next iteration step. To ensure the convergence matrix should be diagonally dominant

$$\sum_{i=1}^{n} |A_{ii}| \geq \sum_{i=1}^{n} \sum_{\substack{j=1 \\ j \neq i}}^{n} |A_{i,j}| \qquad (3.9.2)$$

Diagonal dominancy is the important condition for all iterative formulations. Unfortunately, very big matrices solved in engineering problems are in general fits to this definition.

In order to reduce number of iteration steps, every new calculated x value in the x vector set can be substited into the equation as soon as it is calculated and is not waited the complete set to be calculated as in Jacobi method. This form of iterative process is called Gauss-Seidel iteration.

$$x_i^{k+1} = \frac{1}{A_{ij}}\left(b_i - \sum_{j=1}^{i-1} A_{ij} x_j^{k+1} - \sum_{j=i+1}^{n} A_{ij} x_j^k\right) \quad 1 \leq i \leq n \qquad (3.9.3)$$

Gauss-Seidel and Jacobi methods can be merged to define an intermediate method between two by defining a relaxation coefficient, α, in the equation.

$$x_i^{k+1} = (1-\alpha) x_i^{k+1} + \frac{\alpha}{A_{ij}}\left(b_i - \sum_{j=1}^{i-1} A_{ij} x_j^{k+1} - \sum_{j=i+1}^{n} A_{ij} x_j^k\right) \quad 1 \leq i \leq n \qquad (3.9.4)$$

When α is equal to one equation coverged to the Gauss-Seidel method.

PROGRAM 3.9-1 Iterative relaxation method

```
import java.io.*;

class NA21
{
  public static double [] gauss_seidel(double a[][],double b[])
  {
  double lambda=1.2;
  int n=a.length;
  //starting x vector is zero
```

```
double x[]=new double[n];
return gauss_seidel(a,b,x,lambda);
}

public static double [] gauss_seidel(double a[][],double b[],double x[])
{
double lambda=1.2;
int n=a.length;
return gauss_seidel(a,b,x,lambda);
}

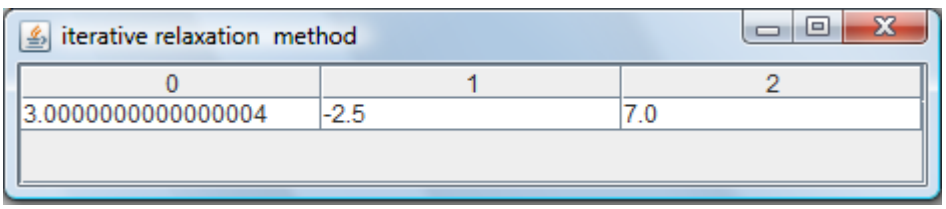
public static double [] gauss_seidel(double a[][],double b[],double x[],double lambda)
{
//lambda is the relaxation coefficient
int n=a.length;
int imax=500;//maximum number of iterations
double es=1.0e-20;
double dummy=0;
double ea=1e-1;
double sum=0;
double old=0;
int iter=0;
int sentinel=0;
System.out.print("iter = "+(iter+1)+" ");
for(int i=0;i<n;i++)
{  dummy=a[i][i];
   for(int j=0;j<n;j++)
     {a[i][j]/=dummy;}
   b[i]/=dummy;
}
for(int i=0;i<n;i++)
{  sum=b[i];
   for(int j=0;j<n;j++)
     {if(i!=j)sum-=a[i][j]*x[j];}
   x[i]=sum;
   System.out.print(x[i]+ "  ");
}
 System.out.println();
iter++;
do
{
sentinel=1;
System.out.print("iter = "+(iter+1)+" ");
for(int i=0;i<n;i++)
   {
            old=x[i];
            sum=b[i];
            for(int j=0;j<n;j++)
               {if(i!=j) sum-=a[i][j]*x[j];}
               x[i]=lambda*sum+(1.0-lambda)*old;
               if(sentinel==1 && x[i]!=0.0 ) {ea=Math.abs((x[i]-old)/x[i])*100.0;}
               if(ea<es) sentinel=0;
             System.out.print(x[i]+ "  ");
   }
System.out.println();
iter++;
}while(sentinel==1 && iter<imax);
```

```
  if(iter>=imax) System.out.println("Maximum number of iteration is exceed, result might not be valid iter = "+iter);
  return x;
  }

  public static void main(String args[]) throws IOException
  {
  double a[][]={{3,-0.1,-0.2},{0.1,7,-0.3},{0.3,-0.2,10}};
  double b[]={7.85,-19.3,71.4};

  double x[]={0,0,0};
  double lambda=0.0;
  Text.print(gauss_seidel(a,b),"iterative relaxation  method");
  }
}
```

```
---------- Capture Output ----------
> "C:\turhan\java\bin\javaw.exe" NA21
iter = 1 2.6166666666666667  -2.7945238095238096  7.005609523809524
iter = 2 3.0653344761904764  -2.4419267678911565  6.997919811665852
iter = 3 2.9890896189795266  -2.5115345924328882  7.000531981165179
iter = 4 3.0017632509999936  -2.49769594964207  6.999885423939555
iter = 5 2.999730345729483  -2.5004620799100574  7.000021532847987
iter = 6 3.00003717028554  -2.4999071138192726  6.999996584568461
iter = 7 2.999996008155598  -2.5000186844552923  7.000000378365779
iter = 8 3.0000000812599312  -2.4999962450431576  7.000000011520452
iter = 9 3.000000134867924  -2.500000752710909  6.999999974775603
iter = 10 2.999999940900027  -2.4999998497419305  7.000000010778672
iter = 11 3.000000018692611  -2.500000029817727  6.999999996455708
iter = 12 2.9999999947852256  -2.4999999941293365  7.0000000010374865
iter = 13 3.00000000136078  -2.500000001144104  6.999999999716058
iter = 14 2.9999999996593645  -2.499999999779943  7.0000000000743325
iter = 15 3.000000000082876  -2.500000000041609  6.999999999981153
iter = 16 2.9999999999802522  -2.4999999999923093  7.000000000004666
iter = 17 3.0000000000046305  -2.500000000001377  6.999999999998867
iter = 18 2.9999999999989284  -2.4999999999997646  7.000000000000272
iter = 19 3.000000000000245  -2.500000000000037  6.999999999999938
iter = 20 2.9999999999999445  -2.4999999999999956  7.000000000000015
iter = 21 3.0000000000000124  -2.5000000000000004  6.999999999999997
iter = 22 2.999999999999997  -2.5  7.000000000000001
iter = 23 3.0000000000000004  -2.5  7.0
```



PROGRAM 3.9-2 Iterative relaxation method, Python code

```
 # -*- coding: utf-8 -*-
"""
Created on Thu Aug 30 09:51:34 2018
@author: Mustafa Turhan Çoban
"""
```

```
def gauss_seidel(a,b,x,lembda):
    lamd=1.2;
    n=len(a);
    imax=500;#maximum number of iterations
    es=1.0e-20;
    dummy=0;
    ea=1e-1;
    sum=0;
    old=0;
    iter=0;
    sentinel=0;
    for i in range(n):
        dummy=a[i][i];
        for j in range(n):
            a[i][j]/=dummy;
        b[i]/=dummy;
    for i in range(n):
        sum=b[i];
        for j in range(n):
            if i!=j:sum-=a[i][j]*x[j];
        x[i]=sum;
    iter=iter+1;
    while iter<imax:
      sentinel=1;
      for i in range(n):
          old=x[i];
          sum=b[i];
          for j in range(n):
              if i!=j:
                  sum-=a[i][j]*x[j];
          x[i]=lamd*sum+(1.0-lamd)*old;
          if sentinel==1 and x[i]!=0.0: ea=abs((x[i]-old)/x[i])*100.0;
          if ea<es: return x;
      iter+=1;
    if iter>=imax: print("Maximum number of iteration is exceed, result might not be valid iter = ",iter);
    return x;

a=[[3,-0.1,-0.2],[0.1,7,-0.3],[0.3,-0.2,10]];
b=[7.85,-19.3,71.4];
x=[0,0,0];
lamd=0.0;
print("iterative relaxation  method",gauss_seidel(a,b,x,lamd));
```

```
runfile('D:/okul/SCO1/gauss_seidel.py', wdir='D:/okul/SCO1')
iterative relaxation  method [3.0000000000000004, -2.5, 7.000000000000001]
```

3.10 METHODS FOR BAND MATRICES

A three-band matrix can be shown as:

$$\begin{bmatrix} f_1 & g_1 & & & & & & \\ e_2 & f_2 & g_1 & & & & & \\ & e_3 & f_3 & g_1 & & & & \\ & & e_4 & f_4 & g_1 & & & \\ & & & e_5 & f_5 & g_1 & & \\ & & & & . & . & . & \\ & & & & & & f_{n-1} & g_{n-1} \\ & & & & & & e_n & f_n \end{bmatrix} \begin{Bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \\ . \\ x_{n-1} \\ x_n \end{Bmatrix} = \begin{Bmatrix} r_1 \\ r_2 \\ r_3 \\ r_4 \\ r_5 \\ . \\ r_{n-1} \\ r_n \end{Bmatrix}$$

This kind of matrix can be solved by using decomposition back and front substitution processes. Following form of basic solition process is called Thomas algorithm

Decomposition step :

```
for(int k=1;k<n;k++)
  {e[k]=e[k]/f[k-1];
   f[k]=f[k]-e[k]*g[k-1];
  }
```

Forward substitution :

```
for(int k=1;k<n;k++)
  {r[k]=r[k]-e[k]*r[k-1];
  }
```

Backward substitution:

```
x[n-1]=r[n-1]/f[n-1];
for(int k=(n-2);k>=0;k--)
  {x[k]=(r[k]-g[k]*x[k+1])/f[k];}
```

The whole Thomas algorithm is defined in the following code

Program 3.10-1 Thomas algorithm for band matrices (Java version)

```
import java.io.*;

class NA23
{
 public static double [] thomas(double a[][],double r[])
 {
 int n=a.length;
 double f[]=new double[n];
 double e[]=new double[n];
 double g[]=new double[n];
 double x[]=new double[n];
 for(int i=0;i<n;i++)     {f[i]=a[i][i];}
 for(int i=0;i<(n-1);i++) {g[i]=a[i][i+1];}
 for(int i=0;i<(n-1);i++) {e[i+1]=a[i+1][i];}
 for(int k=1;k<n;k++)
   {e[k]=e[k]/f[k-1];
    f[k]=f[k]-e[k]*g[k-1];
   }
 for(int k=1;k<n;k++)
   {r[k]=r[k]-e[k]*r[k-1];
   }
 x[n-1]=r[n-1]/f[n-1];
 for(int k=(n-2);k>=0;k--)
   {x[k]=(r[k]-g[k]*x[k+1])/f[k];}
 return x;
 }

 public static double [] thomas(double f[],double e[],double g[],double r[])
 {
 int n=f.length;
 double x[]=new double[n];
 for(int k=1;k<n;k++)
   {e[k]=e[k]/f[k-1];
    f[k]=f[k]-e[k]*g[k-1];
   }
 for(int k=1;k<n;k++)
   {r[k]=r[k]-e[k]*r[k-1];
```

```
    }
  x[n-1]=r[n-1]/f[n-1];
  for(int k=(n-2);k>=0;k--)
    {x[k]=(r[k]-g[k]*x[k+1])/f[k];}
  return x;
  }

  public static void main(String args[]) throws IOException
  {
  double f[]={2.04,2.04,2.04,2.04};
  double g[]={-1,-1,-1,0};
  double e[]={0,-1,-1,-1};
  double b[]={40.8,0.8,0.8,200.8};
  double x[]=thomas(f,e,g,b);
  Text.print(x,"thomas algorithm for 3 band matrix");
  System.out.println("thomas : \n"+Matrix.toStringT(x));
  double a[][]={{2.04,-1,0,0},{-1,2.04,-1,0 },{0,-1,2.04,-1},{0,0,-1,2.04}};
  double d[]={40.8,0.8,0.8,200.8};
  System.out.println("thomas : \n"+Matrix.toStringT(thomas(a,d)));
  }
}
```

```
thomas :
     65.969834366776620
     93.778462108224330
    124.538228334001000
    159.479523693137740
```

Another example with detailed calculation steps:

$$\begin{bmatrix} f_1 & g_1 & 0 \\ e_2 & f_2 & g_2 \\ 0 & e_3 & f_3 \end{bmatrix} \begin{Bmatrix} x_1 \\ x_2 \\ x_3 \end{Bmatrix} = \begin{Bmatrix} r_1 \\ r_2 \\ r_3 \end{Bmatrix}$$

$$\begin{bmatrix} 0.8 & -0.4 & 0 \\ -0.4 & 0.8 & -0.4 \\ 0 & -0.4 & 0.8 \end{bmatrix} \begin{Bmatrix} x_1 \\ x_2 \\ x_3 \end{Bmatrix} = \begin{Bmatrix} 41 \\ 25 \\ 105 \end{Bmatrix}$$

Decomposition:

$$e_k = \frac{e_k}{f_{k-1}}$$

$$f_k = f_k - e_k g_{k-1}$$

$$e_2 = \frac{e_2}{f_1} = \frac{-0.4}{0.8} = -0.5$$

$$f_2 = f_2 - e_2 g_1 = 0.8 - (-0.5)(-0.4) = 0.6$$

$$e_3 = \frac{e_3}{f_3} = \frac{-0.4}{0.6} = -0.666666666$$

$$f_3 = f_3 - e_3 g_3 = 0.8 - (-0.666666666)(-0.4) = 0.533333333$$

Matrix took the form of

$$\begin{bmatrix} 0.8 & -0.4 & 0 \\ -0.4 & 0.6 & -0.4 \\ 0 & -0.666666666 & 0.533333333 \end{bmatrix}$$

LU decomposed form of the matrix :

$$[A] = [L][U] = \begin{bmatrix} 1 & 0 & 0 \\ -0.5 & 1 & 0 \\ 0 & -0.66666 & 1 \end{bmatrix} \begin{bmatrix} 0.8 & -0.4 & 0 \\ 0 & 0.6 & -0.4 \\ 0 & 0 & 0.53333 \end{bmatrix}$$

Forward substitution

$$[L]\{D\} = \{B\}$$

$$\begin{bmatrix} 1 & 0 & 0 \\ -0.5 & 1 & 0 \\ 0 & -0.66666 & 1 \end{bmatrix} \begin{Bmatrix} d_1 \\ d_2 \\ d_3 \end{Bmatrix} = \begin{Bmatrix} 41 \\ 25 \\ 105 \end{Bmatrix}$$

d_1=41
d_2=25-(-0.5)*41=45.5
d_3=105-(-0.66666)*45.5=135.3333333
Backward substitution

$$[U]\{x\} = \{D\}$$

$$\begin{bmatrix} 0.8 & -0.4 & 0 \\ 0 & 0.6 & -0.4 \\ 0 & 0 & 0.53333 \end{bmatrix} \begin{Bmatrix} x_1 \\ x_2 \\ x_3 \end{Bmatrix} = \begin{Bmatrix} 41 \\ 45.5 \\ 135.33333 \end{Bmatrix}$$

X_3=135.33333/0.53333=253.75
X_2=(45.5-(-0.4)* 253.75)/0.6=245
X_1=(41-(-0.4)*245)/0.8=173.75

Python version of the Thomas algorithm is given below:

Program 3.10-2 Thomas algorithm for band matrices (python version)

```
# -*- coding: utf-8 -*-
"""
Created on Sun Jun  3 21:38:38 2018

@author: Mustafa Turhan Çoban
"""
def thomas(a,r):
    n=len(a);
    f=[0.0 for i in range(n)];
    e=[0.0 for i in range(n)];
    g=[0.0 for i in range(n)];
    x=[0.0 for i in range(n)];
    for i in range(0,n): f[i]=a[i][i];
    for i in range(0,n-1): g[i]=a[i][i+1];
    for i in range(0,n-1): e[i+1]=a[i+1][i];
    for k in range(0,n): e[k]=e[k]/f[k-1];f[k]=f[k]-e[k]*g[k-1];
    for k in range(1,n): r[k]=r[k]-e[k]*r[k-1];
    x[n-1]=r[n-1]/f[n-1];
    for k in range(n-2,-1,-1):  x[k]=(r[k]-g[k]*x[k+1])/f[k];
    return x;

def thomas_n(f,e,g,r):
    n=len(f);
    x=[0.0 for i in range(n)];
    for k in range(0,n): e[k]=e[k]/f[k-1];f[k]=f[k]-e[k]*g[k-1];
    for k in range(1,n): r[k]=r[k]-e[k]*r[k-1];
    x[n-1]=r[n-1]/f[n-1];
    for k in range(n-2,-1,-1):  x[k]=(r[k]-g[k]*x[k+1])/f[k];
    return x;

a=[[0.8,-0.4,0],[-0.4,0.8,-0.4],[0,-0.4,0.8]];
b=[41,25,105];
x=thomas(a,b);
print("x=",x);
```

```
runfile('D:/okul/SCO1/thomas.py', wdir='D:/okul/SCO1')
x= [173.75, 244.99999999999997, 253.74999999999997]
```

Five band matrix equation can be written as :

$$\begin{bmatrix} d_1 & c_1 & f_1 & & & & & \\ a_1 & d_2 & c_2 & f_2 & & & & \\ e_1 & a_2 & d_3 & c_3 & f_3 & & & \\ & e_2 & a_3 & d_4 & c_4 & f_4 & & \\ & & e_3 & a_4 & d_5 & c_5 & f_5 & \\ & & & . & . & . & . & f_{n-2} \\ & & & & e_{n-3} & a_{n-2} & d_{n-1} & c_{n-1} \\ & & & & & e_{n-2} & a_{n-1} & d_n \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \\ . \\ x_{n-1} \\ x_n \end{bmatrix} = \begin{bmatrix} b_1 \\ b_2 \\ b_3 \\ b_4 \\ b_5 \\ . \\ b_{n-1} \\ b_n \end{bmatrix}$$

It should be noted that bands are written as vectors, instead of a complete matrix. It will give us storage advantage as well as calculation advantage. A five band algorithm is given in the following program for the system of equation:

$$\mathrm{A} = \begin{bmatrix} 6 & -4 & 1 & 0 & 0 & \cdots \\ -4 & 6 & -4 & 1 & 0 & \cdots \\ 1 & -4 & 6 & -4 & 1 & \cdots \\ & \ddots & \ddots & \ddots & \ddots & \\ \cdots & 0 & 1 & -4 & 6 & -4 \\ \cdots & 0 & 0 & 1 & -4 & 7 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ \vdots \\ x_9 \\ x_{10} \end{bmatrix} = \begin{bmatrix} 3 \\ 0 \\ 0 \\ \vdots \\ 0 \\ 4 \end{bmatrix}$$

Program 3.10-2 five band matrix algorithm

```
import java.io.*;

class NA24
{
 public static double [] fiveband(double A[][],double b[])
 {
 //
 int n=A.length;
 double f[]=new double[n-1];
 double c[]=new double[n-1];
 double d[]=new double[n];
 double a[]=new double[n-1];
 double e[]=new double[n-2];
 double x[]=new double[n];

 for(int i=0;i<n;i++)    {d[i]=A[i][i];}
 for(int i=0;i<(n-1);i++) {c[i]=A[i][i+1];}
 for(int i=0;i<(n-1);i++) {a[i]=A[i+1][i];}
 for(int i=0;i<(n-2);i++) {f[i]=A[i][i+2];}
 for(int i=0;i<(n-2);i++) {e[i]=A[i+2][i];}
 int i;
 double xmult;
 for(i=1;i<(n-1);i++)
 {xmult=a[i-1]/d[i-1];
  d[i]+=-xmult*c[i-1];
  c[i]+=-xmult*f[i-1];
  b[i]+=-xmult*b[i-1];
  xmult=e[i-1]/d[i-1];
  a[i]+=-xmult*c[i-1];
  d[i+1]+=-xmult*f[i-1];
  b[i+1]+=-xmult*b[i-1];
  //System.out.println(i+"d="+d[i]+"c="+c[i]+"b="+b[i]);
 }
 xmult=a[n-2]/d[n-2];
 d[n-1]+=-xmult*c[n-2];
 x[n-1]=(b[n-1]-xmult*b[n-2])/d[n-1];
 x[n-2]=(b[n-2]-c[n-2]*x[n-1])/d[n-2];
 for(i=n-3;i>=0;i--)
```

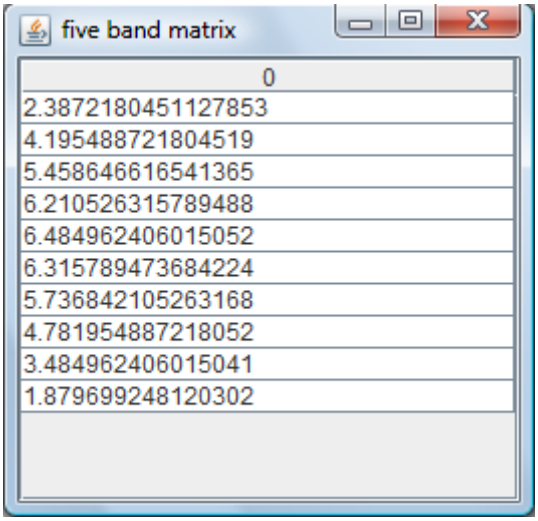
```
 {x[i]=(b[i]-f[i]*x[i+2]-c[i]*x[i+1])/d[i];}
 return x;
 }

  public static double [] fiveband(double f[],double c[],double d[],double a[],double e[],double b[])
 {
 int n=d.length;
 double x[]=new double[n];
 int i;
 double xmult;
 for(i=1;i<(n-1);i++)
 {xmult=a[i-1]/d[i-1];
  d[i]+=-xmult*c[i-1];
  c[i]+=-xmult*f[i-1];
  b[i]+=-xmult*b[i-1];
  xmult=e[i-1]/d[i-1];
  a[i]+=-xmult*c[i-1];
  d[i+1]+=-xmult*f[i-1];
  b[i+1]+=-xmult*b[i-1];
  //System.out.println(i+"d="+d[i]+"c="+c[i]+"b="+b[i]);
 }
 xmult=a[n-2]/d[n-2];
 d[n-1]+=-xmult*c[n-2];
 x[n-1]=(b[n-1]-xmult*b[n-2])/d[n-1];
 x[n-2]=(b[n-2]-c[n-2]*x[n-1])/d[n-2];
 for(i=n-3;i>=0;i--)
 {x[i]=(b[i]-f[i]*x[i+2]-c[i]*x[i+1])/d[i];}
 return x;
 }

 public static void main(String args[]) throws IOException
 {
 double d[]={6,6,6,6,6,6,6,6,6,7};
 double c[]={-4,-4,-4,-4,-4,-4,-4,-4,-4,-4};
 double f[]={1,1,1,1,1,1,1,1,1};
 double a[]={-4,-4,-4,-4,-4,-4,-4,-4,-4,-4};
 double e[]={1,1,1,1,1,1,1,1,1};

 double b[]={3,0,0,0,0,0,0,0,0,4};
 double x[]=fiveband(f,c,d,a,e,b);
 Text.printT(x,"five band matrix");
 }
}
```

The solution vector is :



3.11 CONJUGATE GRADIENT METHOD

If vector x^* is a solution of positive and well defined system of equation Ax = b, this solution is valid when the vector x^* is defined as minimum of function $g(x) = x^T Ax - 2b^T x$. The x vector is defined as a different vector then the solution vector x* . In addition a v vector with values different than 0 and a real number α is defined.

$$g(x + \alpha v) = g(x) + 2\alpha[v^T(Ax - B)] + \alpha^2[v^T(Av)]$$

In this equation, if v and x vector is known, for some value of α function will be minimum (One dimensional minimisation problem). If it is assumed that only independent variable of the function is α

$h(\alpha) = g(x + \alpha v)$ can be written. The minimum of the function, can be defined as the root of the derivative of the function with respect to α

$$h'^{(\alpha)} = 2[v^T(Ax - B)] + 2\alpha[v^T(Av)] = 0$$

From this equation

$$\alpha = \frac{v^T(Ax - B)}{v^T(Av)}$$

To simplify the equation a little bit further definition $r = b - Ax$

r = b – A x can be applied. If this definition is substituted into the previous equation:

$$\alpha = \frac{v^T r}{v^T(Av)}$$

If α is substituted into the iterative process iterative process

$$x_k = x_{k-1} + \alpha_k v_k$$

and for $r_k = b - Ax_k$ value would be

$$\alpha_k = \frac{v^T r_k}{v_k^T(Av_k)}$$

New x_k vector should be give solution to the $Ax_k = b$ equation . When a solution is obtained, it is known that equation will converged to

$$Ax_k = A(x_{k-1} + \alpha_k v_k) = b$$

But we still do not know the search direction vector, v_k . As a starting search direction **steepest descent** optimization direction can be selected. This direction is defined as

$$\nabla g(x) = 2(ax - b) = -2r$$

In this case v_{k+1} can be selected as

$$v_{k+1} = -\nabla g(x_k) = r_k$$

This direction gives the maximum negative change in $g(x_k)$. An alternative selection to the direction vector can be carried out such a vector that for this vector condition $v_i^T A v_j = 0 \quad i \neq j$ will be valid for a non-zero v vector. This type of vectors are called orthogonal vectors. If some orthogonal v vectors can be found, it will simplify our solution. In orthogonal sets terms out of diagonal will be zero so only diagonal terms will remain in the equation. Therefore process of solving matrix system will be converted to simple multiplication and division processes. The orthogonal set suggested by Hesteness and Stiefel has the following form :

$$p_k = \frac{r_k^T r_k}{r_{k-1}^T r_{k-1}}$$

$$v_{k+1} = r_k + p_k v_k$$

As it is seen from the equation, values are derived from the previous sets. In summary this method can be applied as:

In the first iteration steepest descent value is taken.

$r_0 = b - Ax_0$ and $v_0 = r_0$(steepest descent value) .

then for the next iteration sets calculates the values as

for k=1 to n

$$\alpha_k = \frac{r_{k-1}^T r_k}{v_k^T A v_k}$$

$$x_k = x_{k-1} + \alpha_k v_k$$

$$r_k = b - Ax_k$$
$$p_k = \frac{r_k^T r_k}{r_{k-1}^T r_{k-1}}$$
$$v_{k+1} = r_k + p_k v_k$$

As it is seen process is quite a complex process compared to Gauss elimination type of solution. It is not preffered to Gauss elimination in solution of the small systems, but when the system becomes larger, it will be prefferable.

3.11-1 Conjugate-gradient method to solve system of equations

```
import java.util.*;
import java.awt.*;
import java.applet.Applet;
import java.awt.event.*;
import javax.swing.*;
import java.io.*;
//CONJUGATE GRADIENT METHOD
public class NA25
{
public static double VT_V(double [] left)
{
//multiplies a vector transpose with a vector
int n=left.length;
double tot=0;
for(int i=0;i<n;i++)
  {
  tot+=left[i]*left[i];
  }
return tot;
}

public static double[][] V_VT(double [] left)
{
//multiplys a vector transpose with a vector
int n=left.length;
double aa[][]=new double[n][n];
for(int i=0;i<n;i++)
  {
  for(int j=0;j<n;j++)
    {
              aa[i][j]=left[i]*left[j];
              }
  }
return aa;
}

public static double  VT_X(double [] left,double [] right)
{
//multiplys a vector transpose with a vector
int n=left.length;
double tot=0;
for(int i=0;i<n;i++)
  {
  tot+=left[i]*right[i];
  }
return tot;
}
```

```
public static double[] multiply(double[] left,double[][] right)
{
//multiplication of one vector with one matrix
int ii,jj,i,j,k;
int m2=right[0].length;
int n2=right.length;
int m1=left.length;
double[] b;
b=new double[m1];
 if(n2 != m1)
 {
 System.out.println("inner matrix dimensions must agree");
 for(ii=0;ii<n2;ii++)
   {
    b[ii]=0;
   }
  return b;
 }
 for(i=0;i<m2;i++)
   {
    b[i]=0;
    for(k=0;k<m1;k++)
            b[i]+=right[i][k]*left[k];
   }
return b;
//end of multiply of a vector and a matrix
}

public static double[] multiply(double[][] left,double[] right)
{
//multiplication of one matrix with one vector
int ii,jj,i,j,k;
int m1=left[0].length;
int n1=left.length;
int m2=right.length;
double[] b;
b=new double[m2];
 if(n1 != m2)
 {
 System.out.println("inner matrix dimensions must agree");
 for(ii=0;ii<n1;ii++)
   {
    b[ii]=0;
   }
  return b;
 }
 for(i=0;i<m1;i++)
   {
    b[i]=0;
    for(k=0;k<n1;k++)
            b[i]+=left[i][k]*right[k];
   }
return b;
//end of multiply of a matrix and a vector
}
public static double[] multiply(double left,double[] right)
{
```

```
//multiplying a vector with a constant
int i;
int n=right.length;
double b[];
b=new double[n];
for(i=0;i<n;i++)
  {
  b[i]=left*right[i];
  }
return b;
}
public static double[] substract(double[] left,double[] right)
{
//addition of two vectors
int n1=left.length;
int n2=right.length;
int nMax;
int i;
if(n1>=n2) nMax=n1;
else       nMax=n2;
double b[];
b=new double[nMax];
for(i=0;i<n1;i++)
  {
    b[i]=b[i]+left[i];
  }
for(i=0;i<n2;i++)
  {
    b[i]=b[i]-right[i];
  }
return b;
//end of vector substraction method
}
public static double[] add(double[] left,double[] right)
{
//addition of two vectors
int n1=left.length;
int n2=right.length;
int nMax;
int i;
if(n1>=n2) nMax=n1;
else       nMax=n2;
double b[];
b=new double[nMax];
for(i=0;i<n1;i++)
  {
    b[i]=b[i]+left[i];
  }
for(i=0;i<n2;i++)
  {
    b[i]=b[i]+right[i];
  }
return b;
//end of vector addition method
}

public static double[] equals(double[] left)
```

```
{
//addition of two vectors
int n1=left.length;
double b[];
b=new double[n1];
for(int i=0;i<n1;i++)
  {
    b[i]=left[i];
  }
return b;
//end of vector equal method
}

public static double norm(double v[])
{
// vector norm
double total=0;;
for(int i=0;i<v.length;i++)
  {total+=v[i]*v[i];}
return Math.sqrt(total);
}
public static double VT_G_V(double V[],double G[][])
{
  double x1[]=multiply(V,G);
  return VT_X(V,x1);
}

public static double[] conjugate_gradient(double A[][],double b[])
{int n=A.length;
 double x[]=new double[n];
 return conjugate_gradient(x,A,b);
}


public static double[] conjugate_gradient(double x[],double A[][],double b[])
{ // iterative system of equation solution
double eps=1.0e-15;
int maxiter=500;
int n=x.length;
double r[]=new double[n];
double re[]=new double[n];
double v[]=new double[n];
double alpha;
double p;
r=substract(b,multiply(A,x));
v=equals(r);
int k=0;
do
  {
  re=equals(r);
  alpha=VT_V(re)/VT_G_V(v,A);
  x=add(x,multiply(alpha,v));
  r=substract(r,multiply(multiply(alpha,v),A));
  p=VT_V(r)/VT_V(re);
  v=add(r,multiply(p,v));
  k++;
  }while(norm(r)>eps && k<maxiter);
```

```
System.out.println("k = "+k+" norm(r)="+norm(r));
return x;
}

  public static void main(String args[]) throws IOException
  {
    double A[][]={{0.2,0.1,1,1,0},{0.1,4,-1,1,-1},{1,-1,60,0,-2},{1,1,0,8,4},{0,-1,-2,4,700}};
  double b[]={1,2,3,4,5};
  double x[]=conjugate_gradient(A,b);
  System.out.println("conjugate gradient method  : \n"+Matrix.toStringT(x));
  }
}
```

```
---------- Capture Output ----------
> "C:\turhan\java\bin\javaw.exe" NA25
k = 9 norm(r)=5.161554370481792E-18
conjugate gradient method  :
       7.859713075445860
       0.422926408295008
      -0.073592239024046
      -0.540643016894627
       0.010626162854036

> Terminated with exit code 0.
```

Numerical example

To illustrate the conjugate gradient method, we will complete a simple example. Considering the linear system Ax = b given by

$$\mathbf{Ax} = \begin{bmatrix} 4 & 1 \\ 1 & 3 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} 1 \\ 2 \end{bmatrix},$$

we will perform two steps of the conjugate gradient method beginning with the initial guess

$$\mathbf{x}_0 = \begin{bmatrix} 2 \\ 1 \end{bmatrix}$$

in order to find an approximate solution to the system.

Solution

```
k= 0  alpha = 0.25  p=0.0625
r1 =       -0.500000000000000          0.250000000000000
v1 =       -0.437500000000000          0.375000000000000
x1 =        0.250000000000000          0.500000000000000
k= 1 alpha = 0.36363636363636365  p=2.465190328815662E-33
r2 =        0.000000000000000          0.000000000000000
v2 =       -0.000000000000000          0.000000000000000
x2 =        0.090909090909091          0.636363636363636
```

The result, $\mathbf{x}_2$, is a "better" approximation to the system's solution than $\mathbf{x}_1$ and $\mathbf{x}_0$. If exact arithmetic were to be used in this example instead of limited-precision, then the exact solution would theoretically have been reached after $n = 2$ iterations (n being the order of the system).

We will now extend the conjugate gradient method to include ***preconditioning*.** If the matrix A is ill-conditioned, the conjugate gradient method is highly susceptible to rounding errors. So, although the exact answer should be obtained in n steps, this is not usually the case. As a direct method the conjugate gradient method is not as good as Gaussian elimination with pivoting. The main use of the conjugate gradient method is as an iterative method

applied to a better-conditioned system. To apply the method to a better-conditioned system, we want to select a nonsingular conditioning matrix *C* so that system of equation can be converted to a better conditioned system.

$A^* = C^{-1}AC^{-1} \qquad x^* = C^{-1}x \qquad b^* = C^{-1}b \quad A^*x^* = b^*$

As a C conditioning matrix, a matrix with the rootsquare values of the diagonal elements and 0 values for the rest of the elements can be selected. For such a matrix, inverse matrix can be produced just by dividing numbers in the matrix by 1. Another possibility for the conditioning matrix could be unit matrix.

If the whole procedure is given as a recipe:

Input : Matrix A,b and x , C^{-1} inverse of positive conditioning matrix

Step 1 : $r_0 = b - A\,x$

$w_0 = C^{-1}r_0$

$v_0 = C^{-1}\,w_0$

$$\gamma_0 = \sum_{j=1}^{n} w_j^2$$

Step 2 : k=1

Step 3 : while(k<=N) do adim 4-5

Step 4 : $u=Av_k$

$$\alpha_k = \frac{\gamma_{k-1}}{\sum_{j=1}^{n} v_j u_j}$$

$x_k=x_{k-1} + \alpha_k v_k$

$r_k=r_{k-1} + \alpha_k u_k$

$w_k = C^{-1}r_k$

$$\gamma_k = \sum_{j=1}^{n} w_j^2$$

Step 5 : $p_k= \beta_k/\gamma_k$

$v_{k+1} = C^{-1}\,w_k + p_k v_k$

$\gamma_k = \beta_k$

k=k+1

Preconditioned conjugate - gradient program using above recipe is given below

PROBLEM 3.11-2 Conditioned conjugate-gradient method to solve system of equations (java version)

```
import java.util.*;
import java.awt.*;
import java.applet.Applet;
import java.awt.event.*;
import javax.swing.*;
import java.io.*;
//reconditioned Conjugate Gradient
public class NA26
{
public static double VT_V(double [] left)
{
//multiplies a vector transpose with a vector
int n=left.length;
double tot=0;
for(int i=0;i<n;i++)
  {
  tot+=left[i]*left[i];
  }
return tot;
}
```

```
public static double[][] V_VT(double [] left)
{
//multiplys a vector transpose with a vector
int n=left.length;
double aa[][]=new double[n][n];
for(int i=0;i<n;i++)
  {
  for(int j=0;j<n;j++)
    {
                aa[i][j]=left[i]*left[j];
                }
  }
return aa;
}

public static double  VT_X(double [] left,double [] right)
{
//multiplys a vector transpose with a vector
int n=left.length;
double tot=0;
for(int i=0;i<n;i++)
  {
  tot+=left[i]*right[i];
  }
return tot;
}
public static double[][] multiply(double left,double[][] right)
{
//multiplying a matrix with a constant
int i,j;
int n=right.length;
int m=right[0].length;
double b[][];
b=new double[n][m];
for(i=0;i<n;i++)
  {
  for(j=0;j<m;j++)
    b[i][j]=right[i][j]*left;
  }
return b;
//end of multiplying a matrix with a constant double
}
public static double[] multiply(double[] left,double[][] right)
{
//multiplication of one vector with one matrix
int ii,jj,i,j,k;
int m2=right[0].length;
int n2=right.length;
int m1=left.length;
double[] b;
b=new double[m1];
 if(n2 != m1)
 {
 System.out.println("inner matrix dimensions must agree");
 for(ii=0;ii<n2;ii++)
   {
     b[ii]=0;
   }
   return b;
 }
 for(i=0;i<m2;i++)
   {
     b[i]=0;
     for(k=0;k<m1;k++)
```

```
                b[i]+=right[i][k]*left[k];
   }
return b;
//end of multiply of a vector and a matrix
}

public static double[] multiply(double[][] left,double[] right)
{
//multiplication of one matrix with one vector
int ii,jj,i,j,k;
int m1=left[0].length;
int n1=left.length;
int m2=right.length;
double[] b;
b=new double[m2];
 if(n1 != m2)
 {
 System.out.println("inner matrix dimensions must agree");
 for(ii=0;ii<n1;ii++)
   {
    b[ii]=0;
   }
  return b;
 }
 for(i=0;i<m1;i++)
   {
    b[i]=0;
    for(k=0;k<n1;k++)
                b[i]+=left[i][k]*right[k];
   }
return b;
//end of multiply of a matrix and a vector
}
public static double[] multiply(double left,double[] right)
{
//multiplying a vector with a constant
int i;
int n=right.length;
double b[];
b=new double[n];
for(i=0;i<n;i++)
  {
  b[i]=left*right[i];
  }
return b;
}
public static double[] substract(double[] left,double[] right)
{
//addition of two vectors
int n1=left.length;
int n2=right.length;
int nMax;
int i;
if(n1>=n2) nMax=n1;
else       nMax=n2;
double b[];
b=new double[nMax];
for(i=0;i<n1;i++)
  {
   b[i]=b[i]+left[i];
  }
for(i=0;i<n2;i++)
  {
   b[i]=b[i]-right[i];
```

```
  }
return b;
//end of vector substraction method
}
public static double[] add(double[] left,double[] right)
{
//addition of two vectors
int n1=left.length;
int n2=right.length;
int nMax;
int i;
if(n1>=n2) nMax=n1;
else       nMax=n2;
double b[];
b=new double[nMax];
for(i=0;i<n1;i++)
  {
    b[i]=b[i]+left[i];
  }
for(i=0;i<n2;i++)
  {
    b[i]=b[i]+right[i];
  }
return b;
//end of vector addition method
}
public static double[] equals(double[] left)
{
//addition of two vectors
int n1=left.length;
double b[];
b=new double[n1];
for(int i=0;i<n1;i++)
  {
    b[i]=left[i];
  }
return b;
//end of vector equal method
}
public static double norm(double v[])
{return Math.sqrt(normkare(v));}

public static double normkare(double v[])
{
// vector norm
double total=0;;
for(int i=0;i<v.length;i++)
  {total+=v[i]*v[i];}
return total;
}

public static double VT_G_V(double V[],double G[][])
{
  double x1[]=multiply(V,G);
  return VT_X(V,x1);
}

public static double[][] I(int n)
{
//unit matrix
double b[][];
b=new double[n][n];
for(int i=0;i<n;i++)
    b[i][i]=1.0;
```

```
return b;
}

public static double[][] D(double A[][])
{
//Diagonal elements of matrix
int n=A.length;
double b[][];
b=new double[n][n];
for(int i=0;i<n;i++)
   b[i][i]=A[i][i];
return b;
}

public static double[][] invsqrtD(double A[][])
{
//square root of Diagonal elements of a matrix
int n=A.length;
double b[][];
b=new double[n][n];
for(int i=0;i<n;i++)
   b[i][i]=1.0/Math.sqrt(A[i][i]);
return b;
}

public static double[] conjugate_gradient(double x[],double A[][],double b[])
{   int n=A.length;
            double C[][]=new double[n][n];
            C=invsqrtD(A);
   return conjugate_gradient(x,A,b,C);
}

public static double[] conjugate_gradient(double A[][],double b[])
{   int n=A.length;
            double C[][]=new double[n][n];
            C=invsqrtD(A);
            double x[]=new double[n];
            //başlangıç x değerleri 0 olarak alındı
   return conjugate_gradient(x,A,b,C);
}

public static double[] conjugate_gradient(double x[],double A[][],double b[],double C[][])
{ // iterative system of equation solution
  // x is the initial guess
  // C is the conditioning matrix
double eps=1.0e-15;
int maxiter=500;
int n=x.length;
double r[]=new double[n];
double re[]=new double[n];
double v[]=new double[n];
double w[]=new double[n];
double u[]=new double[n];
double alpha;
double gamma,beta,p;
int k=0;
r=substract(b,multiply(A,x));
w=multiply(C,r);
v=multiply(C,w);
gamma=normkare(w);
do
  {
  k++;
  if(norm(v)<eps) return x;
```

```
  re=equals(r);
  u=multiply(A,v);
  alpha=gamma/VT_X(v,u);
  x=add(x,multiply(alpha,v));
  r=substract(r,multiply(alpha,u));
  w=multiply(C,r);
  beta=normkare(w);
  if(beta<eps)
  {if(norm(r)<eps) return x;}
  p=beta/gamma;
  v=add(multiply(C,w),multiply(p,v));
  gamma=beta;
  }while(norm(r)>eps && beta>eps && k<maxiter );
  //System.out.println("k="+k);
return x;
}
  public static void main(String args[]) throws IOException
  {
  double A[][]={{0.2,0.1,1,1,0},{0.1,4,-1,1,-1},{1,-1,60,0,-2},{1,1,0,8,4},{0,-1,-2,4,700}};
  double b[]={1,2,3,4,5};
  double x[]=conjugate_gradient(A,b);
  System.out.println("conditioned conjugate gradient method : \n"+Matrix.toStringT(x));
  }
}
```

```
---------- Capture Output ----------
> "C:\turhan\java\bin\javaw.exe" NA26
conditioned conjugate gradient method :
       7.859713075445862
       0.422926408295008
      -0.073592239024046
      -0.540643016894627
       0.010626162854036
> Terminated with exit code 0.
```

PROBLEM 3.11-3 Conditioned conjugate-gradient method to solve system of equations (python version)

```
# -*- coding: utf-8 -*-
"""
Created on Thu Aug 23 11:49:55 2018

@author: Mustafa Turhan Çoban
"""
from math import *
def VT_V(left):
   #multiplies a vector transpose with a vector
   n=len(left);
   tot=0;
   for i in range(n): tot+=left[i]*left[i];
   return tot;

def V_VT(left):
   #multiplys a vector with a vector             transpose
   n=len(left);
   aa = [[0 for x in range(n)] for y in range(n)];
   for i in range(n):
      for j in range(n): aa[i][j]=left[i]*left[j];
   return aa;

def  VT_X(left,right):
   #multiplys a vector transpose with a vector
   n=len(left);
   tot=0;
   for i in range(n):tot+=left[i]*right[i];
```

```
    return tot;

def multiply_c_m(left,right):
    #multiplying a matrix with a constant
     n=len(rigth);
     m=len(right[0]);
     b = [[0 for x in range(m)] for y in range(n)];
     for i in range(n):
         for j in range(m):
             b[i][j]=right[i][j]*left;
     return b;

def multiply_v_m(left,right):
    #multiplication of one vector with one matrix
    m2=len(right[0]);
    n2=len(right);
    m1=len(left);
    b = [0.0 for i in range(m1)];
    if n2 != m1:
        print("inner matrix dimensions must agree");
        for ii in range(n2):b[ii]=0;
        return b;
    for i in range(m2):
        b[i]=0;
        for k in range(m1):
            b[i]+=right[i][k]*left[k];
    return b;

def multiply_m_v(left,right):
    #multiplication of one matrix with one vector
    m1=len(left[0]);
    n1=len(left);
    m2=len(right);
    b = [0.0 for i in range(m2)];
    if n1 != m2:
        print("inner matrix dimensions must agree");
        for ii in range(n1):b[ii]=0;
        return b;
    for i in range(m1):
        b[i]=0;
        for k in range(n1):
            b[i]+=left[i][k]*right[k];
    return b;

def multiply_c_v(left,right):
    #multiplying a vector with a constant
    n=len(right);
    b = [0.0 for i in range(n)];
    for i in range(n):
        b[i]=left*right[i];
    return b;

def substract(left,right):
    #addition of two vectors
    n1=len(left);
    n2=len(right);
    if n1>=n2: nMax=n1;
    else     : nMax=n2;
    b = [0.0 for i in range(nMax)];
    for i in range(n1):
        b[i]=b[i]+left[i];
    for i in range(n2):
        b[i]=b[i]-right[i];
    return b;
```

```
def add(left,right):
    #addition of two vectors
    n1=len(left);
    n2=len(right);
    if n1>=n2: nMax=n1;
    else     : nMax=n2;
    b = [0.0 for i in range(nMax)];
    for i in range(n1):
        b[i]=b[i]+left[i];
    for i in range(n2):
        b[i]=b[i]+right[i];
    return b;

def equals(left):
    #addition of two vectors
     n1=len(left);
     b = [0.0 for i in range(n1)];
     for i in range(n1):
         b[i]=left[i];
     return b;

def norm(v):
    return sqrt(normkare(v));

def normkare(v):
    #vector norm
    total=0;
    n=len(v);
    for i in range(n):
        total+=v[i]*v[i];
    return total;

def VT_G_V(V,G):
  x1=multiply_v_m(V,G);
  return VT_X(V,x1);

def I(n):
    #unit matrix
    b = [0.0 for i in range(n)];
    for i in range(n):
        b[i][i]=1.0;
    return b;

def D(A):
    #Diagonal elements of matrix
    n=len(A);
    b = [[0 for x in range(n)] for y in range(n)];
    for i in range(n):
        b[i][i]=A[i][i];
    return b;

def invsqrtD(A):
    #square root of Diagonal elements of a matrix
    n=len(A);
    b = [[0 for x in range(n)] for y in range(n)];
    for i in range(n):
        b[i][i]=1.0/sqrt(A[i][i]);
    return b;

def conjugate_gradient(x,A,b):
    n=len(A);
    C = [[0 for x in range(n)] for y in range(n)];
    C=invsqrtD(A);
```

```
    return conjugate_gradient1(x,A,b,C);

def conjugate_gradient(A,b):
    n=len(A);
    C = [[0 for x in range(n)] for y in range(n)];
    C=invsqrtD(A);
    x = [0.0 for i in range(n)];
    return conjugate_gradient1(x,A,b,C);

def conjugate_gradient1(x,A,b,C):
    #iterative system of equation solution
    #x is the initial guess
    #C is the conditioning matrix
    eps=1.0e-15;
    maxiter=500;
    n=len(x);
    x = [0.0 for i in range(n)];
    r = [0.0 for i in range(n)];
    v = [0.0 for i in range(n)];
    w = [0.0 for i in range(n)];
    u = [0.0 for i in range(n)];
    k=0;
    r=substract(b,multiply_m_v(A,x));
    w=multiply_m_v(C,r);
    v=multiply_m_v(C,w);
    beta=0;
    gamma=normkare(w);
    nr=norm(r);
    while (nr>eps and k<maxiter):
        k=k+1;
        if norm(v)<eps: return x;
        re=equals(r);
        u=multiply_m_v(A,v);
        alpha=gamma/VT_X(v,u);
        x=add(x,multiply_c_v(alpha,v));
        r=substract(r,multiply_c_v(alpha,u));
        nr=norm(r);
        w=multiply_m_v(C,r);
        beta=normkare(w);
        if beta<eps:
            nr=norm(r);
            if nr<eps: return x;
        p=beta/gamma;
        v=add(multiply_m_v(C,w),multiply_c_v(p,v));
        gamma=beta;
    return x;

A=[[0.2,0.1,1,1,0],[0.1,4,-1,1,-1],[1,-1,60,0,-2],[1,1,0,8,4],[0,-1,-2,4,700]];
b=[1,2,3,4,5];
x=conjugate_gradient(A,b);
print("conditioned conjugate gradient method : \n",x);
```

```
runfile('D:/okul/SCO1/conjugate_gradient.py', wdir='D:/okul/SCO1')
conditioned conjugate gradient method :
 [7.859713075445861, 0.4229264082950077, -0.07359223902404635, -0.5406430168946268, 0.010626162854036319]
```

3.12 THE RESIDUAL CORRECTION METHOD

We assume that Ax=b has ben solved for an approximate solution $x=x^{(0)}$. Also the LU decomposition along with a record of all row or column interchanges should been stored. Calculate residual

$r^{(0)}=b-Ax^{(0)}$

Define $e^{(0)}=x-x^{(0)}$. Then solve

$Ae^{(0)}=r^{(0)}$

Solve this system using the stored LU decomposition .If the resulting approximate solution is $e^{(0)}$. Define a new approximate solution to Ax=b by

$x^{(1)}=x^{(0)}+e^{(0)}$

define also error term $\varepsilon = \|x\|_e = \sqrt{\sum_{i=0}^{n} x_i^2} < \varepsilon_0$

convert iterations until error term reduce below the given range. As an example Hilbert matrix is taken as one of the ill conditioning matrix example. The n x n Hilbert matrix defined by:

$$H_{ij}^{(n)} = \frac{1}{i+j-1}$$

Hilbert Matrix defined in Mathd class

```
//The Hilbert Matrix
public static double[][] Hilbert(int n)
{
double H[][]=new double[n][n];
for(int i=0;i<n;i++)
{for(int j=0;j<n;j++)
 {H[i][j]=1.0/(i+j+1.0);}
}
return H;
}
```

Problem 3.12-1 The residual correction by using LU decomposition

```
import java.io.*;
import javax.swing.*;
class residual
{
  public static void pivot(double a[][],int o[],double s[],int k)
  {// Gauss(Dolittle) LU decomposition with partial pivoting
    int n=a.length;
    int dummy1=0;
    double buyuk;
    double dummy=0;
    //pivotlama
              int p=k;
              buyuk=Math.abs(a[o[k]][k]);
              for(int ii=k+1;ii<n;ii++)
              {  dummy=Math.abs(a[o[ii]][k]/s[o[ii]]);
                if(dummy > buyuk) {buyuk=dummy;p=ii;}
              }
              dummy1=o[p];
              o[p]=o[k];
              o[k]=dummy1;
  }

  public static int[] sirali_indeks(int n)
  {   // sıralı rakamları taşıyan bir referans vektörü oluşturur
               int o[]=new int[n];
               for(int i=0;i<n;i++) {o[i]=i;}
               return o;
  }

  public static double[][] gaussLU(double ai[][],int o[])
  { //kısmi pivotlu gauss eleme yöntemi
  int n=ai.length;
```

```
double a[][]=new double[n][n];
double s[]=new double[n];
double carpan=0;
double toplam=0;
double absaij;

//gauss eleme
int i,j,k,p,ii,jj;
//System.out.println("Orijinal Matris :\n"+Matrix.toString(ai));
for(i=0;i<n;i++)
{for(j=0;j<n;j++){a[i][j]=ai[i][j];}}
for(i=0;i<n;i++)
{ o[i]=i;s[i]=Math.abs(a[i][0]);
  for(j=1;j<n;j++)
    { absaij=Math.abs(a[i][j]);
      if(absaij>s[i]) {s[i]=absaij;}
    }
}
for(k=0;k<(n-1);k++)
{ pivot(a,o,s,k);
            //pivot referansını değiştir
            //System.out.println("pivotlu Matris : k="+k+"\n"+Matrix.toString(a));
            //gauss elemeyi çözme
  for(i=k+1;i<n;i++)
  {  carpan=a[o[i]][k]/a[o[k]][k];
     a[o[i]][k]=carpan;
     for(j=k+1;j<n;j++)
     {   a[o[i]][j]-=carpan*a[o[k]][j]; }
            }
}
return a;
}

public static double[] yerine_koyma(double ai[][],double bi[])
{   int n=ai.length;
              return yerine_koyma(ai,bi,sirali_indeks(n));
}

public static double[][] yerine_koyma(double ai[][],double bi[][])
{   int n=ai.length;
              return yerine_koyma(ai,bi,sirali_indeks(n));
}

public static double[] yerine_koyma(double ai[][],double bi[],int o[])
{ //gauss LU çözümüyle LU indirgenmesi yapılmış Matris kullanarak linear
  //denklem sistemi kökleri
  int n=ai.length;
  double toplam;
  double x[]=new double[n];
   double a[][]=new double[n][n];
  double b[]=new double[n];
  int i,j;
for(i=0;i<n;i++)
    {for(j=0;j<n;j++){a[i][j]=ai[i][j];};b[i]=bi[i];}
//System.out.println("a=\n "+Matrix.toString(a));
//System.out.println("b=\n "+Matrix.toStringT(b));
for(i=1;i<n;i++)
```

```
    {toplam=b[o[i]];for(j=0;j<i;j++){toplam-=a[o[i]][j]*b[o[j]];};b[o[i]]=toplam;}
 //System.out.println("d=\n "+Matrix.toStringT(b));
x[n-1]=b[o[n-1]]/a[o[n-1]][n-1];
for(i=(n-2);i>=0;i--)
    {toplam=0;
     for(j=i+1;j<n;j++)
      {toplam+=a[o[i]][j]*x[j];};
      x[i]=(b[o[i]]-toplam)/a[o[i]][i];
    }
//System.out.println("d=\n "+Matrix.toStringT(x));
return x;
}

public static double[][] yerine_koyma(double ai[][],double bi[][],int o[])
{
  int n=ai.length;
  int m=bi[0].length;
  double toplam;
  double x[][]=new double[n][m];
  double a[][]=new double[n][n];
  double b[][]=new double[n][m];
  int i,j,k;
for(i=0;i<n;i++)
    {for(j=0;j<n;j++){a[i][j]=ai[i][j];};for(k=0;k<m;k++) {b[i][k]=bi[i][k];}}
//System.out.println("a=\n "+Matrix.toString(a));
//System.out.println("b=\n "+Matrix.toStringT(b));
for(k=0;k<m;k++)
{
for(i=1;i<n;i++)
    {toplam=b[o[i]][k];for(j=0;j<i;j++){toplam-=a[o[i]][j]*b[o[j]][k];};b[o[i]][k]=toplam;}
 //System.out.println("d=\n "+Matrix.toStringT(b));
x[n-1][k]=b[o[n-1]][k]/a[o[n-1]][n-1];
for(i=(n-2);i>=0;i--)
    {toplam=0;
     for(j=i+1;j<n;j++)
      {toplam+=a[o[i]][j]*x[j][k];};
      x[i][k]=(b[o[i]][k]-toplam)/a[o[i]][i];
    }
}//kin sonu
//System.out.println("d=\n "+Matrix.toStringT(x));
return x;
}

public static double [] AXB(double a[][],double b[])
{
//denklem sistemi çözümü b vektörü için
int n=a.length;
int q[]=new int[n];
double c[][]=gaussLU(a,q);
double s[]=yerine_koyma(c,b,q);
return s;
}

public static double [][] AXB(double a[][],double b[][])
{//denklem sistemi çözümü b matrisi için
int n=a.length;
int q[]=new int[n];
```

```
  double c[][]=gaussLU(a,q);
  double s[][]=yerine_koyma(c,b,q);
  return s;
  }


public static double[] multiply(double[][] left,double[] right)
{
//multiplication of one matrix with one vector
int ii,jj,i,j,k;
int n1=left.length;
int m1=left[0].length;

int m2=right.length;
double[] b;
b=new double[m2];
 for(i=0;i<n1;i++)
   {
     b[i]=0;
     for(k=0;k<m1;k++)
               {b[i]+=left[i][k]*right[k];
               }
   }
return b;
//end of multiply of a matrix and a vector
}

public static double[] substract(double[] left,double[] right)
{
//addition of two vectors
int n1=left.length;
int n2=right.length;
int nMax;
int i;
if(n1>=n2) nMax=n1;
else       nMax=n2;
double b[];
b=new double[nMax];
for(i=0;i<n1;i++)
  {
    b[i]=b[i]+left[i];
  }
for(i=0;i<n2;i++)
  {
    b[i]=b[i]-right[i];
  }
return b;
//end of vector substraction method
}
public static double[] add(double[] left,double[] right)
{
//addition of two vectors
int n1=left.length;
int n2=right.length;
int nMax;
int i;
if(n1>=n2) nMax=n1;
```

```
else       nMax=n2;
double b[];
b=new double[nMax];
for(i=0;i<n1;i++)
  {
    b[i]=b[i]+left[i];
  }
for(i=0;i<n2;i++)
  {
    b[i]=b[i]+right[i];
  }
return b;
//end of vector addition method
}

 public static double error(double q[])
 { double err=0;
   for(int i=0;i<q.length;i++)
   {err+=q[i]*q[i];}
   return Math.sqrt(err);
 }

  public static double[] residual_correction(double a[][],double b[],double eps)
  {
  double eps1;
  System.out.println("a\n"+Matrix.toString(a));
  int n=a.length;
  int q[]=new int[n];
  double c[][]=gaussLU(a,q);
  double x[]=yerine_koyma(c,b,q);
  System.out.println("x\n"+Matrix.toString(x));
  double e[]=new double[x.length];
  double r[]=new double[x.length];
  do
  {
  r= substract(b,multiply(a,x));
  e=yerine_koyma(c,r,q);
  System.out.println("r\n"+Matrix.toString(r));
  System.out.println("e\n"+Matrix.toString(e));
  x=add(x,e);
  System.out.println("x\n"+Matrix.toString(x));
  eps1=error(e);
  } while(eps1>eps);
  return x;
  }

  public static void main(String args[]) throws IOException
  {
  double eps=1e-10;
  double a[][]=Mathd.Hilbert(4);

  double b[]={1,0,0,0};
  double x[]=residual_correction(a,b,eps);
  Text.print(x,"Solution of Gauss(Doolittle) decomposition and elimination with residual correction method");
  }
}
```

```
---------- Capture Output ----------
> "D:\java\bin\javaw.exe" residual
a
      1.000000000000000      0.500000000000000      0.333333333333333      0.250000000000000
      0.500000000000000      0.333333333333333      0.250000000000000      0.200000000000000
      0.333333333333333      0.250000000000000      0.200000000000000      0.166666666666667
      0.250000000000000      0.200000000000000      0.166666666666667      0.142857142857143

x
     15.999999999998870000000000          -119.999999999986880000000000          239.99999999968000000000000
139.99999999979020000000000

r
      0.000000000000000000000000          -0.000000000000035527136788          -0.000000000000035527136788
0.000000000000000000000000

e
     -0.000000000004263256414560          0.000000000053290705181999          -0.000000000134292577058639
0.000000000089528384705759

x
     15.999999999984440000000000          -119.999999999815500000000000          239.99999999954580000000000          -
139.99999999970070000000000

> Terminated with exit code 0.
```

3.13 NORMS

Norms are an important tool to evaluate matrices: Several form of norms are defined. General definitions of norms are as follows:

Infinity norm : Maximum value of the sum of the columns

$$\|A\|_\infty = \sum_{i=1}^{n} max\left(\sum_{j=1}^{n}|a_{ij}|\right)$$

Norm: Maximum value of the sum of the raws

$$\|A\| = \sum_{j=1}^{n} max\left(\sum_{i=1}^{n}|a_{ij}|\right)$$

Euclidian norm: square root of the sum of the square of the elements

$$\|A\|_e = \|A\|_2 = \sqrt{\sum_{i=1}^{n}\sum_{j=1}^{n} a_{ij}^2}$$

m norm: mth root of the sum of the mth power of the elements

$$\|A\|_m = \sqrt[m]{\sum_{i=1}^{n}\sum_{j=1}^{n} a_{ij}^m}$$

Of course the same norm definitions can be apply to the vectors as well, please be reminded that vectors are one dimensional matrices.

$$\|x\|_\infty = \sum_{i=1}^{n} max(|x_i|)$$

$$\|x\| = \sum_{i=1}^{n} max(|x_i|)$$

$$\|x\|_e = \|x\|_2 = \sqrt{\sum_{i=1}^{n} x_i^2}$$

$$\|x\|_m = \sqrt[m]{\sum_{i=1}^{n} x_i^m}$$

Basic norm definitions are given in the norm class.

3.13-1 Matrix norm definitions : class norm

```
public class norm
{
//*************norm methods definition*****************
public static double norm(double v[])
{
// vector norm
double total=0;;
for(int i=0;i<v.length;i++)
  {total+=v[i]*v[i];}
return Math.sqrt(total);
}

public static double norm(double v[],int p)
  {
   //p vector norm
  if(p==0) p=1;
  double total=0;
  double x,y;
  for(int i=0;i<v.length;i++)
    {
   x=Math.abs(v[i]);
   total+=Math.pow(x,p);
    }
  return Math.pow(total,(1.0/p));
  }

public static double norminf(double v[])
    {
     //infinite vector norm
    double x;
    double max=0;
    for(int i=0;i<v.length;i++)
      {
      x=Math.abs(v[i]);
      if(x>max) max=x;
      }
    return max;
    }

public static double[] vek(double mat[][],int n)
{ double a[]=new double[mat.length];
  for(int i=0;i<mat.length;i++) {a[i]=mat[i][n];}
  return a;
}

public static double norminf(double a[][])
       {
         //infinite matrix norm
         double x;
         double max = 0;
         double total;
```

```
        int i,j;
        int n=a.length;
        for(i=0;i<n;i++)
        {
        total=0;
        for(j=0;j<n;j++)
        {total+=Math.abs(a[i][j]);}
        x=total;
        if(x>max) max=x;
        }
       return max;
       }

public static double norm(double a[][])
      {
      //matrix norm
              double x;
              double max = 0;
              double total;
              int i,j;
              int n=a.length;
              for(j=0;j<n;i++)
                {
                total=0;
                for(i=0;i<n;j++)
                {total+=Math.abs(a[i][j]);}
                x=total;
                if(x>max) max=x;
                }
              return max;
     }

public static double normE(double a[][])
            {
            //Euclidian matrix norm
            double x;
            double total;
            int i,j;
            total=0;
            int n=a.length;
            for(j=0;j<n;i++)
            {for(i=0;i<n;j++) {total+=a[i][j]*a[i][j];}}
            return Math.sqrt(total);
           }
}
```

3.14 UNIT MATRIX, TRANPOSE MATRIX, DETERMINANT AND MATRIX INVERSION

Unit matrix is a matrix with all diagonal elements are one and all the remaining elements are zero

$a_{ii} = 1 a_{ij, i \neq j} = 0$

As an example for n=4

$$[I] = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Transpose matrix is the matrix with rows and colums are interchanged. Mathematically, it can be shown with a T sign at the top of the matrix. Transpose of a matrix can be taken as in the following method:

```
public static double[][] Transpose(double [][] left)
{
//transpose matrix (if A=a(i,j) Transpose(A)=a(j,i)
int i,j;
int n=left.length;
```

```
int m=left[0].length;
double b[][];
b=new double[m][n];
for(i=0;i<n;i++)
 {
 for(j=0;j<m;j++)
  {
  b[j][i]=left[i][j];
  }
 }
return b;
}
```

Inverse matrix is the matrix when multiplied with the original matrix will give the unit matrix

$$[A][A]^{-1} = [I]$$

Determinant of a matrix can be defined as follows: if a matrix A is given as

$$[A] = \begin{bmatrix} a_{00} & a_{01} & a_{02} \\ a_{10} & a_{11} & a_{12} \\ a_{20} & a_{21} & a_{22} \end{bmatrix}$$

Determinat of this matrix determined as:

$$D[A] = \begin{vmatrix} a_{00} & a_{01} & a_{02} \\ a_{10} & a_{11} & a_{12} \\ a_{20} & a_{21} & a_{22} \end{vmatrix} = a_{00} \begin{vmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{vmatrix} - a_{01} \begin{vmatrix} a_{10} & a_{12} \\ a_{20} & a_{22} \end{vmatrix} + a_{02} \begin{vmatrix} 0 & a_{11} \\ a_{20} & a_{21} \end{vmatrix}$$

$$D[A] = a_{00}(a_{11}a_{22} - a_{12}a_{21}) - a_{01}(a_{10}a_{22} - a_{12}a_{20}) + a_{02}(a_{10}a_{22} - a_{12}a_{20})$$

If matrix is converted to a upper triangular form Determinant will simply be the product of diagonals

$$[A] = \begin{bmatrix} a_{00} & a_{01} & a_{02} \\ 0 & a_{11} & a_{12} \\ 0 & 0 & a_{22} \end{bmatrix}$$

$$D[A] = a_{00}a_{11}a_{11}$$

Inverse matrix can be obtained by using any of the previous system of equation solution methods.

Problem 3.14-1 Inverse matrix program with partially pivoted Gauss LU method

```
import java.io.*;
import java.text.*;
import java.util.Locale;

class NA28
{
 public static void pivot(double a[][],int o[],double s[],int k)
 {// GaussLU  pivoting process
  int n=a.length;
  int dummy1=0;
  double buyuk;
  double dummy=0;
  //pivotlama
            int p=k;
            buyuk=Math.abs(a[o[k]][k]);
            for(int ii=k+1;ii<n;ii++)
            {  dummy=Math.abs(a[o[ii]][k]/s[o[ii]]);
              if(dummy > buyuk) {buyuk=dummy;p=ii;}
            }
            dummy1=o[p];
            o[p]=o[k];
            o[k]=dummy1;
```

```
}

public static int[] pivot_index(int n)
{   // indexed the pivoted matrix
                int o[]=new int[n];
                for(int i=0;i<n;i++) {o[i]=i;}
                return o;
}

public static double[][] gaussLU(double ai[][],int o[])
{ //Gauss LU with pivoting
int n=ai.length;
double a[][]=new double[n][n];
double s[]=new double[n];
double carpan=0;
double sum=0;
double absaij;

//gauss elimination
int i,j,k,p,ii,jj;
for(i=0;i<n;i++)
{for(j=0;j<n;j++){a[i][j]=ai[i][j];}}
for(i=0;i<n;i++)
{ o[i]=i;s[i]=Math.abs(a[i][0]);
  for(j=1;j<n;j++)
    { absaij=Math.abs(a[i][j]);
      if(absaij>s[i]) {s[i]=absaij;}
    }
}
for(k=0;k<(n-1);k++)
{ pivot(a,o,s,k);
                //change pivort reference
                //solve gauss elimination
  for(i=k+1;i<n;i++)
  {  carpan=a[o[i]][k]/a[o[k]][k];
     a[o[i]][k]=carpan;
     for(j=k+1;j<n;j++)
     {   a[o[i]][j]-=carpan*a[o[k]][j]; }
                }
}
return a;
}

public static double[] back_substiution(double ai[][],double bi[])
{   int n=ai.length;
                return back_substiution(ai,bi,pivot_index(n));
}

public static double[][] back_substiution(double ai[][],double bi[][])
{   int n=ai.length;
                return back_substiution(ai,bi,pivot_index(n));
}

public static double[] back_substiution(double ai[][],double bi[],int o[])
{
  int n=ai.length;
  double sum;
  double x[]=new double[n];
  double a[][]=new double[n][n];
  double b[]=new double[n];
  int i,j;
  for(i=0;i<n;i++)
    {for(j=0;j<n;j++){a[i][j]=ai[i][j];};b[i]=bi[i];}
for(i=1;i<n;i++)
```

```
    {sum=b[o[i]];for(j=0;j<i;j++){sum-=a[o[i]][j]*b[o[j]];};b[o[i]]=sum;}
x[n-1]=b[o[n-1]]/a[o[n-1]][n-1];
for(i=(n-2);i>=0;i--)
    {sum=0;
     for(j=i+1;j<n;j++)
      {sum+=a[o[i]][j]*x[j];};
      x[i]=(b[o[i]]-sum)/a[o[i]][i];
    }
return x;
}

public static double[][] back_substiution(double ai[][],double bi[][],int o[])
{
  int n=ai.length;
  int m=bi[0].length;
  double sum;
  double x[][]=new double[n][m];
  double a[][]=new double[n][n];
  double b[][]=new double[n][m];
  int i,j,k;
for(i=0;i<n;i++)
    {for(j=0;j<n;j++){a[i][j]=ai[i][j];};for(k=0;k<m;k++) {b[i][k]=bi[i][k];}}
for(k=0;k<m;k++)
{
for(i=1;i<n;i++)
    {sum=b[o[i]][k];for(j=0;j<i;j++){sum-=a[o[i]][j]*b[o[j]][k];};b[o[i]][k]=sum;}
x[n-1][k]=b[o[n-1]][k]/a[o[n-1]][n-1];
for(i=(n-2);i>=0;i--)
    {sum=0;
     for(j=i+1;j<n;j++)
      {sum+=a[o[i]][j]*x[j][k];};
      x[i][k]=(b[o[i]][k]-sum)/a[o[i]][i];
    }
}
return x;
}

public static double [] AXB(double a[][],double b[])
{
//solution of the system of equation
int n=a.length;
int q[]=new int[n];
double c[][]=gaussLU(a,q);
double s[]=back_substiution(c,b,q);
return s;
}

public static double [][] AXB(double a[][],double b[][])
{//solution of the set of the system  of equations
int n=a.length;
int q[]=new int[n];
double c[][]=gaussLU(a,q);
double s[][]=back_substiution(c,b,q);
return s;
}

public static double[][] I(int n)
{
//Inverse matrix
double b[][];
b=new double[n][n];
for(int i=0;i<n;i++)
  b[i][i]=1.0;
return b;
```

```
  }

  public static double[][] inverse(double a[][])
  {
  //Inverse matrix
  int n=a.length;
  int q[]=new int[n];
  double c[][]=gaussLU(a,q);
  double s[][]=back_substiution(c,I(n),q);
  return s;
  }

  // Matrix output classes
  public static String toString(double[][] left)
  {
  //return a string representation of a matrix
  return toString(left,25,15);
  }
  public static String toString(double[][] left,int w,int d)
  {
  //return a string representation of a matrix
  int n,m;
  String b;
  b="";
  n=left.length;
  m=left[0].length;
  for(int i=0;i<n;i++)
  {
  for(int j=0;j<m;j++)
    {
    b=b+toString(left[i][j],w,d);
    }
    b=b+"\n";
  }
  return b;
  }

  public static String toString(double left, int w, int d)
  // converts a double to a string with given width and decimals.
  {
              Locale us=new Locale("US");
              String s="%"+w+"."+d+"f";
              String s1=String.format(us,s,left);
   return s1;
  }

  public static void main(String args[]) throws IOException
  {
  double a[][]={{3.556,-1.778,0},{-1.778,3.556,-1.778},{0,-1.778,3.556}};
  System.out.println("Matrix a : \n"+toString(a));
  System.out.println("Inverse Matrix of a : \n"+toString(inverse(a)));
  }
}
```

Inverse matrix class NA28 output

```
---------- Capture Output ----------
> "C:\turhan\java\bin\javaw.exe" NA28
Matrix a :
       3.556000000000000      -1.778000000000000       0.000000000000000
      -1.778000000000000       3.556000000000000      -1.778000000000000
       0.000000000000000      -1.778000000000000       3.556000000000000

Inverse Matrix of a :
       0.421822272215973       0.281214848143982       0.140607424071991
```

```
      0.281214848143982      0.562429696287964      0.281214848143982
      0.140607424071991      0.281214848143982      0.421822272215973


> Terminated with exit code 0.
```

Problem 3.14-2 Inverse matrix program with partially pivoted Gauss elimination method in Python

```
# Gauss Elimination with pivoting
# python version

from math import *

def gauss(a,b):
  n=len(a)
  x=[0 for i in range(n)]
  for k in range(n):
    p=k
    buyuk=abs(a[k][k])
    for ii in range(k+1,n):
      dummy=abs(a[ii][k])
      if dummy>buyuk:
        buyuk=dummy
        p=ii
    if p!=k:
      for jj in range(k,n):
        dummy=a[p][jj]
        a[p][jj]=a[k][jj]
        a[k][jj]=dummy
      dummy=b[p]
      b[p]=b[k]
      b[k]=dummy
    for i in range(k+1,n):
      carpan=a[i][k]/a[k][k]
      a[i][k]=0;
      for j in range(k+1,n):
        a[i][j]=a[i][j]-carpan*a[k][j]
      b[i]   =b[i]-carpan*b[k]
  x[n-1]=b[n-1]/a[n-1][n-1]
  for i in range(n-1,-1,-1):
    toplam=0;
    for j in range(i+1,n):
      toplam=toplam+a[i][j]*x[j]
    x[i]=(b[i]-toplam)/a[i][i];
  return x;

def I(n):
    #unit matrix of dimension n
    B = [[0 for x in range(n)] for y in range(n)];
    for i in range(0,n):
        for j in range(0,n):
            if i==j: B[i][j]=1.0;
    return B;

def invA(A):
    #inverse matrix (n=m)
    n=len(A);
    Im= I(n);
    B = [[0 for x in range(n)] for y in range(n)];
    for i in range(n):
        B[i]=gauss(A,Im[i]);
    return B;

A=[[3.556,-1.778,0],[-1.778,3.556,-1.778],[0,-1.778,3.556]];
print("matrix = \n",A);
```

```
AI=invA(A);
print("inverse matrix = \n",AI);
```

```
runfile('D:/okul/SCO1/gauss.py', wdir='D:/okul/SCO1')
matrix =
 [[3.556, -1.778, 0], [-1.778, 3.556, -1.778], [0, -1.778, 3.556]]
inverse matrix =
 [[0.421822272215973, 0.281214848143982, 0.140607424071991], [0.18747656542932134, 0.3749531308586427, 0.0],
[0.140607424071991, 0.281214848143982, 0.421822272215973]]
```

If inverse matrix is determined solution of x unknown vector of a system of equation can be solved by multiplication of inverse matrix and b.

$$[A]\{x\} = \{b\}$$
$$\{x\} = [A]^{-1}\{b\}$$

Let us give an example of this method by an excell spreadsheet.

$$\begin{bmatrix} -8 & 1 & 2 \\ 5 & 7 & -3 \\ 2 & 1 & -2 \end{bmatrix} \begin{bmatrix} X_0 \\ X_1 \\ X_2 \end{bmatrix} = \begin{bmatrix} 0 \\ 10 \\ -2 \end{bmatrix}$$

	B	C	D	E	F
1		A			
2	-8	1	2	x0	0
3	5	7	-3	x1	10
4	2	1	-2	x2	-2
5		A'			
6	-0.14865	0.054054	-0.22973		1
7	0.054054	0.162162	-0.18919	x=	2
8	-0.12162	0.135135	-0.82432		3

In order to solve these system of equation an inverse matrix function **MINVERSE**(If you are using a turkish version function name: **DIZEY_TERS**) and matrix multiplication function **MMULTIPLY**((If you are using a turkish version function name: **DÇARP**) can be used. While using these functions instead of enter key, Ctrl-shift-enter key (ctrl ⇑ ↵) should be used. The cells that inverse matrix will be created is selected first and then MINVERSE(A2:D4) is entered (a2:D is the original space matrix A is located), and then ctrl-shift-enter keys are pushed. Inverse matrix appears at the selected cell section. And than by using **MMULTIPLY(B6:D8;F2:F4)** , where B7:D9 is the space of the inverse matrix and F2:F4 is the space of vector b. We have to use ctrl-shift-enter again with this function to get the result.

3.17 CRAMER'S RULE

Another method to solve linear system of equations is Cramer's rule. The rule based on solutions of determinants. This solution method is not preferred much as a computer solution method, but it is practical for hand solutions of small systems. If a system of equation given as:

$$\begin{bmatrix} a_{11} & a_{12} & \dots & a_{1n} \\ a_{21} & a_{22} & \dots & a_{2n} \\ \dots & \dots & \dots & \dots \\ a_{n1} & a_{n2} & \dots & a_{nn} \end{bmatrix} \begin{Bmatrix} x_1 \\ x_2 \\ \dots \\ x_n \end{Bmatrix} = \begin{Bmatrix} b_1 \\ b_2 \\ \dots \\ b_n \end{Bmatrix}$$

then solution can be given as:

$$D = \begin{vmatrix} a_{11} & a_{12} & \dots & a_{1n} \\ a_{21} & a_{22} & \dots & a_{2n} \\ \dots & \dots & \dots & \dots \\ a_{n1} & a_{n2} & \dots & a_{nn} \end{vmatrix} \quad x_1 = \begin{vmatrix} b_1 & a_{12} & \dots & a_{1n} \\ b_2 & a_{22} & \dots & a_{2n} \\ \dots & \dots & \dots & \dots \\ b_n & a_{n2} & \dots & a_{nn} \end{vmatrix} \frac{1}{D} \quad x_1 = \begin{vmatrix} a_{11} & b_1 & \dots & a_{1n} \\ a_{21} & b_2 & \dots & a_{2n} \\ \dots & \dots & \dots & \dots \\ a_{n1} & b_n & \dots & a_{nn} \end{vmatrix} \frac{1}{D} \quad \dots.$$

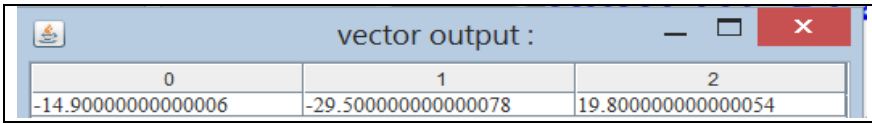
Let us try the method by using an example problem:

Problem 3.17-Cramer' rule for solving system of equations

```
public class cramer
{
public static double[][] insert(double a[][],double b[],int nj)
{
int n=a.length;
int m=a[0].length;
double c[][]=new double[n][m];
for(int i=0;i<n;i++)
{for(int j=0;j<m;j++) {if(j==nj) c[i][j]=b[i];
                 else      c[i][j]=a[i][j];
                }
}
return c;
}
//LU decomposition method
public static double[][] LU(double c[][],int indx[],int d[])
{
//returns LU decomposition of matrix c and index indx
double a[][];
int n=c.length;
a=new double[n][n];
double vv[];
vv=new double[n];
double sum,dum,big,temp;
int i,j,k;
int imax;
int nmax=100;
double tiny=1.0e-40;
imax=0;
for(i=1;i<=n;i++)
{
  for(j=1;j<=n;j++)
     a[i-1][j-1]=c[i-1][j-1];
}
d[0]=1;
for(i=1;i<=n;i++)
  {
  big=0.0;
  for(j=1;j<=n;j++)
    {
    if(Math.abs(a[i-1][j-1])>big) big=Math.abs(a[i-1][j-1]);
    }
  if(big==0) {System.out.println("singular matrix");return a;}
  vv[i-1]=1.0/big;
  }
for(j=1;j<=n;j++)
{
  for(i=1;i<j;i++)
    {
      sum=a[i-1][j-1];
      for(k=1;k<i;k++)
                {
                  sum-=a[i-1][k-1]*a[k-1][j-1];
                }
    a[i-1][j-1]=sum;
    }
  big=0;
  for(i=j;i<=n;i++)
    {
    sum=a[i-1][j-1];
    for(k=1;k<j;k++)
      {
      sum-=a[i-1][k-1]*a[k-1][j-1];
      }
```

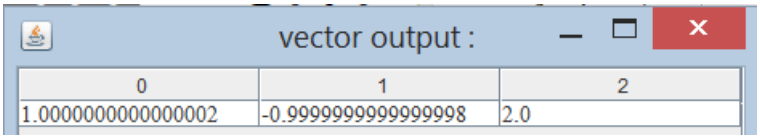
```
    a[i-1][j-1]=sum;
    dum=vv[i-1]*Math.abs(sum);
    if(dum>=big)
      {
      imax=i;
      big=dum;
      }
    } //end of i=0
if(j != imax)
    {
    for(k=1;k<=n;k++)
      {
      dum=a[imax-1][k-1];
      a[imax-1][k-1]=a[j-1][k-1];
      a[j-1][k-1]=dum;
      }
    d[0]=-d[0];
    vv[imax-1]=vv[j-1];
    } //end of if
indx[j-1]=imax;
if(a[j-1][j-1]==0) a[j-1][j-1]=tiny;
if(j!=n)
  {
  dum=1.0/a[j-1][j-1];
  for(i=j+1;i<=n;i++)
    a[i-1][j-1]*=dum;
  }//endif
} //end for j=
return a;
}
public static double D(double a[][])
{
//determinant of a matrix
int n=a.length;
int indx[]=new int[n];
int d[]={1};
double e;
double b[][]=new double[n][n];
b=LU(a,indx,d);
e=d[0];
for(int i=0;i<n;i++)
  e=e*b[i][i];
return e;
}  //end of det

public static double[] cramer(double A[][],double B[])
{double D=D(A);
 int n=B.length;
 double x[]=new double[n];
 double E[][]=A;
 for(int i=0;i<n;i++)
 { E=insert(A,B,i);
   x[i]=D(E)/D;
 }
 return x;
}
public static void main(String arg[])
{double A[][]={{0.3,0.52,1},{0.5,1,1.9},{0.1,0.3,0.5}};
 double B[]={-0.01,0.67,-0.44};
 double x[]=cramer(A,B);
 Text.print(x);
}}
```



Another example

```
public static void main(String arg[])
{double A[][]={{6,-3,1},{2,1,-8},{1,-7,1}};
 double B[]={11,-15,10};
 double x[]=cramer(A,B);
 Text.print(x);
}
```



Even though it is very simple method to solve by hand, it is not recommended as a computer solution method due to heavier calculation load compare to Gauss Elimination and similar methods.

3.16 PROBLEMS

PROBLEM 1

Solve system of equation by using Gauss elimination method.

$$\begin{bmatrix} 7 & 1 & 2 \\ -1 & 4 & -1 \\ 3 & 15 & 20 \end{bmatrix} \begin{Bmatrix} x_0 \\ x_1 \\ x_2 \end{Bmatrix} = \begin{Bmatrix} 47 \\ 19 \\ 87 \end{Bmatrix}$$

PROBLEM 2

Solve system of equation by using

a) By using Gauss elimination
b) By using Gauss-Jordan elimination

$$\begin{bmatrix} 1.1348 & 3.8226 & 1.1651 & 3.4017 \\ 0.5301 & 1.7875 & 2.5330 & 1.5435 \\ 3.4129 & 4.9317 & 8.7643 & 1.3142 \\ 1.2371 & 4.9998 & 10.6721 & 0.0147 \end{bmatrix} \begin{Bmatrix} x_0 \\ x_1 \\ x_2 \\ x_3 \end{Bmatrix} = \begin{Bmatrix} 9.5342 \\ 6.3941 \\ 18.4231 \\ 16.9237 \end{Bmatrix}$$

Exact solution: $\begin{Bmatrix} x_0 \\ x_1 \\ x_2 \\ x_3 \end{Bmatrix} = \begin{Bmatrix} 1 \\ 1 \\ 1 \\ 1 \end{Bmatrix}$

PROBLEM 3

Solve system of equation by using

a) By using Gauss elimination
b) By using Gauss-Jordan elimination

$$\begin{bmatrix} 4 & 3 & -1 \\ 7 & -2 & 3 \\ 5 & -18 & 13 \end{bmatrix} \begin{Bmatrix} x_0 \\ x_1 \\ x_2 \end{Bmatrix} = \begin{Bmatrix} 6 \\ 9 \\ 3 \end{Bmatrix}$$

PROBLEM 4

Solve system of equation by using

a) By using Gauss elimination
b) By using Gauss-Jordan elimination
c) LU decomposition (Dolittle)
d) LU Craut Decomposition

$$\begin{bmatrix} 1 & 1/2 & 1/3 & 1/4 & 1/5 \\ 1/2 & 1/3 & 1/4 & 1/5 & 1/6 \\ 1/3 & 1/4 & 1/5 & 1/6 & 1/7 \\ 1/4 & 1/4 & 1/6 & 1/7 & 1/8 \\ 1/5 & 1/6 & 1/7 & 1/8 & 1/9 \end{bmatrix} \begin{bmatrix} X_0 \\ X_1 \\ X_2 \\ X_3 \\ X_4 \end{bmatrix} = \begin{bmatrix} 137/60 \\ 87/60 \\ 459/60 \\ 743/840 \\ 1879/2520 \end{bmatrix}$$

PROBLEM 5

Solve system of equation by using

a) By using Gauss elimination
b) Gauss-Seidel iteration
c) Conjugate gradient method

$$\begin{bmatrix} 8 & 2 & 3 \\ 3 & -9 & 2 \\ 2 & 3 & 6 \end{bmatrix} \begin{Bmatrix} x_0 \\ x_1 \\ x_2 \end{Bmatrix} = \begin{Bmatrix} 30 \\ 1 \\ 31 \end{Bmatrix}$$

PROBLEM 6 Solve system of equation by using

a) By using Gauss elimination
b) Gauss-Seidel with relaxation
c) Conjugate gradient method

$$\begin{bmatrix} 3 & -5 & 47 & 20 \\ 11 & 16 & 17 & 10 \\ 56 & 22 & 11 & -18 \\ 17 & 66 & -12 & 7 \end{bmatrix} \begin{Bmatrix} x_0 \\ x_1 \\ x_2 \\ x_3 \end{Bmatrix} = \begin{Bmatrix} 18 \\ 26 \\ 34 \\ 82 \end{Bmatrix}$$

PROBLEM 7 Solve system of equation by using Thomas algorithm (3 band matrix method)

$$\begin{bmatrix} -2 & 1 & & & & & & & \\ 1 & -2 & 1 & & & & & & \\ & 1 & -2 & 1 & & & & & \\ & & & \cdots & & & & & \\ & & & 1 & -2 & 1 & & & \\ & & & & 1 & -2 & 1 & & \\ & & & & & 1 & -2 & & \end{bmatrix} \begin{bmatrix} X_0 \\ X_1 \\ X_2 \\ \\ X_7 \\ X_8 \\ X_9 \end{bmatrix} = \begin{bmatrix} -0.5 \\ -1.5 \\ -1.5 \\ \cdots \\ -1.5 \\ -1.5 \\ 0.5 \end{bmatrix}$$

PROBLEM 8 Solve system of equation by using Thomass algorithm (3 band matrix method)

$$\begin{bmatrix} -4 & 1 & & & & & & & \\ 1 & -4 & 1 & & & & & & \\ & 1 & -4 & 1 & & & & & \\ & & & \cdots & & & & & \\ & & & 1 & -4 & 1 & & & \\ & & & & 1 & -4 & 1 & & \\ & & & & & 1 & -4 & & \end{bmatrix} \begin{bmatrix} X_0 \\ X_1 \\ X_2 \\ \\ X_7 \\ X_8 \\ X_9 \end{bmatrix} = \begin{bmatrix} -27 \\ -15 \\ -15 \\ \cdots \\ -15 \\ -15 \\ -15 \end{bmatrix}$$

PROBLEM 9

Solve the following system of equation

$$\begin{bmatrix} 3 & -5 & 6 & 4 & -2 & -3 & 8 \\ 1 & 1 & -9 & 15 & 1 & -9 & 2 \\ 2 & -1 & 7 & 5 & -1 & 6 & 11 \\ -1 & 1 & 3 & 2 & 7 & -1 & -2 \\ 4 & 3 & 1 & -7 & 2 & 1 & 1 \\ 2 & 9 & -8 & 11 & -1 & -4 & -1 \\ 7 & 2 & -1 & 2 & 7 & -1 & 9 \end{bmatrix} \begin{bmatrix} X_0 \\ X_1 \\ X_2 \\ X_3 \\ X_4 \\ X_5 \\ X_6 \end{bmatrix} = \begin{bmatrix} 47 \\ 17 \\ 24 \\ 8 \\ 13 \\ -10 \\ 34 \end{bmatrix}$$

PROBLEM 10

Solve the following system of equation

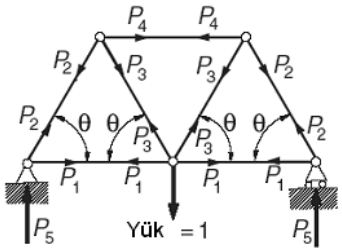
$$\begin{bmatrix} 1 & -1 & 2 & 5 & -7 & -8 \\ 3 & -9 & 1 & -1 & 8 & 1 \\ -1 & 1 & 9 & -9 & 2 & 3 \\ 1 & 7 & 2 & -3 & -1 & 4 \\ 7 & 1 & -2 & 4 & 1 & -1 \\ 2 & 3 & -9 & 12 & -2 & 7 \end{bmatrix} \begin{bmatrix} X_0 \\ X_1 \\ X_2 \\ X_3 \\ X_4 \\ X_5 \end{bmatrix} = \begin{bmatrix} -12 \\ 8 \\ 22 \\ 41 \\ 15 \\ 50 \end{bmatrix}$$

PROBLEM 11

Solve the following system of equation

$$\begin{bmatrix} 10 & -2 & -1 & 2 & 3 & 1 & -4 & 7 \\ 5 & 11 & 3 & 10 & -3 & 3 & 3 & -4 \\ 7 & 12 & 1 & 5 & 3 & -12 & 2 & 3 \\ 8 & 7 & -2 & 1 & 3 & 2 & 2 & 4 \\ 2 & -15 & -1 & 1 & 4 & -1 & 8 & 3 \\ 4 & 2 & 9 & 1 & 12 & -1 & 4 & 1 \\ -1 & 4 & -7 & -1 & 1 & 1 & -1 & -3 \\ -1 & 3 & 4 & 1 & 3 & -4 & 7 & 6 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \\ x_6 \\ x_7 \\ x_8 \end{bmatrix} = \begin{bmatrix} 0 \\ 12 \\ -5 \\ 3 \\ -25 \\ -26 \\ 9 \\ -7 \end{bmatrix}$$

PROBLEM 12

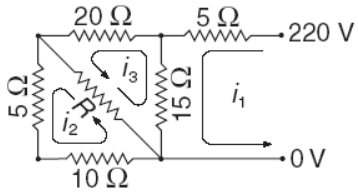


Load Matrix of the figure can be given with the following system of eqautions

$$\begin{bmatrix} c & 1 & 0 & 0 & 0 \\ 0 & s & 0 & 0 & 1 \\ 0 & 0 & 2s & 0 & 0 \\ 0 & -c & c & 1 & 0 \\ 0 & s & s & 0 & 0 \end{bmatrix} \begin{bmatrix} P_1 \\ P_2 \\ P_3 \\ P_4 \\ P_5 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 1 \\ 0 \\ 0 \end{bmatrix}$$

Where c=cos(θ),s=sin(θ) . For θ =60º find the load distribution profile

PROBLEM 13



For the given electrical circuit current density-Voltage eqautions can be given as:

$$5i_1 + 15(i_1 - i_3) = 220\,V$$
$$5i_1 + 10i_2 + R(i_2 - i_3) = 0$$
$$20i_3 + R(i_3 - i_2) + 15(i_3 - i_1) = 0$$

Calculate current densities for R=10 Ω.

PROBLEM 14

$-x_1 + 3x_2 + 5x_3 + x_4 = 8$
$x_1 + 9x_2 + 3x_3 + 4x_4 = 10$
$x_2 + x_4 = 2$
$3x_1 + x_2 + x_3 - x_4 = -4$

Solve the given system of equations. Determine the best method to solve it.

PROBLEM 15 Solve the following system of equation by using Doolittle LU method

$$\begin{bmatrix} 1 & 3 & 5 & 7 \\ 11 & 16 & 17 & 10 \\ 0 & 0 & 2 & 5 \\ -2 & -6 & -3 & 1 \end{bmatrix} \begin{Bmatrix} x_0 \\ x_1 \\ x_2 \\ x_3 \end{Bmatrix} = \begin{Bmatrix} 1 \\ 2 \\ 3 \\ 4 \end{Bmatrix}$$

PROBLEM 16 Solve system of equation by using

a) By using Gauss elimination

b) By using Gauss-Jordan elimination
c) LU decomposition (Dolittle)
d) LU Craut Decomposition

$3x_1 + 5x_2 + 4x_3 + 2x_4 = 11$
$6x_1 + 14x_2 + 11x_3 + 6x_4 = 26$
$9x_1 + 11x_2 + 16x_3 + 5x_4 = 68$
$3x_1 + 13x_2 + 17x_3 + 12x_4 = 43$

PROBLEM 17 Solve system of equation by using

a) By using Gauss elimination
b) By using Gauss-Jordan elimination
c) LU decomposition (Dolittle)
d) LU Craut Decomposition
e) Gauss Seidel iteration
f) Conjugate-Gradian method

$$x_1 + 3x_2 + x_3 + 5x_4 = 4$$
$$2x_1 + x_2 + 3x_4 = 5$$
$$4x_1 + 2x_2 + 2x_3 + x_4 = 11$$
$$-3x_1 + x_2 + 3x_3 + 2x_4 = 3$$

PROBLEM 18 Solve system of equation by using

a) By using Gauss elimination
b) By using Gauss-Jordan elimination
c) LU decomposition (Dolittle)
d) LU Craut Decomposition

$x_1 + x_2 - x_3 = -3$
$6x_1 + 2x_2 + 2x_3 = 2$
$-3x_1 + 4x_2 + x_3 = 1$

PROBLEM 19 Find infinite norm two norm and m=3 norm of the matrix

$$\begin{bmatrix} 7 & 1 & 2 \\ -1 & 4 & -1 \\ 3 & 15 & 20 \end{bmatrix}$$

PROBLEM 20 Find the inverse matrix

$$\begin{bmatrix} 1.1348 & 3.8326 & 1.1651 & 3.4017 \\ 0.5301 & 1.7875 & 2.5330 & 1.5435 \\ 3.4129 & 4.9317 & 8.7643 & 1.3142 \\ 1.2371 & 4.9998 & 10.6721 & 0.0147 \end{bmatrix}$$

PROBLEM 21 Find the inverse matrix

$$\begin{bmatrix} 4 & 3 & -1 \\ 7 & -2 & 3 \\ 5 & -18 & 13 \end{bmatrix}$$

PROBLEM 22

Find two norm and inverse matrix

$$\begin{bmatrix} 1 & 1/2 & 1/3 & 1/4 & 1/5 \\ 1/2 & 1/3 & 1/4 & 1/5 & 1/6 \\ 1/3 & 1/4 & 1/5 & 1/6 & 1/7 \\ 1/4 & 1/4 & 1/6 & 1/7 & 1/8 \\ 1/5 & 1/6 & 1/7 & 1/8 & 1/9 \end{bmatrix}$$

PROBLEM 23

Find two norm, infinite norm and and inverse matrix

$$\begin{bmatrix} 8 & 2 & 3 \\ 1 & -9 & 2 \\ 2 & 3 & 6 \end{bmatrix}$$

PROBLEM 24

Find two norm, infinite norm and and inverse matrix

$$\begin{bmatrix} 3 & -5 & 47 & 20 \\ 11 & 16 & 17 & 10 \\ 56 & 22 & 11 & -18 \\ 17 & 66 & -12 & 7 \end{bmatrix}$$

PROBLEM 25

Find two norm, infinite norm and and inverse matrix

$$\begin{bmatrix} -2 & 1 & & & & & & \\ 1 & -2 & 1 & & & & & \\ & 1 & -2 & 1 & & & & \\ & & & \cdots & & & & \\ & & & 1 & -2 & 1 & & \\ & & & & 1 & -2 & 1 & \\ & & & & & 1 & -2 \end{bmatrix}$$

4 NONLINEAR SYSTEM OF EQUATIONS (ROOTS OF SYSTEM OF EQUATIONS)

4.1 NEWTON-RAPHSON METHOD

One dimensional Newton-Raphson Formula was:

$$x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)} \qquad (4.1.1)$$

If the equation format changed a little, it can be written in the form of

$$\Delta x_n = x_{n+1} - x_n = -\frac{f(x_n)}{f'(x_n)} \qquad (4.1.2)$$

$$f'(x_n)\Delta x_n = -f(x_n) \qquad (4.1.3)$$

Now the same equation can be considered for a system of non-linear equation

$$[\nabla f] = \begin{bmatrix} \frac{\partial f_1}{\partial x_1} & \frac{\partial f_1}{\partial x_2} & \cdots & \frac{\partial f_1}{\partial x_n} \\ \frac{\partial f_2}{\partial x_1} & \frac{\partial f_2}{\partial x_2} & \cdots & \frac{\partial f_2}{\partial x_n} \\ \cdots & \cdots & \cdots & \cdots \\ \frac{\partial f_n}{\partial x_1} & \frac{\partial f_n}{\partial x_2} & \cdots & \frac{\partial f_n}{\partial x_n} \end{bmatrix} \quad (4.1.4) \qquad \{\Delta x\} = \begin{Bmatrix} \Delta x_1 \\ \Delta x_2 \\ \cdots \\ \Delta x_n \end{Bmatrix} \quad \{f\} = \begin{Bmatrix} f_1 \\ f_2 \\ \cdots \\ f_n \end{Bmatrix} \quad (4.1.5)$$

So multidimensional Newton-Raphson equation becomes

$$[\nabla f]\{\Delta x\} = -\{f\} \qquad (4.1.6)$$

The equation can be solved by using tecqniques such as gauss elimination. An initial estimate for all the x values are required to start iterative solution

First example function

$f_0(x_0,x_1)=x_0^2+ x_0\, x_1-10$

$f_1(x_0,x_1)=x_1+3\, x_0\, x_1^2-57$

$$[\nabla f] = \begin{bmatrix} 2x_0 + x_1 & x_0 \\ 3x_1^2 & 1 + 6x_0x_1 \end{bmatrix}$$

PROGRAM 4.1-1 NA29 : Newton Raphson method both function and derivatives are defined

```
// NA29 Newton Raphson method
// to solve non-linear system of equation

import java.util.*;
import java.awt.*;
import java.applet.Applet;
import java.awt.event.*;
import javax.swing.*;
//============= Tanimi gereken fonksiyon ================
abstract class fi_xi
{
  // multifunction multi independent variable
  // vector of dependent variables are returned
  // example f[0]=x[0]+sin(x[1])
  //         f[1]=x[0]*x[0]-x[1]
  // func(x) returns the value of f[0] and f[1]
  // as a two dimensional vector

  abstract double[] func(double x[]);
}

abstract class fij_xi
{
  // multifunction multi independent variable
  // for n independent variable n*n matrix of
  // dependent variables are returned
  // example f[0][0]=x[0]+Math.sin(x[1])   f[0][1]=x[0]-x[1]
  //         f[1][0]=x[0]*x[0]-x[1]       f[1][1]=Math.exp(x[0]+x[1]*x[1]
```

```
  // func(x) returns the value of f[0][0], f[0][1]
  //                              f[1][0], f[1][1]
  // as a two dimensional matrix
  abstract double[][] func(double x[]);
}

class fa extends fi_xi
{
public double[] func(double x[])
{
//system of non-linear equation
double ff[]=new double[2];
ff[0]=x[0]*x[0]+x[0]*x[1]-10.0;
ff[1]=x[1]+3.0*x[0]*x[1]*x[1]-57.0;
return ff;
}
}

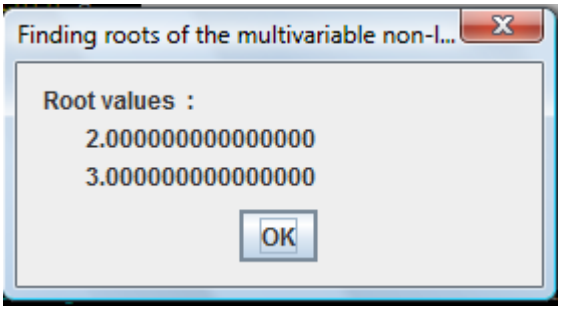
class fb extends fij_xi
{
//derivative of system of non-linear equation
public double[][] func(double x[])
{
double a[][]=new double[2][2];
a[0][0]=2.0*x[0]+x[1];
a[0][1]=x[0];
a[1][0]=3.0*x[1]*x[1];
a[1][1]=1.0+6.0*x[0]*x[1];
return a;
} }

public class NA29
{
// Roots of system of non-linear equations
// By using Newton-Raphson Method
public static double[] newton(fi_xi f,fij_xi df,double x[])
{
//x vector of independent variables
//y vector of dependent variable
//dy derivative of vector of dependent variable
double ti=1.0;
int i;
int nmax=400;
double tolerance=1.0e-20;
int n=x.length;
double b[];
b=new double[n];
for(i=0;i<n;i++)
{
b[i]=1.0;
}
i=0;
while( i++ < nmax && Matrix.abs(b) > tolerance )
{ b=Matrix.multiply(Matrix.divide(f.func(x),df.func(x)),-1.0);
  x=Matrix.add(x,b);
}
if(i >= nmax) JOptionPane.showMessageDialog(null,"Maximum number of iterations are exceeded \n"+
```

```
            " Results might not be valid","MAXIMUM NUMBER OF ITERATIONS
ERROR",JOptionPane.WARNING_MESSAGE);
System.out.println();
return x;
}

public static double[] verigir()
{
   String s=JOptionPane.showInputDialog("ENTER FIRST ROOT ESTIMATES WİTH ONE EMPTY SPPACE IN
BETWEEN: ");
   StringTokenizer token=new StringTokenizer(s);
   int n=token.countTokens()-1;
   int m=n+1;
   double a[]=new double[m];
   int j=0;
   while(token.hasMoreTokens())
   {
   Double ax=new Double(token.nextToken());
   a[j++]=ax.doubleValue();
   }
   return a;
}

    public static void main (String args[])
    {
            fa f =new fa();
    fb df=new fb();
    double [] x0=verigir();
    double [] r= newton(f,df,x0);
    String s=" Root values  : \n"+Matrix.toStringT(r);
    String s1="Finding roots of the multivariable non-linear system of equations : ";
    JOptionPane.showMessageDialog(null,s,s1,JOptionPane.PLAIN_MESSAGE);
    System.exit(0);
    }
}
```



If the same problem is solved by hand starting from the initial guess x={1,2}

Newton-Raphson Non-linear equation solution

Initial guess

x0	1
x1	2

4	1	dx0	7
12	13	dx1	43

3

4	1	dx0	7
0	10	dx1	22

dx0	1.2
dx1	2.2

First iteration result

x0	2.2
x1	4.2

	8.6	2.2	dx0	-4.08
	52.92	56.44	dx1	-63.624
6.153488	8.6	2.2	dx0	-4.08
	0	42.90233	dx1	-38.5178

dx0	-0.24475
dx1	-0.8978

Second iteration results

x0	1.955252
x1	3.302199

	7.212702	1.955252	dx0	-0.27964
	32.71355	39.73977	dx1	-10.2654
4.535547	7.212702	1.955252	dx0	-0.27964
	0	27.70816	dx1	-8.54467

dx0	0.044827
dx1	-0.30838

Third iteration results

x0	2.000079
x1	2.993818

	6.993975	2.000079	dx0	0.011814
	26.88883	36.92723	dx1	0.2264
3.844571	6.993975	2.000079	dx0	0.011814
	0	29.23778	dx1	0.180979

dx0	-8.1E-05
dx1	0.00619

x0	1.999998
x1	3.000008

	7.000003	1.999998	dx0	4.94E-07
	27.00014	37.00005	dx1	-0.00022
3.857161	7.000003	1.999998	dx0	4.94E-07
	0	29.28574	dx1	-0.00022

dx0	2.24E-06
dx1	-7.6E-06

Fourth iteration results

x0	2
x1	3

	7	2	dx0	1.2E-11
	27	37	dx1	-4E-11
3.857143	7	2	dx0	1.2E-11
	0	29.28571	dx1	-8.6E-11

dx0	2.56E-12
dx1	-2.9E-12

Fifth iteration results

x0	2
x1	3

PROGRAM 4.1-2 : Newton Raphson method only function are defined, derivatives are taken numerically

```
// NA30 Newton Raphson method
// to solve non-linear system of equation

import java.util.*;
import java.awt.*;
import java.applet.Applet;
import java.awt.event.*;
import javax.swing.*;
//============= Tanimi gereken fonksiyon ================
abstract class fi_xi
{
  // multifunction multi independent variable
  // vector of dependent variables are returned
  // example f[0]=x[0]+sin(x[1])
  //         f[1]=x[0]*x[0]-x[1]
  // func(x) returns the value of f[0] and f[1]
  // as a two dimensional vector

  abstract double[] func(double x[]);
}

class fa extends fi_xi
{
public double[] func(double x[])
{
//system of non-linear equation
double ff[]=new double[2];
ff[0]=x[0]*x[0]+x[0]*x[1]-10.0;
ff[1]=x[1]+3.0*x[0]*x[1]*x[1]-57.0;
return ff;
}
}

public class NA30
{
// Roots of system of non-linear equations
// By using Newton-Raphson Method

public static double[] newtond( fi_xi f,double x[])
{
//x vector of independent variables
//y vector of dependent variable
//dy derivative of vector of dependent variable
// derivative vector calculated by numerical derivatives
double ti=1.0;
int i,ii,jj;
int nmax=500;
double tolerance=1.0e-15;
int n=x.length;
double b[];
b=new double[n];
double dy[][];
dy=new double[n][n];
i=0;
for(i=0;i<n;i++)
{
b[i]=1.0;
}
while( i++ < nmax && Matrix.abs(b) > tolerance )
{
  for(ii=0;ii<n;ii++)
  {
    for(jj=0;jj<n;jj++)
    {
```

```
        dy[ii][jj]=derivative(f,x,ii,jj);
    }
  }
    b=Matrix.multiply(Matrix.divide(f.func(x),dy),-ti);
    x=Matrix.add(x,b);
}
if(i >= nmax) JOptionPane.showMessageDialog(null,"Maximum number of iterations are exceeded \n"+
            " Results might not be valid","MAXIMUM NUMBER OF ITERATIONS ERROR",JOptionPane.WARNING_MESSAGE);
return x;
}

public static double derivative(fi_xi f,double x[],int denklem_ref,int x_ref)
{
double h0=0.0256;
int i,m;
int n=7;
double f1[];
f1=new double[x.length];
double f2[];
f2=new double[x.length];
double x1[];
x1=new double[x.length];
double x2[];
x2=new double[x.length];
for(i=0;i<x.length;i++)
{
x1[i]=x[i];
x2[i]=x[i];
}
//derivative of a simple function
double T[][];
T=new double[n][n];
double h[];
h=new double[n];
//vector<double> h(n,0);
for(i=0;i<n;i++)
{
  h[i]=0;
  for(int j=0;j<n;j++)
    T[i][j]=0;
}
h[0]=h0;
double r=0.5;
for( i=1;i<n;i++)
{
h[i]=h0*Math.pow(r,i);
}

for(i=0;i<n;i++)
{
x1[x_ref]+=h[i];
x2[x_ref]-=h[i];
f1=f.func(x1);
f2=f.func(x2);
T[i][0]=( f1[denklem_ref] - f2[denklem_ref])/(2.0*h[i]);
x1[x_ref]=x[x_ref];
x2[x_ref]=x[x_ref];
}
for(m=1;m<n;m++)
{
  for(i=0;i<n-m;i++)
  {
  T[i][m]=(h[i]*h[i]*T[i+1][m-1] - h[i+m]*h[i+m]*T[i][m-1])/(h[i]*h[i]
  - h[i+m]*h[i+m]);
```

```
  }
}
double xx=T[0][n-1];
return xx;
}

public static double[] verigir()
{
   String s=JOptionPane.showInputDialog("ENTER FIRST ROOT ESTIMATES WİTH ONE EMPTY SPPACE IN BETWEEN: ");
   StringTokenizer token=new StringTokenizer(s);
   int n=token.countTokens()-1;
   int m=n+1;
   double a[]=new double[m];
   int j=0;
   while(token.hasMoreTokens())
   {
   Double ax=new Double(token.nextToken());
   a[j++]=ax.doubleValue();
   }
   return a;
}

    public static void main (String args[])
    {
            fa f =new fa();
    double [] x0=verigir();
    String s2="Newton raphson method with the numerically calculated derivatives : ";
    double [] r1= newtond(f,x0);
    String s=" root vector  : \n"+Matrix.toStringT(r1);
    JOptionPane.showMessageDialog(null,s,s2,JOptionPane.PLAIN_MESSAGE);
    System.exit(0);
    }
}
```

The problem steps including numerical derivative is evaluated step by step as :

Newton-Raphson Non-linear equation solution

Initial guess						dx0	0.01				
x0	1					x0+dx0	1.01	f1(x0+dx0,x1)	-6.9599	f2(x0+dx0,x1)	-42.88
x1	2					x0+2dx0	1.02	f1(x0+2dx0,x1)	-6.9196	f2(x0+2dx0,x1)	-42.76
						x0-dx0	0.99	f1(x0-dx0,x1)	-7.0399	f2(x0-dx0,x1)	-43.12
	4.0000	1.0000	dx0	7.0000		x0-2dx0	0.98	f1(x0-2dx0,x1)	-7.0796	f2(x0-2dx0,x1)	-43.24
	12.0000	13.0000	dx1	43.0000		x1+dx1	2.01	f1(x0,x1+dx1)	-6.99	f2(x0,x1+dx1)	-42.8697
						x1+2dx1	2.02	f1(x0,x1+2dx1)	-6.98	f2(x0,x1+2dx1)	-42.7388
3	4	1	dx0	7		x1-dx1	1.99	f1(x0,x1-dx1)	-7.01	f2(x0,x1-dx1)	-43.1297
	0	10	dx1	22		x1-2dx1	1.98	f1(x0,x1-2dx1)	-7.02	f2(x0,x1-2dx1)	-43.2588
dx0	1.2										
dx1	2.2										
First iteration result						dx0	0.01				
x0	2.2					x0+dx0	2.21	f1(x0+dx0,x1)	4.1661	f2(x0+dx0,x1)	64.1532
x1	4.2					x0+2dx0	2.22	f1(x0+2dx0,x1)	4.2524	f2(x0+2dx0,x1)	64.6824
						x0-dx0	2.19	f1(x0-dx0,x1)	3.9941	f2(x0-dx0,x1)	63.0948
	8.6000	2.2000	dx0	-4.0800		x0-2dx0	2.18	f1(x0-2dx0,x1)	3.9084	f2(x0-2dx0,x1)	62.5656
	52.9200	56.4400	dx1	-63.6240		x1+dx1	4.21	f1(x0,x1+dx1)	4.102	f2(x0,x1+dx1)	64.18906
						x1+2dx1	4.22	f1(x0,x1+2dx1)	4.124	f2(x0,x1+2dx1)	64.75544
6.153488	8.6000	2.2000	dx0	-4.0800		x1-dx1	4.19	f1(x0,x1-dx1)	4.058	f2(x0,x1-dx1)	63.06026
	0	42.90233	dx1	-38.51777		x1-2dx1	4.18	f1(x0,x1-2dx1)	4.036	f2(x0,x1-2dx1)	62.49784
dx0	-0.244748										
dx1	-0.897801										
Second iteration results						dx0	0.01				
x0	1.955252					x0+dx0	1.9653	f1(x0+dx0,x1)	0.351864	f2(x0+dx0,x1)	10.59255
x1	3.302199					x0+2dx0	1.9753	f1(x0+2dx0,x1)	0.424291	f2(x0+2dx0,x1)	10.91968

					x0-dx0	1.9453	f1(x0-dx0,x1)	0.20761	f2(x0-dx0,x1)	9.938276
	7.2127	1.9553	dx0	-0.279637	x0-2dx0	1.9353	f1(x0-2dx0,x1)	0.135783	f2(x0-2dx0,x1)	9.61114
	32.7135	39.7398	dx1	-10.26541	x1+dx1	3.3122	f1(x0,x1+dx1)	0.29919	f2(x0,x1+dx1)	10.6634
					x1+2dx1	3.3222	f1(x0,x1+2dx1)	0.318742	f2(x0,x1+2dx1)	11.06255
4.535547	7.212702	1.955252	dx0	-0.279637	x1-dx1	3.2922	f1(x0,x1-dx1)	0.260085	f2(x0,x1-dx1)	9.8686
	0	27.70816	dx1	-8.544666	x1-2dx1	3.2822	f1(x0,x1-2dx1)	0.240532	f2(x0,x1-2dx1)	9.472962

dx0	0.044827
dx1	-0.308381

Third iteration results

x0	2.000079
x1	2.993818

					dx0	0.01				
					x0+dx0	2.0101	f1(x0+dx0,x1)	0.058225	f2(x0+dx0,x1)	0.042488
					x0+2dx0	2.0201	f1(x0+2dx0,x1)	0.128465	f2(x0+2dx0,x1)	0.311377
					x0-dx0	1.9901	f1(x0-dx0,x1)	-0.081654	f2(x0-dx0,x1)	-0.495288
	6.9940	2.0001	dx0	0.011814	x0-2dx0	1.9801	f1(x0-2dx0,x1)	-0.151294	f2(x0-2dx0,x1)	-0.764177
	26.8888	36.9272	dx1	0.2264	x1+dx1	3.0038	f1(x0,x1+dx1)	0.008186	f2(x0,x1+dx1)	0.143472
					x1+2dx1	3.0138	f1(x0,x1+2dx1)	0.028187	f2(x0,x1+2dx1)	0.514545
3.844571	6.993975	2.000079	dx0	0.011814	x1-dx1	2.9838	f1(x0,x1-dx1)	-0.031815	f2(x0,x1-dx1)	-0.595072
	0	29.23778	dx1	0.180979	x1-2dx1	2.9738	f1(x0,x1-2dx1)	-0.051816	f2(x0,x1-2dx1)	-0.962544

dx0	-8.09E-05
dx1	0.00619

x0	1.999998
x1	3.000008

Solution given in python language

PROGRAM 4.1-3 : fi_xj abstract class to define function and derivatives of a vector function in python

```
from abc import ABC, abstractmethod
from math import *

class fi_xi(ABC):
  # multifunction multi independent variable
  # vector of dependent variables are returned
  # example f[0]=x[0]+sin(x[1])
  #         f[1]=x[0]*x[1]-x[1]
  # func(x) returns the value of f[0] and f[1]
  # as a two dimensional vector

  @abstractmethod
  def func(self,x):
    pass;

  def xp(self,x,j,m):
     k=len(x);
     h=0.001;
     xx=[0 for i in range(k)];
     for i in range(k):
        if i==m:
           xx[i]=x[i]+j*h;
        else: xx[i]=x[i];
     return xx;

  def dfunc(self,x,n):
     h=0.001;
     k=len(x);
     df=[[0 for i in range(k)] for j in range(k)];
     for i in range(k):
        for j in range(k):
           df[i][j]=self.dfun(x,n,i,j);
     return df;

  def dfun(self,x,n,j,m):
     k=len(x);
     ff=[[0 for i2 in range(k)] for j2 in range(9)];
     xp1=[[0 for i2 in range(k)] for j2 in range(9)];
     h=0.001;
     hh=1/h;
     x1=x[m];
     for jj in range(-4,5):
```

```
        j1=jj+4;
        xp1[j1]=self.xp(x,jj,m);
        ff[j1]=self.func(xp1[j1]);
      if n==0: df=ff[4][j];
      elif n==1:
         df=(3.0*ff[0][j]-32.0*ff[1][j]+168.0*ff[2][j]-672.0*ff[3][j]+672.0*ff[5][j]-168.0*ff[6][j]+32.0*ff[7][j]-3.0*ff[8][j])/840.0*hh;
      elif n==2:
         df=(-14350.0*ff[4][j]-9.0*ff[0][j]+128*ff[1][j]-1008*ff[2][j]+8064*ff[3][j]+8064.0*ff[5][j]-1008.0*ff[6][j]+128.0*ff[7][j]-
9.0*ff[8][j])/5040.0*hh*hh;
      elif n==3:
         df=(-7.0*ff[0][j]+72.0*ff[1][j]-338.0*ff[2][j]+488.0*ff[3][j]-488.0*ff[5][j]+338.0*ff[6][j]-72.0*ff[7][j]+7.0*ff[8][j])/240.0*hh*hh*hh;
      elif n==4:
         df=(2730.0*ff[4][j]+7.0*ff[0][j]-96.0*ff[1][j]+676.0*ff[2][j]-1952.0*ff[3][j]-1952.0*ff[5][j]+676.0*ff[6][j]-
96.0*ff[7][j]+7.0*ff[8][j])/240.0*hh*hh*hh*hh;
      elif n==5:
         df=(ff[0][j]-9.0*ff[1][j]+26.0*ff[2][j]-29.0*ff[3][j]+29.0*ff[5][j]-26.0*ff[6][j]+9.0*ff[7][j]-ff[8][j])/6.0*hh*hh*hh*hh*hh;
      elif n==6:
         df=(-150.0*ff[4][j]-ff[0][j]+12.0*ff[1][j]-52.0*ff[2][j]+116.0*ff[3][j]+116.0*ff[5][j]-52.0*ff[6][j]+12.0*ff[7][j]-
ff[8][j])/4.0*hh*hh*hh*hh*hh*hh;
      elif n==7:
         df=(-ff[0][j]+6.0*ff[1][j]-14.0*ff[2][j]+14.0*ff[3][j]-14.0*ff[5][j]+14.0*ff[6][j]-6.0*ff[7][j]+ff[8][j])/2.0*hh*hh*hh*hh*hh*hh*hh;
      elif n==8:
         df=(70.0*ff[4][j]+ff[0][j]-8.0*ff[1][j]+28.0*ff[2][j]-56.0*ff[3][j]-56.0*ff[5][j]+28.0*ff[6][j]-
8.0*ff[7][j]+ff[8][j])*hh*hh*hh*hh*hh*hh*hh*hh;
      else: df=0;
      return df;
```

PROGRAM 4.1-4 : newtond.py Newton-raphson meyhod to calculate non-linear roots in python

```
from math import *
from fi_xi import *;
from gauss import *;

class f1(fi_xi):
  def func(self,x):
      y=[0 for i in range(2)];
      y[0]=x[0]*x[0]+x[0]*x[1]-10.0;
      y[1]=x[1]+3.0*x[0]*x[1]*x[1]-57.0;
      return y;

def newton(f,x):
   # Newton-Raphson root finding method
   k=len(x);
   nmax=100;
   tolerance=1.0e-10
   for i in range(nmax):
      fx= f.func(x);
      dfx=f.dfunc(x,1);
      dx=gauss(dfx,fx);
      for j in range(k):
         x[j]=x[j]-dx[j];
      total=0;
      for j in range(k):
         total+=fx[j];
      if abs(total)<tolerance: return x
   return x;

f=f1();
x=[0 for i in range(2)];
x[0]=1.0;
x[1]=2.0;
r=newton(f,x)
print("r=",r);
```

```
runfile('D:/okul/SCO1/newtond.py', wdir='D:/okul/SCO1')
r= [2.0, 3.0]
```

4.2 BROYDEN SECANT METHOD WITH SHARMAN-MORRISON FORMULA

We would like to solve f(x)=0 linear system of equation. In these equations x={x1, x1,…, xn-1, xn} is a multidimensional independent variable set. As a second way of solving this system of equations secant method can be used instead of Newton-Raphson method. In Newton-

Raphson approach was defined as :

$$[\nabla f] = \begin{bmatrix} \frac{\partial f_1}{\partial x_1} & \frac{\partial f_1}{\partial x_2} & \cdots & \frac{\partial f_1}{\partial x_n} \\ \frac{\partial f_2}{\partial x_1} & \frac{\partial f_2}{\partial x_2} & \cdots & \frac{\partial f_2}{\partial x_n} \\ \cdots & \cdots & \cdots & \cdots \\ \frac{\partial f_n}{\partial x_1} & \frac{\partial f_n}{\partial x_2} & \cdots & \frac{\partial f_n}{\partial x_n} \end{bmatrix} \quad (4.2.1) \qquad \{\Delta x\} = \begin{Bmatrix} \Delta x_1 \\ \Delta x_2 \\ \cdots \\ \Delta x_n \end{Bmatrix} \quad \{f\} = \begin{Bmatrix} f_1 \\ f_2 \\ \cdots \\ f_n \end{Bmatrix} \quad (4.2.2)$$

equation is used. This equation can be change to a new format:

$$[\nabla f]\{\Delta x\} = -\{f\} \qquad (4.2.3)$$

In this set derivative term $[\nabla f]$ be replaced by difference (numerical derivative)

$$\frac{\partial f_j(x^{(i)})}{\partial x_k} \approx \frac{f_j(x^{(i)}+e_k h)-f_j(x^{(i)})}{h} \quad (4.2.3)$$

h in the equation is finite difference parameter (Δx) and e_k is unit vektor (kth component is 1 the other components are 0). i is the iteration number. Note that secant method is always approach to the roots less quickly compare to Newton-Raphson method. In order to improve number of iterations for the solutions, a modified version of secant method, Broyden method will be used. In Broyden method when x(0) is a given value, first iteration, x(1), will be calculated by using Newton-Raphson method. For the following steps

$$\left[\nabla f(x^{(0)})\right]\left\{x^{(1)} - x^{(0)}\right\} = -\left\{f\left(x^{(0)}\right)\right\} \quad (4.2.4)$$ form of the equation can be changed to be

$$[A_1]\left\{x^{(1)} - x^{(0)}\right\} = \left\{f\left(x^{(1)}\right) - f\left(x^{(0)}\right)\right\} \quad (4.2.5)$$ In order to have this equation valid for every vector z, vector should satisfy following equations

$$[A_1]\{z\} = \left[\nabla f(x^{(0)})\right]z \quad (4.2.6) \qquad \left\{x^{(1)} - x^{(0)}\right\}z = 0 \quad (4.2.7)$$

By using this defination Formula to calculate $[A_1]$ can be derived.

$$[A_1] = \left[\nabla f(x^{(0)})\right] + \frac{[\{f(x^{(1)})-f(x^{(0)})\}-\nabla f(x^{(0)})\{x^{(1)}-x^{(0)}\}]\{x^{(1)}-x^{(0)}\}^t}{\|x^{(1)}-x^{(0)}\|_2^2} \quad (4.2.8)$$

Where the term $\left\|x^{(1)} - x^{(0)}\right\|_2$ (4.2.9) is a two norm. Norms was defined in previous chapter. After finding $[A_1]$ new value for vector x can be constructed.

$$x^{(2)} = x^{(1)} - [A_1]^{-1} f\left(x^{(1)}\right) \quad (4.2.10)$$

When $x^{(2)}$ is found, this value can be used to calculate $x^{(3)}$.General Formula will become:

$$[A_i] = [A_{i-1}] + \frac{[\{f(x^{(i)})-f(x^{(i-1)})\}-\nabla f(x^{(i-1)})\{x^{(i)}-x^{(i-1)}\}]\{x^{(i)}-x^{(i-1)}\}^t}{\|x^{(i)}-x^{(i-1)}\|_2^2} \quad (4.2.11)$$

$$x^{(i+1)} = x^{(i)} - [A_i]^{-1} f\left(x^{(i)}\right) \quad (4.2.12)$$

One difficulty in this equation is the requirement of solving inverse of the matrix each time which requires solving of system of equation by a method such as Gauss elimination. But Gauss method is expensive in calculation time. A beter alternative could be using Sherman-Morrison approach. This approach calculates inverse directly as an iterative process. Matrix A_i^{-1} can directly be calculated from value A_{i-1}^{-1}. Equations of Sherman-Morrison approach is as follows

$$s_i = \left\{x^{(i)} - x^{(i-1)}\right\} \qquad (4.2.13)$$

$$y_i = \{f(x^{(i)}) - f(x^{(i-1)})\} \quad (4.2.14)$$

$$([\mathrm{A}] + xy^t)^t = [A]^{-1} - \frac{[\mathrm{A}]^{-1}xy^t[\mathrm{A}]^{-1}}{1+y^t[\mathrm{A}]^{-1}x} \quad (4.2.15)$$

$$[A_i]^{-1} = \left([A_{i-1}]^{-1} + \frac{y_i - [A_{i-1}]s_i}{\|s_i\|_2^2} s_i^t\right)^{-1} \quad (4.2.16)$$

$$[A_i]^{-1} = [A_i]^{-1} + \frac{[A_i]^{-1}\left(\frac{y_i - [A_{i-1}]s_i}{\|s_i\|_2^2}\right)[A_i]^{-1}}{1+s_i^t[A_i]^{-1}\left(\frac{y_i - [A_{i-1}]s_i}{\|s_i\|_2^2}\right)} s_i^t \quad (4.2.17)$$

$$[A_i]^{-1} = [A_i]^{-1} + \frac{(s_i - [A_{i-1}]^{-1}y_i)[A_i]^{-1}}{s_i^t[A_{i-1}]^{-1}y_i} \quad (4.2.18)$$

Due to the fact that this equation only involves multiplication processes, it is easier to use compare to Gauss elimination method. Algorithm of the method is given below.

Program 4.2-1 Broyden secant method with sherman-Morrison Formula.

```
 // NA30A Broyden secant method with Sherman-Morrison formula to solve inverse matrix
// Solution of non-linear system of equations.
// Referans : Numerical Analysis, 8th edition Richard L. Burden, J. Douglas Faires 3
// page 617
import java.util.*;
import java.awt.*;
import java.applet.Applet;
import java.awt.event.*;
import javax.swing.*;
//============= Tanimi gereken fonksiyon ================
abstract class fi_xi
{
  // multifunction multi independent variable
  // vector of dependent variables are returned
  // example f[0]=x[0]+sin(x[1])
  //         f[1]=x[0]*x[0]-x[1]
  // func(x) returns the value of f[0] and f[1]
  // as a two dimensional vector

  abstract double[] func(double x[]);
}

abstract class fij_xi
{
  // multifunction multi independent variable
  // for n independent variable n*n matrix of
  // dependent variables are returned
  // example f[0][0]=x[0]+Math.sin(x[1])   f[0][1]=x[0]-x[1]
  //         f[1][0]=x[0]*x[0]-x[1]        f[1][1]=Math.exp(x[0]+x[1]*x[1]
  // func(x) returns the value of f[0][0], f[0][1]
  //                              f[1][0], f[1][1]
  // as a two dimensional matrix
  abstract double[][] func(double x[]);
}

class fa extends fi_xi
{
public double[] func(double x[])
```

```
{
//function to be solved.
double ff[]=new double[3];
ff[0]=3.0*x[0]-Math.cos(x[1]*x[2])-0.5;
ff[1]=x[0]*x[0]-81.0*(x[1]+0.1)*(x[1]+0.1)+Math.sin(x[2])+1.06;
ff[2]=Math.exp(-x[0]*x[1])+20.0*x[2]+(10.0*Math.PI-3.0)/3.0;
return ff;
}
}

class fb extends fij_xi
{
fi_xi f;

public fb(fi_xi fi)
{f=fi;}

public double[][] func(double x[])
{
  int n=x.length;
  double a[][]=new double[n][n];
  for(int i=0;i<n;i++)
  {
    for(int j=0;j<n;j++)
    {
       a[i][j]=derivative(x,i,j);
    }
  }
return a;
}

public double derivative(double x[],int denklem_ref,int x_ref)
{
// df(x)/dx[x_ref]  x_ref=0...n
double h0=0.0156;
int i,m;
int n=7;
double f1[];
f1=new double[x.length];
double f2[];
f2=new double[x.length];
double x1[];
x1=new double[x.length];
double x2[];
x2=new double[x.length];
for(i=0;i<x.length;i++)
{
x1[i]=x[i];
x2[i]=x[i];
}
//derivative of a simple function
double T[][];
T=new double[n][n];
double h[];
h=new double[n];
//vector<double> h(n,0);
for(i=0;i<n;i++)
{
  h[i]=0;
  for(int j=0;j<n;j++)
    T[i][j]=0;
}
h[0]=h0;
double r=0.5;
for( i=1;i<n;i++)
{
```

```
h[i]=h0*Math.pow(r,i);
}

for(i=0;i<n;i++)
{
x1[x_ref]+=h[i];
x2[x_ref]-=h[i];
f1=f.func(x1);
f2=f.func(x2);
T[i][0]=( f1[denklem_ref] - f2[denklem_ref])/(2.0*h[i]);
x1[x_ref]=x[x_ref];
x2[x_ref]=x[x_ref];
}
for(m=1;m<n;m++)
{
  for(i=0;i<n-m;i++)
  {
  T[i][m]=(h[i]*h[i]*T[i+1][m-1] - h[i+m]*h[i+m]*T[i][m-1])/(h[i]*h[i]
  - h[i+m]*h[i+m]);
  }
}
double xx=T[0][n-1];
return xx;
}

}

public class NA30A
{
// SCO5E Sherman-Morrison formülü destekli broyden sekant metoduyla
//denklem sistemlerinin köklerini (çözümünü) bulmak

public static double[] multiply(double left,double[] right)
{
//multiplying a vector with a constant
int i;
int n=right.length;
double b[];
b=new double[n];
for(i=0;i<n;i++)
  {b[i]=left*right[i];}
return b;
}

public static double[] multiply(double[][] left,double[] right)
{
//multiplication of one matrix with one vector
int ii,jj,i,j,k;
int m1=left[0].length;
int n1=left.length;
int m2=right.length;
double[] b;
b=new double[m2];
 if(n1 != m2)
 {
 System.out.println("inner matrix dimensions must agree");
 for(ii=0;ii<n1;ii++)
   {
     b[ii]=0;
   }
   return b;
 }
 for(i=0;i<m1;i++)
   {
     b[i]=0;
     for(k=0;k<n1;k++)
```

```
            b[i]+=left[i][k]*right[k];
   }
return b;
//end of multiply of a matrix and a vector
}

public static double[] multiply(double[] left,double[][] right)
{
//multiplication of one vector with one matrix
int ii,jj,i,j,k;
int m2=right[0].length;
int n2=right.length;
int m1=left.length;
double[] b;
b=new double[m1];
 if(n2 != m1)
 {
 System.out.println("inner matrix dimensions must agree");
 for(ii=0;ii<n2;ii++)
   {
     b[ii]=0;
   }
   return b;
 }
 for(i=0;i<m2;i++)
   {
     b[i]=0;
     for(k=0;k<m1;k++)
            b[i]+=right[i][k]*left[k];
   }
return b;
//end of multiply of a vector and a matrix
}

public static double  VT_X(double [] left,double [] right)
{
//multiplys a vector transpose with a vector
int n=left.length;
double tot=0;
for(int i=0;i<n;i++)
  {
  tot+=left[i]*right[i];
  }
return tot;
}

public static double[] add(double[] left,double[] right)
{
//addition of two vectors
int n1=left.length;
int n2=right.length;
int nMax;
int i;
if(n1>=n2) nMax=n1;
else       nMax=n2;
double b[];
b=new double[nMax];
for(i=0;i<n1;i++)
  {
    b[i]=b[i]+left[i];
  }
for(i=0;i<n2;i++)
  {
    b[i]=b[i]+right[i];
  }
return b;
```

```
//end of vector addition method
}

public static double[][] multiply(double left,double[][] right)
{
//multiplying a matrix with a constant
int i,j;
int n=right.length;
int m=right[0].length;
double b[][];
b=new double[n][m];
for(i=0;i<n;i++)
  {
  for(j=0;j<m;j++)
    b[i][j]=right[i][j]*left;
  }
return b;
//end of multiplying a matrix with a constant double
}

public static double[][] add(double[][] left,double[][] right)
{
//addition of two matrices
int n1=left.length;
int m1=left[0].length;
int n2=right.length;
int m2=right[0].length;
int nMax,mMax;
int i,j;
if(m1>=m2) mMax=m1;
else       mMax=m2;
if(n1>=n2) nMax=n1;
else       nMax=n2;
double b[][];
b=new double[nMax][mMax];
for(i=0;i<n1;i++)
  {
  for(j=0;j<m1;j++)
    {
    b[i][j]=b[i][j]+left[i][j];
    }
  }
for(i=0;i<n2;i++)
  {
  for(j=0;j<m2;j++)
    {
    b[i][j]=b[i][j]+right[i][j];
    }
  }
return b;
//end of matrix addition method
}

public static double[] substract(double[] left,double[] right)
{
//addition of two vectors
int n1=left.length;
int n2=right.length;
int nMax;
int i;
if(n1>=n2) nMax=n1;
else       nMax=n2;
double b[];
b=new double[nMax];
for(i=0;i<n1;i++)
  {
```

```
    b[i]=b[i]+left[i];
  }
for(i=0;i<n2;i++)
  {
    b[i]=b[i]-right[i];
  }
return b;
//end of vector addition method
}

public static double[][] inv(double[][] a)
{
// INVERSION OF A MATRIX
// inversion by using gaussian elimination
// with full pivoting
int n=a.length;
int m=a[0].length;
double b[][];
b=new double[n][n];
int indxc[];
int indxr[];
double ipiv[];
indxc=new int[n];
indxr=new int[n];
ipiv=new double[n];
int i,j,k,l,ll,ii,jj;
int icol=0;
int irow=0;
double big,dum,pivinv,temp;
if(n!=m)
{
   System.out.println("Matrix must be square ");
                for(ii=0;ii<n;ii++)
                  for(jj=0;jj<n;jj++)
                        b[ii][jj]=0.0;
                return b;
}
for(i=0;i<n;i++)
   for(j=0;j<n;j++)
      b[i][j]=a[i][j];
for(i=0;i<n;i++)
{
  big=0.0;
  for(j=0;j<n;j++)
  {
    if(ipiv[j] != 1)
      for(k=0;k<n;k++)
      {
              if(ipiv[k] == 0)
              {
                if(Math.abs(b[j][k]) >= big)
                {
                  big=Math.abs(b[j][k]);
                  irow=j;
                  icol=k;
                }
              }
              else if(ipiv[k] > 1 )
                {
                System.out.println("error : inverse of the matrix : singular matrix-1");
                for(ii=0;ii<n;ii++)
                  for(jj=0;jj<n;jj++)
                        b[ii][jj]=0.0;
                return b;
                }
      }
```

```
   }
   ++ ipiv[icol];
   if(irow != icol)
     for(l=0;l<n;l++)
     {
     temp=b[irow][l];
     b[irow][l]=b[icol][l];
     b[icol][l]=temp;
     }
   indxr[i]=irow;
   indxc[i]=icol;
   if(b[icol][icol] == 0.0)
   {
     System.out.println("error : inverse of the matrix : singular matrix-2");
                for(ii=0;ii<n;ii++)
                  for(jj=0;jj<n;jj++)
                          b[ii][jj]=0.0;
     return b;
   }
   pivinv=1.0/b[icol][icol];
   b[icol][icol]=1.0;
   for(l=0;l<n;l++) b[icol][l] *=pivinv;
   for(ll=0;ll<n;ll++)
   if(ll != icol)
   {
     dum=b[ll][icol];
     b[ll][icol]=0.0;
     for(l=0;l<n;l++) b[ll][l]-= b[icol][l]*dum;
   }
}
for(l=n-1;l>=0;l--)
{
  if(indxr[l] != indxc[l])
      for(k=0;k<n;k++)
      {
        temp=b[k][indxc[l]];
        b[k][indxc[l]]=b[k][indxr[l]];
        b[k][indxr[l]]=temp;
      }
}
return b;
}
public static double norm(double v[])
{
// vector norm
double total=0;;
for(int i=0;i<v.length;i++)
  {total+=v[i]*v[i];}
return Math.sqrt(total);
}

public static double[][] multiply(double[] left,double[] right)
{
//multiplication of one vector with one matrix
int ii,jj,i,j,k;
int m2=left.length;
int n2=right.length;
int m1=left.length;
double[][] b;
b=new double[m2][n2];
 for(i=0;i<m2;i++)
   {
     for(k=0;k<n2;k++)
                b[i][k]=left[i]*right[k];
   }
return b;
```

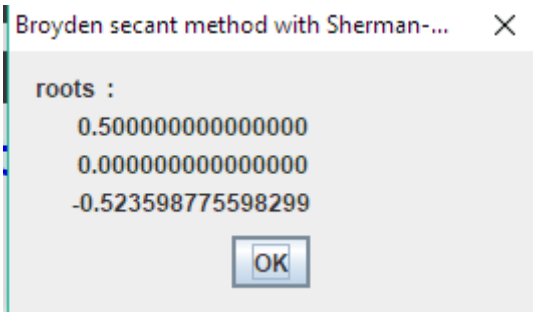
```
//end of multiply of a vector and a matrix
}

public static double[] broyden(fi_xi f,double x[])
{
//=====================================================
// Roots of non-linear system of equations
// Broyden secant method
fb J=new fb(f);
int i;
int nmax=400;
double tolerance=1.0e-15;
int n=x.length;
double b[]=new double[n];
double v[]=new double[n];
double s[]=new double[n];
double w[]=new double[n];
double y[]=new double[n];
double z[]=new double[n];
double p;
double ut[]=new double[n];
double x1[]=new double[n];
double A[][]=new double[n][n];
x1=x;
A=J.func(x1);
v=f.func(x1);
A=inv(A);
s=multiply(-1.0,multiply(A,v));
x=add(x,s);
int k=1;
while(k<nmax)
{
w=v;
v=f.func(x);
y=substract(v,w);
z=multiply(-1.0,multiply(A,y));
p=-VT_X(s,z);
ut=multiply(s,A);  //*********
A=add(A,multiply((1.0/p),multiply(add(s,z),ut)));
s=multiply(-1.0,multiply(A,v));
x=add(x,s);
if(norm(s)<tolerance) break;
k++;
}
return x;
}

public static double[] enterdata()
{
   String s=JOptionPane.showInputDialog("enter first gueses for roots with a sapce in between: ");
   StringTokenizer token=new StringTokenizer(s);
   int n=token.countTokens()-1;
   int m=n+1;
   double a[]=new double[m];
   int j=0;
   while(token.hasMoreTokens())
   {
   Double ax=new Double(token.nextToken());
   a[j++]=ax.doubleValue();
   }
   return a;
}

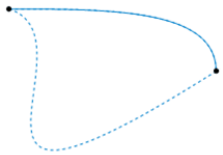
    public static void main (String args[])
    {
            fa f =new fa();
```

```
    double [] x0=enterdata();
    double [] r= broyden(f,x0);
    String s=" roots  : \n"+Matrix.toStringT(r);
    String s1="Broyden secant method with Sherman-Morrison iterative inverse matrix formula : ";
    JOptionPane.showMessageDialog(null,s,s1,JOptionPane.PLAIN_MESSAGE);
    System.exit(0);
    }
}
```



4.3 HOMOTOPY OR CONTINUATION METHODS

In topology, two continuous functions from one topological space to another are called **homotopic**(from Greek ὁμός homós "same, similar" and τόπος tópos "place") if one can be "continuously deformed" into the other, such a deformation being called a **homotopy** between the two functions.



When a problem of system of nonlinear equations of the form F(x)=0 desired to be solved, another interesting solution method is Homotopy or Continuation method. Assume that solution set to be found is x*. Consider a parametric function G(λ,x) in the form of

G(λ,x) = λF(x) + (1-λ)[F(x) - F(x(0))] (4.3.1)

Where λ=0 coresponds to initial guess of the solution , x(0) ,and where λ=1 value correspods the actual solution set x(1)= x* (4.3.2)

It is desired to be found G(λ,x) = 0 therefore for λ=0 equation becomes

G(λ,x) =G(0,x) = F(x) - F(x(0)) and for λ=1 (4.3.4)

0=G(1,x) = F(x) (4.3.4)

Therefore x(1)=x* solution set will be obtained. If a function G(λ,x) satisfies the above equation can be found, it will also give us the solution. Function G is called a homotopy between the function G(0,x) and G(1,x)=F(x) (4.3.5)

In order to find such a function, it is assumed to have a function

G(λ,x)=0 is existed and partial derivative of this function with respect to λ and x will also be zero

$$\frac{\partial G(\lambda,x)}{\partial \lambda} + \frac{\partial G(\lambda,x)}{\partial x} x'(\lambda) = 0 \qquad (4.3.6)$$

if x'(λ) is isolated form this equation, it becomes:

$$x'(\lambda) = -\left[\frac{\partial G(\lambda,x)}{\partial x}\right]^{-1}\left[\frac{\partial G(\lambda,x)}{\partial \lambda}\right] \quad (4.3.7)$$

If $G(\lambda,x)=\lambda F(x)+(1-\lambda)[F(x)-F(x(0))]$ the equation is substituted into the differential equation

$$\left[\frac{\partial G(\lambda,x)}{\partial \lambda}\right] = \begin{bmatrix} \frac{\partial f_1(x(\lambda))}{\partial x_1} & \frac{\partial f_1(x(\lambda))}{\partial x_2} & \frac{\partial f_1(x(\lambda))}{\partial x_{33}} \\ \frac{\partial f_2(x(\lambda))}{\partial x_1} & \frac{\partial f_2(x(\lambda))}{\partial x_2} & \frac{\partial f_2(x(\lambda))}{\partial x_3} \\ \frac{\partial f_3(x(\lambda))}{\partial x_1} & \frac{\partial f_3(x(\lambda))}{\partial x_2} & \frac{\partial f_3(x(\lambda))}{\partial x_3} \end{bmatrix} = J(x(\lambda)) \quad (4.3.8)$$

Forms a Jacobian matrix. and

$\left[\frac{\partial G(\lambda,x)}{\partial x}\right] = F(x(0))$ (4.3.9)

Differential equation becomes

$x'(\lambda) = \frac{dx(\lambda)}{d\lambda} = -[J(x(\lambda)]^{-1}F(x(0))$ $0 \leq \lambda \leq 1$ (4.3.10)

It is possible to solve such a differential equation by using initial value problem approaches, solution at x(1) will be given us the roots of the system of equation. Solutions of initial value problems will be given latter chapters in details, but A fourth order Runge-Kutta solution will be defined here to solve our homotopy problem. If equation

$\frac{dx(\lambda)}{d\lambda} = f(\lambda,x(\lambda))$ (4.3.11) is given the 4th order Runge-Kutta method to numerically solve this differential equation is defined as:

$x_{i+1} = x_i + (1/6)*(k_1 + 2k_2 +2k_3+k_4)h$ (4.3.12)

$k_1=f(\lambda_i,x_i)$

$k_2=f(\lambda_i+0.5h,x_i+0.5k_1h)$

$k_3=f(\lambda_i+0.5h,x_i+0.5k_2h)$

$k_4=f(\lambda_i+h,x_i+k_3h)$

In these equations h is finite difference step size. Solution starts by using the initial value $\lambda=0$, x0(0) and adds h into λ in each iteration step. The example program uses 4th degree and 6th degree Runge-Kutta method to solve homotopy(Continuation problem). It should be note that Homotophy method is less dependent to initial value compare to methods such as Newton-Raphson therefore one possibility is to approachsolution with a relatively rough estimate with homotophy following with a Newton-Raphson type of method. In the following code methods of Homotophy with 4th and 6th order Runge-Kutta methods and combination of Homotophy with 4th order Runge-Kutta and Newton-Raphson method is given.

Program 4.3-1 Homotophy (Contiuation) method (java version)

```
 / Continuation-homotopy
// solution of system of nonlinar equations (roots of equations)
// Referans : Numerical Analysis, 8th edition Richard L. Burden, J. Douglas Faires 3
// page 637
// All the required derivatives calculated numercally

import java.util.*;
import java.awt.*;
import java.applet.Applet;
import java.awt.event.*;
import javax.swing.*;
//============= Functions to be defined ================
class fa extends fi_xi
{
public double[] func(double x[])
{
//çözümü istenen fonksiyon
double ff[]=new double[3];
ff[0]=3.0*x[0]-Math.cos(x[1]*x[2])-0.5;
ff[1]=x[0]*x[0]-81.0*(x[1]+0.1)*(x[1]+0.1)+Math.sin(x[2])+1.06;
ff[2]=Math.exp(-x[0]*x[1])+20.0*x[2]+(10.0*Math.PI-3.0)/3.0;
return ff;
} }

public class NA30C
{
// (Continuity-homotopy) method to solve a system of nonlinear equations
//


public static double[] multiply(double left,double[] right)
{
```

```
//multiplying a vector with a constant
int i;
int n=right.length;
double b[];
b=new double[n];
for(i=0;i<n;i++)
  {b[i]=left*right[i];}
return b;
}

public static double[] multiply(double[][] left,double[] right)
{
//multiplication of one matrix with one vector
int ii,jj,i,j,k;
int m1=left[0].length;
int n1=left.length;
int m2=right.length;
double[] b;
b=new double[m2];
 if(n1 != m2)
 {
 System.out.println("inner matrix dimensions must agree");
 for(ii=0;ii<n1;ii++)
   {
    b[ii]=0;
   }
  return b;
 }
 for(i=0;i<m1;i++)
   {
    b[i]=0;
    for(k=0;k<n1;k++)
            b[i]+=left[i][k]*right[k];
   }
return b;
//end of multiply of a matrix and a vector
}

public static double[] add(double[] left,double[] right)
{
//addition of two vectors
int n1=left.length;
int n2=right.length;
int nMax;
int i;
if(n1>=n2) nMax=n1;
else       nMax=n2;
double b[];
b=new double[nMax];
for(i=0;i<n1;i++)
  {
   b[i]=b[i]+left[i];
  }
for(i=0;i<n2;i++)
  {
   b[i]=b[i]+right[i];
  }
return b;
//end of vector addition method
}

public static double[][] inv(double[][] a)
{
// INVERSION OF A MATRIX
// inversion by using gaussian elimination
// with full pivoting
```

```
int n=a.length;
int m=a[0].length;
double b[][];
b=new double[n][n];
int indxc[];
int indxr[];
double ipiv[];
indxc=new int[n];
indxr=new int[n];
ipiv=new double[n];
int i,j,k,l,ll,ii,jj;
int icol=0;
int irow=0;
double big,dum,pivinv,temp;
if(n!=m)
{
  System.out.println("Matrix must be square ");
            for(ii=0;ii<n;ii++)
              for(jj=0;jj<n;jj++)
                    b[ii][jj]=0.0;
            return b;
}
for(i=0;i<n;i++)
  for(j=0;j<n;j++)
     b[i][j]=a[i][j];
for(i=0;i<n;i++)
{
 big=0.0;
 for(j=0;j<n;j++)
 {
  if(ipiv[j] != 1)
    for(k=0;k<n;k++)
    {
          if(ipiv[k] == 0)
          {
           if(Math.abs(b[j][k]) >= big)
           {
            big=Math.abs(b[j][k]);
            irow=j;
            icol=k;
           }
          }
          else if(ipiv[k] > 1 )
           {
           System.out.println("error : inverse of the matrix : singular matrix-1");
           for(ii=0;ii<n;ii++)
             for(jj=0;jj<n;jj++)
                    b[ii][jj]=0.0;
           return b;
           }
    }
 }
 ++ ipiv[icol];
 if(irow != icol)
  for(l=0;l<n;l++)
  {
  temp=b[irow][l];
  b[irow][l]=b[icol][l];
  b[icol][l]=temp;
  }
 indxr[i]=irow;
 indxc[i]=icol;
 if(b[icol][icol] == 0.0)
 {
  System.out.println("error : inverse of the matrix : singular matrix-2");
            for(ii=0;ii<n;ii++)
```

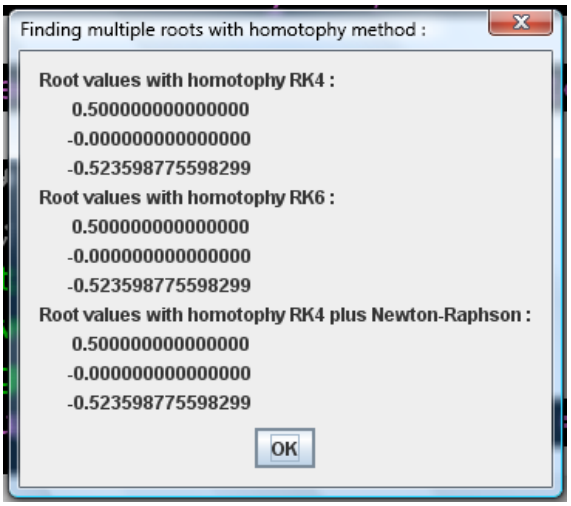
```
                for(jj=0;jj<n;jj++)
                        b[ii][jj]=0.0;
    return b;
   }
   pivinv=1.0/b[icol][icol];
   b[icol][icol]=1.0;
   for(l=0;l<n;l++) b[icol][l] *=pivinv;
   for(ll=0;ll<n;ll++)
   if(ll != icol)
   {
     dum=b[ll][icol];
     b[ll][icol]=0.0;
     for(l=0;l<n;l++) b[ll][l]-= b[icol][l]*dum;
   }
}
for(l=n-1;l>=0;l--)
{
  if(indxr[l] != indxc[l])
     for(k=0;k<n;k++)
     {
       temp=b[k][indxc[l]];
       b[k][indxc[l]]=b[k][indxr[l]];
       b[k][indxr[l]]=temp;
     }
}
return b;
}

public static double[] continuationRK4(fi_xi f,double x[],int N)
{
//=====================================================
// Roots of nonlinear system of equations Homotophy RK6
// 4.th order Runge-Kutta
// yi+1 = yi + (1/6)*( k1 + 2k2 +2k3+k4)h
// k1=f(xi,yi)
// k2=f(xi+0.5h,yi+0.5k1h)
// k3=f(xi+0.5h,yi+0.5k2h)
// k4=f(xi+h,yi +k3h)
//=====================================================
//x vector of independent variables
//y vector of dependent variables
//dy derivative vector of dependent variables
int i;
int nmax=400;
double tolerance=1.0e-20;
int n=x.length;
double h=1.0/(double)N;
double b[]=new double[n];
double x1[]=new double[n];
double k[][]=new double[4][n];
double A[][]=new double[n][n];
b=multiply(-h,f.func(x));
for(i=0;i<N;i++)
{
x1=x;
A=f.dfunc(x1);
k[0]=multiply(inv(A),b);
x1=add(x,multiply(0.5,k[0]));
A=f.dfunc(x1);
k[1]=multiply(inv(A),b);
x1=add(x,multiply(0.5,k[1]));
A=f.dfunc(x1);
k[2]=multiply(inv(A),b);
x1=add(x,k[2]);
A=f.dfunc(x1);
k[3]=multiply(inv(A),b);
```

```
for(int j=0;j<n;j++) {x[j]=x[j]+1.0/6.0*(k[0][j]+2.0*k[1][j]+2.0*k[2][j]+k[3][j]);}
}
return x;
}

public static double[] continuationRK6(fi_xi f,double x[],int N)
{
//=================================================
// Roots of nonlinear system of equations Homotophy RK6
// yi+1 = yi + (1/90)*( 7k1 + 32k3 +12k4+32k5+7k6)h
// k1=f(xi,yi)
// k2=f(xi+0.25h , yi+0.25k1h)
// k3=f(xi+0.25h , yi+0.125k1h+0.125k2h)
// k4=f(xi+0.5h , yi - 0.5k2h+k3h)
// k5=f(xi+0.75h , yi + (3/16)k1h+(9/16)k4h)
// k6=f(xi+h , yi - (3/7)k1h+(2/7)k2h+(12/7)k3h - (12/7)k4h+(8/7)k5h)
//=================================================
//x vector of independent variables
//y vector of dependent variables
//dy derivative vector of dependent variables
int i;
int nmax=400;
double tolerance=1.0e-20;
int n=x.length;
double h=1.0/(double)N;
double b[]=new double[n];
double x1[]=new double[n];
double k[][]=new double[6][n];
double A[][]=new double[n][n];
b=multiply(-h,f.func(x));
for(i=0;i<N;i++)
{
A=f.dfunc(x);
// k1=f(xi,yi)
k[0]=multiply(inv(A),b);
x1=add(x,multiply(0.25,k[0]));
A=f.dfunc(x1);
// k2=f(xi+0.25h , yi+0.25k1h)
k[1]=multiply(inv(A),b);
x1=add(x,add(multiply(0.125,k[0]),multiply(0.125,k[1])));
A=f.dfunc(x1);
// k3=f(xi+0.25h , yi+0.125k1h+0.125k2h)
k[2]=multiply(inv(A),b);
x1=add(x,add(multiply(-0.5,k[1]),k[2]));
A=f.dfunc(x1);
// k4=f(xi+0.5h , yi - 0.5k2h+k3h)
k[3]=multiply(inv(A),b);
x1=add(x,add(multiply((3.0/16.0),k[0]),multiply((9.0/16.0),k[3])));
A=f.dfunc(x1);
// k5=f(xi+0.75h , yi + (3/16)k1h+(9/16)k4h)
k[4]=multiply(inv(A),b);
x1=add(x,
      add(multiply((-3.0/7.0),k[0]),add(multiply((2.0/7.0),k[1]),
      add(multiply((12.0/7.0),k[2]),
         add(multiply((-12.0/7.0),k[3]),multiply((8.0/7.0),k[4]))))));
A=f.dfunc(x1);
// k6=f(xi+h , yi - (3/7)k1h+(2/7)k2h+(12/7)k3h - (12/7)k4h+(8/7)k5h)
k[5]=multiply(inv(A),b);
// yi+1 = yi + (1/90)*( 7k1 + 32k3 +12k4+32k5+7k6)h
for(int j=0;j<n;j++) {x[j]=x[j]+1.0/90.0*(7.0*k[0][j]+32.0*k[2][j]+12.0*k[3][j]+32.0*k[4][j]+7.0*k[5][j]);}
}
return x;
}
public static double[] newton_continuationRK4(fi_xi f,double x[])
{
// Roots of nonlinear system of equations Homotophy RK4 plus Newton-Raphson
```

```
//ti : weight function
//x vector of independent variables
//y vector of dependent variables
//dy derivative vector of dependent variables
x=continuationRK4(f,x,4);
double ti=1.0;
int i;
int nmax=400;
double tolerance=1.0e-10;
int n=x.length;
double b[];
b=new double[n];
for(i=0;i<n;i++)
{
b[i]=1.0;
}
i=0;
while( i++ < nmax && Matrix.abs(b) > tolerance )
{ b=Matrix.multiply(Matrix.divide(f.func(x),f.dfunc(x)),-ti);
  x=Matrix.add(x,b);
}
if(i >= nmax) JOptionPane.showMessageDialog(null,"Warning : Maximum number of iterations are exceeded \n"+
            " Results may not be valid","MAXIMUM ITERATIONS WARNING",JOptionPane.WARNING_MESSAGE);
System.out.println();
return x;
}
public static double[] enterdata()
{
   String s=JOptionPane.showInputDialog("enter initial root estimations with a space in between ");
   StringTokenizer token=new StringTokenizer(s);
   int n=token.countTokens()-1;
   int m=n+1;
   double a[]=new double[m];
   int j=0;
   while(token.hasMoreTokens())
   {
   Double ax=new Double(token.nextToken());
   a[j++]=ax.doubleValue();
   }
   return a;
}
    public static void main (String args[])
    { fa f =new fa();
    double [] x0=enterdata();
    double [] r= continuationRK4(f,x0,4);
    double [] r1= continuationRK6(f,x0,4);
    double [] r2= newton_continuationRK4(f,x0);
    String s=" Root values with homotophy RK4 : \n"+Matrix.toStringT(r);
    s+=" Root values with homotophy RK6 : \n"+Matrix.toStringT(r1);
    s+=" Root values with homotophy RK4 plus Newton-Raphson : \n"+Matrix.toStringT(r2);
    String s1="Finding multiple roots with homotophy method : ";
    JOptionPane.showMessageDialog(null,s,s1,JOptionPane.PLAIN_MESSAGE);
    System.exit(0);
    }
}
```



Program 4.3-2 Homotophy (Contiuation) method (python version)

```
# -*- coding: utf-8 -*-
"""
Created on Thu Aug 23 11:49:55 2018

@author: Mustafa Turhan Çoban
"""
from math import *
from fi_xi import *;
from gauss import *;
from newtond import *;

class fa(fi_xi):
  def func(self,x):

      ff=[0 for i in range(3)];
      ff[0]=3.0*x[0]-cos(x[1]*x[2])-0.5;
      ff[1]=x[0]*x[0]-81.0*(x[1]+0.1)*(x[1]+0.1)+sin(x[2])+1.06;
      ff[2]=exp(-x[0]*x[1])+20.0*x[2]+(10.0*pi-3.0)/3.0;
      return ff;

def unit_matrix(n):
   #unit matrix of dimension n
   B = [[0 for x in range(n)] for y in range(n)];
   for i in range(0,n):
       for j in range(n):
           if i==j: B[i][j]=1.0;
   return B;

def invA(A):
    #inverse matrix (n=m)
    n=len(A);
    I= unit_matrix(n);
    B = [[0 for x in range(n)] for y in range(n)];
    for i in range(n):
        B[i]=gauss(A,I[i]);
    return B;

def continuationRK4(f,x,N):
    #=====================================================
    # Roots of nonlinear system of equations Homotophy RK6
    # 4.th order Runge-Kutta
    # yi+1 = yi + (1/6)*( k1 + 2k2 +2k3+k4)h
    # k1=f(xi,yi)
    # k2=f(xi+0.5h,yi+0.5k1h)
    # k3=f(xi+0.5h,yi+0.5k2h)
    # k4=f(xi+h,yi +k3h)
    #=====================================================
    #x vector of independent variables
    #y vector of dependent variables
    #dy derivative vector of dependent variables
    nmax=400;
```

```
    tolerance=1.0e-20;
    n=len(x);
    h=1.0/float(N);
    b=[0 for i in range(n)];
    x1=[0 for i in range(n)];
    k = [[0 for x in range(n)] for y in range(4)];
    A= [[0 for x in range(n)] for y in range(n)];
    b=multiply_c_v(-h,f.func(x));
    for i in range(N):
       x1=x;
       A=f.dfunc(x1,1);
       k[0]=multiply_m_v(invA(A),b);
       x1=add(x,multiply_c_v(0.5,k[0]));
       A=f.dfunc(x1,1);
       k[1]=multiply_m_v(invA(A),b);
       x1=add(x,multiply_c_v(0.5,k[1]));
       A=f.dfunc(x1,1);
       k[2]=multiply_m_v(invA(A),b);
       x1=add(x,k[2]);
       A=f.dfunc(x1,1);
       k[3]=multiply_m_v(invA(A),b);
       for j in range(n):
          x[j]=x[j]+1.0/6.0*(k[0][j]+2.0*k[1][j]+2.0*k[2][j]+k[3][j]);
    return x;

def continuationRK6(f,x,N):
    #=================================================
    # Roots of nonlinear system of equations Homotophy RK6
    # yi+1 = yi + (1/90)*( 7k1 + 32k3 +12k4+32k5+7k6)h
    # k1=f(xi,yi)
    # k2=f(xi+0.25h , yi+0.25k1h)
    # k3=f(xi+0.25h , yi+0.125k1h+0.125k2h)
    # k4=f(xi+0.5h , yi - 0.5k2h+k3h)
    # k5=f(xi+0.75h , yi + (3/16)k1h+(9/16)k4h)
    # k6=f(xi+h , yi - (3/7)k1h+(2/7)k2h+(12/7)k3h - (12/7)k4h+(8/7)k5h)
    #=================================================
    #x vector of independent variables
    #y vector of dependent variables
    #dy derivative vector of dependent variables
    n=len(x);
    h=1.0/float(N);
    b=[0 for i in range(n)];
    x1=[0 for i in range(n)];
    k = [[0 for x in range(n)] for y in range(6)];
    A= [[0 for x in range(n)] for y in range(n)];
    b=multiply_c_v(-h,f.func(x));
    for i in range(N):
       A=f.dfunc(x,1);
       # k1=f(xi,yi)
       k[0]=multiply_m_v(invA(A),b);
       x1=add(x,multiply_c_v(0.25,k[0]));
       A=f.dfunc(x1,1);
       # k2=f(xi+0.25h , yi+0.25k1h)
       k[1]=multiply_m_v(invA(A),b);
       x1=add(x,add(multiply_c_v(0.125,k[0]),multiply_c_v(0.125,k[1])));
       A=f.dfunc(x1,1);
       # k3=f(xi+0.25h , yi+0.125k1h+0.125k2h)
       k[2]=multiply_m_v(invA(A),b);
       x1=add(x,add(multiply_c_v(-0.5,k[1]),k[2]));
       A=f.dfunc(x1,1);
       # k4=f(xi+0.5h , yi - 0.5k2h+k3h)
       k[3]=multiply_m_v(invA(A),b);
       x1=add(x,add(multiply_c_v((3.0/16.0),k[0]),multiply_c_v((9.0/16.0),k[3])));
       A=f.dfunc(x1,1);
       # k5=f(xi+0.75h , yi + (3/16)k1h+(9/16)k4h)
       k[4]=multiply_m_v(invA(A),b);
       x1=add(x,add(multiply_c_v((-3.0/7.0),k[0]),add(multiply_c_v((2.0/7.0),k[1]),
       add(multiply_c_v((12.0/7.0),k[2]),add(multiply_c_v((-12.0/7.0),k[3]),
       multiply_c_v((8.0/7.0),k[4])))));
       A=f.dfunc(x1,1);
       # k6=f(xi+h , yi - (3/7)k1h+(2/7)k2h+(12/7)k3h - (12/7)k4h+(8/7)k5h)
       k[5]=multiply_m_v(invA(A),b);
       # yi+1 = yi + (1/90)*( 7k1 + 32k3 +12k4+32k5+7k6)h
       for j in range(n):
```

```
        x[j]=x[j]+1.0/90.0*(7.0*k[0][j]+32.0*k[2][j]+12.0*k[3][j]+32.0*k[4][j]+7.0*k[5][j]);
   return x;

f=fa();
x=[1,1,1];
x= continuationRK6(f,x,4);
print("x : \n",x);
r=newton(f,x);
print("r : \n",r);
```

```
runfile('D:/okul/SCO1/homotopy.py', wdir='D:/okul/SCO1')
Reloaded modules: fi_xi, gauss, newtond
x :
 [0.6085068630782303, 0.0859198719076823, -0.44503655626229516]
r :
 [0.5, -4.1299852640664695e-18, -0.5235987755982989]
```

In roots searching first estimate is given as a vector by the user. Another alternative way of searching non-linear roots can be achived by defining a search regoun and then finding a first estimate by random first estimate selection in the given region. This process can be repeated number of times and minimum value of summation of y=f(x) value can be considered as the best possible solution fort he given region. Note that there could be more than one solution set in the given region.

- Detemine search region $x_{i_low}, x_{i_high} \quad i = 0, N$
- Find a starting point randomly falling into search region $x_i = xx_{i_low} + Random\ number(0,1) * (x_{i_{high}} - x_{i_{low}})$
- Find solution set
- Find $\sum_{i=0}^{N} f(x)$
- Compare them with the minimum value found so far
- Repeat the process for a given number of time and select the case with minimum total $\sum_{i=0}^{N} f(x)$ value

```
import java.util.*;
import java.awt.*;
import java.applet.Applet;
import java.awt.event.*;
import javax.swing.*;
import java.lang.Math;
class fa extends fi_xi
{
public double[] func(double x[])
{
//çözümü istenen fonksiyon
double ff[]=new double[3];
ff[0]=3.0*x[0]-Math.cos(x[1]*x[2])-0.5;
ff[1]=x[0]*x[0]-81.0*(x[1]+0.1)*(x[1]+0.1)+Math.sin(x[2])+1.06;
ff[2]=Math.exp(-x[0]*x[1])+20.0*x[2]+(10.0*Math.PI-3.0)/3.0;
return ff;
} }

public class iterative_continuity
{
public int Nlimit;  //search limit
public double low[];  //search area low limits
public double high[]; //search area high limits
public int N;  //Runge Kutta 6 differential eqn solving number of division
public int Neqn;  //number of independent set
// constructor
public iterative_continuity(int nlimiti, int Ni,double lowi[],double highi[])
{low=lowi;high=highi;Nlimit=nlimiti;N=Ni; Neqn=low.length;}
public iterative_continuity(int nlimiti,double lowi[],double highi[])
{low=lowi;high=highi;Nlimit=nlimiti;N=30; Neqn=low.length;}

// (Continuity-homotopy) method to solve a system of nonlinear equations
public  double[] multiply(double left,double[] right)
{
//multiplying a vector with a constant
```

```
int i;
int n=right.length;
double b[];
b=new double[n];
for(i=0;i<n;i++)
  {b[i]=left*right[i];}
return b;
}
public  double[] multiply(double[][] left,double[] right)
{
//multiplication of one matrix with one vector
int ii,jj,i,j,k;
int m1=left[0].length;
int n1=left.length;
int m2=right.length;
double[] b;
b=new double[m2];
 if(n1 != m2)
 {
 System.out.println("inner matrix dimensions must agree");
 for(ii=0;ii<n1;ii++)
   {
    b[ii]=0;
   }
   return b;
 }
 for(i=0;i<m1;i++)
   {
    b[i]=0;
    for(k=0;k<n1;k++)
              b[i]+=left[i][k]*right[k];
   }
return b;
//end of multiply of a matrix and a vector
}
public  double[] add(double[] left,double[] right)
{
//addition of two vectors
int n1=left.length;
int n2=right.length;
int nMax;
int i;
if(n1>=n2) nMax=n1;
else       nMax=n2;
double b[];
b=new double[nMax];
for(i=0;i<n1;i++)
  {
   b[i]=b[i]+left[i];
  }
for(i=0;i<n2;i++)
  {
   b[i]=b[i]+right[i];
  }
return b;
//end of vector addition method
}
public  double[][] inv(double[][] a)
{
// INVERSION OF A MATRIX
// inversion by using gaussian elimination
// with full pivoting
int n=a.length;
int m=a[0].length;
double b[][];
b=new double[n][n];
int indxc[];
int indxr[];
double ipiv[];
indxc=new int[n];
indxr=new int[n];
ipiv=new double[n];
int i,j,k,l,ll,ii,jj;
int icol=0;
```

```
int irow=0;
double big,dum,pivinv,temp;
if(n!=m)
{ System.out.println("Matrix must be square ");
                for(ii=0;ii<n;ii++)
                  for(jj=0;jj<n;jj++)
                                b[ii][jj]=0.0;
                return b;
}
for(i=0;i<n;i++)
   for(j=0;j<n;j++)
       b[i][j]=a[i][j];
for(i=0;i<n;i++)
{
  big=0.0;
  for(j=0;j<n;j++)
  {
    if(ipiv[j] != 1)
      for(k=0;k<n;k++)
      {
                if(ipiv[k] == 0)
                {
                  if(Math.abs(b[j][k]) >= big)
                  {
                    big=Math.abs(b[j][k]);
                    irow=j;
                    icol=k;
                  }
                }
                else if(ipiv[k] > 1 )
                  {
                  System.out.println("error : inverse of the matrix : singular matrix-1");
                  for(ii=0;ii<n;ii++)
                    for(jj=0;jj<n;jj++)
                                b[ii][jj]=0.0;
                  return b;
                  }
      }
  }
  ++ ipiv[icol];
  if(irow != icol)
    for(l=0;l<n;l++)
    {
    temp=b[irow][l];
    b[irow][l]=b[icol][l];
    b[icol][l]=temp;
    }
  indxr[i]=irow;
  indxc[i]=icol;
  if(b[icol][icol] == 0.0)
  {
    System.out.println("error : inverse of the matrix : singular matrix-2");
                for(ii=0;ii<n;ii++)
                  for(jj=0;jj<n;jj++)
                                b[ii][jj]=0.0;
    return b;
  }
  pivinv=1.0/b[icol][icol];
  b[icol][icol]=1.0;
  for(l=0;l<n;l++) b[icol][l] *=pivinv;
  for(ll=0;ll<n;ll++)
  if(ll != icol)
  {
    dum=b[ll][icol];
    b[ll][icol]=0.0;
    for(l=0;l<n;l++) b[ll][l]-= b[icol][l]*dum;
  }
}
for(l=n-1;l>=0;l--)
{
  if(indxr[l] != indxc[l])
      for(k=0;k<n;k++)
      {
        temp=b[k][indxc[l]];
```

```
      b[k][indxc[l]]=b[k][indxr[l]];
      b[k][indxr[l]]=temp;
    }
}
return b;
}
public  double[] continuationRK6(fi_xi f,double x[])
{
//=================================================
// Roots of nonlinear system of equations Homotophy RK6
// yi+1 = yi + (1/90)*( 7k1 + 32k3 +12k4+32k5+7k6)h
// k1=f(xi,yi)
// k2=f(xi+0.25h , yi+0.25k1h)
// k3=f(xi+0.25h , yi+0.125k1h+0.125k2h)
// k4=f(xi+0.5h , yi - 0.5k2h+k3h)
// k5=f(xi+0.75h , yi + (3/16)k1h+(9/16)k4h)
// k6=f(xi+h , yi - (3/7)k1h+(2/7)k2h+(12/7)k3h - (12/7)k4h+(8/7)k5h)
//=================================================
//x vector of independent variables
//y vector of dependent variables
//dy derivative vector of dependent variables
int i;
int nmax=1000;
double tolerance=1.0e-15;
int n=x.length;
double h=1.0/(double)N;
double b[]=new double[n];
double x1[]=new double[n];
double k[][]=new double[6][n];
double A[][]=new double[n][n];
b=multiply(-h,f.func(x));
for(i=0;i<N;i++)
{
A=f.dfunc(x);
// k1=f(xi,yi)
k[0]=multiply(inv(A),b);
x1=add(x,multiply(0.25,k[0]));
A=f.dfunc(x1);
// k2=f(xi+0.25h , yi+0.25k1h)
k[1]=multiply(inv(A),b);
x1=add(x,add(multiply(0.125,k[0]),multiply(0.125,k[1])));
A=f.dfunc(x1);
// k3=f(xi+0.25h , yi+0.125k1h+0.125k2h)
k[2]=multiply(inv(A),b);
x1=add(x,add(multiply(-0.5,k[1]),k[2]));
A=f.dfunc(x1);
// k4=f(xi+0.5h , yi - 0.5k2h+k3h)
k[3]=multiply(inv(A),b);
x1=add(x,add(multiply((3.0/16.0),k[0]),multiply((9.0/16.0),k[3])));
A=f.dfunc(x1);
// k5=f(xi+0.75h , yi + (3/16)k1h+(9/16)k4h)
k[4]=multiply(inv(A),b);
x1=add(x,
      add(multiply((-3.0/7.0),k[0]),add(multiply((2.0/7.0),k[1]),
      add(multiply((12.0/7.0),k[2]),
         add(multiply((-12.0/7.0),k[3]),multiply((8.0/7.0),k[4])))));
A=f.dfunc(x1);
// k6=f(xi+h , yi - (3/7)k1h+(2/7)k2h+(12/7)k3h - (12/7)k4h+(8/7)k5h)
k[5]=multiply(inv(A),b);
// yi+1 = yi + (1/90)*( 7k1 + 32k3 +12k4+32k5+7k6)h
for(int j=0;j<n;j++) {x[j]=x[j]+1.0/90.0*(7.0*k[0][j]+32.0*k[2][j]+12.0*k[3][j]+32.0*k[4][j]+7.0*k[5][j]);}
}
return x;
}

public static double[] newton(fi_xi f,double x[])
{
// Roots of nonlinear system of equations Homotophy RK4 plus Newton-Raphson
//ti : weight function
//x vector of independent variables
//y vector of dependent variables
//dy derivative vector of dependent variables
double ti=1.0;
int i;
```

```
int nmax=1000;
double tolerance=1.0e-15;
int n=x.length;
double b[];
b=new double[n];
for(i=0;i<n;i++)
{
b[i]=1.0;
}
i=0;
while( i++ < nmax && Matrix.abs(b) > tolerance )
{ b=Matrix.multiply(Matrix.divide(f.func(x),f.dfunc(x)),-ti);
  x=Matrix.add(x,b);
}
if(i >= nmax) JOptionPane.showMessageDialog(null,"Warning : Maximum number of iterations are exceeded \n"+
            " Results may not be valid","MAXIMUM ITERATIONS WARNING",JOptionPane.WARNING_MESSAGE);
return x;
}
public  double[] newton_continuationRK6(fi_xi f,double x[])
{
// Roots of nonlinear system of equations Homotophy RK4 plus Newton-Raphson
//ti : weight function
//x vector of independent variables
//y vector of dependent variables
//dy derivative vector of dependent variables
x=continuationRK6(f,x);
double ti=1.0;
int i;
int nmax=1000;
double tolerance=1.0e-15;
int n=x.length;
double b[];
b=new double[n];
for(i=0;i<n;i++)
{
b[i]=1.0;
}
i=0;
while( i++ < nmax && Matrix.abs(b) > tolerance )
{ b=Matrix.multiply(Matrix.divide(f.func(x),f.dfunc(x)),-ti);
  x=Matrix.add(x,b);
}
if(i >= nmax) JOptionPane.showMessageDialog(null,"Warning : Maximum number of iterations are exceeded \n"+
            " Results may not be valid","MAXIMUM ITERATIONS WARNING",JOptionPane.WARNING_MESSAGE);
return x;
}

public static double[] enterdata()
{
   String s=JOptionPane.showInputDialog("enter initial root estimations with a space in between ");
   StringTokenizer token=new StringTokenizer(s);
   int n=token.countTokens()-1;
   int m=n+1;
   double a[]=new double[m];
   int j=0;
   while(token.hasMoreTokens())
   {a[j++]=Double.parseDouble(token.nextToken());}
   return a;
}
public double[][] findContinuity(fi_xi f)
{    double ymin=3e40;
     int nbest=0;
     double x1[]=new double[Neqn];
                double x[][]=new double[Nlimit][Neqn];
                double y[][]=new double[Nlimit][Neqn];
     for(int j=0;j<Nlimit;j++)
     {
                double yy=0;
                for(int i=0;i<Neqn;i++)
                {x1[i]=low[i]+Math.random()*(high[i]-low[i]);}
                x[j]=continuationRK6(f,x1);
                y[j]=f.func(x[j]);
                for(int i=0;i<Neqn;i++)
                {yy+=y[j][i];}
```

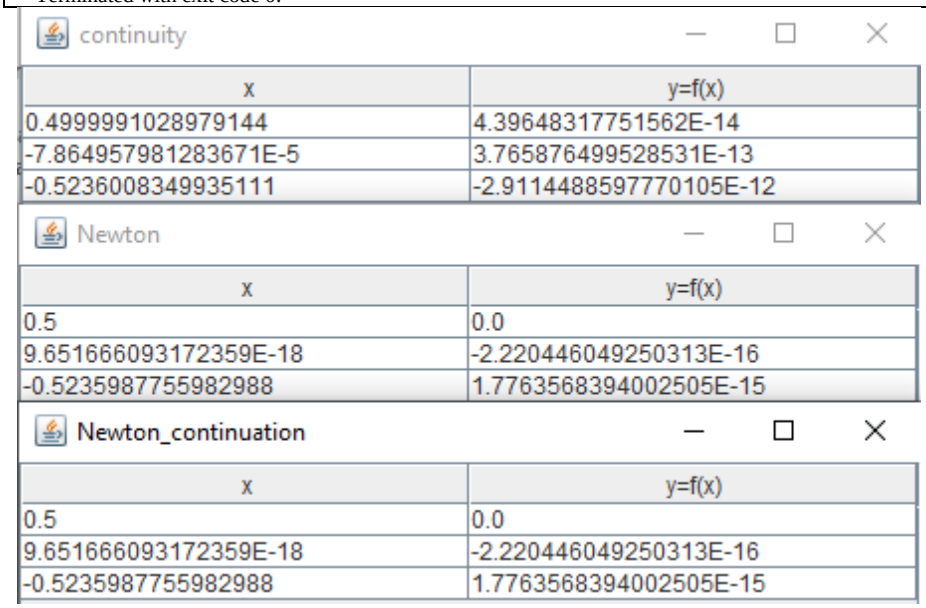
```
                System.out.println("j="+j+"yy="+yy);
                if(yy==0) {nbest=j;break;}
                if(Math.abs(yy)<ymin) {ymin=Math.abs(yy);nbest=j;}
    }
    double zz[][]={x[nbest],y[nbest]};
    return zz;
}
public double[][] findNewton(fi_xi f)
{    double ymin=3e40;
     int nbest=0;
     double x1[]=new double[Neqn];
                double x[][]=new double[Nlimit][Neqn];
                double y[][]=new double[Nlimit][Neqn];
     for(int j=0;j<Nlimit;j++)
     {
                double yy=0;
                for(int i=0;i<Neqn;i++)
                {x1[i]=low[i]+Math.random()*(high[i]-low[i]);}
                x[j]=newton(f,x1);
                y[j]=f.func(x[j]);
                for(int i=0;i<Neqn;i++)
                {yy+=y[j][i];}
                System.out.println("j="+j+"yy="+yy);
                if(yy==0) {nbest=j;break;}
     if(Math.abs(yy)<ymin) {ymin=Math.abs(yy);nbest=j;}
    }
    double zz[][]={x[nbest],y[nbest]};
    return zz;
}

public double[][] findNewtonContinuation(fi_xi f)
{    double ymin=3e40;
     int nbest=0;
     double x1[]=new double[Neqn];
                double x[][]=new double[Nlimit][Neqn];
                double y[][]=new double[Nlimit][Neqn];
     for(int j=0;j<Nlimit;j++)
     {
                double yy=0;
                for(int i=0;i<Neqn;i++)
                {x1[i]=low[i]+Math.random()*(high[i]-low[i]);}
                x[j]=newton_continuationRK6(f,x1);
                y[j]=f.func(x[j]);
                for(int i=0;i<Neqn;i++)
                {yy+=y[j][i];}
                System.out.println("j="+j+"yy="+yy);
                if(yy==0) {nbest=j;break;}
     if(Math.abs(yy)<ymin) {ymin=Math.abs(yy);nbest=j;}
    }
    double zz[][]={x[nbest],y[nbest]};
    return zz;
}
  public static void main(String arg[])
  {  fa fx=new fa();
     double low[]={0,0,0};
    double high[]={1,1,1};
    iterative_continuity itc=new iterative_continuity(30,low,high);
    double z[][]=itc.findContinuity(fx);
    String s[]={"x","y=f(x)"};
    Text.printT(z,s,"continuity");
    double z1[][]=itc.findNewton(fx);
    Text.printT(z1,s,"Newton");
    double z2[][]=itc.findNewtonContinuation(fx);
    Text.printT(z1,s,"Newton_continuation");
  }
}
```

```
---------- Capture Output ----------
> "E:\co\java\bin\javaw.exe" iterative_continuity
j=0yy=9.23383187689808E-8
j=1yy=0.0014655966934973286
j=2yy=2.7940106073209847E-7
j=3yy=8.762042850163354E-6
j=4yy=1.041124964018536E-10
j=5yy=2.4581519429967003E-5
```

j=6yy=4.661132704697479E-6
j=7yy=6.001758694473835E-7
j=8yy=2.3717842578818704E-8
j=9yy=2.4949303163435133E-5
j=10yy=8.468584794307787E-6
j=11yy=0.001434468568131675
j=12yy=1.720759090773072E-10
j=13yy=2.616555025624079E-4
j=14yy=1.0900877761266248E-4
j=15yy=1.993579393455569E-6
j=16yy=2.8659885866866297E-4
j=17yy=9.730782846162356E-9
j=18yy=5.629171173637815E-8
j=19yy=1.510731743028515E-4
j=20yy=0.0011197865506833438
j=21yy=1.289823708949811E-5
j=22yy=2.5530188223654715E-4
j=23yy=1.7026177970713707E-4
j=24yy=4.731952094161329E-5
j=25yy=9.743361971681352E-7
j=26yy=5.590070891955401E-4
j=27yy=5.725897835873184E-7
j=28yy=3.586605644041452E-4
j=29yy=9.07827877439038E-4
j=0yy=1.7763568394002505E-15
j=1yy=-1.7763568394002505E-15
j=2yy=-1.7763568394002505E-15
j=3yy=1.5543122344752192E-15
j=4yy=-1.7763568394002505E-15
j=5yy=-1.7763568394002505E-15
j=6yy=-1.7763568394002505E-15
j=7yy=1.7763568394002505E-15
j=8yy=1.5543122344752192E-15
j=9yy=1.7763568394002505E-15
j=10yy=1.7763568394002505E-15
j=11yy=1.7763568394002505E-15
j=12yy=1.7763568394002505E-15
j=13yy=1.7763568394002505E-15
j=14yy=1.7763568394002505E-15
j=15yy=-1.7763568394002505E-15
j=16yy=1.5543122344752192E-15
j=17yy=1.7763568394002505E-15
j=18yy=-1.7763568394002505E-15
j=19yy=1.5543122344752192E-15
j=20yy=-1.7763568394002505E-15
j=21yy=1.7763568394002505E-15
j=22yy=-1.7763568394002505E-15
j=23yy=1.7763568394002505E-15
j=24yy=1.7763568394002505E-15
j=25yy=-1.7763568394002505E-15
j=26yy=-1.7763568394002505E-15
j=27yy=1.5543122344752192E-15
j=28yy=1.7763568394002505E-15
j=29yy=1.7763568394002505E-15
j=0yy=1.5543122344752192E-15
j=1yy=-1.7763568394002505E-15
j=2yy=-1.7763568394002505E-15
j=3yy=1.7763568394002505E-15
j=4yy=1.7763568394002505E-15
j=5yy=-1.7763568394002505E-15
j=6yy=1.7763568394002505E-15
j=7yy=-1.7763568394002505E-15
j=8yy=-1.7763568394002505E-15
j=9yy=1.7763568394002505E-15
j=10yy=-1.7763568394002505E-15
j=11yy=1.7763568394002505E-15
j=12yy=-1.7763568394002505E-15
j=13yy=1.7763568394002505E-15
j=14yy=1.7763568394002505E-15
j=15yy=-1.7763568394002505E-15
j=16yy=-1.7763568394002505E-15
j=17yy=-1.7763568394002505E-15
j=18yy=1.9984014443252818E-15
j=19yy=1.7763568394002505E-15

```
j=20yy=-1.7763568394002505E-15
j=21yy=-1.7763568394002505E-15
j=22yy=1.7763568394002505E-15
j=23yy=1.5543122344752192E-15
j=24yy=1.7763568394002505E-15
j=25yy=-1.7763568394002505E-15
j=26yy=1.7763568394002505E-15
j=27yy=1.7763568394002505E-15
j=28yy=1.7763568394002505E-15
j=29yy=1.5543122344752192E-15
> Terminated with exit code 0.
```



4.4 JACOBİ, GAUSS-SEIDEL AND RELAXATION METHODS FOR NON-LINEAR SYSTEM OF EQUATIONS

Iterative solution methods of linear system of eqaution were investigated in the previous chapter. The same methods can also be applied to solve non-linear system of equations. There are no guarantee that these type of soutions will be converged. In this method the basic nonlinear set of eqaution first converged as

$f_j(x_i) = 0 \quad i = 0..n \quad j = 0..n$ (4.4.1) from the original form to

$x_j = g_j(x_i) \quad i = 0..n \quad j = 0..n$ or in an open form:

$x_0 = g_0(x_0, x_1, x_2, .., x_n)$

$x_1 = g_1(x_0, x_1, x_2, .., x_n)$

…..

$x_n = g_n(x_0, x_1, x_2, .., x_n)$

So out iterative set of eqaution will be

$x_0^{k+1} = (1-\alpha)x_0^k + \alpha g_0(x_0, x_1, x_2, .., x_n)$

$x_1^{k+1} = (1-\alpha)x_1^k + \alpha g_1(x_0, x_1, x_2, .., x_n)$

$x_2^{k+1} = (1-\alpha)x_2^k + \alpha g_2(x_0, x_1, x_2, .., x_n)$

…..

$x_j^{k+1} = (1-\alpha)x_j^k + \alpha g_j(x_0, x_1, x_2, .., x_n)$

.....

$x_n^{k+1} = (1-\alpha)x_n^k + \alpha g_n(x_0, x_1, x_2, .., x_n)$

The value could be between 0 and 1 or bigger than 1. Let us give an example problem as:

$3x_0 - \cos(x_1 x_2) - 0.5 = 0$

$x_0^2 - 81(x_1 + 0.01) + \sin(x_2) + 1.06 = 0$

$-\exp(-x_0 x_1) + 20x_2 + \frac{10\pi - 3}{3} = 0$

When we converged the system of equation, it will become:

$x_0 = [\cos(x_1 x_2) + 0.5]/3$

$$x_1 = \frac{[x_0^2+\sin(x_2)+1.06]}{81} + 0.01$$
$$x_2 = (-\exp(-x_0 x_1) - \frac{10\pi-3}{3})/20$$

Program list is given as follows:

Program 4.4-1 Jacobi-Gauss_seidel and relaxation method

```
 import java.io.*;
class fa extends fi_xi
{
public double[] func(double x[])
{
// çözümü istenen fonksiyon
double g[]=new double[3];
g[0]=(Math.cos(x[1]*x[2])+0.5)/3.0;
g[1]=Math.sqrt((x[0]*x[0]+Math.sin(x[2]+1.06))/81.0)-0.1;
g[2]=(-Math.exp(-x[0]*x[1])-(10.0*Math.PI-3)/3.0)/20.0;
return g;
} }
class SCO10X
{
  public static double [] gauss_seidel(fi_xi f,double x[],double alpha)
  {
  int n=x.length;
  double xx[]=new double[n];
  double xx1[]=new double[n];
  double delta=0;
  for(int i=0;i<n;i++) xx[i]=x[i];
  double g[]=new double[n];
  int iter=500;
  for(int k=0;k<iter;k++)
  {for(int i=0;i<n;i++) xx1[i]=xx[i];
   delta=0;
  for(int j=0;j<n;j++)
  {   g=f.func(xx);
                xx[j]=(1-alpha)*xx[j]+alpha*g[j];
                delta+=Math.abs(xx[j]-xx1[j]);
                System.out.print("xx["+j+"] = "+xx[j]);
  }
  System.out.println(" k = "+k+" delta = "+delta);
  if(delta<1e-15) break;
  }
  return xx;
  }
  public static void main(String args[]) throws IOException
  {
  double x[]={1.01,0.1,1.0121};
  fa ff=new fa();
  double alpha=0.1;
  Text.print(gauss_seidel(ff,x,alpha));
  }
}
```

Exit 3.7-1 Gauss_seidel rahatlamalı interpolasyon formülü

```
 ---------- Capture Output ----------
> "C:\java\bin\java.exe" SCO10X
xx[0] = 0.9588294212826883xx[1] = 0.09489183495398086xx[2] = 0.8589649657886227 k = 0 delta = 0.20941377797470812
xx[0] = 0.9128358124735219xx[1] = 0.09019866496669672xx[2] = 0.7211037819655458 k = 1 delta = 0.1885479626195275
xx[0] = 0.8714817470867342xx[1] = 0.085824605234896xx[2] = 0.5969938557799842 k = 2 delta = 0.16983805130414997
xx[0] = 0.8342898285193656xx[1] = 0.08169652937067469xx[2] = 0.4852640310069166 k = 3 delta = 0.1530498192046575
xx[0] = 0.8008346544741505xx[1] = 0.07776042319429945xx[2] = 0.3846796198127338 k = 4 delta = 0.1379756914157732
xx[0] = 0.7707362771620705xx[1] = 0.07397773129037596xx[2] = 0.2941288917316395 k = 5 delta = 0.12443179729709779
```

xx[0] = 0.743654758864458xx[1] = 0.07032203418833874xx[2] = 0.2126108822145333 k = 6 delta = 0.11225522491675591
xx[0] = 0.7192855573929079xx[1] = 0.06677618815104167xx[2] = 0.1392243959352476 k = 7 delta = 0.10130153378813286
xx[0] = 0.6973555611327479xx[1] = 0.06332995056219265xx[2] = 0.07315809120795819 k = 8 delta = 0.09144253857629844
xx[0] = 0.6776196472601156xx[1] = 0.059978059978701904xx[2] = 0.013681541973005897 k = 9 delta = 0.08256435369107533
xx[0] = 0.6598576713112414xx[1] = 0.05671871668388067xx[2] = -0.039862816917781055 k = 10 delta = 0.07456567813448237
…….
xx[0] = 0.4999995683317302xx[1] = -0.003073182441921277xx[2] = -0.5236756641509748 k = 388 delta = 8.673617379884035E-19
xx[0] = 0.4999995683317302xx[1] = -0.003073182441921278xx[2] = -0.5236756641509748 k = 389 delta = 8.673617379884035E-19
xx[0] = 0.4999995683317302xx[1] = -0.003073182441921279xx[2] = -0.5236756641509748 k = 390 delta = 8.673617379884035E-19
xx[0] = 0.4999995683317302xx[1] = -0.0030731824419212798xx[2] = -0.5236756641509748 k = 391 delta = 8.673617379884035E-19
xx[0] = 0.4999995683317302xx[1] = -0.0030731824419212806xx[2] = -0.5236756641509748 k = 392 delta = 8.673617379884035E-19
xx[0] = 0.4999995683317302xx[1] = -0.0030731824419212815xx[2] = -0.5236756641509748 k = 393 delta = 8.673617379884035E-19
xx[0] = 0.4999995683317302xx[1] = -0.0030731824419212824xx[2] = -0.5236756641509748 k = 394 delta = 8.673617379884035E-19
xx[0] = 0.4999995683317302xx[1] = -0.0030731824419212824xx[2] = -0.5236756641509748 k = 395 delta = 0.0

4.5 PROBLEMS

PROBLEM 1

Solve the following system of non-linear equations

$4x_1^2-20x_1+0.25x_2^2+8=0$

$0.5x_1x_2^2+2x_1+8=0$

By using

a) Newton raphson method

b) Broyden sekant method

c) Continuity(Homotopy) method

PROBLEM 2

Solve the following system of non-linear equations

$3x_1^2-\cos(x_2x_2) - 0.5=0$

$4x_1^2-6.25x_2^2+2x_3-1=0$

$e^{x1\ x2} +20x_3+(10\pi-3)/3=0$

By using

a) Newton raphson method

b) Broyden sekant method

c) Continuity(Homotopy) method

PROBLEM 3

Solve te following system of non-linear equations

$x_1(1- x_1) + 4x_3 – 12 =0$

$(x_1-2)^2+(2x_2-3)^2-25 =0$

By using

a) Newton raphson method

b) Broyden sekant method

c) Continuity(Homotopy) method

PROBLEM 4

Solve the following system of non-linear equations

$\sin(4\pi x_1x_2)-2x_2-x_1=0$

$\left(\frac{4\pi-1}{4\pi}\right)(e^{2x_1}-e)+4ex_2^2-2ex_1=0$

By using

a) Newton raphson method

b) Broyden sekant method

c) Continuity(Homotopy) method

PROBLEM 5

Solve the following system of non-linear equations

$10x_1-2x_2^2+x_2-2x_3-5=0$

$8x_2^2+4x_3^2-9=0$

$8x_2x_3+4=0$

By using

a) Newton raphson method

b) Broyden sekant method

c) Continuity(Homotopy) method

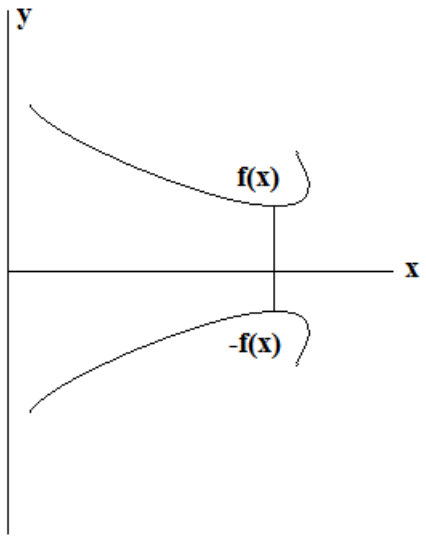
5 OPTIMIZATION

51 DEFINITION OF OPTIMIZATION AND GRAPHIC METHODS

Optimization is the name given to maximum or minimum finding process. When it is considered in mathematically, the root of the derivative of the function defines the optimum point.

$$\frac{df(x)}{dx} = 0$$

As the mathematical process, maximum and minimum is similar processes because
maximum (f(x)) = minimum(-f(x))



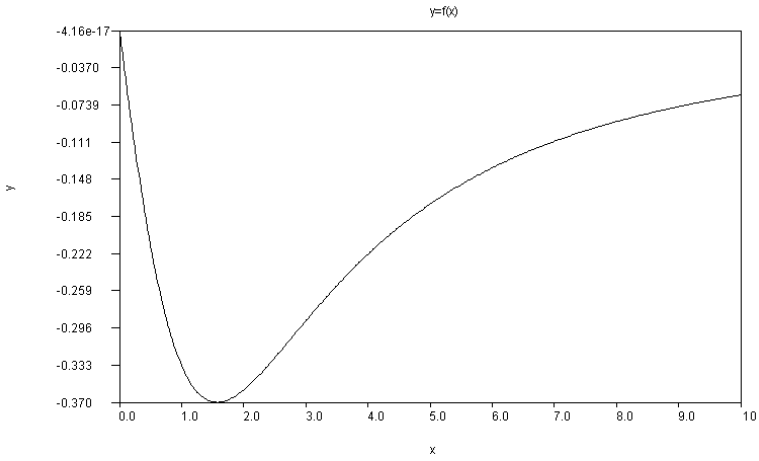
As a first example to one dimensional non-linear optimization, Plotting will be utilised. Mathematically plotting means to subdivide the region into n equal parts and investigate the region of the minimum.

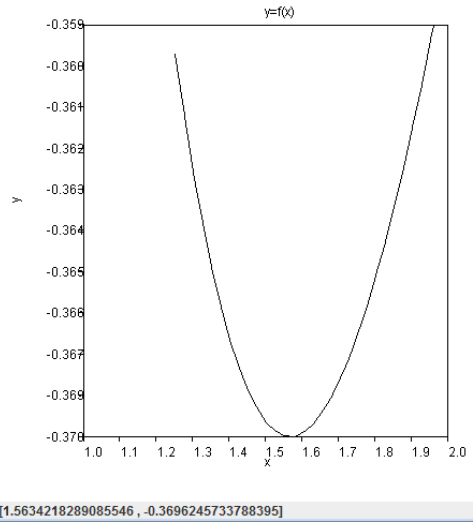
PROGRAM 5.2-1 PlotT02 class as using plots in optimization example

```
import java.io.*;

class fa1 extends f_x
{
  double func (double c)
  {return -(2.0*c/(4.0+0.8*c+c*c+0.2*c*c*c));}
}

class PlotT02
{
    public static void main (String args[]) throws IOException
    {
    fa1 ff1=new fa1();
    Plot pp=new Plot(ff1,0.0,10.0,400,0);
    pp.plot();
    }
}
```





ove [1.5634218289085546 , -0.3696245733788395]

As it is seen from the plot windows, a guess of 1.5634 can be obtained for the optimum point.

5.2 ONE DIMENSIONAL NON-LINEAR FIBONNACHI SEARCH (GOLDEN SEARCH) METHOD

Series 0,1,1,2,3,5,8,13,21,34,… is called fibonnachi series. General definition is in the form of

F(n)=F(n-1)+F(n-2) F(0)=0, F(1)=1

For the big fibonnaci terms the ration of two following fibonnachi number becomes constant. This constant is known as golden ratio,

$$R = \lim_{n\to\infty} \frac{F(n-1)}{F(n)} = \frac{\sqrt{5}-1}{2} = 0.618033989$$

If a minimum being searched in a given region (a,c). In order to determine if a minimum is existed, region should be divided up into three sub-regions. This requires avaluation of two intermediate data points. Neigbors of the minimum point remains and the third region is eliminated. In the new avaluation a new two points will be required. Golden ratio is given us a fine propery so that one of the selected points will always be one of the new search point. So that only one new point will be evaluated in each step of the evaluation.

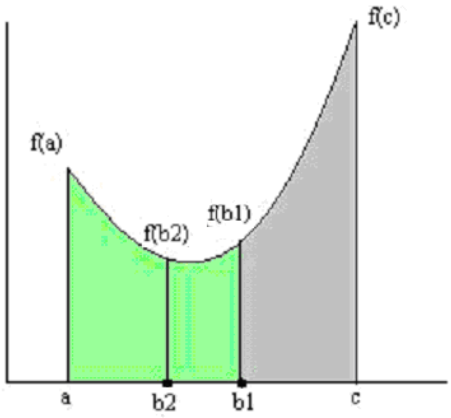


Figure 5.3-1 Golden ratio search

If minimumin in region [a,c] is required
The length of the region will be d0=(c – a) .
Then b1 is taken as b1=a+d0*R
And b2 is taken as b2 = c – d0*R = a + d0*R^2
And if it is assumed that minimum is in point b2, region (b1-c) will be thrown. Our new region will be d1=d0*R olacağindan
new b1 is taken as b1=a+d1*R=a+d0*R^2 which is the same as previous b2.
New b2 is taken n as b2 = c – d1*R = a + d1*R^2 = a + d0*R^3 .

Since one of the point is the same, evaluation of only one new point is required for each iteration step. The Algorithm of Golden search is given in Program 5.3-1

PROGRAM 5.2-1 Golden Ratio or Fibonnachi search

```
/ OPTIMIZATION Golden search or fibonnachi method
import java.util.*;
import java.awt.*;
import java.applet.Applet;
import java.awt.event.*;
import javax.swing.*;
class fb extends f_x
{
  public double func (double c)
  { return -(2.0*c/(4.0+0.8*c+c*c+0.2*c*c*c));}
}

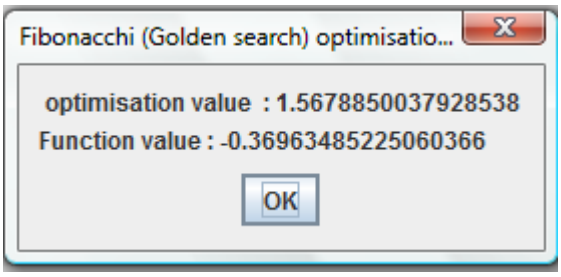
public class NA31
{
public static double golden(f_x f,double a,double b,double epsilon)
{return golden(f,a,b,epsilon,1.0e-10,0);}
public static double golden(f_x f,double a,double b)
{return golden(f,a,b,1.0e-10,1.0e-10,0);}

public static double golden(f_x f,double a,double b,double epsilon,double delta,int print)
{
//  find the minimum of the function
// note  maximum f(x) = minimum (-f(x))
```

```
    double r1 = (Math.sqrt(5.0)-1.0)/2.0; // golden ratio
    double r2 = r1*r1;
    double h = b - a;
    double ya = f.func(a);
    double yb = f.func(b);
    double c = a + r2*h;
    double d = a + r1*h;
    double yc = f.func(c);
    double yd = f.func(d);
    int k = 1;
    double dp,dy,p,yp;
    while ((Math.abs(yb-ya)>epsilon) || (h>delta))
    {
     k++;
     if (yc<yd)
       {
       b = d;
       yb = yd;
       d = c;
       yd = yc;
       h = b - a;
       c = a + r2 * h;
       yc = f.func(c);
       }
     else
      {
        a = c;
        ya = yc;
        c = d;
        yc = yd;
        h = b - a;
        d = a + r1 * h;
        yd = f.func(d);
      }//end of if
    }//end of while
    dp = Math.abs(b-a);
    dy = Math.abs(yb-ya);
    p = a;
    yp = ya;
    if (yb<ya)
    {
     p = b;
     yp = yb;
    }
    if(print==1)
    {System.out.println("x min = "+p+"ymin = "+yp+"error of x ="+dp+"error of y"+dy); }
    return p;
}

     public static void main (String args[])
     {
```

```
        fb f=new fb();
    double a,b;
    a=Double.parseDouble(JOptionPane.showInputDialog(" enter lower limit of the function a : "));
    b=Double.parseDouble(JOptionPane.showInputDialog(" enter upper limit of the function b : "));
    double r;
    System.out.println("x=0.5"+f.func(0.5));
    System.out.println("x=1.5"+f.func(1.5));
    r= golden(f,a,b,1.0e-10,1.0e-5,0);
    JOptionPane.showMessageDialog(null," optimization value  : "+r+"\nFunction value : "+f.func(r),
    "Fibonacchi (Golden search) optimization : ",JOptionPane.PLAIN_MESSAGE);
    System.exit(0);
    }
}
```



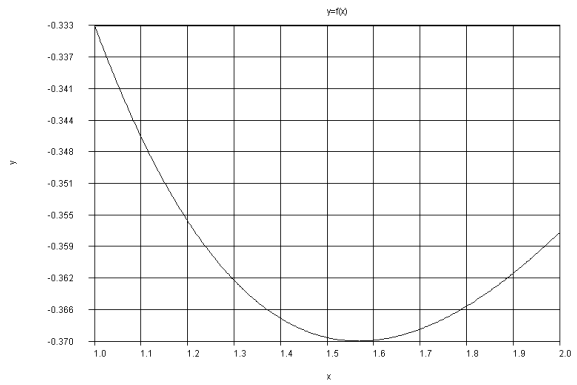
If search method is investigated step by step :

PROGRAM 5.2-1A Golden Ratio or Fibonnachi search by excel

Function f(c)=-(2.0*c/(4.0+0.8*c+c*c+0.2*c*c*c))

a	b	h	c	d	vc	vd
1	2	1	1.381966	1.618034	-0.366410476	-0.369427868
1.381966011	2	0.618034	1.618034	1.763932	-0.369427868	-0.366711114
1.381966011	1.763932	0.381966	1.527864	1.618034	-0.36949646	-0.369427868
1.381966011	1.618034	0.236068	1.472136	1.527864	-0.368819066	-0.36949646
1.472135955	1.618034	0.145898	1.527864	1.562306	-0.36949646	-0.369632208
1.527864045	1.618034	0.09017	1.562306	1.583592	-0.369632208	-0.369614176
1.527864045	1.583592	0.055728	1.54915	1.562306	-0.36960486	-0.369632208
1.549150281	1.583592	0.034442	1.562306	1.570437	-0.369632208	-0.369634304
1.562305899	1.583592	0.021286	1.570437	1.575462	-0.369634304	-0.369630023
1.562305899	1.575462	0.013156	1.567331	1.570437	-0.369634826	-0.369634304
1.562305899	1.570437	0.008131	1.565412	1.567331	-0.369634332	-0.369634826
1.565411519	1.570437	0.005025	1.567331	1.568517	-0.369634826	-0.369634819
1.565411519	1.568517	0.003106	1.566598	1.567331	-0.369634711	-0.369634826
1.56659776	1.568517	0.001919	1.567331	1.567784	-0.369634826	-0.369634851
1.567330897	1.568517	0.001186	1.567784	1.568064	-0.369634851	-0.36963485
1.567330897	1.568064	0.000733	1.567611	1.567784	-0.369634846	-0.369634851

Graphic exit of the function



Matlab (octave) version of the same program

PROGRAM 5.2-1B Golden Ratio or Fibonnachi search by Octave(matlab)

```
function p=golden(f,a,b)
% golden search
r1=(sqrt(5)-1)/2;
r2=r1*r1;
h=b-a;
ya=f(a);
yb=f(b);
c=a+r2*h;
d=a+r1*h;
yc=f(c);
yd=f(d);
k=1;
epsilon=1.0e-6;
delta=1.0e-6;
while (abs(yb-ya) > epsilon & (h>delta))
  k++;
  if yc<yd
     b = d;
      yb = yd;
      d = c;
      yd = yc;
      h = b - a;
      c = a + r2 * h;
      yc = f(c);
   else
       a = c;
       ya = yc;
       c = d;
       yc = yd;
       h = b - a;
       d = a + r1 * h;
       yd = f(d);
   end
end
p=a;
yp=ya;
if yb<ya
  p=b;
  yp=yb;
end
endfunction
```

```
>> y=@(c)-(2.0*c/(4.0+0.8*c+c*c+0.2*c*c*c));
>> golden(y,1,4)
ans =  1.56541151876420
>>
```

Pyhthon version of the same program:

PROGRAM 5.2-1C Golden Ratio or Fibonnachi search by Python language

```
 # Golden search optimisation method
# python version
```

```
from math import *
from math import *
from f_x import *;

class f1(f_x):
  def func(self,x):
      y=x*x-2*x+5;
      return y;

def golden(f,a,b):
  # golden search
  r1=(sqrt(5)-1)/2;
  r2=r1*r1;
  h=b-a;
  ya=f.func(a);
  yb=f.func(b);
  c=a+r2*h;
  d=a+r1*h;
  yc=f.func(c);
  yd=f.func(d);
  k=1;
  epsilon=1.0e-10;
  delta=1.0e-6;
  while abs(yb-ya) > epsilon:
    k=k+1;
    if yc<yd:
       b = d;
       yb = yd;
       d = c;
       yd = yc;
       h = b - a;
       c = a + r2 * h;
       yc = f.func(c);
    else:
       a = c;
       ya = yc;
       c = d;
       yc = yd;
       h = b - a;
       d = a + r1 * h;
       yd = f1(d);
  p=a;
  yp=ya;
  if yb<ya:
     p=b;
     yp=yb;
  return p;

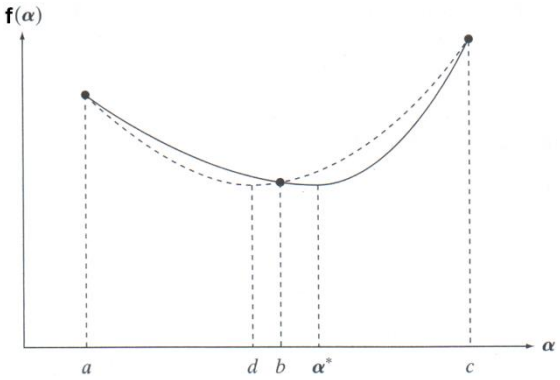
a=float(input("enter lower limit : "));
b=float(input("enter upper limit : "));
f=f1();
r=golden(f,a,b)
print("golden search optimisation  : ",r)
```

```
---------- Capture Output ----------
> "D:\co\python\pythonw.exe" golden.py
enter lower limit : 0
enter upper limit : 1
golden search optimisation  :  1.0

> Terminated with exit code 0.
```

5.3 ONE DIMENSIONAL NON-LINEAR QUADRATIC POLYNOMIAL METHOD

Golden search is a linear search method, it will always Works if a minimum(Maximum) is existed in the given region. But convergence speed is relatively slow. A quadratic search should theoretiacally be given values closer to the actual optimum value.



Minimum value of a quadratic equation is given with the Formula

$$d = b + 0.5\frac{(c-b)^2[f(a)-f(b)]-(a-b)^2[f(c)-f(a)]}{(c-b)[f(a)-f(b)]-(a-b)[f(c)-f(a)]}$$

After finding point d, point b is replaced with this point and iteration continues.

Program 5.3-1 Quadratic polynomial search

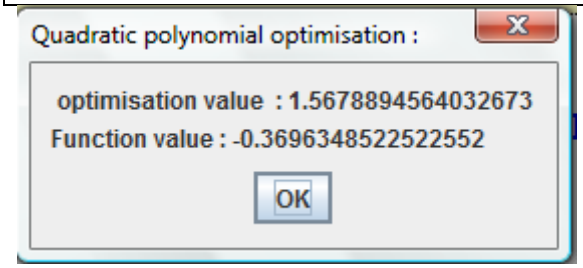
```
// OPTIMIZATION
//  2nd degree polynomial (Quadratic polynomial) optimization
//  of function f(x)
import java.util.*;
import java.awt.*;
import java.applet.Applet;
import java.awt.event.*;
import javax.swing.*;

class fb extends f_x
{
  public double func (double c)
  { return -(2.0*c/(4.0+0.8*c+c*c+0.2*c*c*c));}
}

public class NA32
{
public static double polinom2derece(f_x f,double x0,double x2)
{double x1=(x0+x2)/2.0;
 return polinom2derece(f,x0,x1,x2);
}
public static double polinom2derece(f_x f,double x0,double x1,double x2)
{
double epsilon=1.0e-10;
double delta=1.0e-5;
int print=0;
return polinom2derece(f,x0,x1,x2,epsilon,delta,print);
}
public static double polinom2derece(f_x f,double x0,double x1,double x2,double epsilon,double delta,int print)
```

```
{
int maxit=100;
double f0 = f.func(x0);
   double f1 = f.func(x1);
   double f2 = f.func(x2);
   double f3 = f2;
   double x3 = 0;
   double h = x1 - x0;
   double k=1;
   double dd=Math.abs(f1-f0);
   while ((dd >epsilon) || (h>delta))
   {
    k++;
    x3=(f0*(x1*x1-x2*x2)+f1*(x2*x2-x0*x0)+f2*(x0*x0-x1*x1))/
          (2*f0*(x1-x2)+2.0*f1*(x2-x0)+2.0*f2*(x0-x1));
    f3=f.func(x3);
    if(x3 >= x0 && x3<x1)
      {x2=x1;f2=f1;x1=x3;f1=f3;}
    else if(x3 >= x1 && x3 <x2)
      {x0=x1;f0=f1;x1=x3;f1=f3; }
    else if(x3 >x2)
      {x0=x1;f0=f1;x1=x2;f1=f2;x2=x3;f2=f3; }
    else if(x3 < x0)
      {x0=x3;f0=f3;x1=x0;f1=f0;x2=x1;f2=f1; }
    dd=Math.abs(f1-f0);
    h=Math.abs(x1-x0);
    if(k>maxit) break;
   }//end of while
   if(print==1)
   {System.out.println("x  = "+x3+"f = "+f3+"hata x ="+h+"hata f(x)=y "+dd); }
   return x3;
}

    public static void main (String args[])
    {
            fb f=new fb();
    double x0,x1,x2;
    x0=Double.parseDouble(JOptionPane.showInputDialog(" enter lower limit of the function  x0 : "));
    x1=Double.parseDouble(JOptionPane.showInputDialog(" enter middle value of the function x1 : "));
    x2=Double.parseDouble(JOptionPane.showInputDialog(" enter upper limit of the function  x2 : "));
    double r;
    r= polinom2derece(f,x0,x1,x2,1.0e-10,1.0e-5,1);
   JOptionPane.showMessageDialog(null," optimization value  : "+r+"\nFunction value : "+f.func(r),
    "Quadratic polynomial optimization : ",JOptionPane.PLAIN_MESSAGE);
    System.exit(0);
    }
}
```



5.4 ONE DIMENSIONAL NON-LINEAR CUBIC POLYNOMIAL METHOD

If it is assumed that our polynomial is a cubic polynomial, a method similar to the quadratic polynomial method can be obtained. Hermite cubic polinomial is used for this purpose.

$P(x) = a(x-x_0)^3 + b(x-x_0)^2 + c(x-x_0) + d$
optimum(minimum , maximum or extremum) point of this polynomial in the $[x_0,x_1]$ region wil be calculated from formula

$$x - x_0 = \frac{-b \mp \sqrt{b^2 - 4ac}}{3a}$$

$$x^* = \mp \frac{-b + \sqrt{b^2 - 4ac}}{3a}$$

a b c d coefficients of this equation is unknown. This coefficients can be calculated from:
$p(x_0)=d=f(x_0)$ $p'(x_0)=c=f'(x_0)$
$a=(G-2H)/h$ $h=x_1-x_0$ $b=3H-G$
$F=[f(x_1) - f(x_0)]/h$
$G==[f'(x_1) - f'(x_0)]/h$
$H=[F - f'(x_0)]/h$
This type of formulation is not well suited for hand calculation, but is often highly efficient on the computer.

PROGRAM 5.4-1 Cubic polynomial search

```
//  ======================== //
//  Cubic polynomial optimization    //
//  ======================== //
import java.util.*;
import java.awt.*;
import java.applet.Applet;
import java.awt.event.*;
import javax.swing.*;

//============= Tanimi gereken fonksiyon =================

class fb extends f_x
{
   public double func (double x)
   { return x+25.0/(x*x);}
}
//============================================================
public class NA33
{
public static double df(f_x f,double x,double h)
{ //numerical derivative
  double hh=1.0/h;
  return (-f.func(x+2.0*h)+8.0*f.func(x+h)-8.0*f.func(x-h)+f.func(x-2.0*h))/12.0*hh;
}
public static double cubic_search_opt(f_x f,double x0,double x1)
{
int nmax=500;
double tolerance=1.0e-15;
double f0,f1,df0,df1,fxs,dfxs;
double h=0.00001;
double xs;
int k=0;
double X,F,G,H,a,b,c;
  f0=f.func(x0);
```

```
  f1=f.func(x1);
  df0=df(f,x0,h);
  df1=df(f,x1,h);
while(Math.abs(x0-x1)>tolerance && k<nmax)
{ k++;
  X=1.0/(x1-x0);
  F=X*(f1-f0);
  G=X*(df1-df0);
  H=X*(F-df0);
  a=X*(G-2.0*H);
  b=3.0*H-G;
  c=df0;
  xs=x0+(Math.sqrt(b*b-3.0*a*c)-b)/(3.0*a);
  fxs=f.func(xs);
  dfxs=df(f,xs,h);
  if(dfxs > 0 ) {x1=xs;f1=fxs;df1=dfxs;}
  else         x0=xs;f0=fxs;df0=dfxs;
}
if(k>=nmax)  JOptionPane.showMessageDialog(null,"Uyari maximum iterasyon sayisi aşildi \n"+
   " çözüm geçerli olmiyabilir","MAKSİMUM ITERASYON SAYISI UYARISI",JOptionPane.WARNING_MESSAGE);
return (x0+x1)/2.0;
}
    public static double[] inputdata(String s)
    {
    String s1=JOptionPane.showInputDialog(s);
    StringTokenizer token=new StringTokenizer(s1);
    int n=token.countTokens()-1;
    int m=n+1;
    double a[]=new double[m];
    int j=0;
    while(token.hasMoreTokens())
    {
    Double ax=new Double(token.nextToken());
    a[j++]=ax.doubleValue();
    }
    return a;
    }

    public static void main (String args[])
    {
            fb f=new fb();
    double x[]=inputdata("enter search area limits : xo and x1");
    double r= cubic_search_opt(f,x[0],x[1]);
    JOptionPane.showMessageDialog(null," minimum value  : "+r+"\nFunction value : "+f.func(r),
    "minimisation by using cubic search method : ",JOptionPane.PLAIN_MESSAGE);
    System.exit(0);
    }
}
```

5.5 ONE DIMENSIONAL NON-LINEAR NEWTON-RAPHSON METHOD

Newton- Raphson is investigated as one of the root finding methods. Due to the fact that roots of the derivatives of a function is also given the optimum point makes Newton method a cendidate to find optimum through the roots of the derivative of the function.
Minimum of f(x)

Roots of $g(x) = \frac{df(x)}{dx} = 0$

Newton raphson formula

$$x_{n+1} = x_n - \frac{g(x)}{\frac{dg(x)}{dx}} = x_n - \frac{\frac{df(x)}{dx}}{\frac{d^2f(x)}{dx^2}}$$

Program 5.5-1 Newton-Raphson method

```
/// OPTIMIZATION
//  Newton-Raphson method
import java.util.*;
import java.awt.*;
import java.applet.Applet;
import java.awt.event.*;
import javax.swing.*;

class fc
{
  public double func(double x)
  { return -(2.0*Math.sin(x)-x*x/10.0);}
}

class fb extends f_x
{
  public double func(double x)
  { return -(2.0*Math.cos(x)-x/5.0);}
}
public class NA34
{
public static double derivative(f_x df,double x)
{
// derivative sub-program
// calculates derivative of function df
double h0=0.256;
int i,m;
int n=7;
//derivative of a simple function
double T[][];
T=new double[n][n];
double h[];
h=new double[n];
//vector<double> h(n,0);
for(i=0;i<n;i++)
{
 h[i]=0;
 for(int j=0;j<n;j++)
   T[i][j]=0;
}
h[0]=h0;
double r=0.5;
for( i=1;i<n;i++)
{
h[i]=h0*Math.pow(r,i);
}
```

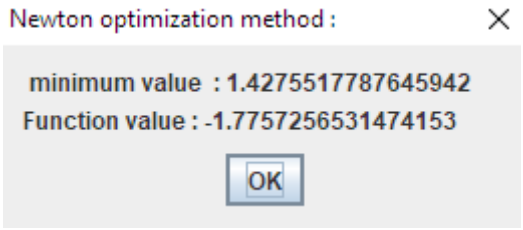
```
for(i=0;i<n;i++)
{
T[i][0]=( df.func(x + h[i]) - df.func( x - h[i]))/(2.0*h[i]);
}
for(m=1;m<n;m++)
{
  for(i=0;i<n-m;i++)
  {
  T[i][m]=(h[i]*h[i]*T[i+1][m-1] - h[i+m]*h[i+m]*T[i][m-1])/
         (h[i]*h[i]- h[i+m]*h[i+m]);
  }
}
double xx=T[0][n-1];
return xx;
}

public static double newton(f_x df,double x)
{
// root finding newton-raphson program

int nmax=500;
double tolerance=1.0e-15;
double fx,dfx;
for(int i=0;i<nmax;i++)
{
fx=df.func(x);
dfx=derivative(df,x);
x-=fx/dfx;
if(Math.abs(fx)<tolerance)
{
return x;
}
}
return x;
}

public static double newton_2nd_derivative(f_x df,double x)
{
// Newton-Raphson with second derivative
int nmax=500;
double dx=1e-3;
double x1m;
double tolerance=1.0e-15;
double fx,fxm,dfx,dfxm,d2fx;
for(int i=0;i<nmax;i++)
{
fx=df.func(x);
fxm=df.func(x-dx);
dfx=derivative(df,x);
dfxm=derivative(df,(x-dx));
d2fx=-6.0/dx/dx*(fx-fxm)+2.0/dx*(2.0*dfx+dfxm);
x-=(fx/dfx+.5*fx*fx/(dfx*dfx*dfx)*d2fx);
if(Math.abs(fx)<tolerance)
{ return x; }
}
return x;
```

```
}
    public static void main (String args[])
    {
            fc  f=new fc();
            fb df=new fb();
    double x0;
    x0=Double.parseDouble(JOptionPane.showInputDialog(" enter guess value  x0 : "));
    double r;
    r= newton(df,x0);
    JOptionPane.showMessageDialog(null," minimum value  : "+r+"\nFunction value : "+f.func(r),
    "Newton optimization method : ",JOptionPane.PLAIN_MESSAGE);
    r= newton_2nd_derivative(df,x0);
    JOptionPane.showMessageDialog(null," minimum value  : "+r+"\nFunction value : "+f.func(r),
    "Newton with second derivative optimization method : ",JOptionPane.PLAIN_MESSAGE);
    System.exit(0);
    }
}
```



Program 5.5 -2 Newton-Raphson method (Python code)

```
from math import *
from f_x import *;

class f1(f_x):
  def func(self,x):
     y=-(2.0*sin(x)-x*x/10.0);
     return y;

def newton(f,x):
   # Newton-Raphson optimisation method
  nmax=100
  tolerance=1.0e-10
  for i in range(0,nmax):
     fx= f.dfunc(x,1);
     dfx=f.dfunc(x,2);
     x=x-fx/dfx
     if abs(fx)<tolerance: return x
  return x

f=f1();
x=float(input("enter initial guess : "))
r=newton(f,x);
print(" ROOT  : ",r,"\nFUNCTION VALUE : ",f.func(r));
```

```
---------- Capture Output ----------
> "D:\co\anaconda\pythonw.exe" newton_opt.py
enter initial guess : 1
 ROOT  :  1.4275517787644865
FUNCTION VALUE :  -1.775725653147415

> Terminated with exit code 0.
```

Newton-Raphson(secant) search by excel

Function: f(c)=-(2.0*c/(4.0+0.8*c+c*c+0.2*c*c*c))

secant							
$\Delta x =$	0.001					$\frac{df(x)}{dx}$	$\frac{d^2f(x)}{dx^2}$
x	$x+\Delta x$	$x-\Delta x$	$f(x)$	$f(x+\Delta x)$	$f(x+\Delta x)$		
1	1.001	0.999	-0.33333	-0.33348	-0.33319	-0.1444445	0.341481
1.422994	1.423994	1.421994	-0.36772	-0.36775	-0.36769	-0.0274624	0.210648
1.553364	1.554364	1.552364	-0.36962	-0.36962	-0.36961	-0.0024853	0.173085
1.567723	1.568723	1.566723	-0.36963	-0.36963	-0.36963	-2.82E-05	0.169166
1.56789	1.56889	1.56689	-0.36963	-0.36963	-0.36963	-3.766E-09	0.16912
1.56789	1.56889	1.56689	-0.36963	-0.36963	-0.36963	2.7756E-14	0.16912
1.56789	1.56889	1.56689	-0.36963	-0.36963	-0.36963	0	0.16912
1.56789	1.56889	1.56689	-0.36963	-0.36963	-0.36963	0	0.16912
1.56789	1.56889	1.56689	-0.36963	-0.36963	-0.36963	0	0.16912
1.56789	1.56889	1.56689	-0.36963	-0.36963	-0.36963	0	0.16912
1.56789	1.56889	1.56689	-0.36963	-0.36963	-0.36963	0	0.16912

5.6 ONE DIMENSIONAL NON-LINEAR BRENT METHOD

It is seen that quadratic formula can used to find with formula

$$x = b + 0.5\frac{(b-a)^2[f(b)-f(a)] - (b-c)^2[f(b)-f(a)]}{(b-a)[f(b)-f(a)] - (b-c)[f(b)-f(c)]}$$

But this Formula can be reached to maximum instead of minimum as well. One more secure approximation could be to use Golden search to approach to the minimum point and then utilise the quadratic polynomial method. This combined method is called Brent method.

Program 5.6-1 Brent method

```
/// OPTIMIZATION
//  one dimensional non-linear functions
//  Brent method
//
import java.util.*;
import java.awt.*;
import java.applet.Applet;
import java.awt.event.*;
import javax.swing.*;

//============= Function to be defined ================
class fb extends f_x
{
  public double func (double x)
  { return -(2.0*Math.sin(x)-x*x/10.0);}
    //- is to obtain maximum value
}
//============================================================

public class NA35
{
    public static double[] inputdata(String s)
    {
    String s1=JOptionPane.showInputDialog(s);
    StringTokenizer token=new StringTokenizer(s1);
    int n=token.countTokens()-1;
    int m=n+1;
    double a[]=new double[m];
```

```
    int j=0;
    while(token.hasMoreTokens())
    {
    Double ax=new Double(token.nextToken());
    a[j++]=ax.doubleValue();
    }
    return a;
    }

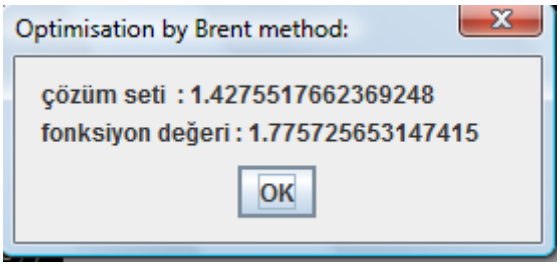
public static double SIGN(double a,double b) {return (b > 0.0 ? Math.abs(a) : -Math.abs(a));}
public static double MAX(double a,double b) {return (a > b ? a : b);}

public  static double brent(f_x f,double ax,double bx,double cx,double tol)
{
 double aa[]=brentf(f,ax,bx,cx,tol);
 return aa[0];
}

public  static double[] brentf(f_x f,double ax,double bx,double cx,double tol)
{
// ax,bx,cx is three initial guesses (bx should be located between ax and cx)
// f : function (defined with f_x abstract class)
// tol tolerance
int ITMAX=100;
double CGOLD=(3.0-Math.sqrt(5))/2.0; //golden ratio
double ZEPS=1.0e-10;
double xmin;
double aa[]=new double[2];
            int iter;
            double a,b,d,etemp,fu,fv,fw,fx,p,q,r,tol1,tol2,u,v,w,x,xm;
            double e=0.0;
            d=0.0;
            a=((ax < cx) ? ax : cx);
            b=((ax > cx) ? ax : cx);
            x=w=v=bx;
            fw=fv=fx=f.func(x);
            for (iter=1;iter<=ITMAX;iter++) {
                        xm=0.5*(a+b);
                        tol2=2.0*(tol1=tol*Math.abs(x)+ZEPS);
                        if (Math.abs(x-xm) <= (tol2-0.5*(b-a))) {
                                    xmin=x;
                  aa[0]=xmin;
                  aa[1]=fx;
                  return aa;
                        }
                        if (Math.abs(e) > tol1) {
                                    r=(x-w)*(fx-fv);
                                    q=(x-v)*(fx-fw);
                                    p=(x-v)*q-(x-w)*r;
                                    q=2.0*(q-r);
                                    if (q > 0.0) p = -p;
                                    q=Math.abs(q);
                                    etemp=e;
                                    e=d;
                                    if (Math.abs(p) >= Math.abs(0.5*q*etemp) || p <= q*(a-x) || p >= q*(b-x))
                                                d=CGOLD*(e=(x >= xm ? a-x : b-x));
                                    else {
```

```
                                        d=p/q;
                                        u=x+d;
                                        if (u-a < tol2 || b-u < tol2)
                                                d=SIGN(tol1,xm-x);
                                }
                        } else {
                                d=CGOLD*(e=(x >= xm ? a-x : b-x));
                        }
                        u=(Math.abs(d) >= tol1 ? x+d : x+SIGN(tol1,d));
                        fu=f.func(u);
                        if (fu <= fx) {
                                if (u >= x) a=x; else b=x;
                                {v=w;w=x;x=u;}
                                {fv=fw;fw=fx;fx=fu;}
                        } else {
                                if (u < x) a=u; else b=u;
                                if (fu <= fw || w == x) {
                                        v=w;
                                        w=u;
                                        fv=fw;
                                        fw=fu;
                                } else if (fu <= fv || v == x || v == w) {
                                        v=u;
                                        fv=fu;
                                }
                        }
                }
                System.out.println("maximum number of iterations are exceeded in BRENT optimization");
                xmin=x;
                aa[0]=xmin;
                aa[1]=fx;
                return aa;
}

    public static void main (String args[])
    {
             String s="enter three initial guess values with a space in between xo x1 x2: ";
    double [] x0=inputdata(s);
    fb f=new fb();
    double r1= brent(f,x0[0],x0[1],x0[2],1.0e-8);
    s=" çözüm seti  : "+r1+"\n fonksiyon değeri : "+(-f.func(r1));
    //fonksiyonun gerçek değerini elde etmek için -1 ile çarpilmiştir
    String s2="Optimization by Brent method:";
    JOptionPane.showMessageDialog(null,s,s2,JOptionPane.PLAIN_MESSAGE);
    Plot pp=new Plot(f,0,2,100);
    pp.plot();
    }
}
```



A second version of Brent method can also consider derivatives of the function to determine movement of the function better is given as the next program.

Program 5.6-2 dbrent method (brent method with derivative)

```
//==============================
//  OPTIMIZATION
//  One dimensional optimization
//  brent method with derivative
//==============================
import java.util.*;
import java.awt.*;
import java.applet.Applet;
import java.awt.event.*;
import javax.swing.*;

//============= Function to be defined ================
class fb extends f_x
{
  public double func (double x)
  { return -(2.0*Math.sin(x)-x*x/10.0);}
}
//============================================================

public class NA35A
{
            public static double SIGN(double a,double b) {return (b > 0.0 ? Math.abs(a) : -Math.abs(a));}
    public static double MAX(double a,double b) {return (a > b ? a : b);}

   public static double[] inputdata(String s)
    {
    String s1=JOptionPane.showInputDialog(s);
    StringTokenizer token=new StringTokenizer(s1);
    int n=token.countTokens()-1;
    int m=n+1;
    double a[]=new double[m];
    int j=0;
    while(token.hasMoreTokens())
    {
    Double ax=new Double(token.nextToken());
    a[j++]=ax.doubleValue();
    }
    return a;
    }
    public static double df(f_x f,double x)
    {
// türev alt programi
// df fonksiyonunun türevini hesaplar
```

```
double h0=0.0256;
int i,m;
int n=7;
//turev of a simple function
double T[][];
T=new double[n][n];
double h[];
h=new double[n];
//vector<double> h(n,0);
for(i=0;i<n;i++)
{
  h[i]=0;
  for(int j=0;j<n;j++)
    T[i][j]=0;
}
h[0]=h0;
double r=0.5;
for( i=1;i<n;i++)
{
h[i]=h0*Math.pow(r,i);
}

for(i=0;i<n;i++)
{
T[i][0]=( f.func(x + h[i]) - f.func( x - h[i]))/(2.0*h[i]);
}
for(m=1;m<n;m++)
{
  for(i=0;i<n-m;i++)
  {
  T[i][m]=(h[i]*h[i]*T[i+1][m-1] - h[i+m]*h[i+m]*T[i][m-1])/
         (h[i]*h[i]- h[i+m]*h[i+m]);
  }
}
double xx=T[0][n-1];
return xx;
}
public  static double dbrent(f_x f,double ax,double bx,double cx,double tol)
{
// ax,bx,cx is three initial guesses (bx should be located between ax and cx)
// f : function (defined with f_x abstract class)
// tol tolerance
int ITMAX=100;
double CGOLD=(3.0-Math.sqrt(5))/2.0; //altin oran
double ZEPS=1.0e-10;
              int iter;
              boolean ok1,ok2;
              double a,b,d,d1,d2,du,dv,dw,dx,e=0.0;
              d=0.0;
              double fu,fv,fw,fx,olde,tol1,tol2,u,u1,u2,v,w,x,xm;
              double xmin=bx;
              a=(ax < cx ? ax : cx);
              b=(ax > cx ? ax : cx);
              x=w=v=bx;
              fw=fv=fx=f.func(x);
              dw=dv=dx=df(f,x);
              for (iter=1;iter<=ITMAX;iter++) {
```

```
xm=0.5*(a+b);
tol1=tol*Math.abs(x)+ZEPS;
tol2=2.0*tol1;
if (Math.abs(x-xm) <= (tol2-0.5*(b-a))) {
        xmin=x;
        return xmin;
}
if (Math.abs(e) > tol1) {
        d1=2.0*(b-a);
        d2=d1;
        if (dw != dx)  d1=(w-x)*dx/(dx-dw);
        if (dv != dx)  d2=(v-x)*dx/(dx-dv);
        u1=x+d1;
        u2=x+d2;
        ok1 = (a-u1)*(u1-b) > 0.0 && dx*d1 <= 0.0;
        ok2 = (a-u2)*(u2-b) > 0.0 && dx*d2 <= 0.0;
        olde=e;
        e=d;
        if (ok1 || ok2) {
                if (ok1 && ok2)
                        d=(Math.abs(d1) < Math.abs(d2) ? d1 : d2);
                else if (ok1)
                        d=d1;
                else
                        d=d2;
                if (Math.abs(d) <= Math.abs(0.5*olde)) {
                        u=x+d;
                        if (u-a < tol2 || b-u < tol2)
                                d=SIGN(tol1,xm-x);
                } else {
                        d=0.5*(e=(dx >= 0.0 ? a-x : b-x));
                }
        } else {
                d=0.5*(e=(dx >= 0.0 ? a-x : b-x));
        }
} else {
        d=0.5*(e=(dx >= 0.0 ? a-x : b-x));
}
if (Math.abs(d) >= tol1) {
        u=x+d;
        fu=f.func(u);
} else {
        u=x+SIGN(tol1,d);
        fu=f.func(u);
        if (fu > fx) {
                xmin=x;
                return xmin;
        }
}
du=df(f,u);
if (fu <= fx) {
        if (u >= x) a=x; else b=x;
        //MOV3(v,fv,dv, w,fw,dw)
        {v=w;fv=fw;dv=dw;}
        //MOV3(w,fw,dw, x,fx,dx)
        {w=x;fw=fx;dw=dx;}
        //MOV3(x,fx,dx, u,fu,du)
```

```
                                {x=u;fx=fu;dx=du;}
                        } else {
                                if (u < x) a=u; else b=u;
                                if (fu <= fw || w == x) {
                                        //MOV3(v,fv,dv, w,fw,dw)
                                        {v=w;fv=fw;dv=dw;}
                                        //MOV3(w,fw,dw, u,fu,du)
                                        {w=u;fw=fu;dw=du;}
                                } else if (fu < fv || v == x || v == w) {
                                        //MOV3(v,fv,dv, u,fu,du)
                                        {v=u;fv=fu;dv=du;}
                                }
                        }
                }
                System.out.println("maximum number of iterations are exceeded in DBRENT optimization");
                return xmin;
}
    public static void main (String args[])
    {
    String s="enter three initial guess values with a space in between xo x1 x2: ";
    double [] x0=inputdata(s);
    fb f=new fb();
    double r1= dbrent(f,x0[0],x0[1],x0[2],1.0e-8);
    s=" optimum value  : "+r1;
    String s2="Optimization by Brent method with derivatives:";
    JOptionPane.showMessageDialog(null,s,s2,JOptionPane.PLAIN_MESSAGE);
    }
}
```

5.7 IMPROVING ONE DIMENSIONAL OPTIMIZATION BRACKET VALUES

Before starting the search for the optimal value search range can be improved, This also can ensure us to have an optimum point in the given range. Method mnbrak takes 3 initial gues points and enlarge the the given zone untill a minimum is existed. It uses golden search in this process.

Program 5.7-1 mnbrak method (barketing the minimum value)

```
// OPTIMIZATION
//  one dimensional non-linear functions
//  mnbark bracket method
//
import java.util.*;
import java.awt.*;
import java.applet.Applet;
import java.awt.event.*;
import javax.swing.*;

//============= Function to be defined ================
class fb extends f_x
{
  public double func (double x)
  { return -(2.0*Math.sin(x)-x*x/10.0);}
}
//=============================================================

public class NA35B
{
public static double SIGN(double a,double b) {return (b > 0.0 ? Math.abs(a) : -Math.abs(a));}
public static double MAX(double a,double b) {return (a > b ? a : b);}
```

```
public  static double[] mnbrak(f_x f,double ax,double dx)
{
return mnbrak(f,ax,(ax-dx),(ax+dx));
}

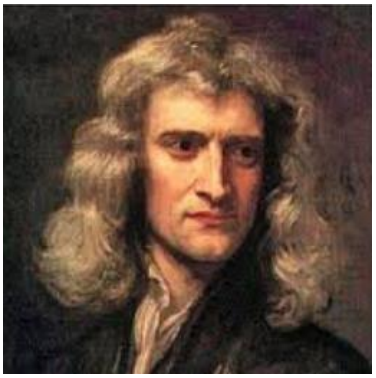
public  static double[] mnbrak(f_x f,double ax)
{
return mnbrak(f,ax,(ax-0.1),(ax+0.1));
}

public  static double[] mnbrak(f_x f,double ax,double bx,double cx)
{

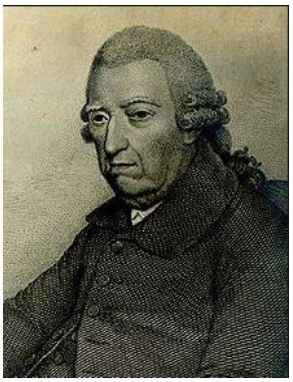
int ITMAX=100;
double xmin;
double GOLD=(Math.sqrt(5.0)+1.0)/2.0;;
double GLIMIT=100.0;
double TINY=1.0e-20;
double fa,fb,fc;
fa=0;
fb=0;
fc=0;
        double ulim,u,r,q,fu,dum;
        double aa[]=new double[3];
        fa=f.func(ax);
        fb=f.func(bx);
        if (fb > fa) {
                //SHFT(dum,*ax,*bx,dum)
                {dum=ax;ax=bx;bx=dum;}
                //SHFT(dum,*fb,*fa,dum)
                {dum=fb;fb=fa;fa=dum;}
        }
        cx=(bx)+GOLD*(bx-ax);
        fc=f.func(cx);
                aa[0]=ax;
                aa[1]=bx;
                aa[2]=cx;
        while (fb > fc) {
                r=(bx-ax)*(fb-fc);
                q=(bx-cx)*(fb-fa);
                u=(bx)-((bx-cx)*q-(bx-ax)*r)/
                        (2.0*SIGN(MAX(Math.abs(q-r),TINY),q-r));
                ulim=(bx)+GLIMIT*(cx-bx);
                if ((bx-u)*(u-cx) > 0.0) {
                        fu=f.func(u);
                        if (fu < fc) {
                                ax=bx;
                                bx=u;
                                fa=fb;
                                fb=fu;
                                return aa;
                        } else if (fu > fb) {
                                cx=u;
                                fc=fu;
                                return aa;
                        }
                        u=(cx)+GOLD*(cx-bx);
                        fu=f.func(u);
                } else if ((cx-u)*(u-ulim) > 0.0) {
                        fu=f.func(u);
                        if (fu < fc) {
                                //SHFT(bx,cx,u,cx+GOLD*(cx-bx))
                                {bx=cx;cx=u;u=cx+GOLD*(cx-bx);}
                                //SHFT(fb,fc,fu,f.func(u))
```

```
                                        {fb=fc;fc=fu;fu=f.func(u);}
                                }
                        } else if ((u-ulim)*(ulim-cx) >= 0.0) {
                                u=ulim;
                                fu=f.func(u);
                        } else {
                                u=cx+GOLD*(cx-bx);
                                fu=f.func(u);
                        }
                        //SHFT(ax,bx,cx,u)
                        {ax=bx;bx=cx;cx=u;}
                        //SHFT(fa,fb,fc,fu)
                        {fa=fb;fb=fc;fc=fu;}
                        aa[0]=ax;
                        aa[1]=bx;
                        aa[2]=cx;
                }
                        return aa;
}
//_____________________________________________________________________
    public static double[] inputdata(String s)
    {
    String s1=JOptionPane.showInputDialog(s);
    StringTokenizer token=new StringTokenizer(s1);
    int n=token.countTokens()-1;
    int m=n+1;
    double a[]=new double[m];
    int j=0;
    while(token.hasMoreTokens())
    {
    Double ax=new Double(token.nextToken());
    a[j++]=ax.doubleValue();
    }
    return a;
    }
    public static void main (String args[])
    {
            String s="enter one or three initial guess value with a space in between: ";
    double [] x0=inputdata(s);
    fb f=new fb();
    double r1[];
    if(x0.length==3)      {r1= mnbrak(f,x0[0],x0[1],x0[2]);}
    else if(x0.length==1) {r1= mnbrak(f,x0[0]);}
    else{System.out.println("Wrong number of first initial guesses are given only the first value will be used");
          r1= mnbrak(f,x0[0]);}
    s=" new bracket values  : \n"+Matrix.toString(r1);
    //fonksiyonun gerçek değerini elde etmek için -1 ile çarpilmiştir
    String s2="bracket values with an ensured optimum value in the given range :";
    JOptionPane.showMessageDialog(null,s,s2,JOptionPane.PLAIN_MESSAGE);
    }
}
```

5.8 NEWTON-RAPHSON METHOD FOR NONLINEAR MULTIVARIABLE OPTIMIZATION



Sir Isaac Newton



Joseph Raphson

Root finding of multivariable functions by using Newton-Raphson method was prevously investigated. By remembering that optimum point is the root of the derivative of the function, the method can be utilise also to find optimum point.

If function $f(x_1,x_2,x_3,\ldots,x_n)$ First and second derivative and independent difference vector of the function can be written as:

$$\nabla f = \begin{Bmatrix} \frac{\partial f}{\partial x_1} \\ \frac{\partial f}{\partial x_2} \\ \frac{\partial f}{\partial x_3} \\ \cdots\cdots \\ \frac{\partial f}{\partial x_n} \end{Bmatrix} \text{ and } \delta^{m+1} = \begin{Bmatrix} (x_1^{m+1} - x_1^m) \\ (x_2^{m+1} - x_2^m) \\ (x_3^{m+1} - x_3^m) \\ \cdots\cdots \\ (x_n^{m+1} - x_n^m) \end{Bmatrix} \text{ and } [H] = \begin{bmatrix} \frac{\partial^2 f}{\partial x_1^2} & \frac{\partial^2 f}{\partial x_1 \partial x_2} & \frac{\partial^2 f}{\partial x_1 \partial x_3} & \cdots & \frac{\partial^2 f}{\partial x_1 \partial x_n} \\ \frac{\partial^2 f}{\partial x_2 \partial x_1} & \frac{\partial^2 f}{\partial x_2^2} & \frac{\partial^2 f}{\partial x_2 \partial x_3} & \cdots & \frac{\partial^2 f}{\partial x_2 \partial x_n} \\ \frac{\partial^2 f}{\partial x_3 \partial x_1} & \frac{\partial^2 f}{\partial x_3 \partial x_2} & \frac{\partial^2 f}{\partial x_3^2} & \cdots & \frac{\partial^2 f}{\partial x_3 \partial x_n} \\ \cdots & \cdots & \cdots & \cdots & \cdots \\ \frac{\partial^2 f}{\partial x_1 \partial x_n} & \frac{\partial^2 f}{\partial x_2 \partial x_n} & \frac{\partial^2 f}{\partial x_3 \partial x_n} & \cdots & \frac{\partial^2 f}{\partial x_n^2} \end{bmatrix}$$

Newton-Raphson iteration for (m+1) th iteration

$$[H^m]\{\delta^{m+1}\} = -\{\nabla f^m\}$$

As it is seen from the equation a linear system of equation is formed. The equation can be solved by using methods such as gauss elimination. An initial estimate for all the x values are required to start iterative solution. Second derivartive matrix is also called a Hessian matrix. First and second derivatives can also be calculated numerically, but that will increase the errors orthe number of iteration steps. The system of equation can also be written as:

$$\{\delta^{m+1}\} = [H^m]^{-1}[-\{\nabla f^m\}]$$

In this format$[H^m]^{-1}$ is the inverse Hessian matrix. If vector δ is written in open form, it becomes

$$\{x^{m+1}\} = \{x^m\} + [H^m]^{-1}[-\{\nabla f^m\}]$$

As a more general definition of this equation a step function α can be added into the term. It becomes

$$\{x^{m+1}\} = \{x^m\} + \alpha^m[H^m]^{-1}[-\{\nabla f^m\}]$$

In Newton-Raphson method step function α is equal to unity. In some other methods different utilisation of this general description will be invetigated.

Program 5.8-1 Abstract class f_xj for function y=f(x0,x1,x2,..) (java version)

```
public abstract class f_xj extends Mathd
{  // single function  multi independent variable
   // a single value is returned indiced to equation_ref
abstract public double func(double x[]);
public double func(double x1,double y1)
{
double x[]=new double[2];
x[0]=x1;
```

```
x[1]=y1;
return func(x);
}
public double[] dfunc(double x[])
{int n=x.length;
 double c[]=new double[n];
 for(int i=0;i<n;i++)
 {c[i]=dfunc(x,i);}
 return c;
}

public double[][] d2func(double x[])
{int n=x.length;
 double c[][]=new double[n][n];

 for(int i=0;i<n;i++)
 {for(int j=0;j<n;j++)
   {   int xref[]={i,j};
                c[i][j]=d2func(x,xref);
   }
 }
 return c;
}
public double d2func(double x[],int x_ref[])
{// derivative of the function with respect to x_ref
double h0=0.0256;
int i,m;
int n=7;
double f1,f2;
double x1[];
x1=new double[x.length];
double x2[];
x2=new double[x.length];
for(i=0;i<x.length;i++)
{
x1[i]=x[i];
x2[i]=x[i];
}
//derivative of a simple function
double T[][];
T=new double[n][n];
double h[];
h=new double[n];
//vector<double> h(n,0);
for(i=0;i<n;i++)
{
  h[i]=0;
  for(int j=0;j<n;j++)
    T[i][j]=0;
}
h[0]=h0;
double r=0.5;
for( i=1;i<n;i++)
{
h[i]=h0*Math.pow(r,i);
}
for(i=0;i<n;i++)
{
x1[x_ref[1]]+=h[i];
x2[x_ref[1]]-=h[i];
f1=dfunc(x1,x_ref[0]);
f2=dfunc(x2,x_ref[0]);
T[i][0]=( f1 - f2)/(2.0*h[i]);
x1[x_ref[1]]=x[x_ref[1]];
```

```
x2[x_ref[1]]=x[x_ref[1]];
}
for(m=1;m<n;m++)
{
  for(i=0;i<n-m;i++)
  {
  T[i][m]=(h[i]*h[i]*T[i+1][m-1] - h[i+m]*h[i+m]*T[i][m-1])/(h[i]*h[i]
  - h[i+m]*h[i+m]);
  }
}
double xx=T[0][n-1];
return xx;
}
public double dfunc(double x[],int x_ref)
{ // derivative of the function with respect to x_ref
double h0=0.015675863;
int i,m;
int n=7;
double f1,f2;
double x1[];
x1=new double[x.length];
double x2[];
x2=new double[x.length];
for(i=0;i<x.length;i++)
{
x1[i]=x[i];
x2[i]=x[i];
}
//derivative of a simple function
double T[][];
T=new double[n][n];
double h[];
h=new double[n];
//vector<double> h(n,0);
for(i=0;i<n;i++)
{
  h[i]=0;
  for(int j=0;j<n;j++)
    T[i][j]=0;
}
h[0]=h0;
double r=0.5;
for( i=1;i<n;i++)
{
h[i]=h0*Math.pow(r,i);
}

for(i=0;i<n;i++)
{
x1[x_ref]+=h[i];
x2[x_ref]-=h[i];
f1=func(x1);
f2=func(x2);
T[i][0]=( f1 - f2)/(2.0*h[i]);
x1[x_ref]=x[x_ref];
x2[x_ref]=x[x_ref];
}
for(m=1;m<n;m++)
{
  for(i=0;i<n-m;i++)
  {
  T[i][m]=(h[i]*h[i]*T[i+1][m-1] - h[i+m]*h[i+m]*T[i][m-1])/(h[i]*h[i]
  - h[i+m]*h[i+m]);
  }
```

```
}
double xx=T[0][n-1];
return xx;
}}
```

Program 5.8-2 Multivariable non-linear Newton-Raphson method (java –abstact class-f-xj version)

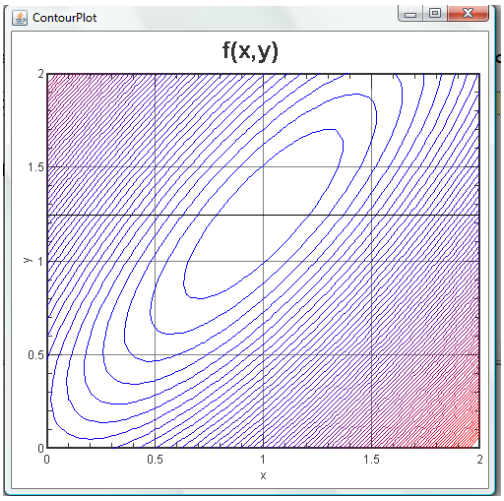
```
// OPTIMIZATION
//  Newton-Raphson method
import java.util.*;
import java.awt.*;
import java.applet.Applet;
import java.awt.event.*;
import javax.swing.*;

class f1 extends f_xj
{
public double func(double x[])
{//optimization function
double y=3.0*x[0]*x[0]-4.0*x[0]*x[1]+2.0*x[1]*x[1]-x[0]-x[1];
return y;
}}

public class NA34A
{
public static double[] gauss(double a[][],double b[])
  { //gauss elimination with partial pivoting
  int n=b.length;
  double x[]=new double[n];
  double carpan=0;
  double toplam=0;
  double buyuk;
  double dummy=0;
  //gauss elimination
  int i,j,k,p,ii,jj;
  for(k=0;k<(n-1);k++)
  { //pivoting
              p=k;
              buyuk=Math.abs(a[k][k]);
              for(ii=k+1;ii<n;ii++)
              {  dummy=Math.abs(a[ii][k]);
                 if(dummy > buyuk) {buyuk=dummy;p=ii;}
              }
              if(p!=k)
              { for(jj=k;jj<n;jj++)
                { dummy=a[p][jj];
                  a[p][jj]=a[k][jj];
                  a[k][jj]=dummy;
                }
                dummy=b[p];
                b[p]=b[k];
                b[k]=dummy;
              }
              //Solving gauss elimination
    for(i=k+1;i<n;i++)
    {  carpan=a[i][k]/a[k][k];
       a[i][k]=0;
       for(j=k+1;j<n;j++)
       {   a[i][j]-=carpan*a[k][j]; }
          b[i]   =b[i]   -carpan*b[k];
              }
  }
  //back substitution
  x[n-1]=b[n-1]/a[n-1][n-1];
  for(i=n-2;i>=0;i--)
```

```
  {
   toplam=0;
   for(j=i+1;j<n;j++)
   {  toplam+=a[i][j]*x[j];}
   x[i]=(b[i]-toplam)/a[i][i];
  }
  return x;
  }
public static double[] newton_opt(f_xj f,double x[])
{ int k=x.length;
  int nmax=100;
  double tolerance=1.0e-10;
  double fx[];
  double dfx[][];
  double dx[];
  double total=0;
  for(int i=0;i<nmax;i++)
  { fx=f.dfunc(x);
    dfx=f.d2func(x);
    dx=gauss(dfx,fx);
    for(int j=0;j<k;j++)
      x[j]-=dx[j];
    total=0;
    for(int j=0;j<k;j++)
      total+=fx[j];
    if(Math.abs(total)<tolerance) return x;
  }
  return x;
}
public static double[] inputdata(String s)
     {
     String s1=JOptionPane.showInputDialog(s);
     StringTokenizer token=new StringTokenizer(s1);
     int n=token.countTokens()-1;
     int m=n+1;
     double a[]=new double[m];
     int j=0;
     while(token.hasMoreTokens())
     {
     Double ax=new Double(token.nextToken());
     a[j++]=ax.doubleValue();
     }
     return a;
     }
     public static void main (String args[])
     {
                f1  f=new f1();
     double x[]=inputdata(" enter guess values  x0 : ");
     double r[];
     r=newton_opt(f,x);
     System.out.println(" minimum value  : "+Matrix.toString(r));
     }
}
```

```
---------- Capture Output ----------
> "D:\co\java\bin\java.exe" NA34A
 minimum value  :        0.999999999999083                    1.249999999998778
> Terminated with exit code 0.
```



Program 5.8-3 if_xj interface (java language)

```
@FunctionalInterface
interface if_xj
{
  // single function  multi independent variable
  // a single value is returned indiced to equation_ref
public double func(double x[]);
default double func(double x1,double y1)
{
double x[]=new double[2];
x[0]=x1;
x[1]=y1;
return func(x);
}
default double[] dfunc(double x[])
{int n=x.length;
 double c[]=new double[n];
 for(int i=0;i<n;i++)
 {c[i]=dfunc(x,i);}
 return c;
}

default double[][] d2func(double x[])
{int n=x.length;
 double c[][]=new double[n][n];
 for(int i=0;i<n;i++)
 {for(int j=0;j<n;j++)
   {   int xref[]={i,j};
                c[i][j]=d2func(x,xref);
   }
 }
 return c;
}

default double d2func(double x[],int x_ref[])
{// derivative of the function with respect to x_ref
double h0=0.0256;
int i,m;
int n=7;
double f1,f2;
double x1[];
x1=new double[x.length];
double x2[];
x2=new double[x.length];
for(i=0;i<x.length;i++)
{
```

```
x1[i]=x[i];
x2[i]=x[i];
}
//derivative of a simple function
double T[][];
T=new double[n][n];
double h[];
h=new double[n];
//vector<double> h(n,0);
for(i=0;i<n;i++)
{
  h[i]=0;
  for(int j=0;j<n;j++)
    T[i][j]=0;
}
h[0]=h0;
double r=0.5;
for( i=1;i<n;i++)
{
h[i]=h0*Math.pow(r,i);
}

for(i=0;i<n;i++)
{
x1[x_ref[1]]+=h[i];
x2[x_ref[1]]-=h[i];
f1=dfunc(x1,x_ref[0]);
f2=dfunc(x2,x_ref[0]);
T[i][0]=( f1 - f2)/(2.0*h[i]);
x1[x_ref[1]]=x[x_ref[1]];
x2[x_ref[1]]=x[x_ref[1]];
}
for(m=1;m<n;m++)
{
  for(i=0;i<n-m;i++)
  {
  T[i][m]=(h[i]*h[i]*T[i+1][m-1] - h[i+m]*h[i+m]*T[i][m-1])/(h[i]*h[i]
  - h[i+m]*h[i+m]);
  }
}
double xx=T[0][n-1];
return xx;
}
default double dfunc(double x[],int x_ref)
{ // derivative of the function with respect to x_ref
double h0=0.0256;
int i,m;
int n=7;
double f1,f2;
double x1[];
x1=new double[x.length];
double x2[];
x2=new double[x.length];
for(i=0;i<x.length;i++)
{
x1[i]=x[i];
x2[i]=x[i];
}
//derivative of a simple function
double T[][];
T=new double[n][n];
double h[];
h=new double[n];
//vector<double> h(n,0);
```

```
for(i=0;i<n;i++)
{
  h[i]=0;
  for(int j=0;j<n;j++)
    T[i][j]=0;
}
h[0]=h0;
double r=0.5;
for( i=1;i<n;i++)
{
h[i]=h0*Math.pow(r,i);
}

for(i=0;i<n;i++)
{
x1[x_ref]+=h[i];
x2[x_ref]-=h[i];
f1=func(x1);
f2=func(x2);
T[i][0]=( f1 - f2)/(2.0*h[i]);
x1[x_ref]=x[x_ref];
x2[x_ref]=x[x_ref];
}
for(m=1;m<n;m++)
{
  for(i=0;i<n-m;i++)
  {
  T[i][m]=(h[i]*h[i]*T[i+1][m-1] - h[i+m]*h[i+m]*T[i][m-1])/(h[i]*h[i]
  - h[i+m]*h[i+m]);
  }
}
double xx=T[0][n-1];
return xx;
}}
```

Program 5.8-4 Multivariable non-linear Newton-Raphson method (java –interface-if-xj – Lambda variable version)

```
/// OPTIMIZATION
//  Newton-Raphson method
import java.util.*;
import java.awt.*;
import java.applet.Applet;
import java.awt.event.*;
import javax.swing.*;

public class NA34B
{
public static double[] gauss(double a[][],double b[])
  { //gauss elimination with partial pivoting
  int n=b.length;
  double x[]=new double[n];
  double carpan=0;
  double toplam=0;
  double buyuk;
  double dummy=0;
  //gauss elimination
  int i,j,k,p,ii,jj;
  for(k=0;k<(n-1);k++)
  { //pivoting
                p=k;
                buyuk=Math.abs(a[k][k]);
                for(ii=k+1;ii<n;ii++)
                {  dummy=Math.abs(a[ii][k]);
                  if(dummy > buyuk) {buyuk=dummy;p=ii;}
```

```
                }
                if(p!=k)
                { for(jj=k;jj<n;jj++)
                  { dummy=a[p][jj];
                    a[p][jj]=a[k][jj];
                    a[k][jj]=dummy;
                  }
                  dummy=b[p];
                  b[p]=b[k];
                  b[k]=dummy;
                }
                //Solving gauss elimination
    for(i=k+1;i<n;i++)
    {  carpan=a[i][k]/a[k][k];
       a[i][k]=0;
       for(j=k+1;j<n;j++)
       {   a[i][j]-=carpan*a[k][j]; }
          b[i]   =b[i]   -carpan*b[k];
                }
  }
  //back substitution
  x[n-1]=b[n-1]/a[n-1][n-1];
  for(i=n-2;i>=0;i--)
  {
    toplam=0;
    for(j=i+1;j<n;j++)
    {  toplam+=a[i][j]*x[j];}
    x[i]=(b[i]-toplam)/a[i][i];
  }
  return x;
  }
public static double[] newton_opt(if_xj f,double x[])
{ int k=x.length;
  int nmax=100;
  double tolerance=1.0e-10;
  double fx[];
  double dfx[][];
  double dx[];
  double total=0;
  for(int i=0;i<nmax;i++)
  { fx=f.dfunc(x);
    dfx=f.d2func(x);
    dx=gauss(dfx,fx);
    for(int j=0;j<k;j++)
      x[j]-=dx[j];
    total=0;
    for(int j=0;j<k;j++)
      total+=Math.abs(fx[j]);
    if(total<tolerance) return x;
  }
  return x;
}
public static double[] inputdata(String s)
    {
     String s1=JOptionPane.showInputDialog(s);
     StringTokenizer token=new StringTokenizer(s1);
     int n=token.countTokens()-1;
     int m=n+1;
     double a[]=new double[m];
     int j=0;
     while(token.hasMoreTokens())
     {
     Double ax=new Double(token.nextToken());
     a[j++]=ax.doubleValue();
```

```
    }
    return a;
    }
    public static void main (String args[])
    {
    if_xj f=(double x[])-> 3.0*x[0]*x[0]-4.0*x[0]*x[1]+2.0*x[1]*x[1]-x[0]-x[1];
    double x[]=inputdata(" enter guess values  x0 : ");
    double r[];
    r=newton_opt(f,x);
    System.out.println(" minimum value  : "+Matrix.toString(r));
    surfacePlot p=new surfacePlot(f,-2.0,2.0,-2.0,2.0,40);
    p.plot();
    }
}
```

Program 5.8-5 Abstract class f_xj for function y=f(x0,x1,x2,..) (Python version)

```
from abc import ABC, abstractmethod
from math import *

class f_xj(ABC):
  # multifunction multi independent variable
  # vector of dependent variables are returned
  # example f[0]=x[0]+sin(x[1])
  #         f[1]=x[0]*x[1]-x[1]
  # func(x) returns the value of f[0] and f[1]
  # as a two dimensional vector

  @abstractmethod
  def func(self,x):
    pass;

  def funcxy(self,x1,y1):
    n=2;
    x=[0 for i in range(n)];
    x[0]=x1;
    x[1]=y1;
    return self.func(x);

  def dfunc_(self,x,x_ref):
     #derivative of the function with respect to x_ref
     h0=0.015675863;
     m=7;
     n=len(x);
     x1=[0 for i in range(n)];
     x2=[0 for i in range(n)];
     for i in range(n):
        x1[i]=x[i];
        x2[i]=x[i];
     #derivative of a simple function
     T=[[0 for i2 in range(m)] for j2 in range(m)];
     h=[0 for i in range(m)];
     for i in range(m):
        h[i]=0;
        for j in range(m):
           T[i][j]=0;
     h[0]=h0;
     r=0.5;
     for i in range(m):
        h[i]=h0*pow(r,i);
     for i in range(m):
        x1[x_ref]+=h[i];
        x2[x_ref]-=h[i];
        f1=self.func(x1);
```

```
        f2=self.func(x2);
        T[i][0]=( f1 - f2)/(2.0*h[i]);
        x1[x_ref]=x[x_ref];
        x2[x_ref]=x[x_ref];
    for k in range(1,m):
         for i in range(0,m-k):
            T[i][k]=(h[i]*h[i]*T[i+1][k-1] - h[i+k]*h[i+k]*T[i][k-1])/(h[i]*h[i]
            - h[i+k]*h[i+k]);
    xx=T[0][n-1];
    return xx;

def d2func_(self,x,x_ref):
    #derivative of the function with respect to x_ref
    h0=0.015675863;
    m=7;
    n=len(x);
    x1=[0 for i in range(n)];
    x2=[0 for i in range(n)];
    for i in range(n):
        x1[i]=x[i];
        x2[i]=x[i];
    #derivative of a simple function
    T=[[0 for i2 in range(m)] for j2 in range(m)];
    h=[0 for i in range(m)];
    for i in range(m):
        h[i]=0;
        for j in range(m):
            T[i][j]=0;
    h[0]=h0;
    r=0.5;
    for i in range(m):
        h[i]=h0*pow(r,i);
    for i in range(m):
        x1[x_ref[1]]+=h[i];
        x2[x_ref[1]]-=h[i];
        f1=self.dfunc_(x1,x_ref[0]);
        f2=self.dfunc_(x2,x_ref[0]);
        T[i][0]=( f1 - f2)/(2.0*h[i]);
        x1[x_ref[1]]=x[x_ref[1]];
        x2[x_ref[1]]=x[x_ref[1]];
    for k in range(1,m):
         for i in range(0,m-k):
            T[i][k]=(h[i]*h[i]*T[i+1][k-1] - h[i+k]*h[i+k]*T[i][k-1])/(h[i]*h[i]
            - h[i+k]*h[i+k]);
    xx=T[0][n-1];
    return xx;

def dfunc(self,x):
    n=len(x);
    c=[0 for i in range(n)];
    for i in range(n):
        c[i]=self.dfunc_(x,i);
    return c;

def d2func(self,x):
    n=len(x);
    x_ref=[0 for i2 in range(2)];
    c=[[0 for i2 in range(n)] for j2 in range(n)];
    for i in range(n):
        for j in range(n):
            x_ref[0]=i;
            x_ref[1]=j;
            c[i][j]=self.d2func_(x,x_ref);
    return c;
```

Program 5.8-4 Multivariable non-linear Newton-Raphson method (Python version)

```
 from math import *
from f_xj import *;
from gauss import *;

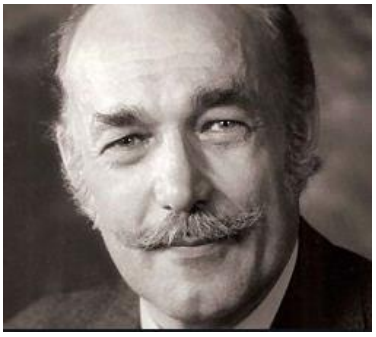
class f1(f_xj):
  def func(self,x):
     y=3.0*x[0]*x[0]-4.0*x[0]*x[1]+2.0*x[1]*x[1]-x[0]-x[1];
     return y;

def newton(f,x):
   # Newton-Raphson root finding method
   k=len(x);
   nmax=100;
   tolerance=1.0e-10
   for i in range(nmax):
      fx= f.dfunc(x);
      dfx=f.d2func(x);
      dx=gauss(dfx,fx);
      for j in range(k):
         x[j]-=dx[j];
      total=0;
      for j in range(k):
         total+=fx[j];
      if abs(total)<tolerance: return x
   return x;

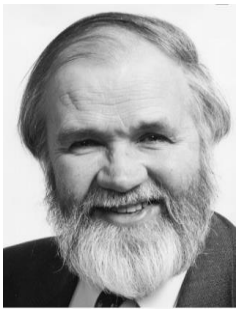
x=[1,1];
f=f1();
r=newton(f,x);
print("r=",r);
```

```
runfile('D:/okul/SCO1/newtond_opt.py', wdir='D:/okul/SCO1')
Reloaded modules: f_xj, gauss
r= [0.9999999999999799, 1.2499999999999776]
```

5.9 NELDER – MEAD SIMPLEX NONLINEAR MULTIVARIABLE OPTIMIZATION METHOD

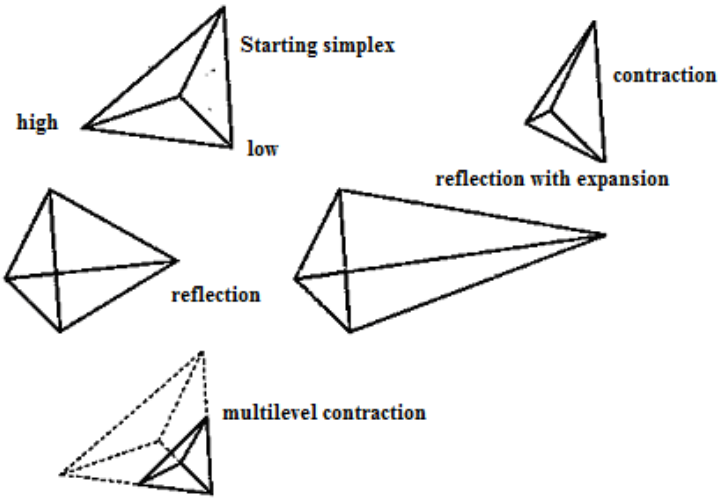


John Nelder



Roger Mead

The simplex method developed by Nelder and Mead method is a multidimensional search method . Simplex is an n dimensional geometric entity. It contains (n+1) points in n dimensional space. The method uses the concept of a simplex, which is a special polytop of $N + 1$ vertices in N dimensions. Examples of simplices include a line segment on a line, a triangle on a plane, a tetrahedron in three-dimensional space and so forth. Required processes for the simplex amobea's movements are shown in the figure



Process can be given as follows

- **1. Order** according to the values of the vertices:
  $\boldsymbol{f(x_1) \leq f(x_2) \leq \cdots \leq f(x_{n+1})}$
- **2. Calculate** x_0, the center of gravity off all points except x_{n+1}
- **3. Reflection:** Compute reflected point $x_r = x_0 + \alpha(x_0 - x_{n+1})$

If the reflected point is better than the second worst, but not better than the best, i.e.:
$\boldsymbol{f(x_1) \leq f(x_2) \leq f(x_n)}$
then obtain a new simplex by replacing the worst point x_{n+1} with the reflected point x_r, and go to step1

- **4. Expansion:**

If the reflected point is the best point so far, $\boldsymbol{f(x_r) < f(x_1)}$ **Then** compute the expanded point
$x_e = x_0 + \gamma(x_0 - x_{n+1})$

If the expanded point is better than the reflected point, $\boldsymbol{f(x_e) < f(x_r)}$ **then** obtain a new simplex by replacing the worst point x_{n+1} with the expanded point x_e, and go to step 1.

Else obtain a new simplex by replacing the worst point x_{n+1} with the reflected point x_r, and go to step 1.

Else (i.e. reflected point is worse than second worst) continue at step 5.

- **5. Contraction:**

Here, it is certain that , $\boldsymbol{f(x_r) < f(x_n)}$
Compute contracted point $x_c = x_{n+1} + \rho(x_0 - x_{n+1})$
If the contracted point is better than the worst point, i.e. $\boldsymbol{f(x_c) < f(x_{n+1})}$
then obtain a new simplex by replacing the worst point x_{n+1} with the contracted point x_c, and go to step 1.
Else go to step 6.

- **6. Reduction:**

For all but the best point with
$x_i = x_1 + \sigma(x_i - x_1)$ for all $i \in \{2, \dots, n+1\}$ goto step 1

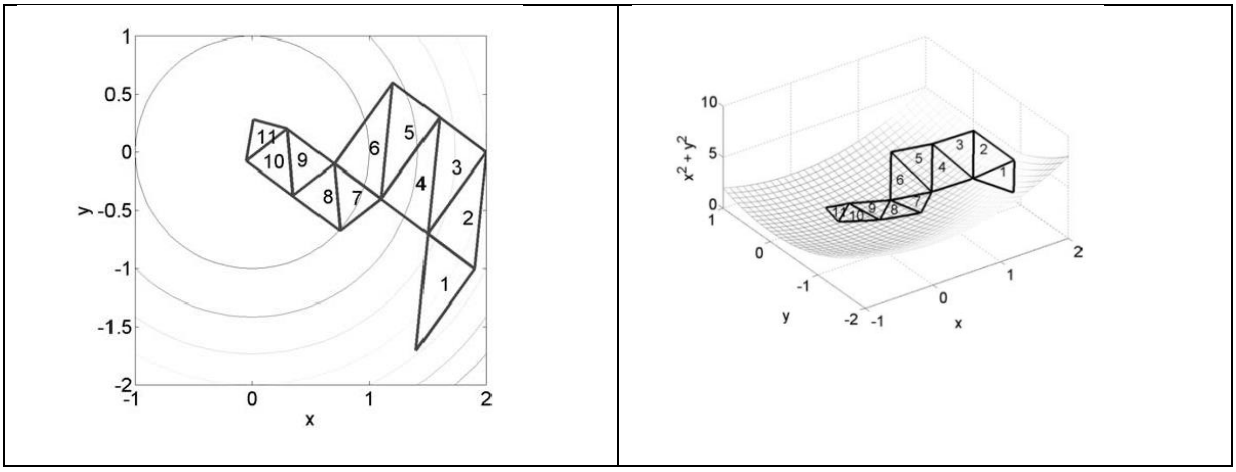
Note: $\boldsymbol{\alpha, \gamma, \rho}$ and σ are respectively the reflection, the expansion, the contraction and the shrink coefficient. Standard values are $\boldsymbol{\alpha = 1, \gamma = 2, \rho = 1/2}$ and $\boldsymbol{\sigma = 1/2}$
For the **reflection**, since x_{n+1} is the vertex with the higher associated value among the vertices, we can expect to find a lower value at the reflection of x_{n+1} in the opposite face formed by all vertices point x_i except x_{n+1}.

For the **expansion**, if the reflection point x_r is the new minimum along the vertices we can expect to find interesting values along the direction from x_o to x_r.
Concerning the **contraction**: If $f(x_r) > f(x_n)$ we can expect that a better value will be inside the simplex formed by all the vertices x_i.
The initial simplex is important, indeed, a too small initial simplex can lead to a local search, consequently the NM can get more easily stuck. So this simplex should depend on the nature of the problem. Instead of given n+1 point same process can be done by giving one point and a change vector by defining
$x_{i-1} = P_i + \lambda\, dx_j$ where if i=j λ=1 else if i≠j λ=0 .
For example:

if $P = \begin{Bmatrix} a \\ b \\ c \end{Bmatrix}$ and $dx = \begin{Bmatrix} da \\ db \\ dc \end{Bmatrix}$

$$x_0 = \begin{Bmatrix} a \\ b \\ c \end{Bmatrix} \quad x_1 = \begin{Bmatrix} a + da \\ b \\ c \end{Bmatrix} \quad x_2 = \begin{Bmatrix} a \\ b + db \\ c \end{Bmatrix} \quad x_3 = \begin{Bmatrix} a \\ b \\ c + dc \end{Bmatrix}$$

Figure 6.2-1 Movements of the nelder-Mead simplex element to find the optimum point



Program 5.9-1 Nelder and Mead simplex method

```
 // Nelder-Mead Simplex optimization method// Nelder-Mead Simplex optimization method
//
import java.util.*;
import java.awt.*;
import java.applet.Applet;
import java.awt.event.*;
import javax.swing.*;


class f1 extends f_xj
{
public double func(double x[])
{
//optimization function
double ff;
ff=(x[0]-2.0)*(x[0]-2.0)*(x[0]-2.0)*(x[0]-2.0)+(x[1]-1.0)*(x[1]-1.0)+
```

```
3.0*(x[2]-1.0)*(x[2]-1.0)*(x[2]-1.0)*(x[2]-1.0)*(x[2]-1.0)*(x[2]-1.0);
return ff;
}}

public class NA41
{
//6-2. Nelder and Mead Simplex multivariable nonlinear optimization method
//  Nelder & Mead 1965 Computer J, v.7, 308-313.
//___abstract class inputs_____________________________________________________________
public static double[] nelder(f_xj fnelder,double a[],double da[],int maxiteration,double tolerance,int printlist)
{
            int i,j;
double x[][]=new double[a.length+1][a.length];
double p[][]=new double[a.length+1][a.length+1];
for(i=0;i<x.length;i++)
  {for(j=0;j<x[0].length;j++)
    {if(i==j){x[i][j]=a[j]+da[j];p[i][j]=x[i][j];}
     else    {x[i][j]=a[j];p[i][j]=x[i][j]; }
    }
    p[i][j] = fnelder.func(p[i]);
  }

  // Inlet variable definitions
  // fnelder : abstract multivariable function f(x)
  // x : independent variable set of n+1 simplex elements
  // maxiteration : maximum iteration number
  // tolerance :
      int NDIMS = x.length-1;
      int NPTS = x.length;
      int FUNC = NDIMS;
      int ncalls = 0;
        ////// construct the starting simplex ///////////////////
        //double p[][]=new double[NPTS][NPTS]; // [row][col] = [whichvx][coord,FUNC]
        double z[]=new double[NDIMS];
        double best = 1E99;
        ///////////////// calculate the first function values for the simplex /////////////////
        int iter=0;
        for (iter=1; iter<maxiteration; iter++)
        {
                  ///////////  define lo, nhi, hi (low high next_to_high //////////////
          int  ilo=0, ihi=0, inhi = -1; // -1 means missing
          double flo = p[0][FUNC];
          double fhi = flo;
          double pavg,sterr;
          for (i=1; i<NPTS; i++)
          {
            if (p[i][FUNC] < flo)
              {flo=p[i][FUNC]; ilo=i;}
            if (p[i][FUNC] > fhi)
              {fhi=p[i][FUNC]; ihi=i;}
          }
          double fnhi = flo;
          inhi = ilo;
          for (i=0; i<NPTS; i++)
            if ((i != ihi) && (p[i][FUNC] > fnhi))
              {fnhi=p[i][FUNC]; inhi=i;}
          ////////// exit criteria //////////////
```

```
if ((iter % 4*NDIMS) == 0)
{
   // calculate the avarage (including maximum value)
   pavg=0;
   for(i=0;i<NPTS;i++)
     pavg+=p[i][FUNC];
   pavg/=NPTS;
   double tot=0;
   if(printlist!=0)
   {  System.out.print(iter);
      for (j=0; j<=NDIMS; j++)
         { System.out.print(p[ilo][j]+" ");}
      System.out.println("");
   }
   for(i=0;i<NPTS;i++)
   { tot=(p[i][FUNC]-pavg)*(p[i][FUNC]-pavg);}
   sterr=Math.sqrt(tot/NPTS);
   //if(sterr < tolerance)
     { for (j=0; j<NDIMS; j++)
          { z[j]=p[ilo][j];}
       //break;
     }
   best = p[ilo][FUNC];
}

///// calculate avarage without maximum value //////

double ave[] = new double[NDIMS];
for (j=0; j<NDIMS; j++)
  ave[j] = 0;
for (i=0; i<NPTS; i++)
  if (i != ihi)
    for (j=0; j<NDIMS; j++)
      ave[j] += p[i][j];
for (j=0; j<NDIMS; j++)
   ave[j] /= (NPTS-1);
///////// reflect ////////////////
double r[] = new double[NDIMS];
for (j=0; j<NDIMS; j++)
  r[j] = 2*ave[j] - p[ihi][j];
double fr = fnelder.func(r);

if ((flo <= fr) && (fr < fnhi))  // in zone: accept
{
   for (j=0; j<NDIMS; j++)
     p[ihi][j] = r[j];
   p[ihi][FUNC] = fr;
   continue;
}
if (fr < flo)  //// expand
{
   double e[] = new double[NDIMS];
   for (j=0; j<NDIMS; j++)
     e[j] = 3*ave[j] - 2*p[ihi][j];
   double fe = fnelder.func(e);
   if (fe < fr)
   {
```

```
            for (j=0; j<NDIMS; j++)
              p[ihi][j] = e[j];
            p[ihi][FUNC] = fe;
            continue;
          }
          else
          {
            for (j=0; j<NDIMS; j++)
              p[ihi][j] = r[j];
            p[ihi][FUNC] = fr;
            continue;
          }
        }
        ////////////// shrink:
        if (fr < fhi)
        {
          double c[] = new double[NDIMS];
          for (j=0; j<NDIMS; j++)
            c[j] = 1.5*ave[j] - 0.5*p[ihi][j];
          double fc = fnelder.func(c);
          if (fc <= fr)
          {
            for (j=0; j<NDIMS; j++)
              p[ihi][j] = c[j];
            p[ihi][FUNC] = fc;
            continue;
          }
          else   /////// shrink
          {
            for (i=0; i<NPTS; i++)
              if (i != ilo)
              {
                for (j=0; j<NDIMS; j++)
                  p[i][j] = 0.5*p[ilo][j] + 0.5*p[i][j];
                p[i][FUNC] = fnelder.func(p[i]);
              }
            continue;
          }
        }

        if (fr >= fhi)   ///
        {
          double cc[] = new double[NDIMS];
          for (j=0; j<NDIMS; j++)
            cc[j] = 0.5*ave[j] + 0.5*p[ihi][j];
          double fcc = fnelder.func(cc);
          if (fcc < fhi)
          {
            for (j=0; j<NDIMS; j++)
              p[ihi][j] = cc[j];
            p[ihi][FUNC] = fcc;
            continue;
          }
          else    /////////
          {
            for (i=0; i<NPTS; i++)
              if (i != ilo)
```

```
                    {
                      for (j=0; j<NDIMS; j++)
                        p[i][j] = 0.5*p[ilo][j] + 0.5*p[i][j];
                      p[i][FUNC] = fnelder.func(p[i]);
                    }
                 }
              }
           }
       return z;
}
    public static double[] nelder(f_xj fnelder,double a[],double da[],double tolerance)
    {return nelder(fnelder,a,da,500,tolerance,0);}

    public static double[] nelder(f_xj fnelder,double a[],double da[])
    {return nelder(fnelder,a,da,500,1.0e-10,0);}

    public static double[] nelder(f_xj fnelder,double a[])
    {
                double [] da=new double[a.length];
          for(int i=0;i<a.length;i++) da[i]=0.1*a[i];
          return nelder(fnelder,a,da);
    }
// if_xj interface methods
public static double[] nelder(if_xj fnelder,double a[],double da[],int maxiteration,double tolerance,int printlist)
{
            int i,j;
double x[][]=new double[a.length+1][a.length];
double p[][]=new double[a.length+1][a.length+1];
for(i=0;i<x.length;i++)
  {for(j=0;j<x[0].length;j++)
    {if(i==j){x[i][j]=a[j]+da[j];p[i][j]=x[i][j];}
     else    {x[i][j]=a[j];p[i][j]=x[i][j]; }
    }
    p[i][j] = fnelder.func(p[i]);
  }

  // Inlet variable definitions
  // fnelder : abstract multivariable function f(x)
  // x : independent variable set of n+1 simplex elements
  // maxiteration : maximum iteration number
  // tolerance :
      int NDIMS = x.length-1;
      int NPTS = x.length;
      int FUNC = NDIMS;
      int ncalls = 0;
        ////// construct the starting simplex ///////////////////
        //double p[][]=new double[NPTS][NPTS]; // [row][col] = [whichvx][coord,FUNC]
        double z[]=new double[NDIMS];
        double best = 1E99;
        ///////////////// calculate the first function values for the simplex /////////////////
        int iter=0;
        for (iter=1; iter<maxiteration; iter++)
        {
                  ////////////  define lo, nhi, hi (low high next_to_high ///////////////
           int  ilo=0, ihi=0, inhi = -1; // -1 means missing
           double flo = p[0][FUNC];
           double fhi = flo;
```

```
double pavg,sterr;
for (i=1; i<NPTS; i++)
{
  if (p[i][FUNC] < flo)
    {flo=p[i][FUNC]; ilo=i;}
  if (p[i][FUNC] > fhi)
    {fhi=p[i][FUNC]; ihi=i;}
}
double fnhi = flo;
inhi = ilo;
for (i=0; i<NPTS; i++)
 if ((i != ihi) && (p[i][FUNC] > fnhi))
   {fnhi=p[i][FUNC]; inhi=i;}
////////// exit criteria //////////////
if ((iter % 4*NDIMS) == 0)
{
   // calculate the avarage (including maximum value)
   pavg=0;
   for(i=0;i<NPTS;i++)
     pavg+=p[i][FUNC];
   pavg/=NPTS;
   double tot=0;
   if(printlist!=0)
   {  System.out.print(iter);
      for (j=0; j<=NDIMS; j++)
        { System.out.print(p[ilo][j]+" ");}
      System.out.println("");
   }
   for(i=0;i<NPTS;i++)
   { tot=(p[i][FUNC]-pavg)*(p[i][FUNC]-pavg);}
   sterr=Math.sqrt(tot/NPTS);
   //if(sterr < tolerance)
     { for (j=0; j<NDIMS; j++)
         { z[j]=p[ilo][j];}
       //break;
      }
   best = p[ilo][FUNC];
}

///// calculate avarage without maximum value //////

double ave[] = new double[NDIMS];
for (j=0; j<NDIMS; j++)
 ave[j] = 0;
for (i=0; i<NPTS; i++)
 if (i != ihi)
   for (j=0; j<NDIMS; j++)
     ave[j] += p[i][j];
for (j=0; j<NDIMS; j++)
  ave[j] /= (NPTS-1);
//////// reflect ////////////////
double r[] = new double[NDIMS];
for (j=0; j<NDIMS; j++)
 r[j] = 2*ave[j] - p[ihi][j];
double fr = fnelder.func(r);

if ((flo <= fr) && (fr < fnhi))  // in zone: accept
```

```
        {
          for (j=0; j<NDIMS; j++)
            p[ihi][j] = r[j];
          p[ihi][FUNC] = fr;
          continue;
        }
        if (fr < flo)  //// expand
        {
          double e[] = new double[NDIMS];
          for (j=0; j<NDIMS; j++)
            e[j] = 3*ave[j] - 2*p[ihi][j];
          double fe = fnelder.func(e);
          if (fe < fr)
          {
            for (j=0; j<NDIMS; j++)
              p[ihi][j] = e[j];
            p[ihi][FUNC] = fe;
            continue;
          }
          else
          {
            for (j=0; j<NDIMS; j++)
              p[ihi][j] = r[j];
            p[ihi][FUNC] = fr;
            continue;
          }
        }
        ////////////// shrink:
        if (fr < fhi)
        {
          double c[] = new double[NDIMS];
          for (j=0; j<NDIMS; j++)
            c[j] = 1.5*ave[j] - 0.5*p[ihi][j];
          double fc = fnelder.func(c);
          if (fc <= fr)
          {
            for (j=0; j<NDIMS; j++)
              p[ihi][j] = c[j];
            p[ihi][FUNC] = fc;
            continue;
          }
          else   /////// shrink
          {
            for (i=0; i<NPTS; i++)
              if (i != ilo)
              {
                for (j=0; j<NDIMS; j++)
                  p[i][j] = 0.5*p[ilo][j] + 0.5*p[i][j];
                p[i][FUNC] = fnelder.func(p[i]);
              }
            continue;
          }
        }

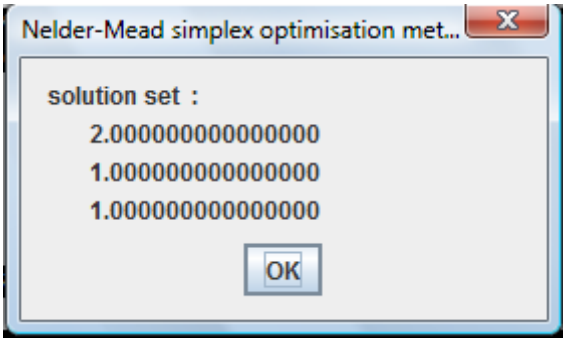
        if (fr >= fhi)   ///
        {
          double cc[] = new double[NDIMS];
```

```
                for (j=0; j<NDIMS; j++)
                  cc[j] = 0.5*ave[j] + 0.5*p[ihi][j];
                double fcc = fnelder.func(cc);
                if (fcc < fhi)
                {
                   for (j=0; j<NDIMS; j++)
                     p[ihi][j] = cc[j];
                   p[ihi][FUNC] = fcc;
                   continue;
                }
                else    /////////
                {
                   for (i=0; i<NPTS; i++)
                     if (i != ilo)
                     {
                        for (j=0; j<NDIMS; j++)
                          p[i][j] = 0.5*p[ilo][j] + 0.5*p[i][j];
                        p[i][FUNC] = fnelder.func(p[i]);
                     }
                }
             }
          }
      return z;
}
    public static double[] nelder(if_xj fnelder,double a[],double da[],double tolerance)
    {return nelder(fnelder,a,da,500,tolerance,0);}

    public static double[] nelder(if_xj fnelder,double a[],double da[])
    {return nelder(fnelder,a,da,500,1.0e-10,0);}

    public static double[] nelder(if_xj fnelder,double a[])
    {
                    double [] da=new double[a.length];
          for(int i=0;i<a.length;i++) da[i]=0.1*a[i];
          return  nelder(fnelder,a,da);
    }
  // input data
  //_____________________________________________________________________
    public static double[] inputdata(String s)
    {
    String s1=JOptionPane.showInputDialog(s);
    StringTokenizer token=new StringTokenizer(s1);
    int n=token.countTokens()-1;
    int m=n+1;
    double a[]=new double[m];
    int j=0;
    while(token.hasMoreTokens())
    {
    Double ax=new Double(token.nextToken());
    a[j++]=ax.doubleValue();
    }
    return a;
    }
            public static void main (String args[])
    {
    double [] x0=inputdata("enter optimum point first estimation values with one empty space in between: ");
    double [] dx0=new double[x0.length];
```

```
    for(int i=0;i<x0.length;i++) dx0[i]=0.2*x0[i];
    f1 f=new f1();
    double [] r1= nelder(f,x0,dx0);
    String s=" solution set  : \n"+Matrix.toStringT(r1);
    String s2="Nelder-Mead simplex optimization method ";
    JOptionPane.showMessageDialog(null,s,s2,JOptionPane.PLAIN_MESSAGE);
    System.exit(0);
    }
}
```



EXAMPLE PROBLEM

Solve the above function by using lambda variables

```
import javax.swing.JOptionPane;
public class NA41A
{  public static void main (String args[])
    {if_xj f=(double x[])->(x[0]-2.0)*(x[0]-2.0)*(x[0]-2.0)*(x[0]-2.0)+(x[1]-1.0)*(x[1]-1.0)+
    3.0*(x[2]-1.0)*(x[2]-1.0)*(x[2]-1.0)*(x[2]-1.0)*(x[2]-1.0)*(x[2]-1.0);
    double [] x0=NA41.inputdata("enter optimum point first estimation values with one empty space in between: ");
    double [] r1=NA41.nelder(f,x0);
    String s=" solution set  : \n"+Matrix.toStringT(r1);
    String s2="Nelder-Mead simplex optimization method ";
    JOptionPane.showMessageDialog(null,s,s2,JOptionPane.PLAIN_MESSAGE);
    System.exit(0);
    }
}
```

Python version of the above problem is given below:

```
from gauss import *;
from math import *;
from f_xj import *;
import numpy as np;
import matplotlib.pyplot as plt;
from mpl_toolkits.mplot3d import Axes3D
from random import *;

class f1(f_xj):
  def func(self,x):
     x1=x[0]-2.0
     x2=x[1]-1.0
     x3=x[2]-1.0
     y=x1*x1*x1*x1+x2*x2+3.0*x3*x3*x3*x3*x3*x3
     return y

class f2(f_xj):
  def func(self,x):
     y=3.0*x[0]*x[0]-4.0*x[0]*x[1]+2.0*x[1]*x[1]-x[0]-x[1]
     return y;
```

```
def  nelder(fnelder,a,da,maxiteration,tolerance,printlist):
    #print("a",a)
    #print("da",da)
    n=len(a)
    m=n+1
    x=[[0.0 for j in range(n)] for i in range(n+1)]
    p=[[0.0 for j in range(m)] for i in range(m)]
    s=""
    NDIMS = len(x)-1
    NPTS = len(x)
    FUNC = NDIMS
    ncalls = 0;
    for i in range(m):
        for j in range(n):
            if i==j:
                x[i][j]=a[j]+da[j]
                p[i][j]=x[i][j]
            else:
                x[i][j]=a[j]
                p[i][j]=x[i][j]
        p[i][FUNC] = fnelder.func(p[i])
        #print("x[",i,"]=",x[i])
        #print("p[",i,"]=",p[i])
    #print("p",p)
    #Inlet variable definitions
    #fnelder : abstract multivariable function f(x)
    #x : independent variable set of n+1 simplex elements
    #maxiteration : maximum iteration number
    #tolerance :

    ##### construct the starting simplex ///////////////////
    z=[0.0 for i in range(NDIMS)]
    best = 1.0E99;
    ################ calculate the first function values for the simplex ////////////////
    iter=0;
    for iter in range(1,maxiteration):
        ########  define lo, nhi, hi (low high next_to_high ///////////////
        ilo=0
        ihi=0
        inhi = -1
        # -1 means missing
        flo = p[0][FUNC];
        fhi = flo
        #print("flo=",flo,"fhi=",fhi)
        #double pavg,sterr;
        for i in range(1,NPTS):
            if p[i][FUNC] < flo:
                flo=p[i][FUNC]
                ilo=i
            if  p[i][FUNC] > fhi:
                fhi=p[i][FUNC]
                ihi=i
            #print("i=",i,"flo=",flo,"fhi=",fhi,"ilo=",ilo,"ihi=",ihi,"pifunc=",p[i][FUNC])
        fnhi = flo
        inhi = ilo
        #print("fnhi=",fnhi,"inhi=",inhi)
        for i in range(NPTS):
            if (i != ihi) and (p[i][FUNC] > fnhi):
                fnhi=p[i][FUNC]
                inhi=i
        ######### exit criteria ///////////////
        if (iter % 4*NDIMS) == 0:
            #calculate the avarage (including maximum value)
            pavg=0
```

```
        for i in range(NPTS):
            pavg=pavg+p[i][FUNC]
        pavg=pavg/NPTS
        tot=0.0
        if printlist!=0:
            print(iter)
            for j in range(NDIMS):
                print(p[ilo][j]," ")
        for i in range(NPTS):
            tot=(p[i][FUNC]-pavg)*(p[i][FUNC]-pavg)
        sterr=sqrt(tot/NPTS)
        for j in range(NDIMS):
            z[j]=p[ilo][j]
        best = p[ilo][FUNC];
    # end of if iter%4*NDMS)==0
    #//// calculate avarage without maximum value //////
    ave=[0.0 for i in range(NDIMS)]
    for j in range(NDIMS):
        ave[j] = 0
    for i in range(NPTS):
        if i != ihi:
            for j in range(NDIMS):
                ave[j] = ave[j]+p[i][j];
    for j in range(NDIMS):
        ave[j]=ave[j]/(NPTS-1)
    #print("ave[",j,"]=",ave[j])
    #//////// reflect ////////////////
    #print("reflect")
    r=[0.0 for i in range(NDIMS)]
    for j in range(NDIMS):
        r[j] = 2.0*ave[j] - p[ihi][j]
    fr = fnelder.func(r)
    if (flo <= fr) and (fr < fnhi): #in zone: accept
        for j in range(NDIMS):
            p[ihi][j] = r[j]
        p[ihi][FUNC] = fr
        continue
    #///(////////////// expand
    if fr < flo:
        #print("expand")
        e=[0.0 for i in range(NDIMS)]
        for j in range(NDIMS):
            e[j] = 3.0*ave[j] - 2.0*p[ihi][j];
        fe = fnelder.func(e);
        if fe < fr:
            for j in range(NDIMS):
                p[ihi][j] = e[j]
            p[ihi][FUNC] = fe
            continue
        else:
            for j in range(NDIMS):
                p[ihi][j] = r[j]
            p[ihi][FUNC] = fr
            continue
    #////////////// shrink:
    if fr < fhi:
        #print("srink")
        c=[0.0 for i in range(NDIMS)]
        for j in range(NDIMS):
            c[j] = 1.5*ave[j] - 0.5*p[ihi][j]
        fc = fnelder.func(c);
        if fc <= fr:
            for j in range(NDIMS):
                p[ihi][j] = c[j];
```

```
            p[ihi][FUNC] = fc
            continue
         else:   #////// shrink
            for i in range(NPTS):
               if i != ilo:
                  for j in range(NDIMS):
                     p[i][j] = 0.5*p[ilo][j] + 0.5*p[i][j]
                  p[i][FUNC] = fnelder.func(p[i])
            continue
      #/////////////// contract
      if fr >= fhi:
         #print("contract")
         cc=[0.0 for i in range(NDIMS)]
         for j in range(NDIMS):
            cc[j] = 0.5*ave[j] + 0.5*p[ihi][j];
         fcc = fnelder.func(cc)
         if fcc < fhi:
            for j in range(NDIMS):
               p[ihi][j] = cc[j];
            p[ihi][FUNC] = fcc
            continue
         else:
            for i in range(NPTS):
               if i != ilo:
                  for j in range(NDIMS):
                     p[i][j] = 0.5*p[ilo][j] + 0.5*p[i][j]
                  p[i][FUNC] = fnelder.func(p[i]);
   return z

a=[1,1,1]
da=[0.5122,0.643,0.25]
f=f1()
r=nelder(f,a,da,100,1.0e-15,0)
print("r = ",r)
```

```
runfile('E:/okul/SCO1/nelder.py', wdir='E:/okul/SCO1')
Reloaded modules: gauss, f_xj
r =  [1.9999749583837203, 0.9999999853635089, 1.002377215650601]
```

EXAMPLE PROBLEM

It is desired to find the minimum of function $\mathbf{f(x_0,x_1)=3x_0^2-4x_0x_1+2x_1^2-x_0-x_1}$ with the initial value of {x}=[-2,2]

Function written in java :

Version 1: abstract class f_xj derived class f1

```
 import javax.swing.JOptionPane;
class f1 extends f_xj
{
public double func(double x[])
{//optimization function
double ff;
ff=3.0*x[0]*x[0]-4.0*x[0]*x[1]+2.0*x[1]*x[1]-x[0]-x[1];
return ff;
}}

public class NA41B
{ public static void main (String args[])
    { f1 f=new f1();
    double [] x0=NA41.inputdata("enter optimum point first estimation values with one empty space in between: ");
    double [] r1=NA41.nelder(f,x0);
    String s=" solution set  : \n"+Matrix.toStringT(r1);
    String s2="Nelder-Mead simplex optimization method ";
    JOptionPane.showMessageDialog(null,s,s2,JOptionPane.PLAIN_MESSAGE);
    System.exit(0);
```

```
    }
}
```

Version 2: interface if_xj derived class f2

```
import javax.swing.JOptionPane;
class f2 implements if_xj
{
public double func(double x[])
{
//optimization function
double ff;
ff=3.0*x[0]*x[0]-4.0*x[0]*x[1]+2.0*x[1]*x[1]-x[0]-x[1];
return ff;
}}
public class NA41C
{public static void main (String args[])
    {f2 f=new f2();
    double [] x0=NA41.inputdata("enter optimum point first estimation values with one empty space in between: ");
    double [] r1=NA41.nelder(f,x0);
    String s=" solution set  : \n"+Matrix.toStringT(r1);
    String s2="Nelder-Mead simplex optimization method ";
    JOptionPane.showMessageDialog(null,s,s2,JOptionPane.PLAIN_MESSAGE);
    System.exit(0);
    }
}
```

Version 3: interface if_xj derived Lambda variable

```
import javax.swing.JOptionPane;

public class NA41D
{public static void main (String args[])
    {if_xj f=(double x[])->3.0*x[0]*x[0]-4.0*x[0]*x[1]+2.0*x[1]*x[1]-x[0]-x[1];
    double [] x0=NA41.inputdata("enter optimum point first estimation values with one empty space in between: ");
    double [] r1=NA41.nelder(f,x0);
    String s=" solution set  : \n"+Matrix.toStringT(r1);
    String s2="Nelder-Mead simplex optimization method ";
    JOptionPane.showMessageDialog(null,s,s2,JOptionPane.PLAIN_MESSAGE);
    System.exit(0);
    }
}
```

The results of first 50 iteration is recorded and results are plotted for first 17 iterations

Output 5.9-1 Sample Nelder-Mead output for the example program

```
iter=1p=
     -2.400000000000000      2.000000000000000      44.879999999999995
     -2.000000000000000      2.400000000000000      42.320000000000000
     -2.000000000000000      2.000000000000000      36.000000000000000
maximum y i=44.879999999999995 0 second maximum y i=42.32 1 minimum y i=36.0 2
iter=2p=
     -1.200000000000000      2.600000000000000      28.920000000000005
     -2.000000000000000      2.400000000000000      42.320000000000000
     -2.000000000000000      2.000000000000000      36.000000000000000
maximum y i=42.32 1 second maximum y i=36.0 2 minimum y i=28.920000000000005 0
iter=3p=
     -1.200000000000000      2.600000000000000      28.920000000000005
     -0.800000000000001      2.100000000000000      16.160000000000014
     -2.000000000000000      2.000000000000000      36.000000000000000
maximum y i=36.0 2 second maximum y i=28.920000000000005 0 minimum y i=16.160000000000014 1
iter=4p=
     -1.200000000000000      2.600000000000000      28.920000000000005
     -0.800000000000001      2.100000000000000      16.160000000000014
      0.999999999999999      3.050000000000002       5.355000000000019

maximum y i=28.920000000000005 0 second maximum y i=16.160000000000014 1 minimum y i=5.355000000000019 2
iter=5p=
```

```
      1.399999999999998       2.550000000000002       0.655000000000011
     -0.800000000000001       2.100000000000000      16.160000000000014
      0.999999999999999       3.050000000000002       5.355000000000019

maximum y i=16.160000000000014 1 second maximum y i=5.355000000000019 2 minimum y i=0.6550000000000114 0
iter=6p=
      1.399999999999998       2.550000000000002       0.655000000000011
      3.199999999999998       3.500000000000003       3.719999999999994
      0.999999999999999       3.050000000000002       5.355000000000019

maximum y i=5.355000000000019 2 second maximum y i=3.7199999999999944 1 minimum y i=0.6550000000000114 0
iter=7p=
      1.399999999999998       2.550000000000002       0.655000000000011
      3.199999999999998       3.500000000000003       3.719999999999994
      1.649999999999998       3.037500000000002       1.885312500000012

maximum y i=3.7199999999999944 1 second maximum y i=1.8853125000000115 2 minimum y i=0.6550000000000114 0
iter=8p=
      1.399999999999998       2.550000000000002       0.655000000000011
      2.362499999999998       3.146875000000002       1.302519531250004
      1.649999999999998       3.037500000000002       1.885312500000012

maximum y i=1.8853125000000115 2 second maximum y i=1.3025195312500042 1 minimum y i=0.6550000000000114 0
iter=9p=
      1.399999999999998       2.550000000000002       0.655000000000011
      2.362499999999998       3.146875000000002       1.302519531250004
      2.112499999999998       2.659375000000002       0.288925781250000

maximum y i=1.3025195312500042 1 second maximum y i=0.6550000000000114 0 minimum y i=0.2889257812500001 2
iter=10p=
      1.399999999999998       2.550000000000002       0.655000000000011
      1.149999999999998       2.062500000000001      -0.224687499999993
      2.112499999999998       2.659375000000002       0.288925781250000

maximum y i=0.6550000000000114 0 second maximum y i=0.2889257812500001 2 minimum y i=-0.22468749999999327 1
iter=11p=
      1.862499999999998       2.171875000000001      -0.374042968750002
      1.149999999999998       2.062500000000001      -0.224687499999993
      2.112499999999998       2.659375000000002       0.288925781250000

maximum y i=0.2889257812500001 2 second maximum y i=-0.22468749999999327 1 minimum y i=-0.3740429687500022 0
iter=12p=
      1.862499999999998       2.171875000000001      -0.374042968750002
      1.149999999999998       2.062500000000001      -0.224687499999993
      0.899999999999998       1.575000000000000      -0.753749999999998

maximum y i=-0.22468749999999327 1 second maximum y i=-0.3740429687500022 0 minimum y i=-0.7537499999999975 2
iter=13p=
      1.862499999999998       2.171875000000001      -0.374042968750002
      1.612499999999998       1.684375000000000      -0.686386718750004
      0.899999999999998       1.575000000000000      -0.753749999999998

maximum y i=-0.3740429687500022 0 second maximum y i=-0.6863867187500037 1 minimum y i=-0.7537499999999975 2
iter=14p=
      0.649999999999998       1.087499999999999      -0.932187499999998
      1.612499999999998       1.684375000000000      -0.686386718750004
      0.899999999999998       1.575000000000000      -0.753749999999998

maximum y i=-0.6863867187500037 1 second maximum y i=-0.7537499999999975 2 minimum y i=-0.9321874999999984 0
iter=15p=
      0.649999999999998       1.087499999999999      -0.932187499999998
      1.193749999999998       1.507812500000000      -1.079252929687500
      0.899999999999998       1.575000000000000      -0.753749999999998
```

```
maximum y i=-0.7537499999999975 2 second maximum y i=-0.9321874999999984 0 minimum y i=-1.0792529296874995 1
iter=16p=
      0.649999999999998      1.087499999999999      -0.932187499999998
      1.193749999999998      1.507812500000000      -1.079252929687500
      0.943749999999998      1.020312499999998      -1.061674804687500

maximum y i=-0.9321874999999984 0 second maximum y i=-1.0616748046875 2 minimum y i=-1.0792529296874995 1
iter=17p=
      0.859374999999998      1.175781249999999      -1.096405029296875
      1.193749999999998      1.507812500000000      -1.079252929687500
      0.943749999999998      1.020312499999998      -1.061674804687500
……..
iter=49p=
      1.000000969021338      1.250005597923464      -1.124999999956208
      0.999993292813279      1.249984113824493      -1.124999999786506
      1.000019269216091      1.250017992060081      -1.124999999625434

maximum y i=-1.1249999996254345 2 second maximum y i=-1.124999999786506 1 minimum y i=-1.1249999999562075 0
```

If we would like to see amobea movement graphically :

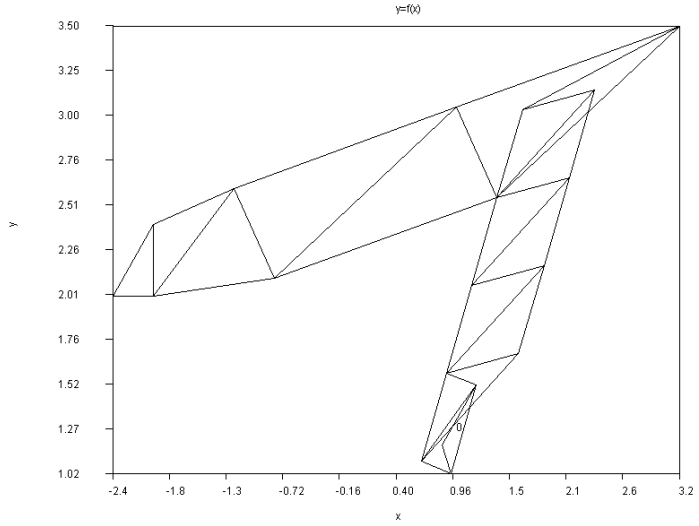
Program 5.9-2 Graphic program to see amobe movement in the example

```
import java.io.*;

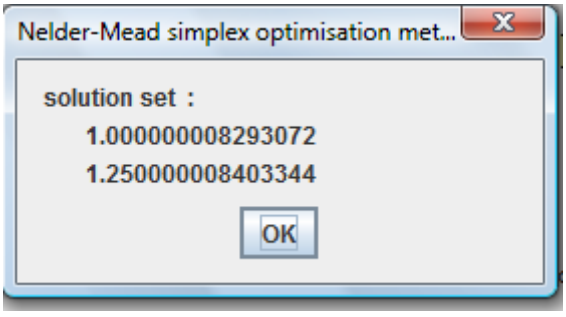
class PlotT01
{
   public static void main (String args[]) throws java.io.IOException
   {
            double x1[]={-2.4,-2,-2,-2.4};
            double y1[]={2,2.4,2,2};
            double x2[]={-1.2,-2,-2,-1.2};
            double y2[]={2.6,2.4,2,2.6};
            double x3[]={-1.2,-0.8,-2,-1.2};
            double y3[]={2.6,2.1,2,2.6};
            double x4[]={-1.2,-0.8,1,-1.2};
            double y4[]={2.6,2.1,3.05,2.6};
            double x5[]={1.4,-0.8,1,1.4};
            double y5[]={2.55,2.1,3.05,2.55};
            double x6[]={1.4,3.2,1,1.4};
            double y6[]={2.55,3.5,3.05,2.55};
            double x7[]={1.4,3.2,1.65,1.4};
            double y7[]={2.55,3.5,3.0375,2.55};
            double x8[]={1.4,2.3625,1.65,1.4};
            double y8[]={2.55,3.146875,3.0375,2.55};
            double x9[]={1.4,2.3625,2.1125,1.4};
            double y9[]={2.55,3.146875,2.659375,2.55};
            double x10[]={1.4,1.15,2.1125,1.4};
            double y10[]={2.55,2.0625,2.659375,2.55};
            double x11[]={1.8625,1.15,2.1125,1.8625};
            double y11[]={2.171875,2.0625,2.659375,2.171875};
            double x12[]={1.8625,1.15,0.9,1.8625};
            double y12[]={2.171875,2.0625,1.575,2.171875};
            double x13[]={1.8625,1.6125,0.9,1.8625};
            double y13[]={2.171875,1.684375,1.575,2.171875};
            double x14[]={0.65,1.6125,0.9,0.65};
            double y14[]={1.0874,1.684375,1.575,1.0874};
            double x15[]={0.65,1.19375,0.9,0.65};
            double y15[]={1.0874,1.5078125,1.575,1.0874};
            double x16[]={0.65,1.19375,0.94375,0.65};
            double y16[]={1.0874,1.5078125,1.0203125,1.0874};
            double x17[]={0.859375,1.19375,0.94375,0.859375};
            double y17[]={1.17578125,1.5078125,1.0203125,1.17578125};
            Plot pp=new Plot(x1,y1);
   pp.addData(x2,y2);
```

```
    pp.addData(x3,y3);
    pp.addData(x4,y4);
    pp.addData(x5,y5);
    pp.addData(x6,y6);
    pp.addData(x7,y7);
    pp.addData(x8,y8);
    pp.addData(x9,y9);
    pp.addData(x10,y10);
    pp.addData(x11,y11);
    pp.addData(x12,y12);
    pp.addData(x13,y13);
    pp.addData(x14,y14);
    pp.addData(x15,y15);
    pp.addData(x16,y16);
    pp.addData(x17,y17);
     pp.plot();
     }
}
```

Nelder-Mead actual amobea movement for the example function



result :



EXAMPLE PROBLEM

As a second example factory space selection problem is taken. In this problem, every city has a number of potential customer. In order to minimize transportation cost of the product new factory should be located in a place minimizing the following function

$$f = \sum_{i=1}^{n} N_i[(x - x_i)^2 + (y - y_i)^2]$$

Where N_i population of the city i and , x_i and y_i are location of the city. x,y are the location of the factory. Following city location function is given as a test function in file city.txt

Population	X coordinate of the city	Y coordinate of the city
10	0	0
10	0	1
10	1	0
10	1	1

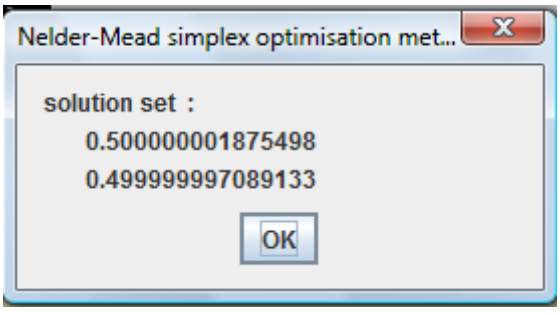
It is clear that optimum point for such a data will be located at [0.5,0.5]. In the example program nelder mead method is taken from class NA41

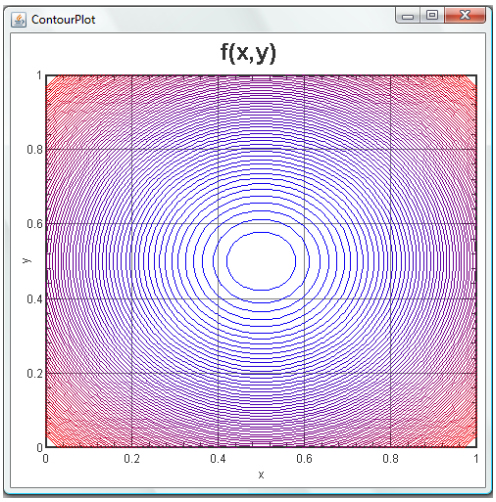
Program 5.9-3 factory placement optimization

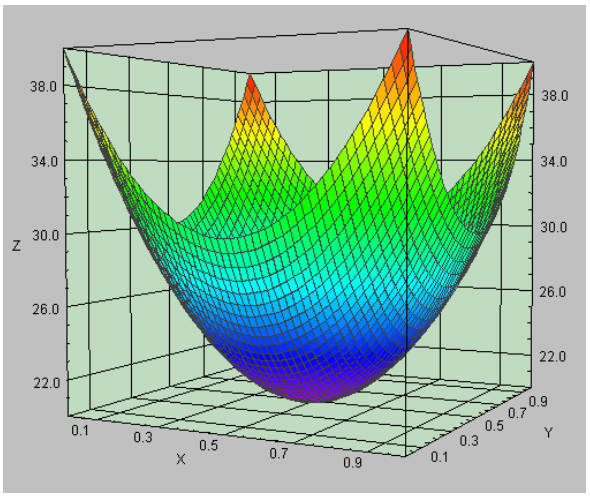
```
 import java.util.*;
import java.awt.*;
import java.applet.Applet;
import java.awt.event.*;
import javax.swing.*;
class ff1 extends f_xj
{
double a[][];
public ff1(String s)
{a=Text.readDoubleT(s);}

public double func(double x[])
{
//optimization function
double ff=0;
int n=a[0].length;
for(int i=0;i<n;i++)
   ff+=a[0][i]*((x[0]-a[1][i])*(x[0]-a[1][i])+(x[1]-a[2][i])*(x[1]-a[2][i]));
return ff;
}
}

public class NA42
{   public static void main (String args[])
    {    String s="enter optimum point first estimation values with one empty space in between: ";
    double [] x0=NA41.inputdata(s);
    double [] dx0=new double[x0.length];
    for(int i=0;i<x0.length;i++) dx0[i]=0.2*x0[i];
    ff1 f=new ff1("city.txt");
    double [] r1= NA41.nelder(f,x0);
    s=" solution set  : \n"+Matrix.toStringT(r1);
    String s2="Nelder-Mead simplex optimization method ";
    JOptionPane.showMessageDialog(null,s,s2,JOptionPane.PLAIN_MESSAGE);
    System.exit(0);
    }
}
```

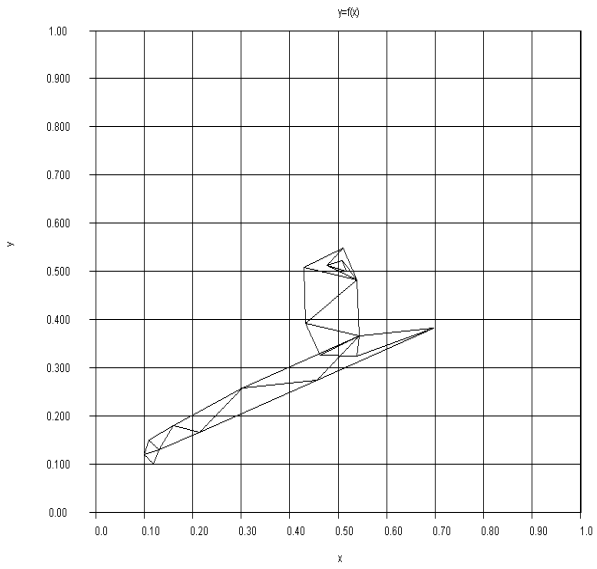







```
---------- Capture Output ----------
> "C:\co\java\bin\java.exe" NA41
iter=1p=
        0.120000000000000       0.100000000000000       32.176000000000000
        0.100000000000000       0.120000000000000       32.176000000000000
        0.100000000000000       0.100000000000000       32.800000000000004
maximum y i=32.800000000000004 2 second maximum y i=32.176 0 minimum y i=32.176 0
iter=2p=
        0.120000000000000       0.100000000000000       32.176000000000000
        0.100000000000000       0.120000000000000       32.176000000000000
        0.130000000000000       0.130000000000000       30.951999999999990
maximum y i=32.176 0 second maximum y i=32.176 1 minimum y i=30.95199999999999 2
iter=3p=
        0.110000000000000       0.150000000000000       30.983999999999995
        0.100000000000000       0.120000000000000       32.176000000000000
        0.130000000000000       0.130000000000000       30.951999999999990
maximum y i=32.176 1 second maximum y i=30.983999999999995 0 minimum y i=30.95199999999999 2
iter=4p=
        0.110000000000000       0.150000000000000       30.983999999999995
        0.160000000000000       0.180000000000000       28.719999999999995
        0.130000000000000       0.130000000000000       30.951999999999990
maximum y i=30.983999999999995 0 second maximum y i=30.95199999999999 2 minimum y i=28.719999999999995 1
iter=5p=
        0.215000000000000       0.165000000000000       27.737999999999985
        0.160000000000000       0.180000000000000       28.719999999999995
        0.130000000000000       0.130000000000000       30.951999999999990
maximum y i=30.95199999999999 2 second maximum y i=28.719999999999995 1 minimum y i=27.737999999999985 0
iter=6p=
        0.215000000000000       0.165000000000000       27.737999999999985
        0.160000000000000       0.180000000000000       28.719999999999995
        0.302500000000000       0.257500000000001       23.912499999999984
maximum y i=28.719999999999995 1 second maximum y i=27.737999999999985 0 minimum y i=23.912499999999984 2
iter=7p=
        0.215000000000000       0.165000000000000       27.737999999999985
        0.456250000000000       0.273750000000001       22.124124999999978
        0.302500000000000       0.257500000000001       23.912499999999984
maximum y i=27.737999999999985 0 second maximum y i=23.912499999999984 2 minimum y i=22.124124999999978 1
iter=8p=
        0.543750000000001       0.366250000000001       20.792124999999988
        0.456250000000000       0.273750000000001       22.124124999999978
        0.302500000000000       0.257500000000001       23.912499999999984
maximum y i=23.912499999999984 2 second maximum y i=22.124124999999978 1 minimum y i=20.792124999999988 0
iter=9p=
        0.543750000000001       0.366250000000001       20.792124999999988
```

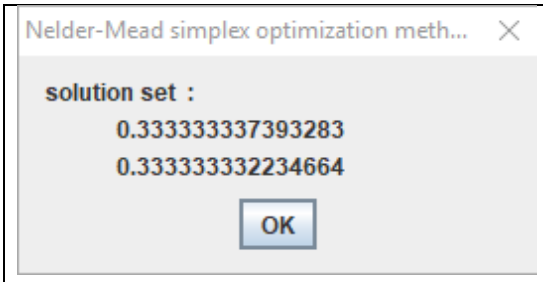
```
    0.456250000000000    0.273750000000001    22.124124999999978
    0.697500000000001    0.382500000000002    22.112499999999997
maximum y i=22.124124999999978 1 second maximum y i=22.112499999999997 2 minimum y i=20.792124999999988 0
iter=10p=
    0.543750000000001    0.366250000000001    20.792124999999988
    0.538437500000001    0.324062500000001    21.297257812499986
    0.697500000000001    0.382500000000002    22.112499999999997
maximum y i=22.112499999999997 2 second maximum y i=21.297257812499986 1 minimum y i=20.792124999999988 0
iter=11p=
    0.543750000000001    0.366250000000001    20.792124999999988
    0.538437500000001    0.324062500000001    21.297257812499986
    0.462890625000000    0.326484375000001    21.259391113281236
maximum y i=21.297257812499986 1 second maximum y i=21.259391113281236 2 minimum y i=20.792124999999988 0
iter=12p=
    0.543750000000001    0.366250000000001    20.792124999999988
    0.433085937500000    0.390976562500001    20.654544067382805
    0.462890625000000    0.326484375000001    21.259391113281236
maximum y i=21.259391113281236 2 second maximum y i=20.792124999999988 0 minimum y i=20.654544067382805 1
iter=13p=
    0.543750000000001    0.366250000000001    20.792124999999988
    0.433085937500000    0.390976562500001    20.654544067382805
    0.539472656250001    0.482871093750001    20.074059600830080
maximum y i=20.792124999999988 0 second maximum y i=20.654544067382805 1 minimum y i=20.07405960083008 2
iter=14p=
    0.428808593750000    0.507597656250001    20.205037628173827
    0.433085937500000    0.390976562500001    20.654544067382805
    0.539472656250001    0.482871093750001    20.074059600830080
maximum y i=20.654544067382805 1 second maximum y i=20.205037628173827 0 minimum y i=20.07405960083008 2
iter=15p=
    0.428808593750000    0.507597656250001    20.205037628173827
    0.509667968750000    0.547363281250001    20.093470001220710
    0.539472656250001    0.482871093750001    20.074059600830080
maximum y i=20.205037628173827 0 second maximum y i=20.09347000122071 1 minimum y i=20.07405960083008 2
iter=16p=
    0.476689453125000    0.511357421875001    20.026894905090334
    0.509667968750000    0.547363281250001    20.093470001220710
    0.539472656250001    0.482871093750001    20.074059600830080
maximum y i=20.09347000122071 1 second maximum y i=20.07405960083008 2 minimum y i=20.026894905090334 0
iter=17p=
    0.476689453125000    0.511357421875001    20.026894905090334
    0.508874511718750    0.522238769531251    20.022932793140413
    0.539472656250001    0.482871093750001    20.074059600830080
maximum y i=20.07405960083008 2 second maximum y i=20.026894905090334 0 minimum y i=20.022932793140413 1
iter=18p=
    0.476689453125000    0.511357421875001    20.026894905090334
    0.508874511718750    0.522238769531251    20.022932793140413
    0.516127319335938    0.499834594726564    20.010404711514713
maximum y i=20.07405960083008 2 second maximum y i=20.026894905090334 0 minimum y i=20.022932793140413 1
iter=52p=
    0.500002657994614    0.499999541116936    20.000000000291020
    0.499999313009312    0.500000148257349    20.000000000019753
    0.499999853685161    0.499998268210034    20.000000000120820
maximum y i=20.00000000029102 0 second maximum y i=20.00000000012082 2 minimum y i=20.000000000019753 1
> Terminated with exit code 0.
```

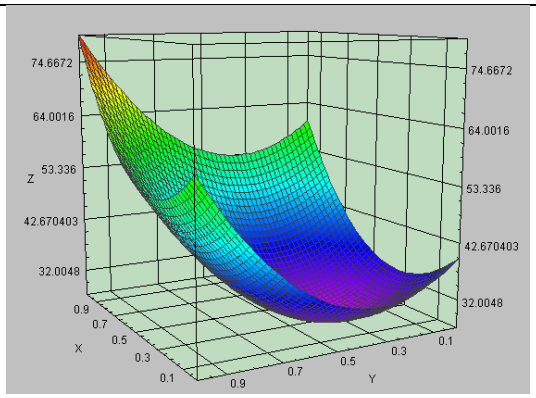


What will happen if the population of one of the city is different, for example

Population	X coordinate of the city	Y coordinate of the city
30	0	0
10	0	1
10	1	0
10	1	1

Then optimum point will move to:





5.10 NON-LINEAR MULTIDIMENSIONAL STEEPEST DESCENT METHOD

For the n dimensional non lineear f(x_0,x_1,x_2,…,x_n) function , we would like to find minimum value of the function. The General definition of the Newton –Raphson approach can be written as:

$$\{x^{m+1}\} = \{x^m\} + \alpha^m [H^m]^{-1} [-\{\nabla f^m\}]$$

where

$$\nabla f(x_1,x_2,x_3,..,x_n) = \begin{Bmatrix} \frac{\partial f(x_1,x_2,x_3,..,x_n)}{\partial x_1} \\ \frac{\partial f(x_1,x_2,x_3,..,x_n)}{\partial x_2} \\ \frac{\partial f(x_1,x_2,x_3,..,x_n)}{\partial x_3} \\ \cdots\cdots \\ \frac{\partial f(x_1,x_2,x_3,..,x_n)}{\partial x_n} \end{Bmatrix} \text{ and } [H] = \begin{bmatrix} \frac{\partial^2 f}{\partial x_1^2} & \frac{\partial^2 f}{\partial x_1 \partial x_2} & \frac{\partial^2 f}{\partial x_1 \partial x_3} & \cdots & \frac{\partial^2 f}{\partial x_1 \partial x_n} \\ \frac{\partial^2 f}{\partial x_2 \partial x_1} & \frac{\partial^2 f}{\partial x_2^2} & \frac{\partial^2 f}{\partial x_2 \partial x_3} & \cdots & \frac{\partial^2 f}{\partial x_2 \partial x_n} \\ \frac{\partial^2 f}{\partial x_3 \partial x_1} & \frac{\partial^2 f}{\partial x_3 \partial x_2} & \frac{\partial^2 f}{\partial x_3^2} & \cdots & \frac{\partial^2 f}{\partial x_3 \partial x_n} \\ \cdots & \cdots & \cdots & \cdots & \cdots \\ \frac{\partial^2 f}{\partial x_1 \partial x_n} & \frac{\partial^2 f}{\partial x_2 \partial x_n} & \frac{\partial^2 f}{\partial x_3 \partial x_n} & \cdots & \frac{\partial^2 f}{\partial x_n^2} \end{bmatrix}$$

is Hessian Matrix, α is a step function. In steepest descent method the value of inverse hessian, $[H]^{-1}$ is taken as unit vector [I] Then the definition of the equation becomes:

$$\{x^{m+1}\} = \{x^m\} + \alpha^m[-\{\nabla f^m\}]$$

In this method, As a first action a starting point for the iteration will be selected. If the starting function is P_0

$P^0 = f(x_1^0, x_2^0, x_3^0, .., x_n^0)$

In each iteration step values of the function and derivatives are calculated.

$d^k = -\nabla f(x_1^k, x_2^k, x_3^k, .., x_n^k)$ k=0..n

Then for each x value is increased by d^k with an unknown multiplier α. In the new equation as only unkown remains is unknown multiplier α (so equation converted to a one dimensional equation as a function of α)

$f(\alpha^k) = f(x^k + \alpha^k d^k) = f\left(\left[x_1^k - \alpha^k \frac{\partial f(x_1^k,x_2^k,x_3^k,..,x_n^k)}{\partial x_1}\right], \left[x_2^k - \alpha^k \frac{\partial f(x_1^k,x_2^k,x_3^k,..,x_n^k)}{\partial x_2}\right], \ldots, \left[x_n^k - \alpha^k \frac{\partial f(x_1^k,x_2^k,x_3^k,..,x_n^k)}{\partial x_n}\right]\right)$

α^k value minimizing function $f(\alpha^k)$ is found by using one of the one dimensional minimisation methods. The new x^{k+1}value calculated by using

$$x^{k+1} = x^k + \alpha^k d^k$$

Process continues until the error is small.

EXAMPLE PROBLEM

Find minimum value of function $f(x_0, x_1) = 3x_0^2 - 4x_0x_1 + 2x_1^2 - x_0 - x_1$with the initial value of $\{x^0\} = \begin{Bmatrix} x_0^0 \\ x_1^0 \end{Bmatrix} = \begin{Bmatrix} -2 \\ 2 \end{Bmatrix}$ by using steepest descent method.

$$\nabla f(x_0, x_1) = \begin{Bmatrix} 6x_0^0 - 4x_1^0 - 1 \\ -4x_0^0 + 4x_1^0 - 1 \end{Bmatrix} = \begin{Bmatrix} -21 \\ 15 \end{Bmatrix}$$

$$\begin{Bmatrix} x_0^1 \\ x_1^1 \end{Bmatrix} = \begin{Bmatrix} x_0^0 \\ x_1^0 \end{Bmatrix} + \alpha^0 \begin{Bmatrix} 6x_0^0 - 4x_1^0 - 1 \\ -4x_0^0 + 4x_1^0 - 1 \end{Bmatrix} = \begin{Bmatrix} -2 \\ 2 \end{Bmatrix} + \alpha^0 \begin{Bmatrix} -21 \\ 15 \end{Bmatrix} = \begin{Bmatrix} -2 - 21\alpha^0 \\ 2 + 15\alpha^0 \end{Bmatrix}$$

If these values are substituted into the function

$f(x_0, x_1) = 3x_0^2 - 4x_0x_1 + 2x_1^2 - x_0 - x_1$ it becomes

$f(\alpha^0) = 3(-2 - 21\alpha^0)^2 - 4(-2 - 21\alpha^0)(2 + 15\alpha^0) + 2(-2 - 21\alpha^0)^2 - (-2 - 21\alpha^0) - (2 + 15\alpha^0)$

$f(\alpha^0) = 36 + 666\alpha^0 + 3033\alpha_0^2$

In order to find minimum of this function, analitical solution will be used in this example

$f'^{(\alpha^0)} = 666 + 6066\alpha^0 = 0 \;\; \alpha^0 = -0.109792284$

If this value is substituted into x_0^1 and x_1^1

For α^0=-0.109792284 $\begin{Bmatrix} x_0^1 \\ x_1^1 \end{Bmatrix} = \begin{Bmatrix} 0.3056379 \\ 0.35311572 \end{Bmatrix}$

For α^1=-1.121212 $\begin{Bmatrix} x_0^2 \\ x_1^2 \end{Bmatrix} = \begin{Bmatrix} 0.9544105 \\ 1.2613973 \end{Bmatrix}$

For α^2=-0.109792284 $\begin{Bmatrix} x_0^3 \\ x_1^3 \end{Bmatrix} = \begin{Bmatrix} 0.9894 \\ 1.2363 \end{Bmatrix}$

For α^3=-1.121212 $\begin{Bmatrix} x_0^4 \\ x_1^4 \end{Bmatrix} = \begin{Bmatrix} 0.9993 \\ 1.250173 \end{Bmatrix}$

The same problem is also solved in the algorithm

Program 5.10-1 steepest descent algorithm

```
 // Multidimensional non-Linear optimization
// Steepest descent method
//
import javax.swing.*;
import java.util.*;
import java.awt.*;
import java.applet.Applet;
import java.awt.event.*;
import java.io.*;


class f1dim implements if_x
{
  // pseudofunction to convert multidimensional function to
  // one dimensional function
  if_xj ff1;
  double pc[];
  double xc[];
  double xt[];
  int n;
  public f1dim(if_xj ff,double pi[],double xi[])
  { n=pi.length;
    ff1=ff;
            pc=new double[n];
    xc=new double[n];
    xt=new double[n];
             for(int i=0;i<n;i++)
               {pc[i]=pi[i];xc[i]=xi[i];}
  }

  public double func(double x)
  {
            for (int j=0;j<n;j++) xt[j]=pc[j]+x*xc[j];
            return ff1.func(xt);
  }
}
public class NA43_i
{
public static double SIGN(double a,double b) {return (b > 0.0 ? Math.abs(a) : -Math.abs(a));}
public static double MAX(double a,double b) {return (a > b ? a : b);}

// 3. brent one dimensional optimization method

public  static double[] brentf(if_x f,double ax,double bx,double cx,double tol)
{
int ITMAX=100;
double CGOLD=(3.0-Math.sqrt(5))/2.0; //altin oran
double ZEPS=1.0e-10;
```

```
double xmin;
double aa[]=new double[2];
        int iter;
        double a,b,d,etemp,fu,fv,fw,fx,p,q,r,tol1,tol2,u,v,w,x,xm;
        double e=0.0;
        d=0.0;
        a=((ax < cx) ? ax : cx);
        b=((ax > cx) ? ax : cx);
        x=w=v=bx;
        fw=fv=fx=f.func(x);
        for (iter=1;iter<=ITMAX;iter++) {
                xm=0.5*(a+b);
                tol2=2.0*(tol1=tol*Math.abs(x)+ZEPS);
                if (Math.abs(x-xm) <= (tol2-0.5*(b-a))) {
                        xmin=x;
            aa[0]=xmin;
            aa[1]=fx;
            return aa;
                }
                if (Math.abs(e) > tol1) {
                        r=(x-w)*(fx-fv);
                        q=(x-v)*(fx-fw);
                        p=(x-v)*q-(x-w)*r;
                        q=2.0*(q-r);
                        if (q > 0.0) p = -p;
                        q=Math.abs(q);
                        etemp=e;
                        e=d;
                        if (Math.abs(p) >= Math.abs(0.5*q*etemp) || p <= q*(a-x) || p >= q*(b-x))
                                d=CGOLD*(e=(x >= xm ? a-x : b-x));
                        else {
                                d=p/q;
                                u=x+d;
                                if (u-a < tol2 || b-u < tol2)
                                        d=SIGN(tol1,xm-x);
                        }
                } else {
                        d=CGOLD*(e=(x >= xm ? a-x : b-x));
                }
                u=(Math.abs(d) >= tol1 ? x+d : x+SIGN(tol1,d));
                fu=f.func(u);
                if (fu <= fx) {
                        if (u >= x) a=x; else b=x;
                        {v=w;w=x;x=u;}
                        {fv=fw;fw=fx;fx=fu;}
                } else {
                        if (u < x) a=u; else b=u;
                        if (fu <= fw || w == x) {
                                v=w;
                                w=u;
                                fv=fw;
                                fw=fu;
                        } else if (fu <= fv || v == x || v == w) {
                                v=u;
                                fv=fu;
                        }
                }
```

```
            }
            System.out.println("maximum iteration limit in BRENT method is exceeded");
            xmin=x;
            aa[0]=xmin;
            aa[1]=fx;
            return aa;
}

public  static double brent(if_x f,double ax,double bx,double cx,double tol)
{
 double aa[]=brentf(f,ax,bx,cx,tol);
 return aa[0];
}

// 3. Steepest descent
//_____________________________________________________________________
public static double[] steepestdescent( if_xj f,double x[])
{
// steepestdescent method
// uses linmin and brent methods
double ti=1.0;
int i,ii,jj;
int nmax=500;
double tolerance=1.0e-15;
int n=x.length;
double b[];
b=new double[n];
double lambda;
double dy[]=new double[n];
i=0;
for(i=0;i<n;i++)
{
b[i]=1.0;
}
i=0;
while( i++ < nmax && Matrix.abs(b) > tolerance )
{
   dy=f.dfunc(x);
   lambda=linmin(f,x,dy);
   b=Matrix.multiply(dy,lambda);
   x=Matrix.add(x,b);
   System.out.println("dy="+Matrix.toString(dy));
   System.out.println("i= "+i+"alfa="+lambda+"\n"+Matrix.toString(x));
}
if(i >= nmax) JOptionPane.showMessageDialog(null,"Uyari maximum iterasyon sayisi aşildi \n"+
  " çözüm geçerli olmiyabilir","MAKSİMUM ITERASYON SAYISI UYARISI",JOptionPane.WARNING_MESSAGE);
return x;
}
public  static double[] mnbrak(if_x f,double ax,double dx)
{
// dx ikinci derece polinom için gerekli iki ek noktayi oluşturmak için
// ana noktadan seçilen uzaklik
return mnbrak(f,ax,(ax-dx),(ax+dx));
}

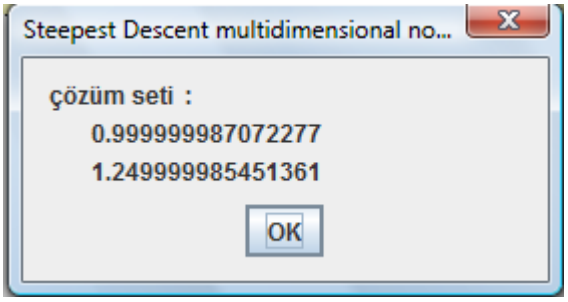
public  static double[] mnbrak(if_x f,double ax)
{
```

```
// dx ikinci derece polinom için gerekli iki ek noktayi oluşturmak için
// ana noktadan seçiln uzaklik
return mnbrak(f,ax,(ax-0.1),(ax+0.1));
}

public  static double[] mnbrak(if_x f,double ax,double bx,double cx)
{
// ax,bx,cx üç adet tahmin noktasi verildiğinde içinde minimum bulunan ax,bx,cx üç adetnoktayi bize arama
// sonucu bulur
// f : fonksiyon (f_x abstract sinifinda tanimlanmiş)
// tol tolerans
int ITMAX=100;
double xmin;
double GOLD=(Math.sqrt(5.0)+1.0)/2.0;;
double GLIMIT=100.0;
double TINY=1.0e-20;
double fa,fb,fc;
fa=0;
fb=0;
fc=0;
        double ulim,u,r,q,fu,dum;
        double aa[]=new double[3];
        fa=f.func(ax);
        fb=f.func(bx);
        if (fb > fa) {
                //SHFT(dum,*ax,*bx,dum)
                {dum=ax;ax=bx;bx=dum;}
                //SHFT(dum,*fb,*fa,dum)
                {dum=fb;fb=fa;fa=dum;}
        }
        cx=(bx)+GOLD*(bx-ax);
        fc=f.func(cx);
                aa[0]=ax;
                aa[1]=bx;
                aa[2]=cx;
        while (fb > fc) {
                r=(bx-ax)*(fb-fc);
                q=(bx-cx)*(fb-fa);
                u=(bx)-((bx-cx)*q-(bx-ax)*r)/
                        (2.0*SIGN(MAX(Math.abs(q-r),TINY),q-r));
                ulim=(bx)+GLIMIT*(cx-bx);
                if ((bx-u)*(u-cx) > 0.0) {
                        fu=f.func(u);
                        if (fu < fc) {
                                ax=bx;
                                bx=u;
                                fa=fb;
                                fb=fu;
                                return aa;
                        } else if (fu > fb) {
                                cx=u;
                                fc=fu;
                                return aa;
                        }
                        u=(cx)+GOLD*(cx-bx);
                        fu=f.func(u);
                } else if ((cx-u)*(u-ulim) > 0.0) {
```

```
                                fu=f.func(u);
                                if (fu < fc) {
                                        //SHFT(bx,cx,u,cx+GOLD*(cx-bx))
                                        {bx=cx;cx=u;u=cx+GOLD*(cx-bx);}
                                        //SHFT(fb,fc,fu,f.func(u))
                                        {fb=fc;fc=fu;fu=f.func(u);}
                                }
                        } else if ((u-ulim)*(ulim-cx) >= 0.0) {
                                u=ulim;
                                fu=f.func(u);
                        } else {
                                u=cx+GOLD*(cx-bx);
                                fu=f.func(u);
                        }
                        //SHFT(ax,bx,cx,u)
                        {ax=bx;bx=cx;cx=u;}
                        //SHFT(fa,fb,fc,fu)
                        {fa=fb;fb=fc;fc=fu;}
                        aa[0]=ax;
                        aa[1]=bx;
                        aa[2]=cx;
        }
                        return aa;
}
public static double linmin(if_xj f,double p[],double xi[])
{
        double tol=2.0e-4;
   int n=p.length;
   double pcom[],xicom[];
        int j;
        double xx,xmin,bx,ax;
        f1dim f1=new f1dim(f,p,xi);
        ax=0.0;
        xx=1.0;
        bx=2.0;
        double aa[]=mnbrak(f1,ax,xx,bx);
        ax=aa[0];
        xx=aa[1];
        bx=aa[2];
        xmin=brent(f1,ax,xx,bx,tol);
        return xmin;
}
//input data method
//_____________________________________________________________
    public static double[] inputdata(String s)
    {
    String s1=JOptionPane.showInputDialog(s);
    StringTokenizer token=new StringTokenizer(s1);
    int n=token.countTokens()-1;
    int m=n+1;
    double a[]=new double[m];
    int j=0;
    while(token.hasMoreTokens())
    {
    Double ax=new Double(token.nextToken());
    a[j++]=ax.doubleValue();
    }
```

```
    return a;
    }
    public static void main (String args[])
    {
    String s="enter optimum point first estimation values with one empty space in between: ";
    double [] x0=inputdata(s);
    if_xj f=(double x[])->3.0*x[0]*x[0]-4.0*x[0]*x[1]+2.0*x[1]*x[1]-x[0]-x[1];
    double [] r1= steepestdescent(f,x0);
    s=" çözüm seti  : \n"+Matrix.toStringT(r1);
    String s2="Steepest Descent multidimensional non-linear optimization method:";
    JOptionPane.showMessageDialog(null,s,s2,JOptionPane.PLAIN_MESSAGE);
    System.exit(0);
    } }
```



Program 5.10-2 steepest descent algorithm (Python version)

```
 from math import *
from f_xj import *;
from f_x import *;
from gauss import *;

class f1(f_xj):
  def func(self,x):
     y=3.0*x[0]*x[0]-4.0*x[0]*x[1]+2.0*x[1]*x[1]-x[0]-x[1];
     return y;

class f1dim(f_x):
   def __init__(self,ff,pi,xi):
     self.name = 'f1dim';
     self.ff1=ff;
     self.pc=pi;
     self.xc=xi;
   def func(self,x):
       n=len(self.pc);
       xt = [0.0 for ii in range(n)];
       for j in range(n):
          xt[j]=self.pc[j]+x*self.xc[j];
       return self.ff1.func(xt);

def linmin(f,p,xi):
   f1=f1dim(f,p,xi);
   xi=1.0;
   xmin=newton(f1,xi);
   return xmin;

def multiply_c_v(left,right):
   #multiplying a vector with a constant
   n=len(right);
   b = [0.0 for i in range(n)];
   for i in range(n):
      b[i]=left*right[i];
```

```
    return b;

def add(left,right):
    #addition of two vectors
    n1=len(left);
    n2=len(right);
    if n1>=n2: nMax=n1;
    else     : nMax=n2;
    b = [0.0 for i in range(nMax)];
    for i in range(n1):
        b[i]=b[i]+left[i];
    for i in range(n2):
        b[i]=b[i]+right[i];
    return b;

def steepestdescent(f,x):
    # steepestdescent method
    ti=1.0;
    nmax=100;
    tolerance=1.0e-10;
    n=len(x);
    b = [0.0 for ii in range(n)];
    dy = [0.0 for ii in range(n)];
    i=0;
    total=0;
    for i in range(n):
        b[i]=1.0;
        total+=abs(b[i]);
    i=0;
    while i<nmax and total>tolerance:
        dy=f.dfunc(x);
        lm=linmin(f,x,dy);
        b= multiply_c_v(lm,dy);
        x=add(x,b);
        total=0;
        for j in range(n):
            total+=abs(b[j]);
        i=i+1;
    return x;

def newton(f,x):
   # Newton-Raphson optimisation method
   # single variable
  nmax=100
  tolerance=1.0e-10
  for i in range(0,nmax):
      fx= f.dfunc(x,1);
      dfx=f.dfunc(x,2);
      x=x-fx/dfx
      if abs(fx)<tolerance: return x
  return x;

x=[1,1];
f=f1();
r=steepestdescent(f,x);
print("r=",r);
```

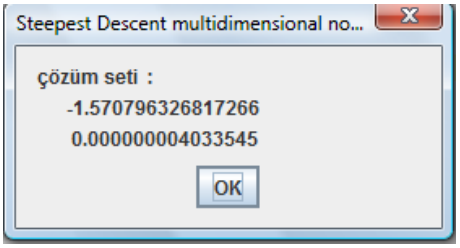
```
runfile('D:/okul/SCO1/steepest_descent_opt.py', wdir='D:/okul/SCO1')
Reloaded modules: f_xj, f_x, gauss
r= [1.0000095779092373, 1.2500087681731902]
```

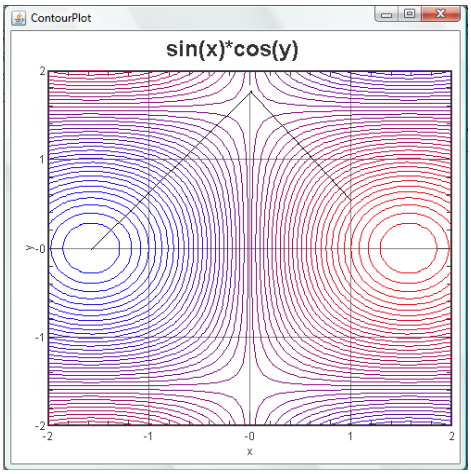
EXAMPLE PROBLEM

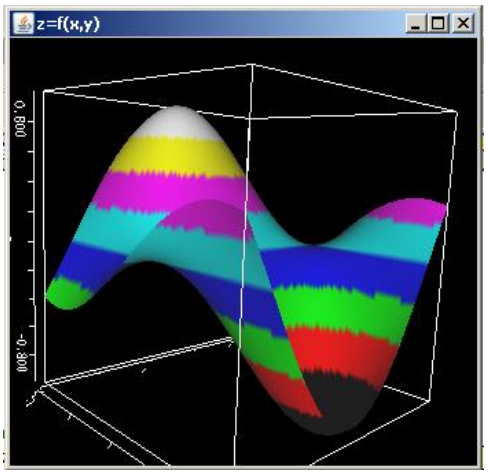
Find the minimum of the function f(x0,x1)=sin(x0)cos(x1) with the initial starting point {x}=[1,0.5]

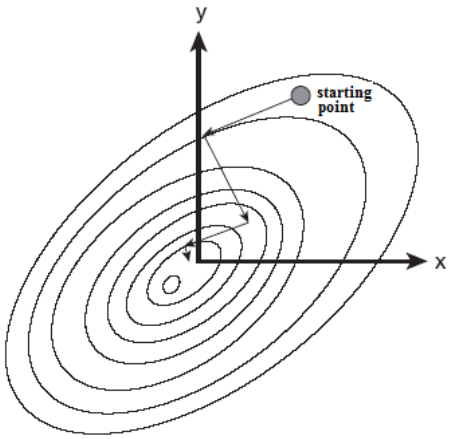
Sample function used in the problem

```
public static void main (String args[])
{
String s="enter optimum point first estimation values with one empty space in between: ";
double [] x0=inputdata(s);
if_xj f=(double x[])->(Math.sin(x[0])*Math.cos(x[1]));
double [] r1= steepestdescent(f,x0);
s=" çözüm seti  : \n"+Matrix.toStringT(r1);
String s2="Steepest Descent multidimensional non-linear optimization method:";
JOptionPane.showMessageDialog(null,s,s2,JOptionPane.PLAIN_MESSAGE);
System.exit(0);
} }
```









5.11 MODIFIED NEWTON METHOD(NEWTON RAPHSON STEEPEST DESCENT COMBINED METHOD)

If the initial guess is close to the actual value Newton-Raphson method performs well. But when it is far from the actual solution it is not behave so well. Modified Newton method combines Newton-Raphson method with steepest descent method to get a better convergence performance. Remembering general definition of optimizing equation

$$\{x^{m+1}\} = \{x^m\} + \alpha^m [H^m]^{-1} [-\{\nabla f^m\}]$$

Now both α and Hessian matrix can be taken into the calculation steps to get a combined method.

Program 5.11-1 Modified Newton Method(Combined Newton-Raphson and Steepest descent method)

```
 // Modified Newton method
// uses Newton-Raphson together with Steepest descent
import java.util.*;
import java.awt.*;
import java.applet.Applet;
import java.awt.event.*;
import javax.swing.*;

public class NA38A
{
public static double SIGN(double a,double b) {return (b > 0.0 ? Math.abs(a) : -Math.abs(a));}
public static double MAX(double a,double b) {return (a > b ? a : b);}
public static double[] dampednewton(if_xj f,double x[])
{
// Modified Newton method
// uses Newton-Raphson together with Steepest descent
// Steepest descent also uses linmin and brent methods of one dimensionaloptimisation
double ti=1.0;
int i,ii,jj;
int nmax=800;
double tolerance=1.0e-14;
int n=x.length;
double b[];
b=new double[n];
double lambda;
double g[]=new double[n];
double G[][]=f.d2func(x);//Hessian matrix
double g1,g2,g3,g4,g5;
//dy=new double[n][n];
i=0;
for(i=0;i<n;i++)
{
b[i]=1.0;
}
while( i++ < nmax && Matrix.abs(b) > tolerance )
{   g=f.dfunc(x);
   g1=Matrix.VT_V(g);
   g2=Matrix.VT_G_V(g,G);
   g1*=g1;
   g3=g1/g2;
   g4=Matrix.VT_G_V(g,Matrix.inverse(G));
   lambda=linmin(f,x,g);
   if(g4>Math.max(g3,0))
       b=Matrix.multiply(Matrix.divide(g,G),lambda);
   else
```

```
        b=Matrix.multiply(g,lambda);
    x=Matrix.add(x,b);
}
if(i >= nmax) JOptionPane.showMessageDialog(null,"WArn,ng : Maximum number of iteration exceeded \n"+
  " result may not be valid","MAXIMUM ITERATION WARNING",JOptionPane.WARNING_MESSAGE);
return x;
}


public  static double[] mnbrak(if_x f,double ax,double dx)
{
return mnbrak(f,ax,(ax-dx),(ax+dx));
}

public  static double[] mnbrak(if_x f,double ax)
{
// dx ikinci derece polinom için gerekli iki ek noktayı oluşturmak için
// ana noktadan seçiln uzaklık
return mnbrak(f,ax,(ax-0.1),(ax+0.1));
}

public  static double[] mnbrak(if_x f,double ax,double bx,double cx)
{
// ax,bx,cx üç adet tahmin noktası verildiğinde içinde minimum bulunan ax,bx,cx üç adetnoktayı bize arama
// sonucu bulur
// f : fonksiyon (if_x abstract sınıfında tanımlanmış)
// tol tolerans
int ITMAX=100;
double xmin;
double GOLD=(Math.sqrt(5.0)+1.0)/2.0;;
double GLIMIT=100.0;
double TINY=1.0e-20;
double fa,fb,fc;
fa=0;
fb=0;
fc=0;
            double ulim,u,r,q,fu,dum;
            double aa[]=new double[3];
            fa=f.func(ax);
            fb=f.func(bx);
            if (fb > fa) {
                        //SHFT(dum,*ax,*bx,dum)
                        {dum=ax;ax=bx;bx=dum;}
                        //SHFT(dum,*fb,*fa,dum)
                        {dum=fb;fb=fa;fa=dum;}
            }
            cx=(bx)+GOLD*(bx-ax);
            fc=f.func(cx);
                        aa[0]=ax;
                        aa[1]=bx;
                        aa[2]=cx;
            while (fb > fc) {
                        r=(bx-ax)*(fb-fc);
                        q=(bx-cx)*(fb-fa);
                        u=(bx)-((bx-cx)*q-(bx-ax)*r)/
                                    (2.0*SIGN(MAX(Math.abs(q-r),TINY),q-r));
                        ulim=(bx)+GLIMIT*(cx-bx);
```

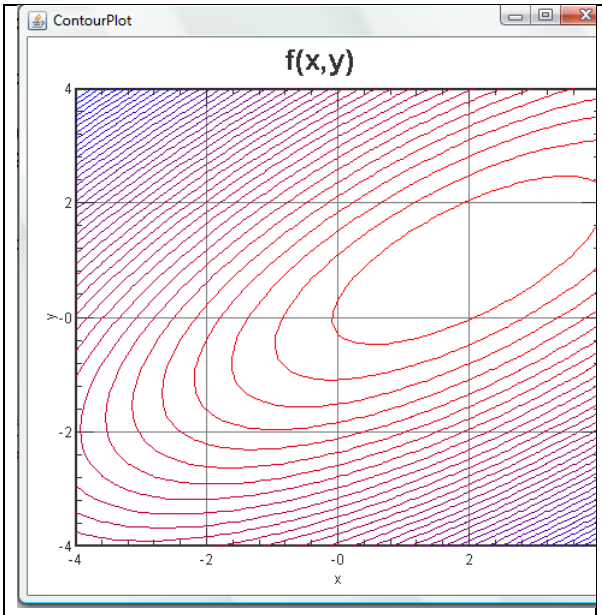
```
                        if ((bx-u)*(u-cx) > 0.0) {
                                fu=f.func(u);
                                if (fu < fc) {
                                        ax=bx;
                                        bx=u;
                                        fa=fb;
                                        fb=fu;
                                        return aa;
                                } else if (fu > fb) {
                                        cx=u;
                                        fc=fu;
                                        return aa;
                                }
                                u=(cx)+GOLD*(cx-bx);
                                fu=f.func(u);
                        } else if ((cx-u)*(u-ulim) > 0.0) {
                                fu=f.func(u);
                                if (fu < fc) {
                                        //SHFT(bx,cx,u,cx+GOLD*(cx-bx))
                                        {bx=cx;cx=u;u=cx+GOLD*(cx-bx);}
                                        //SHFT(fb,fc,fu,f.func(u))
                                        {fb=fc;fc=fu;fu=f.func(u);}
                                }
                        } else if ((u-ulim)*(ulim-cx) >= 0.0) {
                                u=ulim;
                                fu=f.func(u);
                        } else {
                                u=cx+GOLD*(cx-bx);
                                fu=f.func(u);
                        }
                        //SHFT(ax,bx,cx,u)
                        {ax=bx;bx=cx;cx=u;}
                        //SHFT(fa,fb,fc,fu)
                        {fa=fb;fb=fc;fc=fu;}
                        aa[0]=ax;
                        aa[1]=bx;
                        aa[2]=cx;
                }
                        return aa;
}
public  static double[] brentf(if_x f,double ax,double bx,double cx,double tol)
{
// ax,bx,cx üç adet tahmin noktası (bxax ile cx arasında yer almalıdır)
// f : fonksiyon (if_x abstract sınıfında tanımlanmış)
// tol tolerans
int ITMAX=100;
double CGOLD=(3.0-Math.sqrt(5))/2.0; //altın oran
double ZEPS=1.0e-10;
double xmin;
double aa[]=new double[2];
// SHFT(double a,double b,double c,d) (a)=(b);(b)=(c);(c)=(d);
        int iter;
        double a,b,d,etemp,fu,fv,fw,fx,p,q,r,tol1,tol2,u,v,w,x,xm;
        double e=0.0;
        d=0.0;
        a=((ax < cx) ? ax : cx);
        b=((ax > cx) ? ax : cx);
```

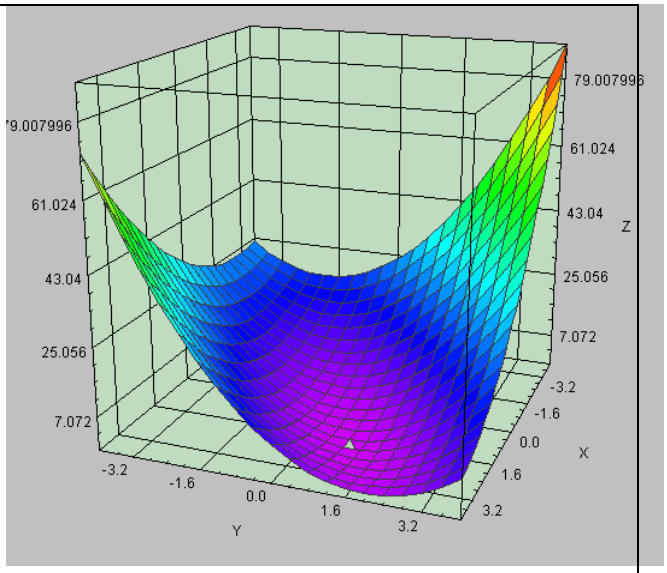
```
		x=w=v=bx;
		fw=fv=fx=f.func(x);
		for (iter=1;iter<=ITMAX;iter++) {
			xm=0.5*(a+b);
			tol2=2.0*(tol1=tol*Math.abs(x)+ZEPS);
			if (Math.abs(x-xm) <= (tol2-0.5*(b-a))) {
				xmin=x;
		aa[0]=xmin;
		aa[1]=fx;
		return aa;
			}
			if (Math.abs(e) > tol1) {
				r=(x-w)*(fx-fv);
				q=(x-v)*(fx-fw);
				p=(x-v)*q-(x-w)*r;
				q=2.0*(q-r);
				if (q > 0.0) p = -p;
				q=Math.abs(q);
				etemp=e;
				e=d;
				if (Math.abs(p) >= Math.abs(0.5*q*etemp) || p <= q*(a-x) || p >= q*(b-x))
					d=CGOLD*(e=(x >= xm ? a-x : b-x));
				else {
					d=p/q;
					u=x+d;
					if (u-a < tol2 || b-u < tol2)
						d=SIGN(tol1,xm-x);
				}
			} else {
				d=CGOLD*(e=(x >= xm ? a-x : b-x));
			}
			u=(Math.abs(d) >= tol1 ? x+d : x+SIGN(tol1,d));
			fu=f.func(u);
			if (fu <= fx) {
				if (u >= x) a=x; else b=x;
				{v=w;w=x;x=u;}
				{fv=fw;fw=fx;fx=fu;}
			} else {
				if (u < x) a=u; else b=u;
				if (fu <= fw || w == x) {
					v=w;
					w=u;
					fv=fw;
					fw=fu;
				} else if (fu <= fv || v == x || v == w) {
					v=u;
					fv=fu;
				}
			}
		}
		System.out.println("Maximum iteration is exceeded in berent optimisation method");
		xmin=x;
		aa[0]=xmin;
		aa[1]=fx;
		return aa;
}
```

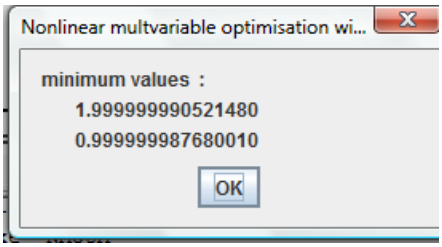
```
public  static double brent(if_x f,double ax,double bx,double cx,double tol)
{
 double aa[]=brentf(f,ax,bx,cx,tol);
 return aa[0];
}
public static double linmin(if_xj f,double p[],double xi[])
{
            double tol=2.0e-4;
   int n=p.length;
   double pcom[],xicom[];
            int j;
            double xx,xmin,bx,ax;
            f1dim f1=new f1dim(f,p,xi);
            ax=0.0;
            xx=1.0;
            bx=2.0;
            double aa[]=mnbrak(f1,ax,xx,bx);
            ax=aa[0];
            xx=aa[1];
            bx=aa[2];
            xmin=brent(f1,ax,xx,bx,tol);
            return xmin;
}

public static double[][] linminiter(if_xj f,double p[],double xi[])
{   // gives new iteration point
   int n=p.length;
   double xmin=linmin(f,p,xi);
   double aa[][]=new double[2][n];
            for (int j=0;j<n;j++)
            {   xi[j] *= xmin;
                        p[j] += xi[j];
                        aa[0][j]=p[j];
                        aa[1][j]=xi[j];
            }
   return aa;
}
// veri giriş çıkış metodu
//_____________________________________________________________________
    public static double[] inputdata(String s)
    {
    String s1=JOptionPane.showInputDialog(s);
    StringTokenizer token=new StringTokenizer(s1);
    int n=token.countTokens()-1;
    int m=n+1;
    double a[]=new double[m];
    int j=0;
    while(token.hasMoreTokens())
    {
    Double ax=new Double(token.nextToken());
    a[j++]=ax.doubleValue();
    }
    return a;
    }
    public static void main (String args[])
    {
            String s="enter optimum point first estimation values with one empty space in between: ";
```

```
    double [] x0=inputdata(s);
    if_xj f=(double x[])->-(2.0*x[0]*x[1]+2.0*x[0]-x[0]*x[0]-2.0*x[1]*x[1]);
    double r1[]=dampednewton(f,x0);
    s=" minimum values  : \n"+Matrix.toStringT(r1);
    String s2="Nonlinear multvariable optimisation with Newton+Steepest descent:";
    JOptionPane.showMessageDialog(null,s,s2,JOptionPane.PLAIN_MESSAGE);
    surfacePlot sp=new surfacePlot(f,-4,4,-4,4);
    sp.plot();
    ContourPlot cp=new ContourPlot(f,-4,4,40,-4,4,40,40,false,"z=f(x,y)","x","y");
    cp.plot();
    }
}
```







5.12 NON-LINEAR MULTIDIMENSIONAL FLETCHER-REEVES CONJUGATE GRADIENT METHOD

$d^0 = g^0$

$g^k = \nabla f\left(x_1^k, x_2^k, x_3^k, .., x_n^k\right), k = 0, .., n$

For k=0 $\{x^{k+1}\} = \{x^k\} + \alpha^k\{d^k\}$

For the next steps $1 \le k \le n$

$$\beta^k = \frac{(g^k)^T(g^k)}{(g^{k-1})^T(g^{k-1})}$$

$$d^k = [-\{g^k\}] + \beta^k d^{k-1}$$
$$\{x^{k+1}\} = \{x^k\} + \alpha^k d^k$$

EXAMPLE PROBLEM

$$\{x^{k+1}\} = \{x^k\} + \alpha^k [H^k]^{-1}[-\{\nabla f^k\}] = \{x^k\} + \alpha^k [H^k]^{-1}[-g^k] = \{x^k\} + \alpha^k \{d^k\}$$

Our problem is finding minimum of function $f(x_0,x_1)=3x_0^2-4x_0x_1+2x_1^2-x_0-x_1$ starting from {x}=[-2,2]

$$\nabla f = \begin{Bmatrix} 6x_0 - 4x_1 - 1 \\ 4x_1 - 4x_0 - 1 \end{Bmatrix} = \begin{Bmatrix} -21 \\ 15 \end{Bmatrix}$$

$$\begin{Bmatrix} x_0^0 \\ x_1^0 \end{Bmatrix} = \begin{Bmatrix} -2 \\ 2 \end{Bmatrix}$$

$$\begin{Bmatrix} x_0^1 \\ x_1^1 \end{Bmatrix} = \begin{Bmatrix} x_0^0 \\ x_1^0 \end{Bmatrix} + \alpha \begin{Bmatrix} 6x_0^0 - 4x_1^0 - 1 \\ 4x_1^0 - 4x_0^0 - 1 \end{Bmatrix}$$

$$\begin{Bmatrix} x_0^1 \\ x_1^1 \end{Bmatrix} = \begin{Bmatrix} -2 \\ 2 \end{Bmatrix} + \alpha^0 \begin{Bmatrix} -21 \\ 15 \end{Bmatrix}$$

$$\begin{Bmatrix} x_0^1 \\ x_1^1 \end{Bmatrix} = \begin{Bmatrix} -2 - 21\alpha^0 \\ 2 + 15\alpha^0 \end{Bmatrix}$$

Substituting this values into the equation

$f(x_0,x_1)=3x_0^2-4x_0x_1+2x_1^2-x_0-x_1$

$y=3(-2-21\ \alpha)^2-4(-2\ -21\alpha)(2\ +15\alpha)+2(2\ +15\alpha)^2-(-2\ -21\alpha)-(2\ +15\alpha)$

$y = 36\ +666\alpha+3033\ \alpha^2$

Minimum of this one dimensional function is needed. In the example programs brent method is used, in the example just analytical derivative method will be use to solve the minimum

$y' = 666+6066\ \alpha = 0 \quad \alpha=-0.109792284$

substituting α back into the x_0^1 and x_1^1 gives

α^0=-0.109792284 için $\begin{Bmatrix} x_0^1 \\ x_1^1 \end{Bmatrix} = \begin{Bmatrix} 0.305637982195846 \\ 0.353115727002967 \end{Bmatrix}$

So far, what is applied was steepest descent method. For the next iteration Flether-Reeves step can be taken

$$g = \nabla f = \begin{Bmatrix} 6x_0 - 4x_1 - 1 \\ 4x_1 - 4x_0 - 1 \end{Bmatrix} = \begin{Bmatrix} -0.578635014836797 \\ -0.810089020771512 \end{Bmatrix}$$

$$\beta^k = \frac{(g^k)^T (g^k)}{(g^{k-1})^T (g^{k-1})} = \frac{0.9910627019697276}{666} = 0.001488082135089681$$

$$d^k = -g^k + \beta^k d^{k-1} = \begin{Bmatrix} 0.60988473967368 \\ 0.787767788745167 \end{Bmatrix}$$

$$x^{k+1} = x^k + \alpha^k d^k = \begin{Bmatrix} 0.3056379 \\ .35311572 \end{Bmatrix} + \alpha \begin{Bmatrix} 0.60988473967368 \\ 0.787767788745167 \end{Bmatrix}$$

$$\alpha^1 = 1.138513513513$$

$$\begin{Bmatrix} x_0^2 \\ x_1^2 \end{Bmatrix} = \begin{Bmatrix} 1.000000000000001 \\ 1.24999999999998 \end{Bmatrix}$$

Since the function taken is almost linear the result is almost equal to solution values.

5.12-1 Fletcher-Reeves Conjugate Gradient method (NA43B.java)

```
import java.util.*;
import java.awt.*;
import java.applet.Applet;
import java.awt.event.*;
```

```
import javax.swing.*;
class f1dim implements if_x
{
  // pseudofunction to convert multidimensional function to
  // one dimensional function
  if_xj ff1;
  double pc[];
  double xc[];
  double xt[];
  int n;
  public f1dim(if_xj ff,double pi[],double xi[])
  { n=pi.length;
    ff1=ff;
            pc=new double[n];
    xc=new double[n];
    xt=new double[n];
             for(int i=0;i<n;i++)
               {pc[i]=pi[i];xc[i]=xi[i];}
  }

  public double func(double x)
  {
            for (int j=0;j<n;j++) xt[j]=pc[j]+x*xc[j];
            return ff1.func(xt);
  }
}
public class NA43B
{
public static double VT_V(double [] left)
{
int n=left.length;
double tot=0;
for(int i=0;i<n;i++)
  {tot+=left[i]*left[i];}
return tot;
}

public static double[][] V_VT(double [] left)
{
//multiplys a vector transpose with a vector
int n=left.length;
double aa[][]=new double[n][n];
for(int i=0;i<n;i++)
  {for(int j=0;j<n;j++) {aa[i][j]=left[i]*left[j];}}
return aa;
}

public static double norm(double v[])
{
// vector norm
double total=0;;
for(int i=0;i<v.length;i++)
  {total+=v[i]*v[i];}
return Math.sqrt(total);
}
//===================
```

```
public static double[] df(if_xj f,double xxi[])
{
//G radian vector
// derivative of f(x0,x1,x2,...xn)
// [df/dx0, df/dx1,df/dx2,....,df/dxn]
// df/dxj  j=0...x.length
//

double a[]=new double[xxi.length];
for(int i=0;i<xxi.length;i++)
   {
   a[i]=dfdx(f,xxi,i);
   }
return a;
}


public static double dfdx(if_xj f,double xxi[],int xref)
{ int i;
              double h=1.0e-3;
  double hh=1.0/h;
  double xp2h[]=new double[xxi.length];
  double xph[]=new double[xxi.length];
  double xm2h[]=new double[xxi.length];
  double xmh[]=new double[xxi.length];
  double fp2h,fph,fm2h,fmh;
  for(i=0;i<xxi.length;i++)
  {
  if(i!=xref)
    {xp2h[i]=xxi[i];xph[i]=xxi[i];xm2h[i]=xxi[i];xmh[i]=xxi[i];}
  else {xp2h[i]=xxi[i]+2.0*h;xph[i]=xxi[i]+h;xm2h[i]=xxi[i]-2.0*h;xmh[i]=xxi[i]-h;}
  }
  fp2h=f.func(xp2h);
  fph=f.func(xph);
  fm2h=f.func(xm2h);
  fmh=f.func(xmh);
  return (-fp2h+8.0*fph-8.0*fmh+fm2h)/12.0*hh;
}

public static double d2fdx2(if_xj f,double xxi[],int xiref,int xjref)
{ //second derivative
  int i;
  double h=1.0e-4;
  double hh=1.0/h;
  int n=xxi.length;
  double ff;
  double xp2h[]=new double[xxi.length];
  double xph[]=new double[xxi.length];
  double xm2h[]=new double[xxi.length];
  double xmh[]=new double[xxi.length];
  double fp2h,fph,fm2h,fmh;
     for(int j=0;j<n;j++)
     {if(j!=xjref) {xp2h[j]=xxi[j];xph[j]=xxi[j];xm2h[j]=xxi[j];xmh[j]=xxi[j];}
      else {xp2h[j]=xxi[j]+2.0*h;xph[j]=xxi[j]+h;xm2h[j]=xxi[j]-2.0*h;xmh[j]=xxi[j]-h;}
     }
     fp2h=dfdx(f,xp2h,xiref);
     fph=dfdx(f,xph,xiref);
```

```
    fm2h=dfdx(f,xm2h,xiref);
    fmh=dfdx(f,xmh,xiref);
    ff=(-fp2h+8.0*fph-8.0*fmh+fm2h)/12.0*hh;
  return ff;
}
//==================
public static double SIGN(double a,double b) {return (b > 0.0 ? Math.abs(a) : -Math.abs(a));}
public static double MAX(double a,double b) {return (a > b ? a : b);}


public static double linmin(if_xj f,double p[],double xi[])
{
        double tol=2.0e-4;
   int n=p.length;
   double pcom[],xicom[];
        int j;
        double xx,xmin,bx,ax;
        f1dim f1=new f1dim(f,p,xi);
        ax=0.0;
        xx=1.0;
        bx=2.0;
        double aa[]=mnbrak(f1,ax,xx,bx);
        ax=aa[0];
        xx=aa[1];
        bx=aa[2];
        xmin=brent(f1,ax,xx,bx,tol);
        return xmin;
}

public static double[][] linminiter(if_xj f,double p[],double xi[])
{
   int n=p.length;
   double xmin=linmin(f,p,xi);
   double aa[][]=new double[2][n];
        for (int j=0;j<n;j++)
        {   xi[j] *= xmin;
                p[j] += xi[j];
                aa[0][j]=p[j];
                aa[1][j]=xi[j];
        }
   return aa;
}
public  static double[] brentf(if_x f,double ax,double bx,double cx,double tol)
{

int ITMAX=100;
double CGOLD=(3.0-Math.sqrt(5))/2.0;
double ZEPS=1.0e-10;
double xmin;
double aa[]=new double[2];
        int iter;
        double a,b,d,etemp,fu,fv,fw,fx,p,q,r,tol1,tol2,u,v,w,x,xm;
        double e=0.0;
        d=0.0;
        a=((ax < cx) ? ax : cx);
        b=((ax > cx) ? ax : cx);
        x=w=v=bx;
```

```
		fw=fv=fx=f.func(x);
		for (iter=1;iter<=ITMAX;iter++) {
			xm=0.5*(a+b);
			tol2=2.0*(tol1=tol*Math.abs(x)+ZEPS);
			if (Math.abs(x-xm) <= (tol2-0.5*(b-a))) {
				xmin=x;
		aa[0]=xmin;
		aa[1]=fx;
		return aa;
			}
			if (Math.abs(e) > tol1) {
				r=(x-w)*(fx-fv);
				q=(x-v)*(fx-fw);
				p=(x-v)*q-(x-w)*r;
				q=2.0*(q-r);
				if (q > 0.0) p = -p;
				q=Math.abs(q);
				etemp=e;
				e=d;
				if (Math.abs(p) >= Math.abs(0.5*q*etemp) || p <= q*(a-x) || p >= q*(b-x))
					d=CGOLD*(e=(x >= xm ? a-x : b-x));
				else {
					d=p/q;
					u=x+d;
					if (u-a < tol2 || b-u < tol2)
						d=SIGN(tol1,xm-x);
				}
			} else {
				d=CGOLD*(e=(x >= xm ? a-x : b-x));
			}
			u=(Math.abs(d) >= tol1 ? x+d : x+SIGN(tol1,d));
			fu=f.func(u);
			if (fu <= fx) {
				if (u >= x) a=x; else b=x;
				{v=w;w=x;x=u;}
				{fv=fw;fw=fx;fx=fu;}
			} else {
				if (u < x) a=u; else b=u;
				if (fu <= fw || w == x) {
					v=w;
					w=u;
					fv=fw;
					fw=fu;
				} else if (fu <= fv || v == x || v == w) {
					v=u;
					fv=fu;
				}
			}
		}
		System.out.println("Maximum iteration limit is exceeded in brent method");
		xmin=x;
		aa[0]=xmin;
		aa[1]=fx;
		return aa;
}
public  static double[] mnbrak(if_x f,double ax,double dx)
{
```

```
return mnbrak(f,ax,(ax-dx),(ax+dx));
}

public  static double[] mnbrak(if_x f,double ax)
{
// dx ikinci derece polinom için gerekli iki ek noktayı oluşturmak için
// ana noktadan seçiln uzaklık
return mnbrak(f,ax,(ax-0.1),(ax+0.1));
}

public  static double[] mnbrak(if_x f,double ax,double bx,double cx)
{

int ITMAX=100;
double xmin;
double GOLD=(Math.sqrt(5.0)+1.0)/2.0;;
double GLIMIT=100.0;
double TINY=1.0e-20;
double fa,fb,fc;
fa=0;
fb=0;
fc=0;
        double ulim,u,r,q,fu,dum;
        double aa[]=new double[3];
        fa=f.func(ax);
        fb=f.func(bx);
        if (fb > fa) {
                //SHFT(dum,*ax,*bx,dum)
                {dum=ax;ax=bx;bx=dum;}
                //SHFT(dum,*fb,*fa,dum)
                {dum=fb;fb=fa;fa=dum;}
        }
        cx=(bx)+GOLD*(bx-ax);
        fc=f.func(cx);
                aa[0]=ax;
                aa[1]=bx;
                aa[2]=cx;
        while (fb > fc) {
                r=(bx-ax)*(fb-fc);
                q=(bx-cx)*(fb-fa);
                u=(bx)-((bx-cx)*q-(bx-ax)*r)/
                        (2.0*SIGN(MAX(Math.abs(q-r),TINY),q-r));
                ulim=(bx)+GLIMIT*(cx-bx);
                if ((bx-u)*(u-cx) > 0.0) {
                        fu=f.func(u);
                        if (fu < fc) {
                                ax=bx;
                                bx=u;
                                fa=fb;
                                fb=fu;
                                return aa;
                        } else if (fu > fb) {
                                cx=u;
                                fc=fu;
                                return aa;
                        }
```

```
                        u=(cx)+GOLD*(cx-bx);
                        fu=f.func(u);
                } else if ((cx-u)*(u-ulim) > 0.0) {
                        fu=f.func(u);
                        if (fu < fc) {
                                //SHFT(bx,cx,u,cx+GOLD*(cx-bx))
                                {bx=cx;cx=u;u=cx+GOLD*(cx-bx);}
                                //SHFT(fb,fc,fu,f.func(u))
                                {fb=fc;fc=fu;fu=f.func(u);}
                        }
                } else if ((u-ulim)*(ulim-cx) >= 0.0) {
                        u=ulim;
                        fu=f.func(u);
                } else {
                        u=cx+GOLD*(cx-bx);
                        fu=f.func(u);
                }
                //SHFT(ax,bx,cx,u)
                {ax=bx;bx=cx;cx=u;}
                //SHFT(fa,fb,fc,fu)
                {fa=fb;fb=fc;fc=fu;}
                aa[0]=ax;
                aa[1]=bx;
                aa[2]=cx;
        }
                return aa;
}
public  static double brent(if_x f,double ax,double bx,double cx,double tol)
{
 double aa[]=brentf(f,ax,bx,cx,tol);
 return aa[0];
}
// 4. fletcher Reeves method
//_____________________________________________________________________
public static double[] fletcher_reeves( if_xj f,double x[])
{
int i,j,ii,jj,k;
int nmax=500;
double gamma=1.0;
double tolerance=1.0e-11;
int n=x.length;
double alpha;
double g[]=new double[n];
double ge[]=new double[n];
double d[]=new double[n];
double beta;
double alpha_d;
double Q[]=new double[n];
int nn=15;
i=0;
j=0;k=0;
while( j < nmax && gamma > tolerance )
{
   g=df(f,x);
   gamma=norm(g);
   if(k<1) {for(i=0;i<n;i++){d[i]=-g[i];};ge=g;}
   else
```

```
    {beta=VT_V(g)/VT_V(ge);
     for(i=0;i<n;i++){Q[i]=beta*d[i];};
    }
    for(i=0;i<n;i++){d[i]=-g[i]+Q[i];}
    alpha=linmin(f,x,d);
    for(i=0;i<n;i++){x[i]+=alpha*d[i];}
    ge=g;
    j++;k++;
    if(k>=nn) k=0; //tekrar bir en dik yaöaç metodu uygulaması
}
if(i >= nmax) JOptionPane.showMessageDialog(null,"Warning : Maximum number of iteration exceeded \n"+
  " result may not be valid","MAXIMUM ITERATION WARNING",JOptionPane.WARNING_MESSAGE);
return x;
}
                public static double[] inputdata(String s)
    {
     String s1=JOptionPane.showInputDialog(s);
     StringTokenizer token=new StringTokenizer(s1);
     int n=token.countTokens()-1;
     int m=n+1;
     double a[]=new double[m];
     int j=0;
     while(token.hasMoreTokens())
     {
     Double ax=new Double(token.nextToken());
     a[j++]=ax.doubleValue();
     }
     return a;
     }

      public static void main (String args[])
     {
                String s="enter optimum point first estimation values with one empty space in between: ";
     double [] x0=inputdata(s);
     if_xj f=(double x[])->3.0*x[0]*x[0]-4.0*x[0]*x[1]+2.0*x[1]*x[1]-x[0]-x[1];
     double [] r1= fletcher_reeves(f,x0);
     s=" minimum values : \n"+Matrix.toStringT(r1);
     String s2="Fletcher-Reeves multivariable optimisation:";
     JOptionPane.showMessageDialog(null,s,s2,JOptionPane.PLAIN_MESSAGE);
     System.exit(0);
     }
}
```

5.13 NON-LINEAR MULTIDIMENSIONAL POLAK-RIBIERE CONJUGATE GRADIENT METHOD

A modified form of Fletcher Reeves method is called **Polak-Ribiere** method.

Fletcher Reeves definition $\beta^k = \frac{(g^k)^T(g^k)}{(g^{k-1})^T(g^{k-1})}$

With a small change of definition it becomes in Polack-Ribiere method

$\beta^k = \frac{(g^{k-1}-g^k)^T(g^k)}{(g^{k-1})^T(g^{k-1})}$

5.13-1 Polak-Ribiere Conjugate Gradient method with numerical derivatives (NA43C.java

```
import java.util.*;
```

```
import java.awt.*;
import java.applet.Applet;
import java.awt.event.*;
import javax.swing.*;

public class NA43C
{

public static double VT_V(double [] left)
{
//multiplies a vector transpose with a vector
int n=left.length;
double tot=0;
for(int i=0;i<n;i++)
  {
  tot+=left[i]*left[i];
  }
return tot;
}

public static double[][] V_VT(double [] left)
{
//multiplys a vector transpose with a vector
int n=left.length;
double aa[][]=new double[n][n];
for(int i=0;i<n;i++)
  {
  for(int j=0;j<n;j++)
    {
              aa[i][j]=left[i]*left[j];
              }
  }
return aa;
}

public static double  VT_X(double [] left,double [] right)
{
//multiplys a vector transpose with a vector
int n=left.length;
double tot=0;
for(int i=0;i<n;i++)
  {
  tot+=left[i]*right[i];
  }
return tot;
}

public static double[] substract(double[] left,double[] right)
{
//addition of two vectors
int n1=left.length;
int n2=right.length;
int nMax;
int i;
if(n1>=n2) nMax=n1;
else       nMax=n2;
double b[];
```

```
b=new double[nMax];
for(i=0;i<n1;i++)
  {
    b[i]=b[i]+left[i];
  }
for(i=0;i<n2;i++)
  {
    b[i]=b[i]-right[i];
  }
return b;
//end of vector substraction method
}

public static double norm(double v[])
{
// vector norm
double total=0;;
for(int i=0;i<v.length;i++)
  {total+=v[i]*v[i];}
return Math.sqrt(total);
}
//===================

//===================
public static double SIGN(double a,double b) {return (b > 0.0 ? Math.abs(a) : -Math.abs(a));}
public static double MAX(double a,double b) {return (a > b ? a : b);}


public static double linmin(if_xj f,double p[],double xi[])
{
   double tol=2.0e-4;
   int n=p.length;
   double pcom[],xicom[];
   int j;
   double xx,xmin,bx,ax;
   f1dim f1=new f1dim(f,p,xi);
   ax=0.0;
   xx=1.0;
   bx=2.0;
   double aa[]=mnbrak(f1,ax,xx,bx);
   ax=aa[0];
   xx=aa[1];
   bx=aa[2];
   xmin=brent(f1,ax,xx,bx,tol);
            return xmin;
}

public static double[][] linminiter(if_xj f,double p[],double xi[])
{   // yeni iterasyon noktasını verir
   int n=p.length;
   double xmin=linmin(f,p,xi);
   double aa[][]=new double[2][n];
            for (int j=0;j<n;j++)
            {   xi[j] *= xmin;
                        p[j] += xi[j];
                        aa[0][j]=p[j];
                        aa[1][j]=xi[j];
```

```
            }
   return aa;
}
public  static double[] brentf(if_x f,double ax,double bx,double cx,double tol)
{
int ITMAX=100;
double CGOLD=(3.0-Math.sqrt(5))/2.0; //altın oran
double ZEPS=1.0e-10;
double xmin;
double aa[]=new double[2];
// SHFT(double a,double b,double c,d) (a)=(b);(b)=(c);(c)=(d);
            int iter;
            double a,b,d,etemp,fu,fv,fw,fx,p,q,r,tol1,tol2,u,v,w,x,xm;
            double e=0.0;
            d=0.0;
            a=((ax < cx) ? ax : cx);
            b=((ax > cx) ? ax : cx);
            x=w=v=bx;
            fw=fv=fx=f.func(x);
            for (iter=1;iter<=ITMAX;iter++) {
                        xm=0.5*(a+b);
                        tol2=2.0*(tol1=tol*Math.abs(x)+ZEPS);
                        if (Math.abs(x-xm) <= (tol2-0.5*(b-a))) {
                                    xmin=x;
                  aa[0]=xmin;
                  aa[1]=fx;
                  return aa;
                        }
                        if (Math.abs(e) > tol1) {
                                    r=(x-w)*(fx-fv);
                                    q=(x-v)*(fx-fw);
                                    p=(x-v)*q-(x-w)*r;
                                    q=2.0*(q-r);
                                    if (q > 0.0) p = -p;
                                    q=Math.abs(q);
                                    etemp=e;
                                    e=d;
                                    if (Math.abs(p) >= Math.abs(0.5*q*etemp) || p <= q*(a-x) || p >= q*(b-x))
                                                d=CGOLD*(e=(x >= xm ? a-x : b-x));
                                    else {
                                                d=p/q;
                                                u=x+d;
                                                if (u-a < tol2 || b-u < tol2)
                                                            d=SIGN(tol1,xm-x);
                                    }
                        } else {
                                    d=CGOLD*(e=(x >= xm ? a-x : b-x));
                        }
                        u=(Math.abs(d) >= tol1 ? x+d : x+SIGN(tol1,d));
                        fu=f.func(u);
                        if (fu <= fx) {
                                    if (u >= x) a=x; else b=x;
                                    {v=w;w=x;x=u;}
                                    {fv=fw;fw=fx;fx=fu;}
                        } else {
                                    if (u < x) a=u; else b=u;
                                    if (fu <= fw || w == x) {
```

```
                                        v=w;
                                        w=u;
                                        fv=fw;
                                        fw=fu;
                                } else if (fu <= fv || v == x || v == w) {
                                        v=u;
                                        fv=fu;
                                }
                        }
                }
                System.out.println("Maximum iteration in BRENT is exceeded");
                xmin=x;
                //minumum değer a[0] ile fonksiyon değeri a[1] ile geridöndürülmektedir
                aa[0]=xmin;
                aa[1]=fx;
                return aa;
}
public  static double[] mnbrak(if_x f,double ax,double dx)
{return mnbrak(f,ax,(ax-dx),(ax+dx));}

public  static double[] mnbrak(if_x f,double ax)
{return mnbrak(f,ax,(ax-0.1),(ax+0.1));}

public  static double[] mnbrak(if_x f,double ax,double bx,double cx)
{
int ITMAX=100;
double xmin;
double GOLD=(Math.sqrt(5.0)+1.0)/2.0;;
double GLIMIT=100.0;
double TINY=1.0e-20;
double fa,fb,fc;
fa=0;
fb=0;
fc=0;
                double ulim,u,r,q,fu,dum;
                double aa[]=new double[3];
                fa=f.func(ax);
                fb=f.func(bx);
                if (fb > fa) {
                        //SHFT(dum,*ax,*bx,dum)
                        {dum=ax;ax=bx;bx=dum;}
                        //SHFT(dum,*fb,*fa,dum)
                        {dum=fb;fb=fa;fa=dum;}
                }
                cx=(bx)+GOLD*(bx-ax);
                fc=f.func(cx);
                        aa[0]=ax;
                        aa[1]=bx;
                        aa[2]=cx;
                while (fb > fc) {
                        r=(bx-ax)*(fb-fc);
                        q=(bx-cx)*(fb-fa);
                        u=(bx)-((bx-cx)*q-(bx-ax)*r)/
                                (2.0*SIGN(MAX(Math.abs(q-r),TINY),q-r));
                        ulim=(bx)+GLIMIT*(cx-bx);
                        if ((bx-u)*(u-cx) > 0.0) {
                                fu=f.func(u);
```

```
                        if (fu < fc) {
                                ax=bx;
                                bx=u;
                                fa=fb;
                                fb=fu;
                                return aa;
                        } else if (fu > fb) {
                                cx=u;
                                fc=fu;
                                return aa;
                        }
                        u=(cx)+GOLD*(cx-bx);
                        fu=f.func(u);
                } else if ((cx-u)*(u-ulim) > 0.0) {
                        fu=f.func(u);
                        if (fu < fc) {
                                //SHFT(bx,cx,u,cx+GOLD*(cx-bx))
                                {bx=cx;cx=u;u=cx+GOLD*(cx-bx);}
                                //SHFT(fb,fc,fu,f.func(u))
                                {fb=fc;fc=fu;fu=f.func(u);}
                        }
                } else if ((u-ulim)*(ulim-cx) >= 0.0) {
                        u=ulim;
                        fu=f.func(u);
                } else {
                        u=cx+GOLD*(cx-bx);
                        fu=f.func(u);
                }
                //SHFT(ax,bx,cx,u)
                {ax=bx;bx=cx;cx=u;}
                //SHFT(fa,fb,fc,fu)
                {fa=fb;fb=fc;fc=fu;}
                aa[0]=ax;
                aa[1]=bx;
                aa[2]=cx;
        }
                return aa;
}
public  static double brent(if_x f,double ax,double bx,double cx,double tol)
{ double aa[]=brentf(f,ax,bx,cx,tol);
  return aa[0];
}
// 4. Polak Ribiere Method
//_____________________________________________________________________
public static double[] polak_ribiere( if_xj f,double x[])
{
int i,j,ii,jj,k;
int nmax=500;
double gamma=1.0;
double tolerance=1.0e-7;
int n=x.length;
double alpha;
double g[]=new double[n];
double ge[]=new double[n];
double d[]=new double[n];
double beta;
double alpha_d;
```

```
double Q[]=new double[n];
int nn=15;
i=0;
j=0;k=0;
while( j < nmax && gamma > tolerance )
{   g=f.dfunc(x);
   gamma=norm(g);
   if(k<1) {for(i=0;i<n;i++){d[i]=-g[i];};ge=g;}
   else
   {   // Polack-Ribiere conjugate gradient definition
            beta=VT_X(substract(ge,g),g)/VT_V(ge);
    for(i=0;i<n;i++){Q[i]=beta*d[i];};
   }
   for(i=0;i<n;i++){d[i]=-g[i]+Q[i];}
   alpha=linmin(f,x,d);
   for(i=0;i<n;i++){x[i]+=alpha*d[i];}
   ge=g;
   j++;k++;
   if(k>=nn) k=0; //re-apply a step of steepest descent
}
if(i >= nmax) JOptionPane.showMessageDialog(null,"Warning : Maximum number of iteration exceeded \n"+
  " result may not be valid","MAXIMUM ITERATION WARNING",JOptionPane.WARNING_MESSAGE);
return x;
}
            public static double[] verigir(String s)
    {
    String s1=JOptionPane.showInputDialog(s);
    StringTokenizer token=new StringTokenizer(s1);
    int n=token.countTokens()-1;
    int m=n+1;
    double a[]=new double[m];
    int j=0;
    while(token.hasMoreTokens())
    {
    Double ax=new Double(token.nextToken());
    a[j++]=ax.doubleValue();
    }
    return a;
    }

            public static void main (String args[])
    {
            String s="enter optimum point first estimation values with one empty space in between: ";
    double [] x0=verigir(s);
     if_xj f=(double x[])->3.0*x[0]*x[0]-4.0*x[0]*x[1]+2.0*x[1]*x[1]-x[0]-x[1];
    double [] r1= polak_ribiere(f,x0);
    s=" minimum values : \n"+Matrix.toStringT(r1);
    String s2="Polak-Ribiere multivariable non linear optimisation:";
    JOptionPane.showMessageDialog(null,s,s2,JOptionPane.PLAIN_MESSAGE);
    System.exit(0);
    }
}
```

5.14 MULTIDIMENSIONAL NONLINEAR DAVIDON-FLETCHER-POWELL METHOD

The general equation was in the form of

$$\{x^{k+1}\} = \{x^k\} + \alpha^k [H^k]^{-1}[-\{\nabla f^k\}] = \{x^k\} + \alpha^k\{-g^k\} = \{x^k\} - \alpha^k\{d^k\}$$

Where

$$\{g^k\} = \nabla f\left(x_1^k, x_2^k, x_3^k, \dots x_n^k\right), \;\; k = 0, \dots, n$$

and

$$\{d^k\} = [H^k]^{-1}\{Q_k\}$$
$$\{x^{k+1}\} = \{x^k\} - \alpha^k\{d^k\}$$

In this method, instead of Hessian an approximation to Hessian will be constructed. Let us define

$$\{Q_k\} = [H^k]^{-1} \;, k \geq 0$$
$$\{d^k\} = \{Q_k\}\{g_k\}$$
$$\{x^{k+1}\} = \{x^k\} - \alpha^k\{Q_k\}\{g_k\} = \{x^k\} - \alpha^k\{d^k\}$$

To calculate Q in Davidon – Fletcher – Powell metod ,

$$\Delta g^k = g^{k+1} - g^k$$
$$r^k = Q_k \Delta g^k$$
$$\Delta x^k = x^{k+1} - x^k$$
$$Q_{k+1} = Q_k - \frac{r^k\left(r^k\right)^T}{(r^k)^T \Delta g^k} + \frac{\left(\Delta x^k\right)\left(\Delta x^k\right)^T}{(\Delta x^k)^T \Delta g^k}, \qquad 0 \leq k \leq n$$

Formulas are used. Program definition of these is given in Program 5.14-1. A second version of the formulas are also existed as follows

$$Q_{k+1} = \beta_k \left[Q_k - \frac{r^k\left(r^k\right)^T}{(r^k)^T \Delta g^k} \right] + \frac{\left(\Delta x^k\right)\left(\Delta x^k\right)^T}{(\Delta x^k)^T \Delta g^k}, \qquad 0 \leq k \leq n$$

$$\beta_k = \frac{\left(\Delta x^k\right)^T\left(\Delta x^k\right)}{(r^k)^T \Delta g^k}, \qquad 0 \leq k \leq n$$

In this iteratif solution method, iteration starts as Q equals to unit matrix.

$$Q_0 = I$$
$$x^{k+1} = x^k - \alpha_k Q_k g^k = \{x^k\} - \alpha^k d^k$$

$\alpha_k$again is used by using one of the one variable optimization method. In our program Brent method is used. First step is always a steepest descent method also in this approach. After in every n step a steepest descent step is used

EXAMPLE PROBLEM

Problem is to find minimum of

$$f(x_0, x_1) = 3x_0^2 - 4x_0x_1 + 2x_1^2 - x_0 - x_1$$

with initial guess $\{x\} = [-2,2]$

$$g^0 = \nabla f = \begin{Bmatrix} 6x_0 - 4x_1 - 1 \\ 4x_1 - 4x_0 - 1 \end{Bmatrix} = \begin{Bmatrix} -21 \\ 15 \end{Bmatrix}$$
$$\begin{Bmatrix} x_0^0 \\ x_1^0 \end{Bmatrix} = \begin{Bmatrix} -2 \\ 2 \end{Bmatrix}$$
$$\begin{Bmatrix} x_0^1 \\ x_1^1 \end{Bmatrix} = \begin{Bmatrix} x_0^0 \\ x_1^0 \end{Bmatrix} + \alpha \begin{Bmatrix} 6x_0^0 - 4x_1^0 - 1 \\ 4x_1^0 - 4x_0^0 - 1 \end{Bmatrix}$$
$$\begin{Bmatrix} x_0^1 \\ x_1^1 \end{Bmatrix} = \begin{Bmatrix} -2 \\ 2 \end{Bmatrix} + \alpha \begin{Bmatrix} -21 \\ 15 \end{Bmatrix} = \begin{Bmatrix} -2 - 21\alpha \\ 2 + 15\alpha \end{Bmatrix}$$

If these values are substituted into the function

$$f(x_0, x_1) = 3x_0^2 - 4x_0x_1 + 2x_1^2 - x_0 - x_1$$
$$y = 3(-2 - 21\alpha)^2 - 4(-2 - 21\alpha)(2 + 15\alpha) + 2(2 + 15\alpha)^2 - (-2 - 21\alpha) - (2 + 15\alpha)$$

The function becomes;

$$y = 36 + 666\alpha + 3033\alpha^2$$

Minimum of the function ca be find with analytical method as:

$$y' = 666 + 6066\alpha = 0$$

$$\alpha = -0.109792284$$

If this value is substituted in x_0^1 andx_1^1

For

$$\alpha^0 = -0.109792284$$

$$\begin{Bmatrix} x_0^1 \\ x_1^1 \end{Bmatrix} = \begin{Bmatrix} 0.305637982195846 \\ 0.35311572002967 \end{Bmatrix}$$

This first step was steepest descent, as a second step Davidon-Fletcher-Powell can be applied

$$g^1 = \nabla f = \begin{Bmatrix} 6x_0 - 4x_1 - 1 \\ 4x_1 - 4x_0 - 1 \end{Bmatrix} = \begin{Bmatrix} -0.578635014836797 \\ -0.810089020771512 \end{Bmatrix}$$

$$\Delta g^0 = g^1 - g^0 = \begin{Bmatrix} -0.578635014836797 - (-21) \\ -0.810089020771512 - 15 \end{Bmatrix} = \begin{Bmatrix} 20.4213649851632 \\ -15.810089020771512 \end{Bmatrix}$$

$$Q_0 = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$$

$$r^k = Q_k \Delta g^k$$

$$r^0 = Q_0 \Delta g^0 = \begin{Bmatrix} 20.4213649851632 \\ -15.810089020771512 \end{Bmatrix}$$

$$\Delta x^{k+1} = x^{k+1} - x^k$$

$$\Delta x^1 = x^1 - x^0 = \begin{Bmatrix} -0.305637982195846 \\ 0.35311572002967 \end{Bmatrix} - \begin{Bmatrix} -2 \\ 2 \end{Bmatrix} = \begin{Bmatrix} 2.305637982195846 \\ -1.646884272997033 \end{Bmatrix}$$

$$Q_{k+1} = Q_k - \frac{r^k (r^k)^T}{(r^k)^T \Delta g^k} + \frac{(\Delta x^k)(\Delta x^k)^T}{(\Delta x^k)^T \Delta g^k}$$

$$Q_1 = \begin{bmatrix} 0.447456314953212 & 0.422131073481232 \\ 0.422131073481232 & 0.662335969913257 \end{bmatrix}$$

$$d_1 = Q_1 g^1 = \begin{bmatrix} 0.447456314953212 & 0.422131073481232 \\ 0.422131073481232 & 0.662335969913257 \end{bmatrix} \begin{Bmatrix} -0.578635014836797 \\ -0.810089020771512 \end{Bmatrix}$$

$$d_1 = \begin{Bmatrix} 0.608978529603123 \\ 0.786597267404034 \end{Bmatrix}$$

$$\begin{Bmatrix} x_0^2 \\ x_1^2 \end{Bmatrix} = \begin{Bmatrix} x_0^1 \\ x_1^1 \end{Bmatrix} + \alpha^1 d_1$$

$$\begin{Bmatrix} x_0^2 \\ x_1^2 \end{Bmatrix} = \begin{Bmatrix} 0.305637982195846 \\ 0.35311572002967 \end{Bmatrix} + \alpha \begin{Bmatrix} 0.608978529603123 \\ 0.786597267404034 \end{Bmatrix}$$

$$\begin{Bmatrix} x_0^2 \\ x_1^2 \end{Bmatrix} = \begin{Bmatrix} 0.305637982195846 + 0.608978529603123\alpha \\ 0.35311572002967 + 0.786597267404034\alpha \end{Bmatrix}$$

If these values are back substituted into the function

$$f(x_0, x_1) = 3x_0^2 - 4x_0x_1 + 2x_1^2 - x_0 - x_1$$

and minimum is solved again it is found that

$$\begin{Bmatrix} x_0^2 \\ x_1^2 \end{Bmatrix} = \begin{Bmatrix} 1 \\ 1.25 \end{Bmatrix}$$

Problem 5.14-1 Davidon_Fletcher-Powell method (NA43D.java)

```
// Davidon-Fletcher-Powell optimization method
// Metod Referansi : Numerical Analysis, 8th edition Richard L. Burden, J. Douglas Faires 3
// sayfa 617
import java.util.*;
import java.awt.*;
import java.applet.Applet;
```

```
import java.awt.event.*;
import javax.swing.*;

class f1 implements if_xj
{
public double func(double x[])
{
//çözümü istenen fonksiyon
double ff;
ff=3.0*x[0]*x[0]-4.0*x[0]*x[1]+2.0*x[1]*x[1]-x[0]-x[1];
return ff;
}}

public class NA43D
{
public static double VT_V(double [] left)
{
//multiplys a vector transpose with a vector
int n=left.length;
double tot=0;
for(int i=0;i<n;i++)
  {
  tot+=left[i]*left[i];
  }
return tot;
}
public static double[][] V_VT(double [] left)
{
//multiplys a vector transpose with a vector
int n=left.length;
double aa[][]=new double[n][n];
for(int i=0;i<n;i++)
  {
  for(int j=0;j<n;j++)
    {
              aa[i][j]=left[i]*left[j];
              }
  }
return aa;
}
public static double  VT_X(double [] left,double [] right)
{
//multiplys a vector transpose with a vector
int n=left.length;
double tot=0;
for(int i=0;i<n;i++)
  {
  tot+=left[i]*right[i];
  }
return tot;
}
public static double norm(double v[])
{
// vector norm
double total=0;;
for(int i=0;i<v.length;i++)
  {total+=v[i]*v[i];}
```

```
return Math.sqrt(total);
}
public static double turev(if_xj f,double x[],int denklem_ref,int x_ref)
{
double h0=0.256808;
int i,m;
int n=7;
double f1[];
f1=new double[x.length];
double f2[];
f2=new double[x.length];
double x1[];
x1=new double[x.length];
double x2[];
x2=new double[x.length];
for(i=0;i<x.length;i++)
{
x1[i]=x[i];
x2[i]=x[i];
}
//turev of a simple function
double T[][];
T=new double[n][n];
double h[];
h=new double[n];
//vector<double> h(n,0);
for(i=0;i<n;i++)
{
  h[i]=0;
  for(int j=0;j<n;j++)
    T[i][j]=0;
}
h[0]=h0;
double r=0.5;
for( i=1;i<n;i++)
{
h[i]=h0*Math.pow(r,i);
}
for(i=0;i<n;i++)
{
x1[x_ref]+=h[i];
x2[x_ref]-=h[i]*1.0094847;
f1=df(f,x1);
f2=df(f,x2);
T[i][0]=( f1[denklem_ref] - f2[denklem_ref])/(2.0*h[i]);
x1[x_ref]=x[x_ref];
x2[x_ref]=x[x_ref];
}
for(m=1;m<n;m++)
{
  for(i=0;i<n-m;i++)
  {
  T[i][m]=(h[i]*h[i]*T[i+1][m-1] - h[i+m]*h[i+m]*T[i][m-1])/(h[i]*h[i]
  - h[i+m]*h[i+m]);
  }
}
double xx=T[0][n-1];
```

```
return xx;
}
public static double[] df(if_xj f,double xxi[])
{
//Gredyen vektörü
// f(x0,x1,x2,...xn) founksiyonunun derivative vektörü
// [df/dx0, df/dx1,df/dx2,....,df/dxn]
// df/dxj  j=0...x.length
//
// bu fonksiyon bilgisayar tarafindan hesaplanmaktadir
// kullanici tarafindan tanimlanan f(x) fonksiyonunun
// sayisal olarak alinan derivativeidir.

//
double a[]=new double[xxi.length];
for(int i=0;i<xxi.length;i++)
  {
  a[i]=turev(f,xxi,i);
  }
return a;
}
public static double turev(if_xj f,double xxi[],int x_ref)
{
// df/dxj
double h0=0.256808;
int i,m;
int n=7;
double x1[];
x1=new double[xxi.length];
double x2[];
x2=new double[xxi.length];
for(i=0;i<xxi.length;i++)
{
x1[i]=xxi[i];
x2[i]=xxi[i];
}
//derivative of a simple function
double T[][];
T=new double[n][n];
double h[];
h=new double[n];
for(i=0;i<n;i++)
{
 h[i]=0;
 for(int j=0;j<n;j++)
   T[i][j]=0;
}
h[0]=h0;
double r=0.5;
for( i=1;i<n;i++)
{
//h[i]=h0*Math.pow(r,i);
h0/=2.0;
h[i]=h0;
}
//first turev (difference formula)
for(i=0;i<n;i++)
```

```
{
x1[x_ref]+=h[i];
x2[x_ref]-=h[i];
T[i][0]=(f.func(x1)-f.func(x2))/(2.0*h[i]);
x1[x_ref]=xxi[x_ref];
x2[x_ref]=xxi[x_ref];
}
for(m=1;m<n;m++)
{
  for(i=0;i<n-m;i++)
  {
  T[i][m]=(h[i]*h[i]*T[i+1][m-1] -
  h[i+m]*h[i+m]*T[i][m-1])/(h[i]*h[i] - h[i+m]*h[i+m]);
  }
}
double xx=T[0][n-1];
return xx;
}
public static double[] turev(if_xj f_deriv,double x[])
{
// df/dxj  j=0...x.length
// This method calculates derivative of a function with more than one independent variable.
// Accuracy of method can be adjusted by changing variables h0 and n
// function input should be in the form given in abstract class
// if_xj,j=0...x.length = df/dx(x_ref)
double a[]=new double[x.length];
for(int x_ref=0;x_ref<x.length;x_ref++)
   {
   a[x_ref]=turev(f_deriv,x,x_ref);
   }
return a;
}
public static double SIGN(double a,double b) {return (b > 0.0 ? Math.abs(a) : -Math.abs(a));}
public static double MAX(double a,double b) {return (a > b ? a : b);}
public static double linmin(if_xj f,double p[],double xi[])
{
            double tol=2.0e-4;
    int n=p.length;
    double pcom[],xicom[];
            int j;
            double xx,xmin,bx,ax;
            f1dim f1=new f1dim(f,p,xi);
            ax=0.0;
            xx=1.0;
            bx=2.0;
            double aa[]=mnbrak(f1,ax,xx,bx);
            ax=aa[0];
            xx=aa[1];
            bx=aa[2];
            xmin=brent(f1,ax,xx,bx,tol);
            return xmin;
}

public static double[][] linminiter(if_xj f,double p[],double xi[])
{   // yeni iterasyon noktasini verir
    int n=p.length;
    double xmin=linmin(f,p,xi);
```

```
    double aa[][]=new double[2][n];
            for (int j=0;j<n;j++)
            {   xi[j] *= xmin;
                    p[j] += xi[j];
                    aa[0][j]=p[j];
                    aa[1][j]=xi[j];
            }
    return aa;
}
public  static double[] brentf(if_x f,double ax,double bx,double cx,double tol)
{
// ax,bx,cx üç adet tahmin noktasi (bxax ile cx arasinda yer almalidir)
// f : fonksiyon (if_x abstract sinifinda tanimlanmiş)
// tol tolerans
int ITMAX=100;
double CGOLD=(3.0-Math.sqrt(5))/2.0; //altin oran
double ZEPS=1.0e-10;
double xmin;
double aa[]=new double[2];
// SHFT(double a,double b,double c,d) (a)=(b);(b)=(c);(c)=(d);
            int iter;
            double a,b,d,etemp,fu,fv,fw,fx,p,q,r,tol1,tol2,u,v,w,x,xm;
            double e=0.0;
            d=0.0;
            a=((ax < cx) ? ax : cx);
            b=((ax > cx) ? ax : cx);
            x=w=v=bx;
            fw=fv=fx=f.func(x);
            for (iter=1;iter<=ITMAX;iter++) {
                    xm=0.5*(a+b);
                    tol2=2.0*(tol1=tol*Math.abs(x)+ZEPS);
                    if (Math.abs(x-xm) <= (tol2-0.5*(b-a))) {
                            xmin=x;
                aa[0]=xmin;
                aa[1]=fx;
                return aa;
                    }
                    if (Math.abs(e) > tol1) {
                            r=(x-w)*(fx-fv);
                            q=(x-v)*(fx-fw);
                            p=(x-v)*q-(x-w)*r;
                            q=2.0*(q-r);
                            if (q > 0.0) p = -p;
                            q=Math.abs(q);
                            etemp=e;
                            e=d;
                            if (Math.abs(p) >= Math.abs(0.5*q*etemp) || p <= q*(a-x) || p >= q*(b-x))
                                    d=CGOLD*(e=(x >= xm ? a-x : b-x));
                            else {
                                    d=p/q;
                                    u=x+d;
                                    if (u-a < tol2 || b-u < tol2)
                                            d=SIGN(tol1,xm-x);
                            }
                    } else {
                            d=CGOLD*(e=(x >= xm ? a-x : b-x));
                    }
```

```
                        u=(Math.abs(d) >= tol1 ? x+d : x+SIGN(tol1,d));
                        fu=f.func(u);
                        if (fu <= fx) {
                                if (u >= x) a=x; else b=x;
                                {v=w;w=x;x=u;}
                                {fv=fw;fw=fx;fx=fu;}
                        } else {
                                if (u < x) a=u; else b=u;
                                if (fu <= fw || w == x) {
                                        v=w;
                                        w=u;
                                        fv=fw;
                                        fw=fu;
                                } else if (fu <= fv || v == x || v == w) {
                                        v=u;
                                        fv=fu;
                                }
                        }
                }
                System.out.println("BRENT metodunda maksimum iterasyon sayisi aşildi");
                xmin=x;
                //minumum değer a[0] ile fonksiyon değeri a[1] ile geridöndürülmektedir
                aa[0]=xmin;
                aa[1]=fx;
                return aa;
}
public  static double[] mnbrak(if_x f,double ax,double dx)
{
// dx ikinci derece polinom için gerekli iki ek noktayi oluşturmak için
// ana noktadan seçilen uzaklik
return mnbrak(f,ax,(ax-dx),(ax+dx));
}
public  static double[] mnbrak(if_x f,double ax)
{
// dx ikinci derece polinom için gerekli iki ek noktayi oluşturmak için
// ana noktadan seçiln uzaklik
return mnbrak(f,ax,(ax-0.1),(ax+0.1));
}

public  static double[] mnbrak(if_x f,double ax,double bx,double cx)
{
// ax,bx,cx üç adet tahmin noktasi verildiğinde içinde minimum bulunan ax,bx,cx üç adetnoktayi bize arama
// sonucu bulur
// f : fonksiyon (if_x abstract sinifinda tanimlanmiş)
// tol tolerans
int ITMAX=100;
double xmin;
double GOLD=(Math.sqrt(5.0)+1.0)/2.0;;
double GLIMIT=100.0;
double TINY=1.0e-20;
double fa,fb,fc;
fa=0;
fb=0;
fc=0;
                double ulim,u,r,q,fu,dum;
                double aa[]=new double[3];
                fa=f.func(ax);
```

```
	fb=f.func(bx);
	if (fb > fa) {
		//SHFT(dum,*ax,*bx,dum)
		{dum=ax;ax=bx;bx=dum;}
		//SHFT(dum,*fb,*fa,dum)
		{dum=fb;fb=fa;fa=dum;}
	}
	cx=(bx)+GOLD*(bx-ax);
	fc=f.func(cx);
		aa[0]=ax;
		aa[1]=bx;
		aa[2]=cx;
	while (fb > fc) {
		r=(bx-ax)*(fb-fc);
		q=(bx-cx)*(fb-fa);
		u=(bx)-((bx-cx)*q-(bx-ax)*r)/
			(2.0*SIGN(MAX(Math.abs(q-r),TINY),q-r));
		ulim=(bx)+GLIMIT*(cx-bx);
		if ((bx-u)*(u-cx) > 0.0) {
			fu=f.func(u);
			if (fu < fc) {
				ax=bx;
				bx=u;
				fa=fb;
				fb=fu;
				return aa;
			} else if (fu > fb) {
				cx=u;
				fc=fu;
				return aa;
			}
			u=(cx)+GOLD*(cx-bx);
			fu=f.func(u);
		} else if ((cx-u)*(u-ulim) > 0.0) {
			fu=f.func(u);
			if (fu < fc) {
				//SHFT(bx,cx,u,cx+GOLD*(cx-bx))
				{bx=cx;cx=u;u=cx+GOLD*(cx-bx);}
				//SHFT(fb,fc,fu,f.func(u))
				{fb=fc;fc=fu;fu=f.func(u);}
			}
		} else if ((u-ulim)*(ulim-cx) >= 0.0) {
			u=ulim;
			fu=f.func(u);
		} else {
			u=cx+GOLD*(cx-bx);
			fu=f.func(u);
		}
		//SHFT(ax,bx,cx,u)
		{ax=bx;bx=cx;cx=u;}
		//SHFT(fa,fb,fc,fu)
		{fa=fb;fb=fc;fc=fu;}
		aa[0]=ax;
		aa[1]=bx;
		aa[2]=cx;
	}
		return aa;
```

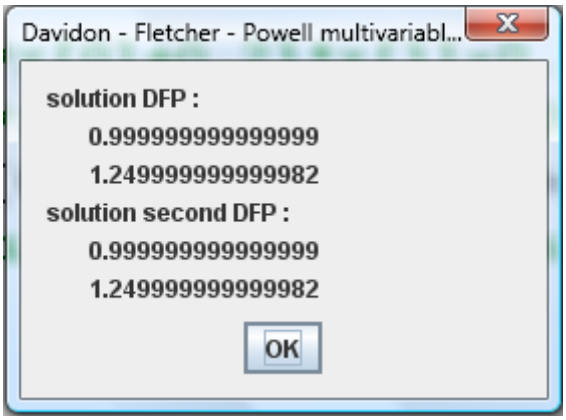
```
}
public  static double brent(if_x f,double ax,double bx,double cx,double tol)
{
 double aa[]=brentf(f,ax,bx,cx,tol);
 return aa[0];
}

public static double[][] I(int n)
{
//unit matrix
double b[][];
b=new double[n][n];
for(int i=0;i<n;i++)
   b[i][i]=1.0;
return b;
}
// Davidon - Fletcher-Powell metodu
//_____________________________________________________________________
public static double[] davidon_fletcher_powell( if_xj f,double x[])
{
int i,j,k;
int nmax=500;
double gamma=1.0;
double tolerance=1.0e-12 ;
int n=x.length;
double alpha;
double dx[]=new double[n];
double g[] =new double[n];
double ge[]=new double[n];
double dg[]=new double[n];
double d[] =new double[n];
double r[] =new double[n];
double rdg,dxdg;
double beta;
double alpha_d;
double Q[][]=I(n);
double Q1[][]=new double[n][n];
double Q2[][]=new double[n][n];
int nn=15;
i=0;
j=0;k=0;
g=turev(f,x);
while( j++ < nmax && gamma > tolerance )
{
   for(int ii=0;ii<n;ii++)
   {
    d[ii]=0.0;
    for(int kk=0;kk<n;kk++)
               {d[ii]=d[ii]-Q[ii][kk]*g[kk];}
   }
   System.out.println("j="+j+"d="+Matrix.toString(d));
   alpha=linmin(f,x,d);
   System.out.println("alfa="+alpha);
   if(alpha==0) break;
   for(i=0;i<n;i++){dx[i]=alpha*d[i];x[i]+=dx[i];ge[i]=g[i];}
   System.out.println("j="+j+"dx=\n"+Matrix.toString(dx));
   System.out.println("j="+j+"x=\n"+Matrix.toString(x));
```

```
   g=turev(f,x);
   System.out.println("j="+j+"g=\n"+Matrix.toString(g));
   for(i=0;i<n;i++){dg[i]=g[i]-ge[i];}
   gamma=norm(g);
   for(int ii=0;ii<n;ii++)
   {r[ii]=0.0;
            for(int kk=0;kk<n;kk++)
            {r[ii]+=Q[ii][kk]*dg[kk];}
   }
   rdg=VT_X(r,dg);
   dxdg=VT_X(dx,dg);
   Q1=V_VT(r);
   Q2=V_VT(dx);
   for(int ii=0;ii<n;ii++)
     for(int jj=0;jj<n;jj++)
       Q[ii][jj]=Q[ii][jj]-Q1[ii][jj]/rdg+Q2[ii][jj]/dxdg;
   gamma=norm(dx);
   System.out.println("j="+j+"Q=\n"+Matrix.toString(Q));
j++;k++;
if(k>=nn) {k=0;Q=I(n);}
}
if(j >= nmax) JOptionPane.showMessageDialog(null,"Uyari maximum iterasyon sayisi aşildi \n"+
  " çözüm geçerli olmiyabilir","MAKSİMUM ITERASYON SAYISI UYARISI",JOptionPane.WARNING_MESSAGE);
return x;
}
public static double[] davidon_fletcher_powell_V1( if_xj f,double x[])
{
int i,j,k;
int nmax=500;
double gamma=1.0;
double tolerance=1.0e-12 ;
int n=x.length;
double alpha;
double dx[]=new double[n];
double g[] =new double[n];
double ge[]=new double[n];
double dg[]=new double[n];
double d[] =new double[n];
double r[] =new double[n];
double rdg,dxdg;
double beta;
double alpha_d;
double Q[][]=I(n);
double Q1[][]=new double[n][n];
double Q2[][]=new double[n][n];
int nn=15;
i=0;
j=0;k=0;
g=turev(f,x);
while( j++ < nmax && gamma > tolerance )
{

   for(int ii=0;ii<n;ii++)
   {
    d[ii]=0.0;
    for(int kk=0;kk<n;kk++)
            {d[ii]=d[ii]-Q[ii][kk]*g[kk];}
```

```
    }
    alpha=linmin(f,x,d);
    if(alpha==0) break;
    for(i=0;i<n;i++){dx[i]=alpha*d[i];x[i]+=dx[i];ge[i]=g[i];}
    g=turev(f,x);
    for(i=0;i<n;i++){dg[i]=g[i]-ge[i];}
    gamma=norm(g);
    for(int ii=0;ii<n;ii++)
    {for(int kk=0;kk<n;kk++)
                {r[ii]=Q[ii][kk]*dg[kk];}
    }
    rdg=VT_X(r,dg);
    dxdg=VT_X(dx,dg);
    beta=dxdg/rdg;
    Q1=V_VT(r);
    Q2=V_VT(dx);
    for(int ii=0;ii<n;ii++)
      for(int jj=0;jj<n;jj++)
        Q[ii][jj]=beta*(Q[ii][jj]-Q1[ii][jj]/rdg)+Q2[ii][jj]/dxdg;
    gamma=norm(dx);
j++;k++;
if(k>=nn) {k=0;Q=I(n);}
}
if(j >= nmax) JOptionPane.showMessageDialog(null,"Uyari maximum iterasyon sayisi aşildi \n"+
  " çözüm geçerli olmiyabilir","MAKSİMUM ITERASYON SAYISI UYARISI",JOptionPane.WARNING_MESSAGE);
return x;
}
                public static double[] inputdata(String s)
    {
    String s1=JOptionPane.showInputDialog(s);
    StringTokenizer token=new StringTokenizer(s1);
    int n=token.countTokens()-1;
    int m=n+1;
    double a[]=new double[m];
    int j=0;
    while(token.hasMoreTokens())
    {
    Double ax=new Double(token.nextToken());
    a[j++]=ax.doubleValue();
    }
    return a;
    }
                public static void main (String args[])
    {
              String s="enter optimum point first estimation values with one empty space in between: ";
    double [] x0=inputdata(s);
    f1 f=new f1();
    double [] r1= davidon_fletcher_powell(f,x0);
    double [] r2= davidon_fletcher_powell_V1(f,x0);
    s=" solution DFP : \n"+Matrix.toStringT(r1);
    s+=" solution second DFP : \n"+Matrix.toStringT(r2);
    String s2="Davidon - Fletcher - Powell multivariable nonlinear function optimization:";
    JOptionPane.showMessageDialog(null,s,s2,JOptionPane.PLAIN_MESSAGE);
    System.exit(0);
    }
}
```



5.15 MULTIVARIABLE NON-LINEAR BROYDEN-FLETCHER-GOLDBERG-SHANNO (BFGS) METHOD

Conjugate gradient methods, Fletcher-Reeves, Polak-Ribiere and Davidon-Fletcher-Powell were investigated inprevious subchapters. These methods were constructing a matrix to represent inverse Hessian matrix. Broyden-Fletcher-Goldberg-Shano methods are also similar, but instead of constructing an inverse hessian representation, it construct directly a hessian matrix representation. Then inverse is calculated by using some form of equation solving method. Algorithm of such a method can be summarised as:

1. Give an initial estimate vector, $x^{(0)}$. As the first guess of the Hessianunit matrix take unit matrix $H^{(0)} = I$. Define conversion factor ε and maximum number of iterations, nmax, start to iteration with the iteration counter k=0 Calculate the gradian vector $c^{(0)} = \nabla f(x^{(0)})$
2. Calculate the norm of gradian vector, $\|c^{(k)}\|$. If norm is smaller than conversion factor stop iteration $\|c^{(k)}\| < \varepsilon$ Result is obtained.
3. Solve vector d from linear system of equation $H^{(k)}d^{(k)} = -c^{(k)}$
4. Find $\alpha^{(k)}$ parameter in d direction from minimizing of one independent variable function $f(x^{(k)} + \alpha^{(k)}d^{(k)})$
5. It should be noted in here that the first iteration is a steepest descent iteration.
6. Calculate vector $x^{(k+1)} + \alpha^{(k)}d^{(k)}$
7. Calculate new Hessian Matrix as:
8. $H^{(k+1)} = H^{(k)} + D^{(k)} + E^{(k)}$

   $D^{(k)} = \frac{y^{(k)}[y^{(k)}]^T}{(y^{(k)}s^{(k)})}$ $\quad E^{(k)} = \frac{c^{(k)}[c^{(k)}]^T}{(c^{(k)}d^{(k)})}$

   $s^{(k)} = \alpha^{(k)}d^{(k)}$ $\quad y^{(k)} = c^{(k+1)} - c^{(k)}$

   $c^{(k+1)} = \nabla f(x^{(k+1)})$

9. Increase iteration parameter k and continiue to the next iteration.

In this algorithm derivatives can be calculated numericlly if so desired. As the example problem the same problem previously investigeted will be looked at: function $f(x_0,x_1)=3x_0^2-4x_0x_1+2x_1^2-x_0-x_1$ with initial conditions {x}=[-2,2]

Program 5.15-1 Broyden - Fletcher-Goldfarb-Shanno (BFGS) method

```
// Broyden - Fletcher-Goldberg-Shanno (BFGS) method
// Method Reference : Intruduction to Optimum Design, Second Edition
// Jasbir S. Arora, Elsevier, ISBN: 0-12-064155-0
import java.util.*;
import java.awt.*;
import java.applet.Applet;
import java.awt.event.*;
import javax.swing.*;
```

```
class f1 implements if_xj
{
public double func(double x[])
{
//çözümü istenen fonksiyon
double ff;
ff=3.0*x[0]*x[0]-4.0*x[0]*x[1]+2.0*x[1]*x[1]-x[0]-x[1];
//ff=5.0*x[0]*x[0]+2.0*x[0]*x[1]+x[1]*x[1]+7.0;
return ff;
}
}

public class NA43E
{
public static double VT_V(double [] left)
{
//multiplys a vector transpose with a vector
int n=left.length;
double tot=0;
for(int i=0;i<n;i++)
  {
  tot+=left[i]*left[i];
  }
return tot;
}

public static double[][] V_VT(double [] left)
{
//multiplys a vector transpose with a vector
int n=left.length;
double aa[][]=new double[n][n];
for(int i=0;i<n;i++)
  {
  for(int j=0;j<n;j++)
    {
            aa[i][j]=left[i]*left[j];
            }
  }
return aa;
}

public static double[][] V_VT_y_s(double [] left,double [] y,double s[])
{
//multiplies a vector transpose with a vector
int n=left.length;
double aa[][]=new double[n][n];
double ss=y_s(y,s);
double x1=y_s(y,s);
for(int i=0;i<n;i++)
  {
  for(int j=0;j<n;j++)
    {aa[i][j]=left[i]*left[j]/ss;}
  }
return aa;

}
```

```
public static double y_s(double [] y,double [] s)
{
//scalar multiply two vectors
int n=y.length;
double aa=0;
for(int i=0;i<n;i++)
  {aa+=y[i]*s[i];}
return aa;
}

public static double  VT_X(double [] left,double [] right)
{
//multiplies a vector transpose with a vector
int n=left.length;
double tot=0;
for(int i=0;i<n;i++)
  {
  tot+=left[i]*right[i];
  }
return tot;
}

public static double norm(double v[])
{
// vector norm
double total=0;;
for(int i=0;i<v.length;i++)
  {total+=v[i]*v[i];}
return Math.sqrt(total);
}

public static double SIGN(double a,double b) {return (b > 0.0 ? Math.abs(a) : -Math.abs(a));}
public static double MAX(double a,double b) {return (a > b ? a : b);}


public static double linmin(if_xj f,double p[],double xi[])
{
        double tol=2.0e-10;
   int n=p.length;
   double pcom[],xicom[];
        int j;
        double xx,xmin,bx,ax;
        f1dim f1=new f1dim(f,p,xi);
        ax=0.0;
        xx=1.0;
        bx=2.0;
        double aa[]=mnbrak(f1,ax,xx,bx);
        ax=aa[0];
        xx=aa[1];
        bx=aa[2];
        xmin=brent(f1,ax,xx,bx,tol);
        return xmin;
}

public static double[][] linminiter(if_xj f,double p[],double xi[])
{
   int n=p.length;
```

```
    double xmin=linmin(f,p,xi);
    double aa[][]=new double[2][n];
            for (int j=0;j<n;j++)
            {   xi[j] *= xmin;
                    p[j] += xi[j];
                    aa[0][j]=p[j];
                    aa[1][j]=xi[j];
            }
    return aa;
}
public  static double[] brentf(if_x f,double ax,double bx,double cx,double tol)
{
int ITMAX=100;
double CGOLD=(3.0-Math.sqrt(5))/2.0; //altın oran
double ZEPS=1.0e-12;
double xmin;
double aa[]=new double[2];
            int iter;
            double a,b,d,etemp,fu,fv,fw,fx,p,q,r,tol1,tol2,u,v,w,x,xm;
            double e=0.0;
            d=0.0;
            a=((ax < cx) ? ax : cx);
            b=((ax > cx) ? ax : cx);
            x=w=v=bx;
            fw=fv=fx=f.func(x);
            for (iter=1;iter<=ITMAX;iter++) {
                    xm=0.5*(a+b);
                    tol2=2.0*(tol1=tol*Math.abs(x)+ZEPS);
                    if (Math.abs(x-xm) <= (tol2-0.5*(b-a))) {
                            xmin=x;
                aa[0]=xmin;
                aa[1]=fx;
                return aa;
                    }
                    if (Math.abs(e) > tol1) {
                            r=(x-w)*(fx-fv);
                            q=(x-v)*(fx-fw);
                            p=(x-v)*q-(x-w)*r;
                            q=2.0*(q-r);
                            if (q > 0.0) p = -p;
                            q=Math.abs(q);
                            etemp=e;
                            e=d;
                            if (Math.abs(p) >= Math.abs(0.5*q*etemp) || p <= q*(a-x) || p >= q*(b-x))
                                    d=CGOLD*(e=(x >= xm ? a-x : b-x));
                            else {
                                    d=p/q;
                                    u=x+d;
                                    if (u-a < tol2 || b-u < tol2)
                                            d=SIGN(tol1,xm-x);
                            }
                    } else {
                            d=CGOLD*(e=(x >= xm ? a-x : b-x));
                    }
                    u=(Math.abs(d) >= tol1 ? x+d : x+SIGN(tol1,d));
                    fu=f.func(u);
                    if (fu <= fx) {
```

```
                                if (u >= x) a=x; else b=x;
                                {v=w;w=x;x=u;}
                                {fv=fw;fw=fx;fx=fu;}
                        } else {
                                if (u < x) a=u; else b=u;
                                if (fu <= fw || w == x) {
                                        v=w;
                                        w=u;
                                        fv=fw;
                                        fw=fu;
                                } else if (fu <= fv || v == x || v == w) {
                                        v=u;
                                        fv=fu;
                                }
                        }
                }
                System.out.println("maximum iteration is exceeded in BRENT");
                xmin=x;
                aa[0]=xmin;
                aa[1]=fx;
                return aa;
}
public  static double[] mnbrak(if_x f,double ax,double dx)
{
return mnbrak(f,ax,(ax-dx),(ax+dx));
}

public  static double[] mnbrak(if_x f,double ax)
{
return mnbrak(f,ax,(ax-0.1),(ax+0.1));
}

public  static double[] mnbrak(if_x f,double ax,double bx,double cx)
{
int ITMAX=100;
double xmin;
double GOLD=(Math.sqrt(5.0)+1.0)/2.0;;
double GLIMIT=100.0;
double TINY=1.0e-20;
double fa,fb,fc;
fa=0;
fb=0;
fc=0;
                double ulim,u,r,q,fu,dum;
                double aa[]=new double[3];
                fa=f.func(ax);
                fb=f.func(bx);
                if (fb > fa) {
                        //SHFT(dum,*ax,*bx,dum)
                        {dum=ax;ax=bx;bx=dum;}
                        //SHFT(dum,*fb,*fa,dum)
                        {dum=fb;fb=fa;fa=dum;}
                }
                cx=(bx)+GOLD*(bx-ax);
                fc=f.func(cx);
                        aa[0]=ax;
                        aa[1]=bx;
```

```
                aa[2]=cx;
        while (fb > fc) {
                r=(bx-ax)*(fb-fc);
                q=(bx-cx)*(fb-fa);
                u=(bx)-((bx-cx)*q-(bx-ax)*r)/
                        (2.0*SIGN(MAX(Math.abs(q-r),TINY),q-r));
                ulim=(bx)+GLIMIT*(cx-bx);
                if ((bx-u)*(u-cx) > 0.0) {
                        fu=f.func(u);
                        if (fu < fc) {
                                ax=bx;
                                bx=u;
                                fa=fb;
                                fb=fu;
                                return aa;
                        } else if (fu > fb) {
                                cx=u;
                                fc=fu;
                                return aa;
                        }
                        u=(cx)+GOLD*(cx-bx);
                        fu=f.func(u);
                } else if ((cx-u)*(u-ulim) > 0.0) {
                        fu=f.func(u);
                        if (fu < fc) {
                                //SHFT(bx,cx,u,cx+GOLD*(cx-bx))
                                {bx=cx;cx=u;u=cx+GOLD*(cx-bx);}
                                //SHFT(fb,fc,fu,f.func(u))
                                {fb=fc;fc=fu;fu=f.func(u);}
                        }
                } else if ((u-ulim)*(ulim-cx) >= 0.0) {
                        u=ulim;
                        fu=f.func(u);
                } else {
                        u=cx+GOLD*(cx-bx);
                        fu=f.func(u);
                }
                //SHFT(ax,bx,cx,u)
                {ax=bx;bx=cx;cx=u;}
                //SHFT(fa,fb,fc,fu)
                {fa=fb;fb=fc;fc=fu;}
                aa[0]=ax;
                aa[1]=bx;
                aa[2]=cx;
        }
                return aa;
}
public  static double brent(if_x f,double ax,double bx,double cx,double tol)
{
 double aa[]=brentf(f,ax,bx,cx,tol);
 return aa[0];
}

public static double[][] I(int n)
{
//unit matrix
double b[][];
```

```
b=new double[n][n];
for(int i=0;i<n;i++)
   b[i][i]=1.0;
return b;
}

// Broyden - Fletcher-Goldfarb-Shanno (BFGS) method
//
// only function is given derivatives are calculated numerically
//_____________________________________________________________

public static double[] BFGS( if_xj f,double x[])
{
int i,j,k;
int nmax=100;
double gamma=1.0;
double tolerance=1.0e-6 ;
int n=x.length;
double alpha;
double s[]=new double[n];
double c[] =new double[n];
double c1[] =new double[n];
double d[] =new double[n];
double y[] =new double[n];
double alpha_d;
double H[][]=I(n);
double H1[][]=I(n);
//double H[][]={{1,0},{0,0}};
double D[][]=new double[n][n];
double E[][]=new double[n][n];
int nn=15;
j=0;k=0;
c=f.dfunc(x);
for(i=0;i<c.length;i++)
{d[i]=-c[i];} //first iteration : steepest descent
gamma=norm(c);
while( k++ < nmax && gamma > tolerance )
{
   alpha=linmin(f,x,d);
   for(i=0;i<n;i++)
   {s[i]=alpha*d[i];
    x[i]+=s[i];
   }
   c1=f.dfunc(x);
   for(i=0;i<n;i++)
   {y[i]=c1[i]-c[i];}
   D=V_VT_y_s(y,y,s);
   E=V_VT_y_s(c,c,d);
   for(i=0;i<n;i++)
   {for(j=0;j<n;j++)
   {H[i][j]=H[i][j]+D[i][j]+E[i][j];
    H1[i][j]=H[i][j];
   }
   }
   for(i=0;i<n;i++)
   {c[i]=c1[i];
    c1[i]=-c1[i];
```

```
    }
    d=pivotlugauss(H1,c1);
    gamma=norm(c);
}
if(k >= nmax) JOptionPane.showMessageDialog(null,"Warning : Maximum number of iteration exceeded \n"+
  " result may not be valid","MAXIMUM ITERATION WARNING",JOptionPane.WARNING_MESSAGE);
return x;
}

// fonksiyon ve türev fonksiyonu verilmiş
//_____________________________________________________________________

public static double[] BFGS( if_xj f,fi_xi df,double x[])
{
int i,j,k;
int nmax=100;
double gamma=1.0;
double tolerance=1.0e-6 ;
int n=x.length;
double alpha;
double s[]=new double[n];
double c[] =new double[n];
double c1[] =new double[n];
double d[] =new double[n];
double y[] =new double[n];
double alpha_d;
double H[][]=I(n);
double H1[][]=I(n);
//double H[][]={{1,0},{0,0}};
double D[][]=new double[n][n];
double E[][]=new double[n][n];
int nn=15;
j=0;k=0;
c=df.func(x);
for(i=0;i<c.length;i++)
{d[i]=-c[i];} //first iteration : steepest descent
gamma=norm(c);
while( k++ < nmax && gamma > tolerance )
{
    alpha=linmin(f,x,d);
    for(i=0;i<n;i++)
    {s[i]=alpha*d[i];
     x[i]+=s[i];
    }
    c1=df.func(x);
    for(i=0;i<n;i++)
    {y[i]=c1[i]-c[i];}
    D=V_VT_y_s(y,y,s);
    E=V_VT_y_s(c,c,d);
    for(i=0;i<n;i++)
    {for(j=0;j<n;j++)
    {H[i][j]=H[i][j]+D[i][j]+E[i][j];
     H1[i][j]=H[i][j];
    }
    }
    for(i=0;i<n;i++)
    {c[i]=c1[i];
```

```
     c1[i]=-c1[i];
    }
    d=pivotlugauss(H1,c1);
    gamma=norm(c);
}
if(k >= nmax) JOptionPane.showMessageDialog(null,"Uyarı maximum iterasyon sayısı aşıldı \n"+
  " çözüm geçerli olmıyabilir","MAKSİMUM ITERASYON SAYISI UYARISI",JOptionPane.WARNING_MESSAGE);
return x;
}


public static double[] multiply(double[] left,double right)
{
//multiplying a vector with a constant
int i;
int n=left.length;
double b[];
b=new double[n];
for(i=0;i<n;i++)
  {
  b[i]=right*left[i];
  }
return b;
}


public static double[] divide(double[] left,double[][] right)
{
return pivotlugauss(right,left);
}

  //en kucuk kareler metodu
  public static double[] pivotlugauss(double a[][],double b[])
  { //kısmi pivotlu gauss eleme yöntemi
  int n=b.length;
  double x[]=new double[n];
  double carpan=0;
  double toplam=0;
  double buyuk;
  double dummy=0;
  //gauss eleme
  int i,j,k,p,ii,jj;
  for(k=0;k<(n-1);k++)
  { //pivotlama
              p=k;
              buyuk=Math.abs(a[k][k]);
              for(ii=k+1;ii<n;ii++)
              {  dummy=Math.abs(a[ii][k]);
                 if(dummy > buyuk) {buyuk=dummy;p=ii;}
              }
              if(p!=k)
              { for(jj=k;jj<n;jj++)
                { dummy=a[p][jj];
                  a[p][jj]=a[k][jj];
                  a[k][jj]=dummy;
                }
                dummy=b[p];
```

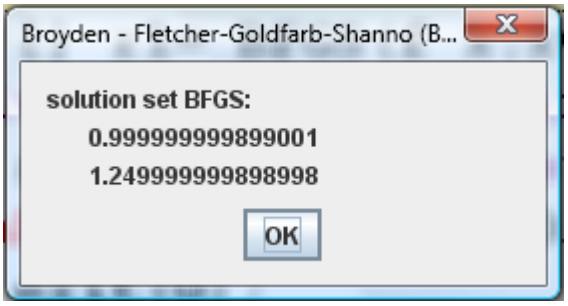
```
                b[p]=b[k];
                b[k]=dummy;
               }
               //gauss elemeyi çözme
    for(i=k+1;i<n;i++)
    {  carpan=a[i][k]/a[k][k];
       a[i][k]=0;
       for(j=k+1;j<n;j++)
       {   a[i][j]-=carpan*a[k][j]; }
          b[i]   =b[i]   -carpan*b[k];
               }
  }
  //geriye doğru yerine koyma
  x[n-1]=b[n-1]/a[n-1][n-1];
  for(i=n-2;i>=0;i--)
  {
    toplam=0;
    for(j=i+1;j<n;j++)
    {  toplam+=a[i][j]*x[j];}
    x[i]=(b[i]-toplam)/a[i][i];
  }
  return x;
}

public static double[] add(double[] left,double[] right)
{
//addition of two vectors
int n1=left.length;
int n2=right.length;
int nMax;
int i;
if(n1>=n2) nMax=n1;
else       nMax=n2;
double b[];
b=new double[nMax];
for(i=0;i<n1;i++)
  {
    b[i]=b[i]+left[i];
  }
for(i=0;i<n2;i++)
  {
    b[i]=b[i]+right[i];
  }
return b;
//end of vector addition method
}

public static double[][] add(double[][] left,double[][] right)
{
//addition of two matrices
int n1=left.length;
int m1=left[0].length;
int n2=right.length;
int m2=right[0].length;
int nMax,mMax;
int i,j;
if(m1>=m2) mMax=m1;
```

```
else      mMax=m2;
if(n1>=n2) nMax=n1;
else      nMax=n2;
double b[][];
b=new double[nMax][mMax];
for(i=0;i<n1;i++)
  {
  for(j=0;j<m1;j++)
    {
    b[i][j]=left[i][j]+right[i][j];
    }
  }
return b;
//end of matrix addition method
}
public static double[] inputdata(String s)
    {
    String s1=JOptionPane.showInputDialog(s);
    StringTokenizer token=new StringTokenizer(s1);
    int n=token.countTokens()-1;
    int m=n+1;
    double a[]=new double[m];
    int j=0;
    while(token.hasMoreTokens())
    {
    Double ax=new Double(token.nextToken());
    a[j++]=ax.doubleValue();
    }
    return a;
    }

            public static void main (String args[])
    {
            String s="enter optimum point first estimation values with one empty space in between: ";
    double [] x0=inputdata(s);
    f1 f=new f1();
    double [] r1= BFGS(f,x0);
    s=" solution set BFGS: \n"+Matrix.toStringT(r1);
    String s2="Broyden - Fletcher-Goldberg-Shanno (BFGS) multivariable optimisation:";
    JOptionPane.showMessageDialog(null,s,s2,JOptionPane.PLAIN_MESSAGE);
    System.exit(0);
    }
}
```



5.16 HOMOTOPY OR CONTINUATION METHODS

Continuation method is defined at chapter 4.3 to solve non-linear system of equations. As we can remember from Newton-Raphson method optimization problem can be defined as solution of system of nonlinear equation for the derivative of the given function. System of equations were defined as:

$$\nabla f = \begin{Bmatrix} \frac{\partial f}{\partial x_1} \\ \frac{\partial f}{\partial x_2} \\ \frac{\partial f}{\partial x_3} \\ \cdots\cdots \\ \frac{\partial f}{\partial x_n} \end{Bmatrix} \text{ and } \delta^{m+1} = \begin{Bmatrix} (x_1^{m+1} - x_1^m) \\ (x_2^{m+1} - x_2^m) \\ (x_3^{m+1} - x_3^m) \\ \cdots\cdots \\ (x_n^{m+1} - x_n^m) \end{Bmatrix} \text{and}$$

$$[H] = \begin{bmatrix} \frac{\partial^2 f}{\partial x_1^2} & \frac{\partial^2 f}{\partial x_1 \partial x_2} & \frac{\partial^2 f}{\partial x_1 \partial x_3} & \cdots & \frac{\partial^2 f}{\partial x_1 \partial x_n} \\ \frac{\partial^2 f}{\partial x_2 \partial x_1} & \frac{\partial^2 f}{\partial x_2^2} & \frac{\partial^2 f}{\partial x_2 \partial x_3} & \cdots & \frac{\partial^2 f}{\partial x_2 \partial x_n} \\ \frac{\partial^2 f}{\partial x_3 \partial x_1} & \frac{\partial^2 f}{\partial x_3 \partial x_2} & \frac{\partial^2 f}{\partial x_3^2} & \cdots & \frac{\partial^2 f}{\partial x_3 \partial x_n} \\ \cdots & \cdots & \cdots & \cdots & \cdots \\ \frac{\partial^2 f}{\partial x_1 \partial x_n} & \frac{\partial^2 f}{\partial x_2 \partial x_n} & \frac{\partial^2 f}{\partial x_3 \partial x_n} & \cdots & \frac{\partial^2 f}{\partial x_n^2} \end{bmatrix}$$

for the (k+1)th iteration

$[H^m]\{\delta^{m+1}\} = -\{\nabla f^m\}$

And for the inverse of the Hessian matrix, it can be written as

$\{\delta^{m+1}\} = [H^m]^{-1}[-\{\nabla f^m\}]$. In this equation the term $[H^m]^{-1}$ is inverse Hessian matrix .

In order to solve this equation Homotopy (Continuation) method given in chapter 4.3 will be used. Please refer to this chapter for the details of the method.

5.16-1 Continuation method for optimization (NA34F.java)

```
// NA34F Continuation-homotopy optimization
// Method Referance : Numerical Analysis, 8th edition Richard L. Burden, J. Douglas Faires 3
// page 637

import java.util.*;
import java.awt.*;
import java.applet.Applet;
import java.awt.event.*;
import javax.swing.*;

public class NA34F
{
// Continuity-homotopy method for optimisation
// (Solution of root of derivatives of function)
public static double[] multiply(double left,double[] right)
{
//multiplying a vector with a constant
int i;
int n=right.length;
double b[];
b=new double[n];
```

```
for(i=0;i<n;i++)
  {b[i]=left*right[i];}
return b;
}

public static double[] multiply(double[][] left,double[] right)
{
//multiplication of one matrix with one vector
int ii,jj,i,j,k;
int m1=left[0].length;
int n1=left.length;
int m2=right.length;
double[] b;
b=new double[m2];
 if(n1 != m2)
 {
 System.out.println("inner matrix dimensions must agree");
 for(ii=0;ii<n1;ii++)
   {
     b[ii]=0;
   }
   return b;
 }
 for(i=0;i<m1;i++)
   {
     b[i]=0;
     for(k=0;k<n1;k++)
              b[i]+=left[i][k]*right[k];
   }
return b;
//end of multiply of a matrix and a vector
}

public static double[] add(double[] left,double[] right)
{
//addition of two vectors
int n1=left.length;
int n2=right.length;
int nMax;
int i;
if(n1>=n2) nMax=n1;
else       nMax=n2;
double b[];
b=new double[nMax];
for(i=0;i<n1;i++)
  {
    b[i]=b[i]+left[i];
  }
for(i=0;i<n2;i++)
  {
    b[i]=b[i]+right[i];
  }
return b;
//end of vector addition method
}

public static double[][] inv(double[][] a)
```

```
{
// INVERSION OF A MATRIX
// inversion by using gaussian elimination
// with full pivoting
int n=a.length;
int m=a[0].length;
double b[][];
b=new double[n][n];
int indxc[];
int indxr[];
double ipiv[];
indxc=new int[n];
indxr=new int[n];
ipiv=new double[n];
int i,j,k,l,ll,ii,jj;
int icol=0;
int irow=0;
double big,dum,pivinv,temp;
if(n!=m)
{
  System.out.println("Matrix must be square ");
              for(ii=0;ii<n;ii++)
                for(jj=0;jj<n;jj++)
                        b[ii][jj]=0.0;
              return b;
}
for(i=0;i<n;i++)
  for(j=0;j<n;j++)
     b[i][j]=a[i][j];
for(i=0;i<n;i++)
{
 big=0.0;
 for(j=0;j<n;j++)
 {
  if(ipiv[j] != 1)
    for(k=0;k<n;k++)
    {
            if(ipiv[k] == 0)
            {
              if(Math.abs(b[j][k]) >= big)
              {
                big=Math.abs(b[j][k]);
                irow=j;
                icol=k;
              }
            }
            else if(ipiv[k] > 1 )
              {
              System.out.println("error : inverse of the matrix : singular matrix-1");
              for(ii=0;ii<n;ii++)
                for(jj=0;jj<n;jj++)
                        b[ii][jj]=0.0;
              return b;
              }
    }
 }
 ++ ipiv[icol];
```

```
  if(irow != icol)
    for(l=0;l<n;l++)
    {
    temp=b[irow][l];
    b[irow][l]=b[icol][l];
    b[icol][l]=temp;
    }
  indxr[i]=irow;
  indxc[i]=icol;
  if(b[icol][icol] == 0.0)
  {
    System.out.println("error : inverse of the matrix : singular matrix-2");
                for(ii=0;ii<n;ii++)
                  for(jj=0;jj<n;jj++)
                          b[ii][jj]=0.0;
    return b;
  }
  pivinv=1.0/b[icol][icol];
  b[icol][icol]=1.0;
  for(l=0;l<n;l++) b[icol][l] *=pivinv;
  for(ll=0;ll<n;ll++)
  if(ll != icol)
  {
    dum=b[ll][icol];
    b[ll][icol]=0.0;
    for(l=0;l<n;l++) b[ll][l]-= b[icol][l]*dum;
  }
}
for(l=n-1;l>=0;l--)
{
  if(indxr[l] != indxc[l])
     for(k=0;k<n;k++)
     {
       temp=b[k][indxc[l]];
       b[k][indxc[l]]=b[k][indxr[l]];
       b[k][indxr[l]]=temp;
     }
}
return b;
}
public static double SIGN(double a,double b) {return (b > 0.0 ? Math.abs(a) : -Math.abs(a));}
public static double MAX(double a,double b) {return (a > b ? a : b);}


public static double[] continuationRK4(if_xj f,double x[],int N)
{
//=================================================
// Roots of nonlinear system of equation with Homotophy-Continuaton method
// 4.th degree  Runge-Kutta equations
// yi+1 = yi + (1/6)*( k1 + 2k2 +2k3+k4)h
// k1=f(xi,yi)
// k2=f(xi+0.5h,yi+0.5k1h)
// k3=f(xi+0.5h,yi+0.5k2h)
// k4=f(xi+h,yi +k3h)
//=================================================
// x  : independent variable vector
// y  : dependent variable vetor
```

```
// dy : derivativeof the dependent vector
// N  : number of points in Runge-Kutta differential equation solution

int i;
int nmax=400;
double tolerance=1.0e-20;
int n=x.length;
double h=1.0/(double)N;
double b[]=new double[n];
double x1[]=new double[n];
double k[][]=new double[4][n];
double A[][]=new double[n][n];
b=multiply(-h,f.dfunc(x));
for(i=0;i<N;i++)
{
x1=x;
A=f.d2func(x1);
k[0]=multiply(inv(A),b);
x1=add(x,multiply(0.5,k[0]));
A=f.d2func(x1);
k[1]=multiply(inv(A),b);
x1=add(x,multiply(0.5,k[1]));
A=f.d2func(x1);
k[2]=multiply(inv(A),b);
x1=add(x,k[2]);
A=f.d2func(x1);
k[3]=multiply(inv(A),b);
for(int j=0;j<n;j++) {x[j]=x[j]+1.0/6.0*(k[0][j]+2.0*k[1][j]+2.0*k[2][j]+k[3][j]);}
}
return x;
}

public static double[] continuationRK6(if_xj f,double x[],int N)
{
//=====================================================
// Roots of nonlinear system of equation with Homotophy-Continuaton method
// 6.th degree  Runge-Kutta equations
// yi+1 = yi + (1/90)*( 7k1 + 32k3 +12k4+32k5+7k6)h
// k1=f(xi,yi)
// k2=f(xi+0.25h , yi+0.25k1h)
// k3=f(xi+0.25h , yi+0.125k1h+0.125k2h)
// k4=f(xi+0.5h , yi - 0.5k2h+k3h)
// k5=f(xi+0.75h , yi + (3/16)k1h+(9/16)k4h)
// k6=f(xi+h , yi - (3/7)k1h+(2/7)k2h+(12/7)k3h - (12/7)k4h+(8/7)k5h)
//=====================================================
// x  : independent variable vector
// y  : dependent variable vetor
// dy : derivativeof the dependent vector
// N  : number of points in Runge-Kutta differential equation solution

int i;
int nmax=400;
double tolerance=1.0e-20;
int n=x.length;
double h=1.0/(double)N;
double b[]=new double[n];
double x1[]=new double[n];
```

```
double k[][]=new double[6][n];
double A[][]=new double[n][n];
b=multiply(-h,f.dfunc(x));
for(i=0;i<N;i++)
{
x1=x;
A=f.d2func(x1);
// k1=f(xi,yi)
k[0]=multiply(inv(A),b);
x1=add(x,multiply(0.25,k[0]));
A=f.d2func(x1);
// k2=f(xi+0.25h , yi+0.25k1h)
k[1]=multiply(inv(A),b);
x1=add(x,add(multiply(0.125,k[0]),multiply(0.125,k[1])));
A=f.d2func(x1);
// k3=f(xi+0.25h , yi+0.125k1h+0.125k2h)
k[2]=multiply(inv(A),b);
x1=add(x,add(multiply(-0.5,k[1]),k[2]));
A=f.d2func(x1);
// k4=f(xi+0.5h , yi - 0.5k2h+k3h)
k[3]=multiply(inv(A),b);
x1=add(x,add(multiply((3.0/16.0),k[0]),multiply((9.0/16.0),k[3])));
A=f.d2func(x1);
// k5=f(xi+0.75h , yi + (3/16)k1h+(9/16)k4h)
k[4]=multiply(inv(A),b);
x1=add(x,
      add(multiply((-3.0/7.0),k[0]),add(multiply((2.0/7.0),k[1]),
      add(multiply((12.0/7.0),k[2]),
         add(multiply((-12.0/7.0),k[3]),multiply((8.0/7.0),k[4])))));
A=f.d2func(x1);
// k6=f(xi+h , yi - (3/7)k1h+(2/7)k2h+(12/7)k3h - (12/7)k4h+(8/7)k5h)
k[5]=multiply(inv(A),b);
// yi+1 = yi + (1/90)*( 7k1 + 32k3 +12k4+32k5+7k6)h
for(int j=0;j<n;j++) {x[j]=x[j]+1.0/90.0*(7.0*k[0][j]+32.0*k[2][j]+12.0*k[3][j]+32.0*k[4][j]+7.0*k[5][j]);}
}
return x;
}

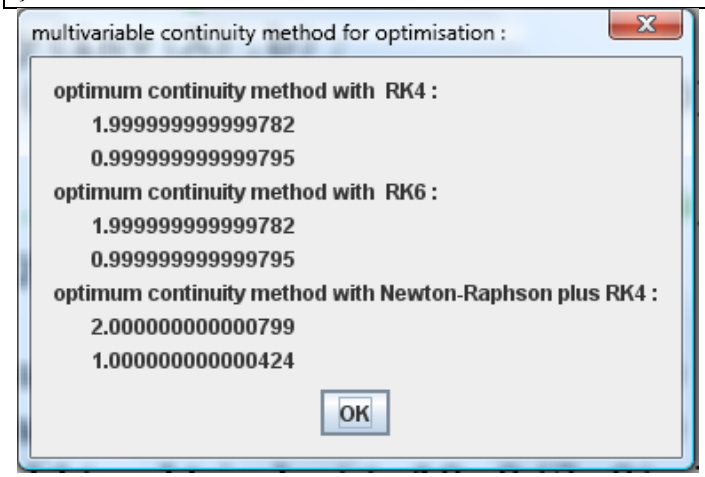
public static double[] newton_continuationRK4(if_xj f,double x[],int N)
{
// Newton-Raphson combined with Continuation (RK4) method

x=continuationRK4(f,x,4);
double ti=1.0;
int i;
int nmax=400;
double tolerance=1.0e-10;
int n=x.length;
double b[];
b=new double[n];
for(i=0;i<n;i++)
{
b[i]=1.0;
}
i=0;
while( i++ < nmax && Matrix.abs(b) > tolerance )
{ b=Matrix.multiply(Matrix.divide(f.dfunc(x),f.d2func(x)),-ti);
```

```
  x=Matrix.add(x,b);
}
if(i >= nmax) JOptionPane.showMessageDialog(null,"Uyarı maximum iterasyon sayısı aşıldı \n"+
            " çözüm geçerli olmıyabilir","MAKSİMUM ITERASYON SAYISI
UYARISI",JOptionPane.WARNING_MESSAGE);
System.out.println();
return x;
}

public static double[] inputdata()
{
   String s=JOptionPane.showInputDialog("kök tahmin değerlerini arada bir boşluk bırakarak giriniz: ");
   StringTokenizer token=new StringTokenizer(s);
   int n=token.countTokens()-1;
   int m=n+1;
   double a[]=new double[m];
   int j=0;
   while(token.hasMoreTokens())
   {
   Double ax=new Double(token.nextToken());
   a[j++]=ax.doubleValue();
   }
   return a;
}

    public static void main (String args[])
    {
    double [] x0=inputdata();
    if_xj ff=(double x[])->2.0*x[0]*x[1]+2.0*x[0]-x[0]*x[0]-2.0*x[1]*x[1];
    double [] r= continuationRK4(ff,x0,10);
    double [] r1=continuationRK6(ff,x0,10);
    double [] r2=newton_continuationRK4(ff,x0,10);
    String s=" optimum continuity method with  RK4 : \n"+Matrix.toStringT(r);
    s+=" optimum continuity method with  RK6 : \n"+Matrix.toStringT(r1);
    s+=" optimum continuity method with Newton-Raphson plus RK4 : \n"+Matrix.toStringT(r2);
    String s1="multivariable continuity method for optimisation : ";
    JOptionPane.showMessageDialog(null,s,s1,JOptionPane.PLAIN_MESSAGE);
    System.exit(0);
    }
}
```



5.17 MULTIVARIABLE NONLINEAR BROYDEN SECANT METHOD WITH SHERMAN MORRISON FORMULA

Broyden secant method was defined in chapter 4.2. It is basically secant root finding method for multivariable functions.

As we can remember from Newton-Raphson method optimization problem can be defined as solution of system of nonlinear equation for the derivative of the given function. System of equations were defined as:

$$\nabla f = \begin{Bmatrix} \frac{\partial f}{\partial x_1} \\ \frac{\partial f}{\partial x_2} \\ \frac{\partial f}{\partial x_3} \\ \cdots\cdots \\ \frac{\partial f}{\partial x_n} \end{Bmatrix} \text{ and } \delta^{m+1} = \begin{Bmatrix} (x_1^{m+1} - x_1^m) \\ (x_2^{m+1} - x_2^m) \\ (x_3^{m+1} - x_3^m) \\ \cdots\cdots \\ (x_n^{m+1} - x_n^m) \end{Bmatrix} \text{ and } [H] = \begin{bmatrix} \frac{\partial^2 f}{\partial x_1^2} & \frac{\partial^2 f}{\partial x_1 \partial x_2} & \frac{\partial^2 f}{\partial x_1 \partial x_3} & \cdots & \frac{\partial^2 f}{\partial x_1 \partial x_n} \\ \frac{\partial^2 f}{\partial x_2 \partial x_1} & \frac{\partial^2 f}{\partial x_2^2} & \frac{\partial^2 f}{\partial x_2 \partial x_3} & \cdots & \frac{\partial^2 f}{\partial x_2 \partial x_n} \\ \frac{\partial^2 f}{\partial x_3 \partial x_1} & \frac{\partial^2 f}{\partial x_3 \partial x_2} & \frac{\partial^2 f}{\partial x_3^2} & \cdots & \frac{\partial^2 f}{\partial x_3 \partial x_n} \\ \cdots & \cdots & \cdots & \cdots & \cdots \\ \frac{\partial^2 f}{\partial x_1 \partial x_n} & \frac{\partial^2 f}{\partial x_2 \partial x_n} & \frac{\partial^2 f}{\partial x_3 \partial x_n} & \cdots & \frac{\partial^2 f}{\partial x_n^2} \end{bmatrix}$$

$[H^m]\{\delta^{m+1}\} = -\{\nabla f^m\}$

And for the inverse of the Hessian matrix, it can be written as

$\{\delta^{m+1}\} = [H^m]^{-1}[-\{\nabla f^m\}]$. In this equation the term $[H^m]^{-1}$ is inverse Hessian matrix .

Hesian matrix can be replaced by numerical differencedefinition of the essian $[H]$ as

$$\frac{\partial g_j(x^{(i)})}{\partial x_k} = \frac{g_j(x^{(i)} + e_k h) - g_j(x^{(i)})}{h}$$

h in the equation is finite difference parameter (Δx) and ek is unit vektor (kth componet is 1 the other components are 0). i is the iteration number. Note that secant method is always approach to the roots less quickly compare to Newton-Raphson method. In order to improve number of iterations for the solutions, a modified version of secant method, Broyden method will be used. In Broyden method when x(0) is a given value, first iteration, x(1), will be calculated by using Newton-Raphson method. For the following steps

$$\left[H(x^{(0)}\right]\left\{x^{(")} - x^{(0)}\right\} = -\left\{g(x^{(0)}\right\}$$

form of the equation can be changed to be

$$[A_1]\left\{x^{(1)} - x^{(0)}\right\} = \left\{g(x^{(1)} - g(x^{(0)}\right\}$$

In order to have this equation valid for every vector z, vector should satisfy following equations

$[A_1]\{z\} = \left[\nabla g(x^{(0)}\right]z$

By using this defination Formula to calculate $[A_1]$ can be derived.

$$[A_1] = H\left(x^{(0)}\right) + \frac{\left[\left\{g(x^{(1)} - g(x^{(0)}\right\} - H\left(x^{(0)}\right)\left(x^{(1)} - x^{(0)}\right)\right]\left(x^{(1)} - x^{(0)}\right)^t}{\|x^{(1)} - x^{(0)}\|_2^2}$$

Where the term $\left\|x^{(1)} - x^{(0)}\right\|_2$ is a two norm. Norms was defined in previous chapter. After finding $[A_1]$ new value for vector x can be constructed.

$x^{(2)} = x^{(1)} - [A_1]^{-1} g(x^{(1)})$

When $x^{(2)}$ is found, this value can be used to calculate$x^{(3)}$.General Formula will become:

$$[A_i] = [A_{i-1}] + \frac{\left[\left\{g(x^{(i)} - g(x^{(i-1)}\right\} - [A_{i-1}]\left(x^{(i)} - x^{(i-1)}\right)\right]\left(x^{(i)} - x^{(i-1)}\right)^t}{\|x^{(i)} - x^{(i-1)}\|_2^2}$$

$x^{(i+1)} = x^{(i)} - [A_i]^{-1} g(x^{(i)})$

One difficulty in this equation is the requirement of solving inverse of the matrix each time which requires solving of system of equation by a method such as Gauss elimination. But Gauss method is expensive in calculation time. A beter alternative could be using Sherman-Morrison approach. This approach calculates inverse directly as an iterative process. Matrix A_i^{-1} can directly be calculated from value A_{i-1}^{-1}. Equations of Sherman-Morrison approach is as follows

$$s_i = \{x^{(i)} - x^{(i-1)}\}$$
$$y = \{f(x^{(i)}) - f(x^{(i-1)})\}$$
$$(A + xy^t)^{-1} = A^{-1} - \frac{A^{-1}xy^tA^{-1}}{1 + y^tA^{-1}x}$$

$$A_i^{-1} = \left(A_{i-1}^{-1} + \frac{y_i - A_{i-1}s_i}{\|s_i\|_2^2}\right)^{-1}$$

$$A_i^{-1} = A_{i-1}^{-1} + \frac{A_{i-1}^{-1}\left(A_{i-1}^{-1} + \frac{y_i - A_{i-1}s_i}{\|s_i\|_2^2}s_i^t\right)A_{i-1}^{-1}}{1 + s_i^tA_{i-1}^{-1}\left(\frac{y_i - A_{i-1}s_i}{\|s_i\|_2^2}s_i^t\right)}s_i^t$$
$$A_i^{-1} = A_{i-1}^{-1} + \frac{(s_i - A_{i-1}^{-1}y_i)s_i^tA_{i-1}^{-1}}{s_i^tA_{i-1}^{-1}y_i}$$

```
 // NA34G Broyden secant method with Sherman-Morrison formula
// optimizasyon
// Reference : Numerical Analysis, 8th edition Richard L. Burden, J. Douglas Faires 3
// page 617

import java.util.*;
import java.awt.*;
import java.applet.Applet;
import java.awt.event.*;
import javax.swing.*;
//============= Function to be defined ================

class f1 extends f_xj
{
public double func(double x[])
{
//çözümü istenen fonksiyon
double ff;
//ff=2.0*x[0]*x[1]+2.0*x[0]-x[0]*x[0]-2.0*x[1]*x[1];
ff=3.0*x[0]*x[0]-4.0*x[0]*x[1]+2.0*x[1]*x[1]-x[0]-x[1];
return ff;
}
}

class fxb extends fi_xi
{
public double[] func(double x[])
{
double ff[]=new double[2];
```

```
ff[0]=6.0*x[0]-4.0*x[1]-1.0;
ff[1]=-4.0*x[0]+4.0*x[1]-1.0;
return ff;
}
}

class fb extends fij_xi
{
fi_xi f;

public fb(fi_xi fi)
{f=fi;}

public double[][] func(double x[])
{
  int n=x.length;
  double a[][]=new double[n][n];
  for(int i=0;i<n;i++)
  {
    for(int j=0;j<n;j++)
    {
       a[i][j]=turev(x,i,j);
    }
  }
return a;
}

public double turev(double x[],int denklem_ref,int x_ref)
{
double h0=0.0256;
int i,m;
int n=7;
double f1[];
f1=new double[x.length];
double f2[];
f2=new double[x.length];
double x1[];
x1=new double[x.length];
double x2[];
x2=new double[x.length];
for(i=0;i<x.length;i++)
{
x1[i]=x[i];
x2[i]=x[i];
}
double T[][];
T=new double[n][n];
double h[];
h=new double[n];
for(i=0;i<n;i++)
{
  h[i]=0;
  for(int j=0;j<n;j++)
    T[i][j]=0;
}
h[0]=h0;
double r=0.5;
```

```
for( i=1;i<n;i++)
{
h[i]=h0*Math.pow(r,i);
}

for(i=0;i<n;i++)
{
x1[x_ref]+=h[i];
x2[x_ref]-=h[i];
f1=f.func(x1);
f2=f.func(x2);
T[i][0]=( f1[denklem_ref] - f2[denklem_ref])/(2.0*h[i]);
x1[x_ref]=x[x_ref];
x2[x_ref]=x[x_ref];
}
for(m=1;m<n;m++)
{
  for(i=0;i<n-m;i++)
  {
  T[i][m]=(h[i]*h[i]*T[i+1][m-1] - h[i+m]*h[i+m]*T[i][m-1])/(h[i]*h[i]
  - h[i+m]*h[i+m]);
  }
}
double xx=T[0][n-1];
return xx;
}

}

public class NA34G
{
//Broyden secant method with Sherman-Morrison formula
public static double[] multiply(double left,double[] right)
{
//multiplying a vector with a constant
int i;
int n=right.length;
double b[];
b=new double[n];
for(i=0;i<n;i++)
  {b[i]=left*right[i];}
return b;
}

public static double[] multiply(double[][] left,double[] right)
{
//multiplication of one matrix with one vector
int ii,jj,i,j,k;
int m1=left[0].length;
int n1=left.length;
int m2=right.length;
double[] b;
b=new double[m2];
 if(n1 != m2)
 {
 System.out.println("inner matrix dimensions must agree");
 for(ii=0;ii<n1;ii++)
```

```
   {
     b[ii]=0;
   }
   return b;
 }
 for(i=0;i<m1;i++)
   {
     b[i]=0;
     for(k=0;k<n1;k++)
              b[i]+=left[i][k]*right[k];
   }
return b;
//end of multiply of a matrix and a vector
}

public static double[] multiply(double[] left,double[][] right)
{
//multiplication of one vector with one matrix
int ii,jj,i,j,k;
int m2=right[0].length;
int n2=right.length;
int m1=left.length;
double[] b;
b=new double[m1];
 if(n2 != m1)
 {
 System.out.println("inner matrix dimensions must agree");
 for(ii=0;ii<n2;ii++)
   {
     b[ii]=0;
   }
   return b;
 }
 for(i=0;i<m2;i++)
   {
     b[i]=0;
     for(k=0;k<m1;k++)
              b[i]+=right[i][k]*left[k];
   }
return b;
//end of multiply of a vector and a matrix
}

public static double  VT_X(double [] left,double [] right)
{
//multiplies a vector transpose with a vector
int n=left.length;
double tot=0;
for(int i=0;i<n;i++)
  {
  tot+=left[i]*right[i];
  }
return tot;
}

public static double[] add(double[] left,double[] right)
{
```

```
//addition of two vectors
int n1=left.length;
int n2=right.length;
int nMax;
int i;
if(n1>=n2) nMax=n1;
else       nMax=n2;
double b[];
b=new double[nMax];
for(i=0;i<n1;i++)
  {
    b[i]=b[i]+left[i];
  }
for(i=0;i<n2;i++)
  {
    b[i]=b[i]+right[i];
  }
return b;
//end of vector addition method
}

public static double[][] multiply(double left,double[][] right)
{
//multiplying a matrix with a constant
int i,j;
int n=right.length;
int m=right[0].length;
double b[][];
b=new double[n][m];
for(i=0;i<n;i++)
  {
  for(j=0;j<m;j++)
    b[i][j]=right[i][j]*left;
  }
return b;
//end of multiplying a matrix with a constant double
}

public static double[][] add(double[][] left,double[][] right)
{
//addition of two matrices
int n1=left.length;
int m1=left[0].length;
int n2=right.length;
int m2=right[0].length;
int nMax,mMax;
int i,j;
if(m1>=m2) mMax=m1;
else       mMax=m2;
if(n1>=n2) nMax=n1;
else       nMax=n2;
double b[][];
b=new double[nMax][mMax];
for(i=0;i<n1;i++)
  {
  for(j=0;j<m1;j++)
    {
```

```
    b[i][j]=b[i][j]+left[i][j];
    }
  }
for(i=0;i<n2;i++)
  {
  for(j=0;j<m2;j++)
    {
    b[i][j]=b[i][j]+right[i][j];
    }
  }
return b;
//end of matrix addition method
}

public static double[] substract(double[] left,double[] right)
{
//addition of two vectors
int n1=left.length;
int n2=right.length;
int nMax;
int i;
if(n1>=n2) nMax=n1;
else       nMax=n2;
double b[];
b=new double[nMax];
for(i=0;i<n1;i++)
  {
    b[i]=b[i]+left[i];
  }
for(i=0;i<n2;i++)
  {
    b[i]=b[i]-right[i];
  }
return b;
//end of vector addition method
}

public static double[][] inv(double[][] a)
{
// INVERSION OF A MATRIX
// inversion by using gaussian elimination
// with full pivoting
int n=a.length;
int m=a[0].length;
double b[][];
b=new double[n][n];
int indxc[];
int indxr[];
double ipiv[];
indxc=new int[n];
indxr=new int[n];
ipiv=new double[n];
int i,j,k,l,ll,ii,jj;
int icol=0;
int irow=0;
double big,dum,pivinv,temp;
if(n!=m)
```

```
{
  System.out.println("Matrix must be square ");
              for(ii=0;ii<n;ii++)
                for(jj=0;jj<n;jj++)
                        b[ii][jj]=0.0;
              return b;
}
for(i=0;i<n;i++)
  for(j=0;j<n;j++)
     b[i][j]=a[i][j];
for(i=0;i<n;i++)
{
 big=0.0;
 for(j=0;j<n;j++)
 {
  if(ipiv[j] != 1)
    for(k=0;k<n;k++)
    {
            if(ipiv[k] == 0)
            {
              if(Math.abs(b[j][k]) >= big)
              {
                big=Math.abs(b[j][k]);
                irow=j;
                icol=k;
              }
            }
            else if(ipiv[k] > 1 )
              {
              System.out.println("error : inverse of the matrix : singular matrix-1");
              for(ii=0;ii<n;ii++)
                for(jj=0;jj<n;jj++)
                        b[ii][jj]=0.0;
              return b;
              }
    }
 }
 ++ ipiv[icol];
 if(irow != icol)
  for(l=0;l<n;l++)
  {
  temp=b[irow][l];
  b[irow][l]=b[icol][l];
  b[icol][l]=temp;
  }
 indxr[i]=irow;
 indxc[i]=icol;
 if(b[icol][icol] == 0.0)
 {
  System.out.println("error : inverse of the matrix : singular matrix-2");
              for(ii=0;ii<n;ii++)
                for(jj=0;jj<n;jj++)
                        b[ii][jj]=0.0;
  return b;
 }
 pivinv=1.0/b[icol][icol];
 b[icol][icol]=1.0;
```

```
  for(l=0;l<n;l++) b[icol][l] *=pivinv;
  for(ll=0;ll<n;ll++)
  if(ll != icol)
  {
    dum=b[ll][icol];
    b[ll][icol]=0.0;
    for(l=0;l<n;l++) b[ll][l]-= b[icol][l]*dum;
  }
}
for(l=n-1;l>=0;l--)
{
  if(indxr[l] != indxc[l])
     for(k=0;k<n;k++)
     {
       temp=b[k][indxc[l]];
       b[k][indxc[l]]=b[k][indxr[l]];
       b[k][indxr[l]]=temp;
     }
}
return b;
}
public static double norm(double v[])
{
// vector norm
double total=0;;
for(int i=0;i<v.length;i++)
  {total+=v[i]*v[i];}
return Math.sqrt(total);
}

public static double[][] multiply(double[] left,double[] right)
{
//multiplication of one vector with one matrix
int ii,jj,i,j,k;
int m2=left.length;
int n2=right.length;
int m1=left.length;
double[][] b;
b=new double[m2][n2];
 for(i=0;i<m2;i++)
   {
     for(k=0;k<n2;k++)
                b[i][k]=left[i]*right[k];
   }
return b;
//end of multiply of a vector and a matrix
}

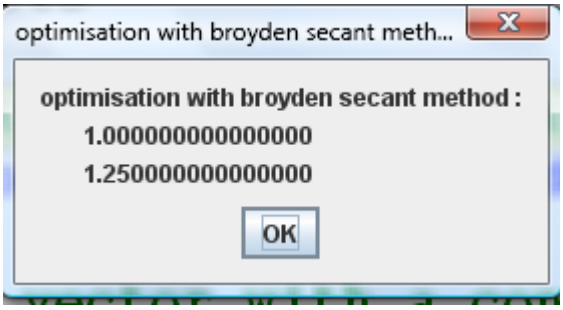
public static double[] broyden(fi_xi f,double x[])
{
//=================================================
// roots of non-linear system of equation
fb J=new fb(f);
int i;
int nmax=400;
double tolerance=1.0e-15;
int n=x.length;
```

```
double b[]=new double[n];
double v[]=new double[n];
double s[]=new double[n];
double w[]=new double[n];
double y[]=new double[n];
double z[]=new double[n];
double p;
double ut[]=new double[n];
double x1[]=new double[n];
double A[][]=new double[n][n];
x1=x;
A=J.func(x1);
v=f.func(x1);
A=inv(A);
s=multiply(-1.0,multiply(A,v));
x=add(x,s);
int k=1;
while(k<nmax)
{
w=v;
v=f.func(x);
y=substract(v,w);
z=multiply(-1.0,multiply(A,y));
p=-VT_X(s,z);
ut=multiply(s,A);  //*********
A=add(A,multiply((1.0/p),multiply(add(s,z),ut)));
s=multiply(-1.0,multiply(A,v));
x=add(x,s);
if(norm(s)<tolerance) break;
k++;
}
return x;
}

public static double[] inputdata()
{
   String s=JOptionPane.showInputDialog("enter optimum point first estimation values with one empty space in between:
");
   StringTokenizer token=new StringTokenizer(s);
   int n=token.countTokens()-1;
   int m=n+1;
   double a[]=new double[m];
   int j=0;
   while(token.hasMoreTokens())
   {
   Double ax=new Double(token.nextToken());
   a[j++]=ax.doubleValue();
   }
   return a;
}

    public static void main (String args[])
    {
            fa f =new fa();
            fxb f1=new fxb();
    double [] x0=inputdata();
    double [] r= broyden(f1,x0);
```

```
    String s=" optimisation with broyden secant method : \n"+Matrix.toStringT(r);
    String s1="optimisation with broyden secant method  : ";
    JOptionPane.showMessageDialog(null,s,s1,JOptionPane.PLAIN_MESSAGE);
    System.exit(0);
    }
}
```



5.18 SCHOLASTIC METHODS : GENETIC ALGORITHMS

5.18.1 NATURAL SELECTION AND SYSTEM OF GENES IN NATURE

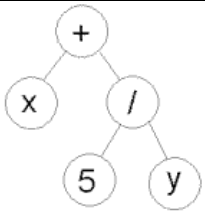
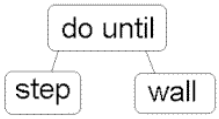
Genetic algorithms are a widely use modern programming method. This method based on natural selection mechanism in nature. Characteristics are carried in GENES, Father and Mother passes their genes to the child. Individuals carrying genes that given better characteristics to fit the nature has a better chance of survival and to give offsprings, therefore more of their gene characteristics will be carried into the next generation. Sometimes gene structure passing throgh from parents can be changed by a mechanism called mutation. It is usuall a very small change. The result of this change could be deadly(less survival potential in environment) or sometimes make the individual fit the envronment better. If that is the case, his chance of passing his genes including this mutated one will be improved.

5.18.2 GENETIC ALGORITHMS, GENERAL DEFINITIONS

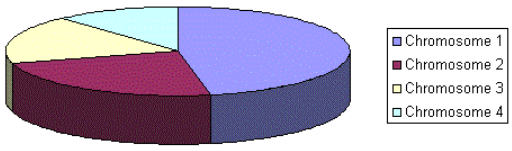
In general an algorithm of optimization by genetic algorithms can be summerized as:

[START] Set up a population of m member (one chromosome for each member, x gene-vector of independent variables)
[START] for each chromosome (vector of independent variables) calculate *f(x)* fitting value, repeat this for each individual in population
[NEW GENERATION] Generate new generations according to the following steps
[SELECTION] Select parents from population by considering their fitting values. Better fits should have a better chance to be selected but no one is eliminated completely from the selection process.
[PRODUCTION] Produce a new individual(s) from the parents. Replace parent generation with the new generation. Some of the successful parents can retain in the new generation
[MUTATION] mutate new generations with a ceratain (given) mutation rate
[ACCEPT] Add new individuals to the population.
[REPEAT] use the new generations to create new populations by using the same rule
[TEST] when the required conditions are occured or enough generations are generated stop the simulation
[LOOP] If the required conditions are not occured continue the process

Gens an choromosomes are the basic calculating units of the genetic calculations. Mathematically all the independent variales are an individual gene. . Chromsome is the vector of genes(independent varables). SAmple of some chromosome types utilised fort he computer simulation are given as the following table

chromosome A **chromosome B** **chromosome C**	**5.3243 0.4556 2.3293 2.4545** **ABDJEIFJDHDIERJFDLDFLFEGT** **(back), (foward), (right), (left), (turn)**
chromosome D	 (+ x (/ 5 y))
chromosome E	 (do_until step wall)

In the natural genes the fitting function is the survival in the nature, in mathmatical genes optimization (Maximisation) of the fitting function will be the base. Fitting functions usually added up and converted into the percantage form

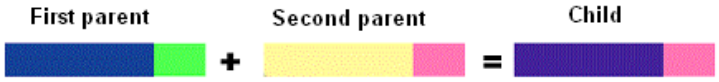


This percentage is called converted fitting function. Percentage in the total pipe is biger for the individuals with a beter fitting functions. Parents selected by using converted fitting functions and random number, so that individuals with a beter fitting has a beter chance to be selected. From the selected population of parent candidates actual parents are also selected by random selection. Then from parents gene the gene of th new population is created. Mathematically this process is called crossing. In crossing process genes of each parents are randomly selected to create the new child. For example if the gene is binary numbers and if single crossing is used, both parents gene is divided from a randomly selected point the first part of first parent is combined with the second part of second parent for the second part of the first parent combined with the first part of the second parent to crate a child.

For example for a binary group of 8 bit:

If 8*random(0-1) = 5 and if the first 5 bit is taken from the first parent and the remaining bits from the second parent : The new child:

11001011 + 11011111 = 11001111



Chromosome 1 (First Parent)	11011 | 00100110110
Chromosome 2 (Second parent)	11011 | 11000011110
1. Child	11011 | 11000011110
2. Child	11011 | 00100110110

If double crossing applied, two random numbers are generated and genes are divided into three parts. The first part is taken from the first parent, second part is taken from the second parent and the third part is taken from the first parent(or vice versa)

For example for a binary group of 8 bit i:

Random1=8*random(0-1) = 2

Random2=8*random(0-1) = 6

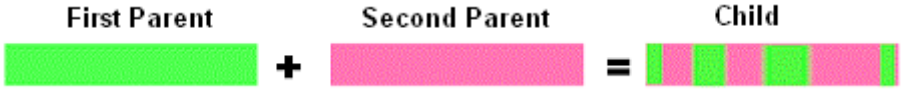
First 2 bit is taken from the first parent, from third to 6th bit is taken from the second parent and 7th and 8th bit is taken from the first parent.

11001011 + 11011111 = 11011111



As a third method every gene is selected by a random number decision process. If generated random number is smaller than 0.5 the first parent's binary digit is selected, if not second parents binary digit is selected.

11001011 + 11011111 = 11001011



5.18.3 GENETIC ALGORITHMS, OPTIMIZATION

In this section actual optimization process with genetic algorithms will be given in details. AS it is seen earlier in this chapter optimization process is minimisation or maximisation of a multivariable function

$$f(x_0, x_1, x_2, x_3, .., x_i)$$

It should again be reminded that maximisation and minimisation is the same process with a sign change

$$\max\left(f(x) = \min\left(-f(x)\right) = \min\left(g(x)\right)\right.$$

$$g(x) = -f(x)$$

Another mathematical fact that will be used in genetic algorithms is that addition of a constant to the function will not change the optimum point

$$\max\left(f(x) = \max(f(x) + C) = \max\left(g(x)\right)\right.$$

$$\min f(x) = \min(f(x) + C) = \min\left(g(x)\right)$$

$$g(x) = f(x) + C$$

$$\frac{dg(x)}{dx} = \frac{df(x)}{dx}$$

In genetic algorithm applications the nature of the algorithm is to find the maximum, in order to get correct percentages constant numbers are added into the function to ensure the resulting value is always positive.

In engineering optimization we usually face the real numbers. If a real number is to be represented as a binary sequence, a conversion process between binary digits and real numbers are needed. If n+1 independent variable is existed as

$$(x_0, x_1, x_2, x_3, .., x_i)$$

In order to apply genetic algorithms lower and upper bounds of each variable should be known.

xo : xo_minimum xo_maximum
x1 : x1_minimum x1_maximum
….
xi : xi_minimum xi_maximum
…
xn : xn_minimum xn_maximum

definitins should be given. Ifbinary digit system is used number of representation digits also should be known. If the independent variable is defined as

$$x_{i_min} \leq x_i \leq x_{i_max} \quad N\ bit\ array$$

Binary equivalent of N digits can be calculated as follows:
Binary group with n digit first converted into a 10 base integer number as

$$(b_{N-1}, b_{N-2}, \dots, b_1, b_0) = \left(\sum_{i=0}^{N-1} b_i 2^i\right)_{10} = x'$$

Then the integer number is converted into a binary digit by using the following conversion process

$$x = x_{min} + x'\left(\frac{x_{max} - x_{min}}{2^N - 1}\right)$$

Inverse process can also be apply to convert digits back to the binary digits. Each independent variables can be given with one gene, and the set of independent variables can be given b a a vector of genes. Evaluation function is assumd to be in the following form

```
class fa extends f_xj
{
 double func (double x[])
 { return (21.5+x[0]*Math.sin(4.0*Math.PI*x[0])+x[1]*Math.sin(20.0*Math.PI*x[1]));
 }}
```

The value sets of independent variables as whole together with the evaluated function value will be called as genotype (as borrowed from nature's genetic system). Number of chromosomes in each individual can be one or more. Number of individual will be set by the user of the program. For each individual starting variable values set by using random selection. Function values are calculated for each genotype. It is the fitting value of the genome. All fitting values are adde and then normalise to get percantage base fitting values. Percantage base fitting values are adde by starting from zero to 1and created cumulative fitting scale for all population. Cumulative fitting values are based on the scale of the fitting function. If a random number between 0 to 1 is selected, the chance of this number is fall in a region with a bigger fitting function is grater than that of a small one. After an individual in the population is selected as a parent cendidate, it can be removed from the selection list or it can remain in the list(a second selection can be allowed). From the selected parent candidates two parents are selected randomly to create a new child.In the new child population a selected percentage can be allowed (selected again randomly) to mutate. Mutation in a binary code can be achived to flip an

indivudual binary digit(from 0 to 1 or from 1 to 0). Process continues for a defined number of generations.

5.18.4 GENETIC ALGORITHMS, CLASS GENE

Genetic algorithms mimic the natural selection process of genes. The first structure used here to mimic natural genes are used binary numbers as defined in previous subchapter. Constructor of gene class can accept either double or binary numbers. Total bit length is a variable but it is limited to 64 bits (0—63). Gene class consist of gene crossing and mutation processes as well.

Program 5.18.4-1 Gene class

```
import java.util.*;
import java.io.*;

public class Gene extends Object
{
//Gene class generates digital(binary) array to represent a double number
//actual binary defination changes depends on number of bits, xmin and xmax
//binary data stored in b BitSet array
int N;      //number of binary digits(number of bits in a byte)
        //N should be less than 64
double xN;//Limit value for integer(2^N-1)
double xmin,xmax;//maximum and minimum limits of double number
BitSet b; //binary array that strore the integer

public Gene(double ixmin,double ixmax,int iN)
{//randomly generate the gene
 N=iN;
 if(N>63) N=63;
 b=new BitSet(N);
 xN=Math.pow(2.0,N)-1;
 xmin=ixmin;
 xmax=ixmax;
 //generate random numbers to fill binary array
 setbits();
}

public Gene(String s,double ixmin,double ixmax)
{
//generate gene equal to a given string
 N=s.length();
 b=new BitSet(N);
 xN=Math.pow(2.0,N)-1;
 xmin=ixmin;
 xmax=ixmax;
 for(int i=N-1;i>=0;i--)
   if(s.charAt(i)=='1') b.set(N-1-i);
}

public Gene(double number,double ixmin,double ixmax,int iN)
{
//Generate a gene from a given double number
 N=iN;//number of binary digits(number of bits in a byte)
 if(N>63) N=63;
 b=new BitSet(N);
 xN=StrictMath.pow(2.0,N)-1;
```

```
 xmin=ixmin;
 xmax=ixmax;
 //z : integer equivalent of the double number
long z=(long)((number-xmin)/(xmax-xmin)*xN);
//System.out.println("inside double"+z);
long y=(long)xN;
double r=z;
for(int i=0;i<N;i++)
   {
   if((z-(z/2)*2)!=0) {b.set(i);}
   z/=2;
   }
}

public Gene()
{
//empty Gene
N=1;
b=new BitSet(N);
xN=StrictMath.pow(2.0,N)-1;
xmin=0;
xmax=1.0;
b.set(0);
}

public Gene(Gene g)
{
// copy a Gene from another one
copyGene(g);
}

public void copyGene(Gene g)
{
 N=g.N;
 b=(BitSet)g.b.clone();
 xN=Math.pow(2.0,N)-1;
 xmin=g.xmin;
 xmax=g.xmax;

}

public Gene copyGene()
{
 Gene g=new Gene(this);
 return g;
}
public Gene(Gene Father,Gene Mother)
{
//crosover type will be selected randomly
double ix=Math.random();
if(ix<0.1) setGene(Father,Mother,"regular");
else if(ix>0.9) setGene(Father,Mother,"single");
else setGene(Father,Mother,"double");
}
public void  setGene(Gene Father,Gene Mother,String crossovertype)
{
xmin=Father.xmin;
```

```
xmax=Father.xmax;
N=Father.N;
xN=Father.xN;
b=new BitSet(N);
int ir1,ir2,ir3;
if(crossovertype.equals("regular"))
{
for(int i=0;i<N;i++)
  {if(boolean_random()) { if(Father.b.get(i)) b.set(i);}
   else { if(Mother.b.get(i)) b.set(i);}
  }
}
else if(crossovertype.equals("single"))
{
   ir1=int_random();
   if(boolean_random())
   {   for(int i=0;i<ir1;i++)
         { if(Father.b.get(i)) b.set(i);}
       for(int i=ir1;i<N;i++)
         { if(Mother.b.get(i)) b.set(i);}
   }
   else
   {   for(int i=0;i<ir1;i++)
         { if(Mother.b.get(i)) b.set(i);}
       for(int i=ir1;i<N;i++)
         { if(Father.b.get(i)) b.set(i);}
   }
}
else if(crossovertype.equals("double"))
{
   ir1=int_random();
   ir2=int_random();
   int ix;
   if(ir1>ir2) {ix=ir2;ir2=ir1;ir1=ix;}
   if(boolean_random())
   {   for(int i=0;i<ir1;i++)
         { if(Father.b.get(i)) b.set(i);}
       for(int i=ir1;i<ir2;i++)
         { if(Mother.b.get(i)) b.set(i);}
       for(int i=ir2;i<N;i++)
         { if(Father.b.get(i)) b.set(i);}
   }
   else
   {   for(int i=0;i<ir1;i++)
         { if(Mother.b.get(i)) b.set(i);}
       for(int i=ir1;i<ir2;i++)
         { if(Father.b.get(i)) b.set(i);}
       for(int i=ir2;i<N;i++)
         { if(Mother.b.get(i)) b.set(i);}
   }
}
}
public void cross1(double crossratio,Gene Father,Gene Mother)
{// create gene from two parent genes
 // crossover process
//  Genes from Father and Mother should have the same limit values
xmin=Father.xmin;
```

```
xmax=Father.xmax;
N=Father.N;
xN=Father.xN;
b=new BitSet(N);
int ir1=(int)(crossratio*N+0.5);
for(int i=0;i<ir1;i++)
  { if(Father.b.get(i)) b.set(i);}
for(int i=ir1;i<N;i++)
  { if(Mother.b.get(i)) b.set(i);}
}
public void cross2(double crossratio,Gene Father,Gene Mother)
{// create gene from two parent genes
 // crossover process
//  Genes from Father and Mother should have the same limit values
xmin=Father.xmin;
xmax=Father.xmax;
N=Father.N;
xN=Father.xN;
b=new BitSet(N);
int ir1=(int)(crossratio*N+0.5);
for(int i=0;i<ir1;i++)
  { if(Mother.b.get(i)) b.set(i);}
for(int i=ir1;i<N;i++)
  { if(Father.b.get(i)) b.set(i);}
}
public long getXP()
{
//returns intermediate integer of double number equivalent
long y=1;
double r=xN;
long z=0;
for(int i=0;i<N;i++)
  {
  if(b.get(i)) {z+=y;}
  y*=2;
  }
 return z;
}
public double getX()
{//return double number equivalent of bit set number
 return (xmin+(xmax-xmin)/xN*getXP());
}

public void clear()
{//fill the entire population with zero
b.clear(0,N);}
public void fill()
{//fill the entire population with ones
b.clear(0,N);b.flip(0,N);}
public String toBitSetString()
{//return bit set array values as defined in class BitSet
return b.toString();
}
public String toString()
{
// return bit set array values as a binary number definition
// plus double value of array
```

```
 String s="(";
 for(int i=N-1;i>=0;i--)
    if(b.get(i)) s+="1";
    else         s+="0";
    s+=") "+getX();
  return s;
}
public boolean boolean_random()
{
if(Math.random()<=0.5) return false;
else return true;
}
public int int_random()
{
  int x1=(int)(Math.random()*N);
  return x1;
}
public double double_random()
{
  return (xmin+Math.random()*(xmax-xmin));
}
public void setbits()
{
b.clear();
for(int i=0;i<N;i++)
   {
   if(boolean_random()) b.set(i);
   }
}
public void flip(int i)
{
b.flip(i);
}
public void mutate()
{
  //fip one bit of data
  int i1=int_random();
  if(i1<0) i1=0;
  b.flip(i1);
}
}
```

Few examples are given to explain gene processes. In the first example 5 63 bit gene is created and the results are listed

Program 5.18.4-2 GeneTest class creating random genes 63 bits between 0 to 10 limit values

```
public class geneTest
{
  public static void main(String arg[])
  {
  Gene x1=new Gene(0,10,63);
  for(int i=0;i<5;i++)
  { x1.setbits();System.out.println(x1.toString());}
  }
}
```

```
---------- Capture Output ----------
> "D:\java\bin\javaw.exe" geneTest
(10100000010111010101000111100111000000000001011011010101011010 1) 6.264239491429992
(11001001000001101110110001110001111100110100111111011101000011 0) 7.852618959616957
(00111000000110000101110101010110101000010010111101110100101001 1) 2.191217743394967
(10110100011101000110101001111010011111011011000010000101101001 0) 7.049013661432487
(10000101100110011011100110111111111001100100101000110100111000 0) 5.218769162659464
> Terminated with exit code 0.
```

Program 5.18.4-3 GeneTest1 class creating 63 bits between 0 to 10 limit values, value of 3.2

```
public class geneTest1
{ public static void main(String arg[])
  { Gene x1=new Gene(3.2,0,10,63);System.out.println(x1.toString());}
}
```

```
---------- Capture Output ----------
> "D:\java\bin\javaw.exe" geneTest1
(0101000111101011100001010001111010111000010100011111011000000000) 3.2
> Terminated with exit code 0.
```

Program 5.18.4-3 GeneTest1 class creating 63 bits between 0 to 10 limit values, value of 3.2

```
public class geneTest2
{ public static void main(String arg[])
  {Gene x1=new Gene("0101000111101011100001010001111010111000010100011111011000000000",0,10);
    System.out.println(x1.toString());}
}
```

```
---------- Capture Output ----------
> "D:\java\bin\javaw.exe" geneTest2
(0101000111101011100001010001111010111000010100011111011000000000) 3.2

> Terminated with exit code 0.
```

5.18.5 GENETIC ALGORITHMS, CLASSES GENOTYPE1 AND GENETIC1

Genotype1 can be considered as vector of Gene class variables. In means of Genetics, it represents a chromosome. In mathematically, it is a set of independent variables **(x0,x1,x2,..xi..xn)**
The main class of genetic algorithms is class Genetic1. Genetic optimization process with all the stages defined above calculated in this class. Some of the variables of Genetic1 should be defined by user. These variables are:

```
int    POPSIZE; //number of individuals in population
int    MAXGENS; //  maximum number of generations
int    NVARS;   // number of variables in chromosome (number of independent variables)
 int    N;       //bit size of the binary genes
double PXOVER; //probability of cross-over;
double PMUTATION; //probability of mutation
```

method evaluate is the main method to do genetic algorithm calculations. It should be emhised again that all the function values in Genetic algorithms should be positive, if it is not the case, please add up a constant to convert the values into positive values. This proses does not change the optimum.

Maximum of function $\mathbf{f(x_1,x_2,x_3,\ldots,x_n)}$ is same as the maximum of $\mathbf{g(x_1,x_2,x_3,\ldots,x_n) = f(x_1,x_2,x_3,\ldots,x_n) + C}$.
This process help to avoid errors during the calculations of cumulative percentage fitting function.

Program 5.18.5-1 Genotype1 and Genetic1 classes

```
 import java.io.*;
import java.text.*;
import java.util.Locale;
class genotype1i
{
// each genotype1 is member of the population
// Genotype is like a chromosone in real life genetic systems
// it can be thought as an evaluation point of a function with n variable
// f(xi)=f(x1,x2,x3,..,x_nvars)
public Gene G[]      ;  // string of variables makes a genotype1
public double Fitness;  // genotype1's fitness f(xi)
public double Upper[];  // gene's upper bound; xi_lower < xi < xi_upper
public double Lower[];  // Gene's lower bound; xi_lower < xi < xi_upper
public double RFitness; // the relative fitness
public double CFitness; // cumulative fitness
public int NVARS;       // Number of variables(Genes)
public int N;           // Number of bits defining each gene
public boolean Breeder;

 public genotype1i(int iNVARS,double low[],double high[],int iN,if_xj fi)
 {setgenotype1(iNVARS,low,high,iN,fi);}

 public void setgenotype1(int iNVARS,double low[],double high[],int iN,if_xj fi)
 {
 setN(iN);
 setNVARS(iNVARS);
 G=new Gene[NVARS];
 Upper=new double[NVARS];
 Lower=new double[NVARS];
 setLower(low);
 setUpper(high);
 setGene();
 setFitness(fi);
 }

 public genotype1i(genotype1i gi,if_xj fi)
 {setgenotype1(gi,fi);}

 public void setgenotype1(genotype1i gi,if_xj fi)
 {
 setN(gi.N);
 setNVARS(gi.NVARS);
 G=new Gene[NVARS];
 Upper=new double[NVARS];
 Lower=new double[NVARS];
 setLower(gi.Upper);
 setUpper(gi.Lower);
 for(int i=0;i<NVARS;i++)
   G[i]=gi.G[i].copyGene();
 setFitness(fi);
 }
```

```
  public genotype1i copygenotype(if_xj fi)
  {
  genotype1i g1=new genotype1i(this,fi);
  return g1;
  }

  public void setN(int iN)    {N=iN;}
  public void setNVARS(int iNVARS)    {NVARS=iNVARS;}
  public void setLower(double lbound[]) {for(int i=0;i<NVARS;i++) Lower[i]=lbound[i];}
  public void setUpper(double ubound[]) {for(int i=0;i<NVARS;i++) Upper[i]=ubound[i];}
  public void setFitness(if_xj fi)
  {            Fitness=fi.func(getGene());}
  public void setRFitness(double isum)  {RFitness=Fitness/isum;}
  public void setCFitness(double ix)  {CFitness=ix;}
  public double getLower(int j) {return Lower[j];}
  public double getUpper(int j) {return Upper[j];}
  public double getFitness() {return Fitness;}
  public double getRFitness(){return RFitness;}
  public double getCFitness(){return CFitness;}
  public void breed() {Breeder=true;}
  public void not_breed() {Breeder=false;}
  public boolean isBreed() {return Breeder;}


  public void setGene(int val)
  {  G[val]=new Gene(Lower[val],Upper[val],N);}

  public void setGene()
  {  for(int j=0;j<NVARS;j++)
                setGene(j);
  }

  public double getGene(int val)
  { return G[val].getX();}

  public double[] getGene()
  {   double x[]=new double[NVARS];
              for(int j=0;j<NVARS;j++)
                x[j]=getGene(j);
              return x;
  }

  public String toString()
  {
  //x values, Fitness,relative fitness and cumulative fitness
  String s="";
  for(int j=0;j<NVARS;j++)
              s+=G[j].toString()+" F:";
  s+=getFitness();
  s+="RF:"+getRFitness();
  s+="CF:"+getCFitness();
  return s;
  }
}  //end of class genotype1

public class Genetic1i
```

```
{
        // Maximizing a function
        int    POPSIZE; //population size
        int    MAXGENS; //maximum number of generations;
        int    NVARS;   //number of independent variables of function
  int   N;       //number of genes in each variable
        double PXOVER; //probability of crossover;
  double PMUTATION; //proportion of mutated variables
  int Generation; //Current generation number
  int NBreeder; //Number of survivors in a generation
  int Best; //The best genotype1 in the population
  genotype1i Population[];
  //if_xj f;  // function to be avaluated by genetic algorithm

  public Genetic1i(int iPOPSIZE,int iMAXGENS,int iNVARS,int iN,
  double iPXOVER,double iPMUTATION)
  {
  POPSIZE=iPOPSIZE;
  MAXGENS=iMAXGENS;
  NVARS=iNVARS;
  N=iN;
  PXOVER=iPXOVER;
  PMUTATION=iPMUTATION;
  NBreeder=0;
  Population=new genotype1i[POPSIZE+1];
        }
        public void setPopulation(double low[],double up[],if_xj fi)
        {
        for(int i=0;i<POPSIZE+1;i++)
          { Population[i]=new genotype1i(NVARS,low,up,N,fi);}
        }

        public genotype1i[] copyPopulation(if_xj fi)
        {
        genotype1i Po[]=new genotype1i[POPSIZE+1];
        for(int i=0;i<POPSIZE+1;i++)
          { Po[i]=new genotype1i(Population[i],fi);
            Po[i].Fitness=Population[i].Fitness;
                  Po[i].RFitness=Population[i].RFitness;
            Po[i].CFitness=Population[i].CFitness;
          }
        return Po;
        }

        public void evaluate(if_xj fi)
        {
          int mem;
          int j;
          double x[]=new double[NVARS];
          for(mem=0;mem<POPSIZE;mem++)
          {
          //System.out.println(Population[mem].toString());
          Population[mem].setFitness(fi);
          if(Population[mem].getFitness() > Population[POPSIZE].getFitness())
            {
                  Best=mem;
                  Population[POPSIZE]=Population[mem];
```

```
            for(j=0;j<NVARS;j++)
              {
              Population[POPSIZE].G[j].copyGene(Population[mem].G[j]);
                    }//end of for(j=0..
                  }//end of if
          }//end of for(mem=0;..
        }


        public void setRFCF(genotype1i Pi[])
        {
        //calculates relative and cumulative fitness functions
        int mem;
        int PN=Pi.length;
        double sum=0.0;
        //total fitness of population
        for(mem=0;mem<PN;mem++)
          {sum+=Pi[mem].Fitness;}
        //System.out.println("sum="+sum);
        //calculate relative fitness of each genotype1
        for(mem=0;mem<PN;mem++)
          {Pi[mem].setRFitness(sum);}
        //calculate cumulative fitness
        Pi[0].setCFitness(Pi[0].getRFitness());
for(mem=1;mem<PN;mem++)
          {Pi[mem].setCFitness(Pi[mem-1].getCFitness()+Pi[mem].getRFitness());
          }
        }

        public void toString(genotype1i Pi[])
        {
        int mem;
        int PN=Pi.length;
        for(mem=0;mem<PN;mem++)
        {
        //list them
        System.out.println("Population["+mem+"]="+Pi[mem].toString());
        }
        }

        public void select(if_xj fi)
        {
        //select the new generation members of population
        double r;
        int mem;
        setRFCF(Population);
        //create a new population;
        genotype1i Po[]=copyPopulation(fi);
setRFCF(Po);
for(int i=0;i<POPSIZE;i++)
{   mem=0;
   r=Math.random();
   double gf1=Population[i].getCFitness();
            double gf2;
   if (gf1 > r) Po[i]=Population[mem].copygenotype(fi);
            for(mem=1;mem<POPSIZE;mem++)
     {
```

```
                gf2=Population[mem].getCFitness();
                if( gf2>=r && gf1< r ) {Po[i]=Population[mem].copygenotype(fi);break;}
        }
 }
 setRFCF(Po);
          Population=Po;
          evaluate(fi);
 //toString(Population);
 }

public void crossover(if_xj fi)
{
int i,count;
int mem;
int POne,a;
int PTwo,b;
int point;
//select members for breeding
int iselect[]=new int[POPSIZE];
int counter=0;
double r;
 for(mem=0;mem<POPSIZE;mem++)
          {
           r=Math.random();
           if(r < PXOVER)
             {
                    iselect[counter++]=mem;
             Population[mem].breed();
                    NBreeder++;
             }
           else
             Population[mem].not_breed();
          }
          //System.out.println("iselect="+Matrix.toString(iselect));
          //let also best of the population to breed
          Population[Best].breed();

//loop through the population select breeding pairs
 for(mem=0;mem<POPSIZE;mem++)
    {
      //select two in popolation in random
      a=(int)(Math.random()*NBreeder)+1;
      b=(int)(Math.random()*NBreeder)+1;
      count=0;
      POne=0;
      PTwo=0;
      //select two individuals for breeding
      for(i=0;i<POPSIZE;i++)
        {
        if(Population[i].isBreed())
          {
          count++;
          if(count==a) POne=count;
          if(count==b) PTwo=count;
                  }
                }
              //perform a crossover;
```

```
                genotype1i Po[]=new genotype1i[2];
                        Po[0]=Population[POne].copygenotype(fi);
                        Po[1]=Population[PTwo].copygenotype(fi);
                for(i=0;i<NVARS;i++)
                {
                        Population[POne].G[i].cross1(Math.random(),Po[0].G[i],Po[1].G[i]);
                        Population[PTwo].G[i].cross1(Math.random(),Po[0].G[i],Po[1].G[i]);
                        }
                        Population[POne].setFitness(fi);
                        Population[PTwo].setFitness(fi);
              }
            }

public void mutate(if_xj fi)
{
int i;
double lbound,hbound;
int nmutations;
int member;
int var;
nmutations=(int)((double)(POPSIZE*NVARS*PMUTATION*N));
for(i=0;i<nmutations;i++)
  {
  member=(int)(Math.random()*POPSIZE);
  var=(int)(Math.random()*NVARS);
  //replace the old value with a new mutated one
  Population[member].G[var].mutate();
        //after mutation recalculate the function value
  Population[member].setFitness(fi);
  }
}

public String report()
{
String s;
int i;
double best_value;
double avg;
double stdev;
double sum_square;
double square_sum;
double sum;
double xx;
sum=0;
sum_square=0.0;
for(i=0;i<POPSIZE;i++)
  {
        xx= Population[i].getFitness();
  sum+=xx;
  sum_square=xx*xx;
        }
avg=sum/(double)POPSIZE;
square_sum=sum*sum/(double)POPSIZE;
stdev=Math.sqrt(square_sum);
best_value=Population[POPSIZE].getFitness();
double aa[]=new double[NVARS+1];
for(i=0;i < NVARS;i++)
```

```
    {
     aa[i]=Population[POPSIZE].getGene(i);
               }
               aa[NVARS]=Population[POPSIZE].getFitness();

  //s="Generation = "+Generation+"best value = "+Matrix.toString(aa)+"Average = "+avg+"Standart Deviation = "+stdev;
  s="Generation = "+Generation+"best value = "+Matrix.toString(aa);
  return s;
  }

  public double[] getBest()
  {
  double aa[]=new double[NVARS+1];
  for(int i=0;i < NVARS;i++)
    {
     aa[i]=Population[POPSIZE].getGene(i);
               }
               aa[NVARS]=Population[POPSIZE].getFitness();
  return aa;
  }

  public double[] calculate(if_xj fi,boolean report)
  {
  evaluate(fi);
  Generation=0;
  while(Generation<MAXGENS)
    {
     Generation++;
     NBreeder=(int)(0.8*POPSIZE);
     select(fi);
     crossover(fi);
     mutate(fi);
     evaluate(fi);
     //if(report)
        //System.out.println(report());
    }
  return getBest();
  }
}
```

Now that a Genetic algorithm is available to us, optimization of actual functions can be carry out. The first example problem will be $f(x_0,x_1)=2x_0\ x_1+2x_0-x_0^2-2x_1^2+10$. This function was used in previous sub-sections of optimization, but an additional constant value of 10 is added to ensure positive cumulative numbers. Please remember that we do not have any dificulty solving this problem with previously given geometrical based optimization algorithms. Note that the result is probabilistic so that it will never be exactly the same like geometric methods.

Program 5.18.5-2 Genetic1 class test program NA104i

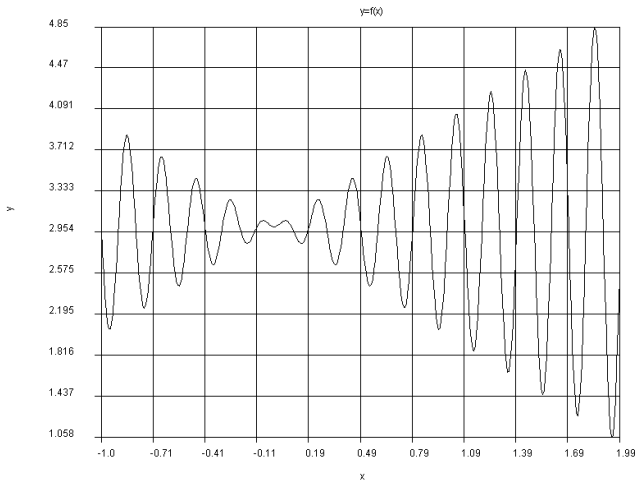
```
 import java.io.*;

class NA104i
{
   public static void main(String args[]) throws IOException
   {
```

```
            if_xj fx=(double x[])->2.0*x[0]*x[1]+2.0*x[0]-x[0]*x[0]-2.0*x[1]*x[1]+10.0;
   int N=30; //number of digits in binary genes
            int POPSIZE=200; //population size
   int MAXGENS=200; //maximum number of generations
   int NVARS=2; //number of variables (member of genes in genome
   double PXOVER=0.3; //probability of crossover
   double PMUTATION=0.02; //probability of mutation
   Genetic1i xx=new Genetic1i(POPSIZE,MAXGENS,NVARS,N,PXOVER,PMUTATION);
   double low[]=new double[NVARS];
   double high[]=new double[NVARS];
   low[0]=-1.0;high[0]=5.0;
   low[1]=-1.0;high[1]=5.0;
   xx.setPopulation(low,high,fx);
   System.out.println("best=\n"+Matrix.toStringT(xx.calculate(fx,false)));
   }
}
```

```
---------- Capture Output ----------
> "E:\co\java\bin\javaw.exe" NA104i
best=
          1.998048807492525
          0.998046871272891
         11.999996185284427
> Terminated with exit code 0.
```

As a second example a one dimensional function $f(x_0)=x_0 \sin(10\pi x_0)+1$ function will be investigated . The plot of the function shows the difficulty of this function: it has many local maximum. It is very hard for geometric methods to get a solution of such function unless some jump mechanisms built into the solution.



Program 5.18.5-3 Genetic1 class test program NA105

```
 mport java.io.*;

class NA104i
{
   public static void main(String args[]) throws IOException
   {
            if_xj fx=(double x[])->(x[0]*Math.sin(10.0*Math.PI*x[0])+3.0);
   int N=30; //number of digits in binary genes
```

```
            int POPSIZE=200; //population size
   int MAXGENS=200; //maximum number of generations
   double low[]={-1};
   double high[]={1};
   int NVARS=low.length; //number of variables (member of genes in genome
   double PXOVER=0.3; //probability of crossover
   double PMUTATION=0.02; //probability of mutation
   Genetic1i xx=new Genetic1i(POPSIZE,MAXGENS,NVARS,N,PXOVER,PMUTATION);

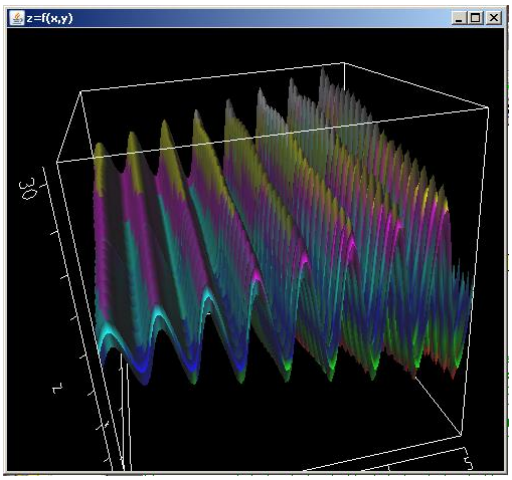
   xx.setPopulation(low,high,fx);
   System.out.println("best=\n"+Matrix.toStringT(xx.calculate(fx,false)));
   }
}
```

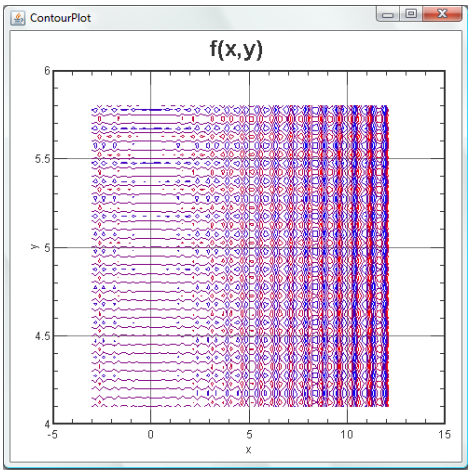
```
 ---------- Capture Output ----------
> "E:\co\java\bin\javaw.exe" NA104i
best=
        0.851189793880274
        3.850595242962603
> Terminated with exit code 0.
```

As a last example of class Genetic1 a two variable nonlimear function is defined as

f(x,y)=21.5+x*sin(4.0*π*y)+y*Math.sin(20.0* π *y)

The 3D plot of the function





It is clear from the plot that this problem also have the same kind of difficulty as the previous problem.

Program 5.18.5-4 Genetic1 class test program NA106

```
 import java.io.*;
class NA104i
{
   public static void main(String args[]) throws IOException
   {
   if_xj fx=(double x[])->21.5+x[0]*Math.sin(4.0*Math.PI*x[0])+x[1]*Math.sin(20.0*Math.PI*x[1]);
   int N=50; //number of digits in binary genes
   int POPSIZE=300; //population size
   int MAXGENS=300; //maximum number of generations
   double low[]={-3,4.1};
   double high[]={12.1,5.8};
   int NVARS=low.length; //number of variables (member of genes in genome
   double PXOVER=0.2; //probability of crossover
```

```
    double PMUTATION=0.01; //probability of mutation
    Genetic1i xx=new Genetic1i(POPSIZE,MAXGENS,NVARS,N,PXOVER,PMUTATION);
    xx.setPopulation(low,high,fx);
    System.out.println("best=\n"+Matrix.toStringT(xx.calculate(fx,false)));
    }
}
```

```
---------- Capture Output ----------
> "E:\co\java\bin\javaw.exe" NA104i
best=
        11.126757812500871
         5.720312499999952
        38.098041101991460
> Terminated with exit code 0.
```

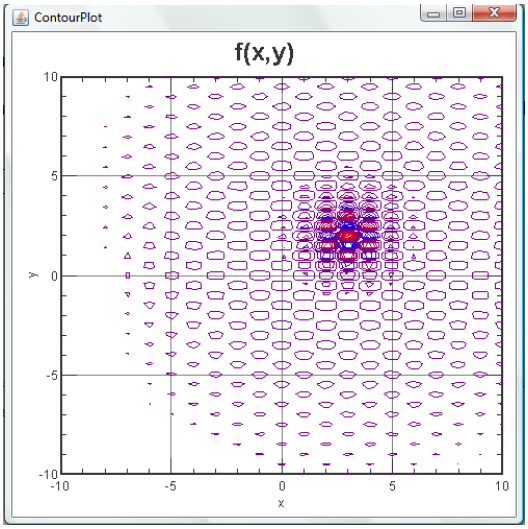
To see the statistical effect on the result if the same problem re-run with exactly the same parameters:

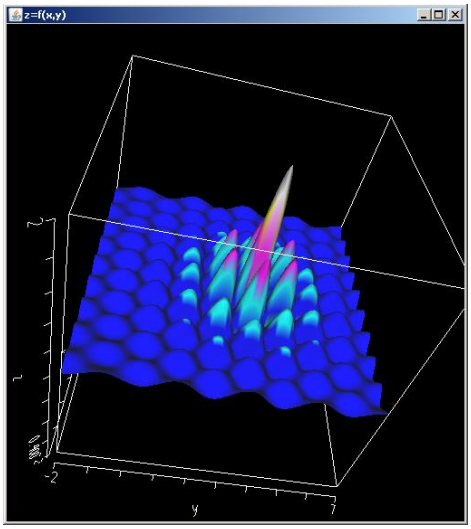
```
 ---------- Capture Output ----------
> "E:\co\java\bin\javaw.exe" NA106i
best=
        11.625358296807466
         4.923434156166672
        38.024865436893660


> Terminated with exit code 0.
```

Another interesting equation to optimise

z=f(x,y)= 1.0+cos(π*(x-3.0))*cos(2.0* π *(y-2.0))/(1+(x-3.0)*(x-3.0)+(y-2.0)*(y-2.0));





Program 5.18.5-4 Genetic1 class test program NA107i

```
import java.io.*;
class NA104i
{
   public static void main(String args[]) throws IOException
   {
            if_xj fx=(double x[])->1.0+Math.cos(Math.PI*(x[0]-3.0))*Math.cos(2.0*Math.PI*(x[1]-2.0))/(1+(x[0]-3.0)*(x[0]-3.0)+(x[1]-
```

```
2.0)*(x[1]-2.0));

   int N=63; //number of digits in binary genes
            int POPSIZE=500; //population size
   int MAXGENS=500; //maximum number of generations
   double low[]={-10,-10};
   double high[]={10,10};

   int NVARS=low.length; //number of variables (member of genes in genome
   double PXOVER=0.2; //probability of crossover
   double PMUTATION=0.01; //probability of mutation
   Genetic1i xx=new Genetic1i(POPSIZE,MAXGENS,NVARS,N,PXOVER,PMUTATION);

   xx.setPopulation(low,high,fx);
   System.out.println("best=\n"+Matrix.toStringT(xx.calculate(fx,false)));
   }
}
```

```
---------- Capture Output ----------
> "E:\co\java\bin\javaw.exe" NA104i
best=
        2.999992370604135
        1.999998092650443
        1.999999999579100


> Terminated with exit code 0.
```

5.18.6 GENETİC ALGORITHMS BY USING GAUSS RANDOM NUMBERS

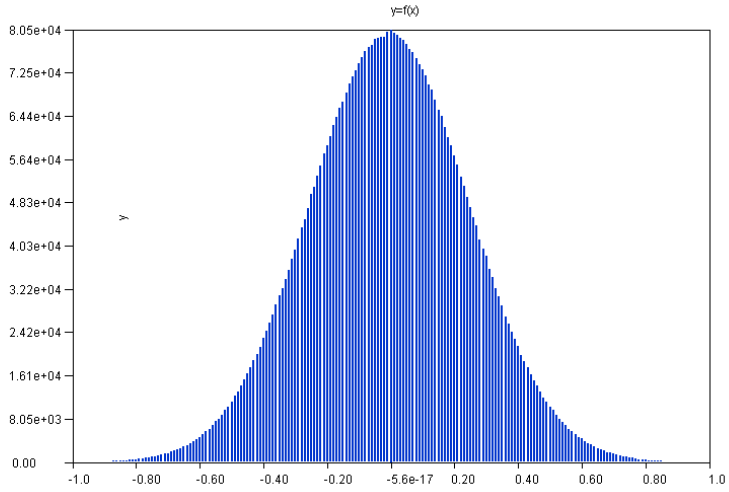
In the previous model, binary numbers are used as the base of constructing the genetic structure. In this second model Gauss random numbers will be used as the basis of the model. Gauss random numbers construct a bell shape distribution around the given point. We will start to create a Gauss random number algorithm.

Program 5.18.6-1Gausrandom method example

```
import java.io.*; //java girdi cikti sinifini cagir
class randeneme
{
public static double  gaussrandom(double xort,double r)
{ int iset=0;
 double gset=0;
 double fac,rsq,v1,v2;
 double aret=0;
 if(iset==0)
 {
   do{
            v1=2.0*Math.random()-1.0;
            v2=2.0*Math.random()-1.0;
            rsq=v1*v1+v2*v2;
    }while(rsq>=1.0 || rsq==0.0);
  fac=Math.sqrt(-2.0*Math.log(rsq)/rsq);
  gset=v1*fac;
  iset=1;
  aret=v2*fac;
 }
 else
 {
 iset=0;
```

```
 aret=gset;
 }
 return aret/8.0*r+xort;
 }

public static void main(String args[])
 {
  double xort,r;
  double gr=0;
  double x[]=new double[201];
  double y[]=new double[201];
  xort=0.0;
  r=2.0;
  for(int i=0;i<=200;i++) {x[i]=xort-r/2+r*(double)i/200.0;}
  for(int i=0;i<=5000000;i++)
    {gr=gaussrandom(xort,r);for(int j=0;j<200;j++){if(gr>=x[j] && gr< x[j+1]){y[j]+=1.0;}}}
  Plot pp=new Plot(x,y);
  pp.setColor(0,0,51,204);
  pp.setPlotType(0,41);
  pp.plot();
}
 }
```



In the example program 5000000 Gauss random points are generated and plotted. Now genetic classes can be generated by using this Gauss Random numbers concept. This set of program is very similar except definition of the genes. With this type of gene definitions when cross-over of the genes are calculated border values should be considered carefully and not to exceed.

Program 5.18.6-2Gausrandom Gene class Gen.java

```
 // Dr. M. Turhan ÇOBAN
// EGE University School of Engineering, Mechanical Engineering Department
// package : Numerical analysis package
// subject : optimization
// group   : Genetic Algorithms gauss random method
// Gen Class
// Gen class generates a double number represent a gene

import java.util.*;
import java.io.*;
```

```
public class Gen extends Object
{
public double xmin,xmax;//maximum and minimum limits of double number
public double x;

public  double  gaussrandom(double ixmin,double ixmax,double ix)
{
  double d1=ixmax-ix;
  double d2=ix-ixmin;
  double d=Math.abs(Math.min(d1,d2));
  ix=gaussrandom(ix,d);
  return ix;
}

public  void  gaussrandom()
{
  double d1=xmax-x;
  double d2=x-xmin;
  double d=Math.abs(Math.min(d1,d2));
  x=gaussrandom(x,d);
}

public  double  gaussrandom(double xort,double r)
{ //r yayılma bölgesi
   //xort ortalama değer
 int iset=0;
 double gset=0;
 double fac,rsq,v1,v2;
 double aret=0;
 if(iset==0)
 {
   do{
                v1=2.0*Math.random()-1.0;
                v2=2.0*Math.random()-1.0;
                rsq=v1*v1+v2*v2;
     }while(rsq>=1.0 || rsq==0.0);
   fac=Math.sqrt(-2.0*Math.log(rsq)/rsq);
   gset=v1*fac;
   iset=1;
   aret=v2*fac;
 }
 else
 {
 iset=0;
 aret=gset;
 }
 return aret/4.0*r+xort;
}

public Gen(double ixmin,double ixmax)
{//randomly generate the gene
 xmin=ixmin;
 xmax=ixmax;
 x=double_random();
}

public Gen(double s,double ixmin,double ixmax)
{
xmin=ixmin;
xmax=ixmax;
//generate gene equal to a given string
if((s>=xmin)&& (s<=xmax)) x=s;
else if(s<xmin) x=xmin;
```

```
else if(s>xmax) x=xmax;
}


public Gen()
{
xmin=0;
xmax=1.0;
x=xmin+Math.random();
}

public Gen(Gen g)
{
xmin=g.xmin;
xmax=g.xmax;
x=g.x;
}

public void copyGen(Gen g)
{
xmin=g.xmin;
xmax=g.xmax;
x=g.x;
}

public Gen copyGen()
{
 Gen g=new Gen(this);
 return g;
}

public Gen(Gen Father,Gen Mother)
{setGen(Father,Mother);}

public void  setGen(Gen Father,Gen Mother)
{
xmin=Father.xmin;
xmax=Father.xmax;
double r1=xmax-xmin;
double r=Math.random();

double d1=xmax-Father.x;
double d2=Father.x-xmin;
double d=Math.abs(Math.min(d1,d2));

double r2=gaussrandom(Father.x,d);
d1=xmax-Mother.x;
d2=Mother.x-xmin;
d=Math.abs(Math.min(d1,d2));
double r3=gaussrandom(Mother.x,d);
x=r2*r+r3*(1-r);
System.out.println("r2="+r2+"r3="+r3+"r="+r);
if(x<xmin) x=xmin;
else if(x>xmax) x=xmax;

}

public void cross1(double crossratio,Gen Father,Gen Mother)
{xmin=Father.xmin;
xmax=Father.xmax;
double r1=xmax-xmin;
double d1=xmax-Father.x;
double d2=Father.x-xmin;
double d=Math.abs(Math.min(d1,d2));
```

```
double gr1=gaussrandom(Father.x,d);
d1=xmax-Mother.x;
d2=Mother.x-xmin;
d=Math.abs(Math.min(d1,d2));
double gr2=gaussrandom(Mother.x,d);
x=gr1*crossratio+gr2*(1-crossratio);
if(x<xmin) x=xmin;
else if(x>xmax) x=xmax;
}

public void cross2(double crossratio,Gen Father,Gen Mother)
{xmin=Father.xmin;
xmax=Father.xmax;
double r1=xmax-xmin;
double d1=xmax-Father.x;
double d2=Father.x-xmin;
double d=Math.abs(Math.min(d1,d2));
double gr1=gaussrandom(Father.x,d);
d1=xmax-Mother.x;
d2=Mother.x-xmin;
d=Math.abs(Math.min(d1,d2));
double gr2=gaussrandom(Mother.x,d);
x=gr1*(1.0-crossratio)+gr2*crossratio;
if(x<xmin) x=xmin;
else if(x>xmax) x=xmax;
}
public double getX()
{//return double number equivalent of bit set number
 return x;
}
public boolean boolean_random()
{
if(Math.random()<=0.5) return false;
else return true;
}

public double double_random()
{ return (xmin+Math.random()*(xmax-xmin));}

public void mutate()
{
  double r=(xmax-xmin);
  double d1=xmax-x;
  double d2=x-xmin;
  double d=Math.abs(Math.min(d1,d2));
  x=gaussrandom(x,d);
}}
```

Program 5.18.6-3 Gausrandom Genome (vector of gene) class class gentype.java

```
 import java.io.*;
import java.text.*;
import java.util.Locale;


class gentypei
{
// each genotype1 is member of the population
// Genotype is like a chromosone in real life genetic systems
// it can be thought as an evaluation point of a function with n variable
// f(xi)=f(x1,x2,x3,..,x_nvars)
public Gen G[]     ;  // string of variables makes a genotype1
public double Fitness;  // genotype1's fitness f(xi)
```

```
public double Upper[];  // gene's upper bound; xi_lower < xi < xi_upper
public double Lower[];  // Gene's lower bound; xi_lower < xi < xi_upper
public double RFitness; // the relative fitness
public double CFitness; // cumulative fitness
public int NVARS;       // Number of variables(Genes)
public int N;           // Number of bits defining each gene
public boolean Breeder;

 public gentypei(int iNVARS,double low[],double high[],int iN,if_xj fi)
 {setgentypei(iNVARS,low,high,iN,fi);}

 public void setgentypei(int iNVARS,double low[],double high[],int iN,if_xj fi)
 {
 setN(iN);
 setNVARS(iNVARS);
 G=new Gen[NVARS];
 Upper=new double[NVARS];
 Lower=new double[NVARS];
 setLower(low);
 setUpper(high);
 setGen();
 setFitness(fi);
 }

 public gentypei(gentypei gi,if_xj fi)
 {setgentypei(gi,fi);}

 public void setgentypei(gentypei gi,if_xj fi)
 {
 setN(gi.N);
 setNVARS(gi.NVARS);
 G=new Gen[NVARS];
 Upper=new double[NVARS];
 Lower=new double[NVARS];
 setLower(gi.Upper);
 setUpper(gi.Lower);
 for(int i=0;i<NVARS;i++)
   G[i]=gi.G[i].copyGen();
 setFitness(fi);
 }

 public gentypei copygentypei(if_xj fi)
 {
 gentypei g1=new gentypei(this,fi);
 return g1;
 }

 public void setN(int iN)    {N=iN;}
 public void setNVARS(int iNVARS)    {NVARS=iNVARS;}
 public void setLower(double lbound[]) {for(int i=0;i<NVARS;i++) Lower[i]=lbound[i];}
 public void setUpper(double ubound[]) {for(int i=0;i<NVARS;i++) Upper[i]=ubound[i];}
 public void setFitness(if_xj fi)
 {              Fitness=fi.func(getGen());}
 public void setRFitness(double isum)  {RFitness=Fitness/isum;}
 public void setCFitness(double ix)  {CFitness=ix;}
 public double getLower(int j) {return Lower[j];}
 public double getUpper(int j) {return Upper[j];}
 public double getFitness() {return Fitness;}
 public double getRFitness(){return RFitness;}
 public double getCFitness(){return CFitness;}
 public void breed() {Breeder=true;}
 public void not_breed() {Breeder=false;}
 public boolean isBreed() {return Breeder;}
```

```
  public void setGen(int val)
  {  G[val]=new Gen(Lower[val],Upper[val]);}

  public void setGen()
  {  for(int j=0;j<NVARS;j++)
                  setGen(j);
  }

  public double getGen(int val)
  { return G[val].getX();}

  public double[] getGen()
  {   double x[]=new double[NVARS];
               for(int j=0;j<NVARS;j++)
                  x[j]=getGen(j);
               return x;
  }

  public String toString()
  {
  //x values, Fitness,relative fitness and cumulative fitness
  String s="";
  for(int j=0;j<NVARS;j++)
               s+=G[j].toString()+" F:";
  s+=getFitness();
  s+="RF:"+getRFitness();
  s+="CF:"+getCFitness();
  return s;
  }
}  //end of class gentypei
```

Program 5.18.6-4 Gausrandom Genetic algorithm Genetic2i.java

```
public class Genetic2
{
               // Maximizing a function
               int    POPSIZE; //population size
               int    MAXGENS; //maximum number of generations;
               int    NVARS;   //number of independent variables of function
   int    N;       //number of genes in each variable
               double PXOVER; //probability of crossover;
   double PMUTATION; //proportion of mutated variables
   int Generation; //Current generation number
   int NBreeder; //Number of survivors in a generation
   int Best; //The best gentype in the population
   gentype Population[];
   //f_xj f;  // function to be avaluated by genetic algorithm

   public Genetic2(int iPOPSIZE,int iMAXGENS,int iNVARS,int iN,
   double iPXOVER,double iPMUTATION)
   {
   POPSIZE=iPOPSIZE;
   MAXGENS=iMAXGENS;
   NVARS=iNVARS;
   N=iN;
   PXOVER=iPXOVER;
   PMUTATION=iPMUTATION;
   NBreeder=0;
   Population=new gentype[POPSIZE+1];
               }
               public void setPopulation(double low[],double up[],f_xj fi)
               {
```

```
        for(int i=0;i<POPSIZE+1;i++)
          { Population[i]=new gentype(NVARS,low,up,N,fi);}
        }

        public gentype[] copyPopulation(f_xj fi)
        {
        gentype Po[]=new gentype[POPSIZE+1];
        for(int i=0;i<POPSIZE+1;i++)
          { Po[i]=new gentype(Population[i],fi);
            Po[i].Fitness=Population[i].Fitness;
                    Po[i].RFitness=Population[i].RFitness;
            Po[i].CFitness=Population[i].CFitness;
          }
        return Po;
        }

        public void evaluate(f_xj fi)
        {
          int mem;
          int j;
          double x[]=new double[NVARS];
          for(mem=0;mem<POPSIZE;mem++)
          {
          //System.out.println(Population[mem].toString());
          Population[mem].setFitness(fi);
          if(Population[mem].getFitness() > Population[POPSIZE].getFitness())
            {
                    Best=mem;
                    Population[POPSIZE]=Population[mem];
            for(j=0;j<NVARS;j++)
              {
              Population[POPSIZE].G[j].copyGen(Population[mem].G[j]);
                      }//end of for(j=0..
                    }//end of if
          }//end of for(mem=0;..
        }


        public void setRFCF(gentype Pi[])
        {
        //calculates relative and cumulative fitness functions
        int mem;
        int PN=Pi.length;
        double sum=0.0;
        //total fitness of population
        for(mem=0;mem<PN;mem++)
          {sum+=Pi[mem].Fitness;}
        //System.out.println("sum="+sum);
        //calculate relative fitness of each gentype
        for(mem=0;mem<PN;mem++)
          {Pi[mem].setRFitness(sum);}
        //calculate cumulative fitness
        Pi[0].setCFitness(Pi[0].getRFitness());
for(mem=1;mem<PN;mem++)
          {Pi[mem].setCFitness(Pi[mem-1].getCFitness()+Pi[mem].getRFitness());
          }
        }

        public void toString(gentype Pi[])
        {
        int mem;
        int PN=Pi.length;
        for(mem=0;mem<PN;mem++)
        {
```

```
            //list them
            System.out.println("Population["+mem+"]="+Pi[mem].toString());
            }
            }

            public void select(f_xj fi)
            {
            //select the new generation members of population
            double r;
            int mem;
            setRFCF(Population);
            //create a new population;
            gentype Po[]=copyPopulation(fi);
 setRFCF(Po);
 for(int i=0;i<POPSIZE;i++)
 {   mem=0;
    r=Math.random();
    double gf1=Population[i].getCFitness();
                double gf2;
    if (gf1 > r) Po[i]=Population[mem].copygentype(fi);
                for(mem=1;mem<POPSIZE;mem++)
      {
                  gf2=Population[mem].getCFitness();
                  if( gf2>=r && gf1< r ) {Po[i]=Population[mem].copygentype(fi);break;}
      }
 }
 setRFCF(Po);
            Population=Po;
            evaluate(fi);
 //toString(Population);
 }

public void crossover(f_xj fi)
{
int i,count;
int mem;
int POne,a;
int PTwo,b;
int point;
//select members for breeding
int iselect[]=new int[POPSIZE];
int counter=0;
double r;
 for(mem=0;mem<POPSIZE;mem++)
            {
              r=Math.random();
              if(r < PXOVER)
                {
                        iselect[counter++]=mem;
                Population[mem].breed();
                        NBreeder++;
                }
              else
                Population[mem].not_breed();
            }
            //System.out.println("iselect="+Matrix.toString(iselect));
            //let also best of the population to breed
            Population[Best].breed();

//loop through the population select breeding pairs
 for(mem=0;mem<POPSIZE;mem++)
    {
      //select two in popolation in random
      a=(int)(Math.random()*NBreeder)+1;
```

```
        b=(int)(Math.random()*NBreeder)+1;
        count=0;
        POne=0;
        PTwo=0;
        //select two individuals for breeding
        for(i=0;i<POPSIZE;i++)
          {
          if(Population[i].isBreed())
            {
            count++;
            if(count==a) POne=count;
            if(count==b) PTwo=count;
                        }
                    }
                  //perform a crossover;
                    gentype Po[]=new gentype[2];
                              Po[0]=Population[POne].copygentype(fi);
                              Po[1]=Population[PTwo].copygentype(fi);
                    for(i=0;i<NVARS;i++)
                    {
                              Population[POne].G[i].cross1(Math.random(),Po[0].G[i],Po[1].G[i]);
                              Population[PTwo].G[i].cross1(Math.random(),Po[0].G[i],Po[1].G[i]);
                              }
                              Population[POne].setFitness(fi);
                              Population[PTwo].setFitness(fi);
                  }
                }

public void mutate(f_xj fi)
{
int i;
double lbound,hbound;
int nmutations;
int member;
int var;
nmutations=(int)((double)(POPSIZE*NVARS*PMUTATION*N));
for(i=0;i<nmutations;i++)
  {
  member=(int)(Math.random()*POPSIZE);
  var=(int)(Math.random()*NVARS);
  //replace the old value with a new mutated one
  Population[member].G[var].mutate();
            //after mutation recalculate the function value
  Population[member].setFitness(fi);
  }
}

public String report()
{
String s;
int i;
double best_value;
double avg;
double stdev;
double sum_square;
double square_sum;
double sum;
double xx;
sum=0;
sum_square=0.0;
for(i=0;i<POPSIZE;i++)
  {
            xx= Population[i].getFitness();
  sum+=xx;
```

```
    sum_square=xx*xx;
                }
  avg=sum/(double)POPSIZE;
  square_sum=sum*sum/(double)POPSIZE;
  stdev=Math.sqrt(square_sum);
  best_value=Population[POPSIZE].getFitness();
  double aa[]=new double[NVARS+1];
  for(i=0;i < NVARS;i++)
   {
   aa[i]=Population[POPSIZE].getGen(i);
               }
               aa[NVARS]=Population[POPSIZE].getFitness();

  //s="Generation = "+Generation+"best value = "+Matrix.toString(aa)+"Average = "+avg+"Standart Deviation = "+stdev;
  s="Generation = "+Generation+"best value = "+Matrix.toString(aa);
  return s;
  }

  public double[] getBest()
  {
  double aa[]=new double[NVARS+1];
  for(int i=0;i < NVARS;i++)
   {
   aa[i]=Population[POPSIZE].getGen(i);
               }
               aa[NVARS]=Population[POPSIZE].getFitness();
  return aa;
  }

  public double[] calculate(f_xj fi,boolean report)
  {
  evaluate(fi);
  Generation=0;
  while(Generation<MAXGENS)
   {
   Generation++;
   NBreeder=(int)(0.8*POPSIZE);
   select(fi);
   crossover(fi);
   mutate(fi);
   evaluate(fi);
   //if(report)
     //System.out.println(report());
   }
  return getBest();
  }
}
```

Program 5.18.5-5 Gausrandom Genetic algorithm test program class NA108.java

```
 import java.io.*;

class NA108

{
   public static void main(String args[]) throws IOException
   {
   int N=50;
            int POPSIZE=600;
   int MAXGENS=600;
   int NVARS=2;
   double PXOVER=0.3;
```

```
    double PMUTATION=0.02;
    if_xj fx=(double x[])->2.0*x[0]*x[1]+2.0*x[0]-x[0]*x[0]-2.0*x[1]*x[1]+10.0;
    Genetic2i xx=new Genetic2i(POPSIZE,MAXGENS,NVARS,N,PXOVER,PMUTATION);
    double low[]={-1,-1};
    double high[]={5,5};
    xx.setPopulation(low,high,fx);
    System.out.println("best=\n"+Matrix.toStringT(xx.calculate(fx,false)));
    }
}
```

The output of the test program:

```
---------- Capture Output ----------
> "E:\co\java\bin\javaw.exe" NA108
best=
          2.009221343823353
          1.001161197373327
         11.999933685659865


> Terminated with exit code 0.
```

5.18.7 GENETİC ALGORITHMS BY USING A LONG TYPE VARIABLE AS A GENE

In a gene structure number of digits in the gene should be constant. It is similar in the natural system as well. In order to produce a child gene structure of both parents should be similar in structure . **A long integer with a setted number of digits** will be used as a gene structure in the next genetic algorithm code. In this two variable systems long type integer value bits can take all the possible values 0 to 9 in each digit, while double value can only corresponds the working space. As a maximum long number 18 digit x'_max=9999999999999999L is taken. In this case the relation between long number x' and double number x can be given as:

$$x = x_{min} + x'\left(\frac{x_{max} - x_{min}}{999999999999999999L}\right)$$

The double number can be converted back to the long integer gene number the same way.

$$x' = 999999999999999999L\left(\frac{x - x_{min}}{x_{max} - x_{min}}\right)$$

The remaining of genetic algorithm is the same as the previous algorithms. It should be note that even though the number of the gens are constant (18) for this algorithm, gen number input variable is kept in the program constructor, in order to make it same as the previous versions with the variable gene numbers.

Program 5.18.7-1 Gene constructed from long type integer numbers of 18 digits.

```
// Dr. M. Turhan ÇOBAN
// EGE University School of Engineering, Mechanical Engineering Department
// package : Numerical analysis package
// subject : optimization
// group   : Genetic Algorithms
// Gene13 Class
// Gene3 class generates integer array to represent a double number
// actual binary defination changes depends on number of bits, xmin and xmax
// binary data stored in b BitSet array

import java.util.*;
import java.io.*;

public class Gene3 extends Object
{
```

```
//long imin=0;
int N=18;
public long imax=999999999999999999L;
public long ix;
public double x;//Limit value for integer(2^N-1)
public double xmin,xmax;//maximum and minimum limits of double number

public Gene3(double ixmin,double ixmax)
{//randomly generate the gene
 xmin=ixmin;
 xmax=ixmax;
 ix=(long)(imax*Math.random());
 x=xmin+(xmax-xmin)*(double)ix/(double)imax;
}


public Gene3(long ixi,double ixmin,double ixmax)
{//randomly generate the gene
 xmin=ixmin;
 xmax=ixmax;
 ix=ixi;
 x=xmin+(double)ix*(xmax-xmin)/(double)imax;
}

public Gene3(Gene3 g)
{
// copy a Gene from another one
copyGene(g);
}

public void copyGene(Gene3 g)
{
 ix=g.ix;
 x=g.x;
 xmin=g.xmin;
 xmax=g.xmax;
}

public Gene3 copyGene()
{Gene3 g=new Gene3(x,xmin,xmax);
 return g;
}

public Gene3(Gene3 Father,Gene3 Mother)
{cross(Math.random(),Father,Mother);}

public Gene3(double xi,double maxi,double mini)
{setGene(xi,maxi,mini);}


public Gene3(double crossratio,Gene3 Father,Gene3 Mother)
{   if(boolean_random()) cross(crossratio,Father,Mother);
   else                  cross(crossratio,Mother,Father);
}

public void  setGene(long iix,double ixmin,double ixmax)
{
 xmin=ixmin;
 xmax=ixmax;
 ix=iix;
 x=xmin+(xmax-xmin)*(double)ix/(double)imax;
}

public void  setGene(double idx,double ixmin,double ixmax)
```

```
{
 xmin=ixmin;
 xmax=ixmax;
 if(idx>=xmin && idx<=xmax)  x=idx;
 else if(idx<xmin) x=xmin;
 else x=xmax;
 ix=(long)((x-xmin)/(xmax-xmin)*imax+0.99);
}

public void  setGene(double idx)
{

 if(idx>=xmin && idx<=xmax)  x=idx;
 else if(idx<xmin) x=xmin;
 else x=xmax;
 ix=(long)((x-xmin)/(xmax-xmin)*imax);
}

public void cross(Gene3 Father,Gene3 Mother)
{double crossratio=Math.random();
cross(crossratio,Father,Mother);
}

public double getX()
{return x;}

public void cross(double crossratio,Gene3 Father,Gene3 Mother)
{
Long IntFx=new Long(Father.ix);
String SFx=IntFx.toString();
int nSFx=SFx.length();
Long IntMx=new Long(Mother.ix);
String SMx=IntMx.toString();
int nSMx=SMx.length();
if(nSFx<N)
{String ss="";for(int i=0;i<(N-nSFx);i++){ss+="0";};SFx=ss+SFx;}
if(nSMx<N)
{String ss="";for(int i=0;i<(N-nSMx);i++){ss+="0";};SMx=ss+SMx;}
int N2 = (int)(crossratio*N+0.02)/2;
int N1=2*N2;
int N3=N1+(N-N1)/2;
StringBuffer dest = new StringBuffer(N);
int i;
for (i = 0; i <N2; i++)
{ dest.append(SFx.charAt(i));
  dest.append(SMx.charAt(i));
}
for (i = N1; i <N3; i++)
{ dest.append(SFx.charAt(i));}
for (i = N3; i <N; i++)
{ dest.append(SMx.charAt(i));}
String s=dest.toString();
long iix=Long.parseLong(s);
ix=iix;
xmin=Father.xmin;
xmax=Father.xmax;
x=xmin+(xmax-xmin)*(double)ix/(double)imax;
}


public boolean boolean_random()
{
if(Math.random()<=0.5) return false;
else return true;
```

```
}

public void mutate()
{
int N1 = (int)(N*Math.random());
Long IntFx=new Long(ix);
String SFx=IntFx.toString();
int nSFx=SFx.length();
if(nSFx<N)
{String ss="";for(int i=0;i<(N-nSFx);i++){ss+="0";};SFx=ss+SFx;}
StringBuffer dest = new StringBuffer(N);
int i;
for (i = 0; i <N; i++)
{  if(i==N1)
   {  int i1=(int)(9.0*Math.random());
      String s2=Integer.toString(i1);
      dest.append(s2.charAt(0));
   }
  else {dest.append(SFx.charAt(i));}
}
String s=dest.toString();
long iix=Long.parseLong(s);
ix=iix;
x=xmin+(xmax-xmin)*(double)ix/(double)imax;
//==================
}

public String toString()
{String s;
 s="x="+x+"ix="+ix+"xmin="+xmin+"xmax="+xmax;
 return s;
}
}
```

Program 5.18.7-2 Genetic algorithm constructed from long type integer numbers of 18 digits.

```
 import java.io.*;
import java.text.*;
import java.util.Locale;



class genotype3i
{
// each genotype3i is member of the population
// Genotype is like a chromosone in real life genetic systems
// it can be thought as an evaluation point of a function with n variable
// f(xi)=f(x1,x2,x3,..,x_nvars)
public Gene3 G[]     ;  // string of variables makes a genotype3i
public double Fitness;  // genotype3i's fitness f(xi)
public double Upper[];  // gene's upper bound; xi_lower < xi < xi_upper
public double Lower[];  // Gene's lower bound; xi_lower < xi < xi_upper
public double RFitness; // the relative fitness
public double CFitness; // cumulative fitness
public int NVARS;       // Number of variables(Genes)
public int N;           // Number of bits defining each gene
public boolean Breeder;
  public genotype3i(int iNVARS,double low[],double high[],int iN,if_xj fi)
  {setgenotype(iNVARS,low,high,iN,fi);}
  public void setgenotype(int iNVARS,double low[],double high[],int iN,if_xj fi)
  {
  setN(iN);
  setNVARS(iNVARS);
  G=new Gene3[NVARS];
  Upper=new double[NVARS];
```

```
Lower=new double[NVARS];
setLower(low);
setUpper(high);
setGene();
setFitness(fi);
}

public genotype3i(genotype3i gi,if_xj fi)
{setgenotype(gi,fi);}

public void setgenotype(genotype3i gi,if_xj fi)
{
setN(gi.N);
setNVARS(gi.NVARS);
G=new Gene3[NVARS];
Upper=new double[NVARS];
Lower=new double[NVARS];
setLower(gi.Upper);
setUpper(gi.Lower);
for(int i=0;i<NVARS;i++)
  G[i]=gi.G[i].copyGene();
setFitness(fi);
}

public genotype3i copygenotype(if_xj fi)
{
genotype3i g1=new genotype3i(this,fi);
return g1;
}

public void setN(int iN)    {N=iN;}
public void setNVARS(int iNVARS)    {NVARS=iNVARS;}
public void setLower(double lbound[]) {for(int i=0;i<NVARS;i++) Lower[i]=lbound[i];}
public void setUpper(double ubound[]) {for(int i=0;i<NVARS;i++) Upper[i]=ubound[i];}
public void setFitness(if_xj fi)
{              Fitness=fi.func(getGene());}
public void setRFitness(double isum)  {RFitness=Fitness/isum;}
public void setCFitness(double ix)  {CFitness=ix;}
public double getLower(int j) {return Lower[j];}
public double getUpper(int j) {return Upper[j];}
public double getFitness() {return Fitness;}
public double getRFitness(){return RFitness;}
public double getCFitness(){return CFitness;}
public void breed() {Breeder=true;}
public void not_breed() {Breeder=false;}
public boolean isBreed() {return Breeder;}


public void setGene(int val)
{  G[val]=new Gene3(Lower[val],Upper[val]);}

public void setGene()
{  for(int j=0;j<NVARS;j++)
                setGene(j);
}

public double getGene(int val)
{ return G[val].getX();}

public double[] getGene()
{   double x[]=new double[NVARS];
              for(int j=0;j<NVARS;j++)
                x[j]=getGene(j);
              return x;
```

```
  }

  public String toString()
  {
  //x values, Fitness,relative fitness and cumulative fitness
  String s="";
  for(int j=0;j<NVARS;j++)
              s+=G[j].toString()+" F:";
  s+=getFitness();
  s+="RF:"+getRFitness();
  s+="CF:"+getCFitness();
  return s;
  }
}  //end of class genotype3i

public class Genetic3i
{
              // Maximizing a function
              int    POPSIZE; //population size
              int    MAXGENS; //maximum number of generations;
              int    NVARS;   //number of independent variables of function
    int    N;       //number of genes in each variable
              double PXOVER; //probability of crossover;
    double PMUTATION; //proportion of mutated variables
    int Generation; //Current generation number
    int NBreeder; //Number of survivors in a generation
    int Best; //The best genotype3i in the population
    genotype3i Population[];
    //if_xj f;  // function to be avaluated by genetic algorithm

    public Genetic3i(int iPOPSIZE,int iMAXGENS,int iNVARS,int iN,
    double iPXOVER,double iPMUTATION)
    {
    POPSIZE=iPOPSIZE;
    MAXGENS=iMAXGENS;
    NVARS=iNVARS;
    N=iN;
    PXOVER=iPXOVER;
    PMUTATION=iPMUTATION;
    NBreeder=0;
    Population=new genotype3i[POPSIZE+1];
              }
              public void setPopulation(double low[],double up[],if_xj fi)
              {
              for(int i=0;i<POPSIZE+1;i++)
                { Population[i]=new genotype3i(NVARS,low,up,N,fi);}
              }

              public genotype3i[] copyPopulation(if_xj fi)
              {
              genotype3i Po[]=new genotype3i[POPSIZE+1];
              for(int i=0;i<POPSIZE+1;i++)
                { Po[i]=new genotype3i(Population[i],fi);
                  Po[i].Fitness=Population[i].Fitness;
                            Po[i].RFitness=Population[i].RFitness;
                  Po[i].CFitness=Population[i].CFitness;
                }
              return Po;
              }

              public void evaluate(if_xj fi)
              {
                int mem;
                int j;
```

```
            double x[]=new double[NVARS];
            for(mem=0;mem<POPSIZE;mem++)
            {
            //System.out.println(Population[mem].toString());
            Population[mem].setFitness(fi);
            if(Population[mem].getFitness() > Population[POPSIZE].getFitness())
              {
                      Best=mem;
                      Population[POPSIZE]=Population[mem];
              for(j=0;j<NVARS;j++)
                {
                Population[POPSIZE].G[j].copyGene(Population[mem].G[j]);
                        }//end of for(j=0..
                      }//end of if
            }//end of for(mem=0;..
          }


          public void setRFCF(genotype3i Pi[])
          {
          //calculates relative and cumulative fitness functions
          int mem;
          int PN=Pi.length;
          double sum=0.0;
          //total fitness of population
          for(mem=0;mem<PN;mem++)
            {sum+=Pi[mem].Fitness;}
          //System.out.println("sum="+sum);
          //calculate relative fitness of each genotype3i
          for(mem=0;mem<PN;mem++)
            {Pi[mem].setRFitness(sum);}
          //calculate cumulative fitness
          Pi[0].setCFitness(Pi[0].getRFitness());
for(mem=1;mem<PN;mem++)
            {Pi[mem].setCFitness(Pi[mem-1].getCFitness()+Pi[mem].getRFitness());
            }
          }

          public void toString(genotype3i Pi[])
          {
          int mem;
          int PN=Pi.length;
          for(mem=0;mem<PN;mem++)
          {
          //list them
          System.out.println("Population["+mem+"]="+Pi[mem].toString());
          }
          }

          public void select(if_xj fi)
          {
          //select the new generation members of population
          double r;
          int mem;
          setRFCF(Population);
          //create a new population;
          genotype3i Po[]=copyPopulation(fi);
setRFCF(Po);
for(int i=0;i<POPSIZE;i++)
{   mem=0;
   r=Math.random();
   double gf1=Population[i].getCFitness();
              double gf2;
   if (gf1 > r) Po[i]=Population[mem].copygenotype(fi);
```

```
                for(mem=1;mem<POPSIZE;mem++)
        {
                  gf2=Population[mem].getCFitness();
                  if( gf2>=r && gf1< r ) {Po[i]=Population[mem].copygenotype(fi);break;}
        }
 }
 setRFCF(Po);
             Population=Po;
             evaluate(fi);
 //toString(Population);
 }

public void crossover(if_xj fi)
{
int i,count;
int mem;
int POne,a;
int PTwo,b;
int point;
//select members for breeding
int iselect[]=new int[POPSIZE];
int counter=0;
double r;
 for(mem=0;mem<POPSIZE;mem++)
             {
               r=Math.random();
               if(r < PXOVER)
                 {
                          iselect[counter++]=mem;
                   Population[mem].breed();
                          NBreeder++;
                 }
               else
                 Population[mem].not_breed();
             }
             //System.out.println("iselect="+Matrix.toString(iselect));
             //let also best of the population to breed
             Population[Best].breed();

//loop through the population select breeding pairs
 for(mem=0;mem<POPSIZE;mem++)
    {
       //select two in popolation in random
       a=(int)(Math.random()*NBreeder)+1;
       b=(int)(Math.random()*NBreeder)+1;
       count=0;
       POne=0;
       PTwo=0;
       //select two individuals for breeding
       for(i=0;i<POPSIZE;i++)
         {
        if(Population[i].isBreed())
           {
           count++;
           if(count==a) POne=count;
           if(count==b) PTwo=count;
                     }
                   }
                 //perform a crossover;
                   genotype3i Po[]=new genotype3i[2];
                            Po[0]=Population[POne].copygenotype(fi);
                            Po[1]=Population[PTwo].copygenotype(fi);
                   for(i=0;i<NVARS;i++)
                   {
```

```
                        Population[POne].G[i].cross(Math.random(),Po[0].G[i],Po[1].G[i]);
                        Population[PTwo].G[i].cross(Math.random(),Po[0].G[i],Po[1].G[i]);
                        }
                        Population[POne].setFitness(fi);
                        Population[PTwo].setFitness(fi);
                }
              }

public void mutate(if_xj fi)
{
int i;
double lbound,hbound;
int nmutations;
int member;
int var;
nmutations=(int)((double)(POPSIZE*NVARS*PMUTATION*N));
for(i=0;i<nmutations;i++)
  {
  member=(int)(Math.random()*POPSIZE);
  var=(int)(Math.random()*NVARS);
  //replace the old value with a new mutated one
  Population[member].G[var].mutate();
          //after mutation recalculate the function value
  Population[member].setFitness(fi);
  }
}

public String report()
{
String s;
int i;
double best_value;
double avg;
double stdev;
double sum_square;
double square_sum;
double sum;
double xx;
sum=0;
sum_square=0.0;
for(i=0;i<POPSIZE;i++)
  {
          xx= Population[i].getFitness();
  sum+=xx;
  sum_square=xx*xx;
          }
avg=sum/(double)POPSIZE;
square_sum=sum*sum/(double)POPSIZE;
stdev=Math.sqrt(square_sum);
best_value=Population[POPSIZE].getFitness();
double aa[]=new double[NVARS+1];
for(i=0;i < NVARS;i++)
 {
 aa[i]=Population[POPSIZE].getGene(i);
          }
          aa[NVARS]=Population[POPSIZE].getFitness();

//s="Generation = "+Generation+"best value = "+Matrix.toString(aa)+"Average = "+avg+"Standart Deviation = "+stdev;
s="Generation = "+Generation+"best value = "+Matrix.toString(aa);
return s;
}

public double[] getBest()
{
```

```
  double aa[]=new double[NVARS+1];
  for(int i=0;i < NVARS;i++)
    {
   aa[i]=Population[POPSIZE].getGene(i);
               }
               aa[NVARS]=Population[POPSIZE].getFitness();
  return aa;
  }

  public double[] calculate(if_xj fi,boolean report)
  {
  evaluate(fi);
  Generation=0;
  while(Generation<MAXGENS)
    {
    Generation++;
    NBreeder=(int)(0.8*POPSIZE);
    select(fi);
    crossover(fi);
    mutate(fi);
    evaluate(fi);
    //if(report)
       //System.out.println(report());
    }
  return getBest();
  }
  public double[] solution(if_xj fi)
  {
  evaluate(fi);
  Generation=0;
  while(Generation<MAXGENS)
    {
    Generation++;
    NBreeder=(int)(0.8*POPSIZE);
    select(fi);
    crossover(fi);
    mutate(fi);
    evaluate(fi);
    }
  double a[]=getBest();
  double b[]=new double[a.length-1];
  for(int i=0;i<b.length;i++)       {b[i]=a[i];}
  return b;
  }
}
```

Program 5.18.7-3 Test program NA109 for Gene constructed from long type integer numbers of 9 digits.

```
 import java.io.*;

class NA109i
{
   public static void main(String args[]) throws IOException
   {
   int N=63;
              int POPSIZE=500;
   int MAXGENS=500;
   double low[]={-10,-10};
   double high[]={10,10};
   int NVARS=low.length;
   double PXOVER=0.3;
   double PMUTATION=0.02;
   Genetic3i xx=new Genetic3i(POPSIZE,MAXGENS,NVARS,N,PXOVER,PMUTATION);
```

```
   if_xj fx=(double x[])->1.0+Math.cos(Math.PI*(x[0]-3.0))*Math.cos(2.0*Math.PI*(x[1]-2.0))/
   (1+(x[0]-3.0)*(x[0]-3.0)+(x[1]-2.0)*(x[1]-2.0));

   xx.setPopulation(low,high,fx);
   System.out.println("best=\n"+Matrix.toStringT(xx.calculate(fx,false)));
   }
}
```

```
---------- Capture Output ----------
> "E:\co\java\bin\javaw.exe" NA109i
best=
        3.000000022142125
        2.000000000720926
        1.999999999999997
> Terminated with exit code 0.
```

5.19 STOCHASTIC METHODS: MONTE-CARLO METHOD

Monte carlo optimization as a method is very simple. Points between the given limits are selected randomly and evaluated. When number of points increase to very high numbers, statistically there will be a homogeneous coverege of all the search region. Record the best optimum found so far.What is achived is similar to plotting data, but in monte carlo method taken constant steps and walk through the steps are not needed because randomness takes care of uniformity of search when the total number is big enough.

Program 5.19-1 Monte carlo stochastic optimization method

```
 / Dr. M. Turhan ÇOBAN
//
// EGE University School of Engineering, Mechanical Engineering Department
// package : Numerical analysis package
// subject : optimization
// group   : Monte-Carlo optimization(maximum) Algorithms
// monte_carlo_opt Class
// class selects random points at the given area and selects the best
// results found
//
import java.util.*;
import java.io.*;


public class monte_carlo_opti extends Object
{
int N;
public double max[];
public double min[];
public double x;

public monte_carlo_opti(int Ni,double imin[],double imax[])
{
 max=new double[imin.length];
 min=new double[imin.length];
 N=Ni;
 int n=imin.length;
 for(int i=0;i<n;i++)
 {min[i]=imin[i];max[i]=imax[i];}
}

public double[] monte_carlo_max(if_xj fi)
```

```
{
 int n=min.length;
 double x[]=new double[n];
 double xmax[]=new double[n];
 double max_number=-1e99;
 double f;
 for(int k=0;k<N;k++)
 {
 for(int i=0;i<n;i++)
 {x[i]=min[i]+(max[i]-min[i])*Math.random();}
 f=fi.func(x);
 if(f>max_number)
   {  max_number=f;
      for(int i=0;i<n;i++)
        {xmax[i]=x[i];}
   }
 }
 return xmax;
}}
```

Program 5.19-2 Monte carlo stochastic optimization method example

```
 import java.io.*;

class NA110
{
   public static void main(String args[]) throws IOException
   {
   double low[]={-1};
   double high[]={2};
   monte_carlo_opti mco=new monte_carlo_opti(100000,low,high);
   if_xj fx=(double x[])->x[0]*Math.sin(10.0*Math.PI*x[0])+3.0;
   double xmax[]=mco.monte_carlo_max(fx);
   for(int i=0;i<xmax.length;i++)
   {System.out.println(xmax[i]);}
   }
}
```

```
 ---------- Capture Output ----------
> "E:\co\java\bin\javaw.exe" NA110
1.850556252949647

> Terminated with exit code 0.
```

Program 5.19-3 Monte carlo stochastic optimization method example

```
 import java.io.*;

class NA111
{ public static void main(String args[]) throws IOException
   {
   double low[]={-10,-10};
   double high[]={10,10};
   monte_carlo_opti mco=new monte_carlo_opti(100000,low,high);
   if_xj fx=(double x[])->1.0+Math.cos(Math.PI*(x[0]-3.0))*Math.cos(2.0*Math.PI*(x[1]-2.0))/
   (1+(x[0]-3.0)*(x[0]-3.0)+(x[1]-2.0)*(x[1]-2.0));
   double xmax[]=mco.monte_carlo_max(fx);
   for(int i=0;i<xmax.length;i++)
   {System.out.println(xmax[i]);}
   }
}
```

```
---------- Capture Output ----------
```

```
> "E:\co\java\bin\javaw.exe" NA111
2.9960487170622745
1.99874133664329

> Terminated with exit code 0.
```

5.20 STOCHASTIC METHODS: ITERATIVE HILL CLIMBING

Iterative Hill climbing method, a global maximum finding method similar to genetic algorithms and monte carlo. It is a simple method, yet it is more complicated than the Monte-Carlo method. As a basic strategy a point is selected and evaluated. Furthermore more point (but not excessively more) in the neigborhood of the point is evaluated then jumped to a new point and repeat the same process. If the method is given as an algorithm:

```
Iterative hill climbing algorithm
start
    t=0
    while
        local=false
        select a random point vc
        calculate the function for vc
        while
            select approximately 30 points in neighbourhood of vc and evaluate the function values
            In between these points select the maximum one as the solution vector
            if f(vc) < f(vn)  then  vc = vn
            else  local=true
        until local
        t=t+1
     until t=MAXIMUM
finish
```

The basic problem in this code is the definition of neighbourhood region. In reality Genetic engineering Gene concept is just establishing this, a child gene is always in the neighbourhood of the parent genes. By using this property the method can be establish. A few example of describing the gene was given in genetic engineering method, the same methods can be utilised for iterative hill climbing as well. In our first class vector(point) vc is made by using binary gene

Program 5.20-1 Iterative hill climbing with binary gene

```
 import java.io.*;
import java.text.*;
import java.util.Locale;

public class iteratedhillclimberi
{
    // Maximizing a function
   //iterated hill climber
   // a version steepest ascent with statistical approach to variable
   genotype1i Vc;
   genotype1i Vn[];
   genotype1i best;
   int MAX;
   double t;
   boolean local;
   int POPSIZE;
   int NVARS;
   int N;//number of bit representation for each variable

   //if_xj f;  // function to be avaluated by genetic algorithm
```

```
public iteratedhillclimberi(int iPOPSIZE,int iNVARS,int iN)
{
POPSIZE=iPOPSIZE;//Vn size number of neigbour variable f.e. 30
NVARS=iNVARS; // number of variables equation to solve
MAX=20;
N=iN;           // number of bits represent each variable for example 32
Vn=new genotype1i[POPSIZE+1];
t=0;
local=false;
        }

        public void setVc(double low[],double up[],if_xj fi)
        {
        //select originalgene andomly
        Vc=new genotype1i(NVARS,low,up,N,fi);
        }
        public void setBest(genotype1i gi,if_xj fi)
        {
        //select originalgene andomly
        best=new genotype1i(gi,fi);
        }
public void setVc(genotype1i gi,if_xj fi)
        {
        //select original gene from other gene
        Vc=new genotype1i(gi,fi);
        }
        public void setVn(if_xj fi)
        {//select neighbouring genes similar to the original one
        for(int i=0;i<POPSIZE+1;i++)
          { Vn[i]=new genotype1i(Vc,fi);}
           mutate(fi);
        }


        public void evaluate(if_xj fi)
        {
          int mem;
          int j;
          double x[]=new double[NVARS];
          for(mem=0;mem<POPSIZE;mem++)
          {
          //System.out.println(Vn[mem].toString());
          Vn[mem].setFitness(fi);
          if(Vn[mem].getFitness() > Vn[POPSIZE].getFitness())
             {
                        Vn[POPSIZE]=Vn[mem];
             for(j=0;j<NVARS;j++)
               {
               Vn[POPSIZE].G[j].copyGene(Vn[mem].G[j]);
                         }//end of for(j=0..
                        }//end of if
          }//end of for(mem=0;..
        }

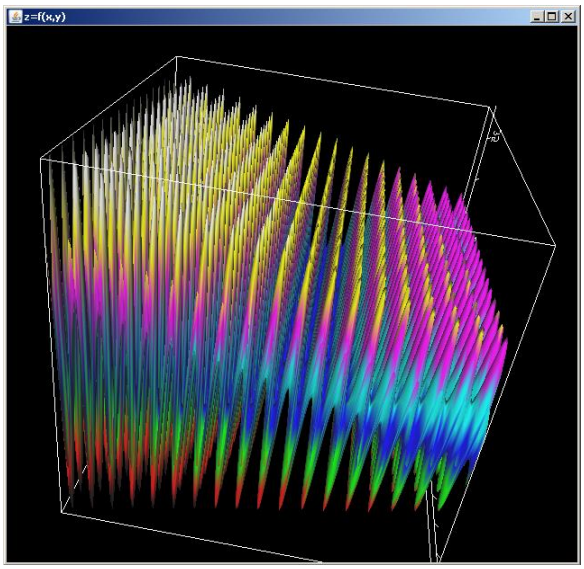
        public void toString(genotype1i Pi[])
        {
        int mem;
        int PN=Pi.length;
        for(mem=0;mem<PN;mem++)
        {
        //list them
        System.out.println("Vn["+mem+"]="+Pi[mem].toString());
        }
```

```
            }


public void mutate(if_xj fi)
{
int i;
double lbound,hbound;
int nmutations;
int member;
int var;
nmutations=(int)((double)(POPSIZE*NVARS*N));
for(i=0;i<nmutations;i++)
  {
  member=(int)(Math.random()*POPSIZE);
  var=(int)(Math.random()*NVARS);
  //replace the old value with a new mutated one
  Vn[member].G[var].mutate();
  //after mutation recalculate the function value
  Vn[member].setFitness(fi);
  }
}


public double[] getBest(genotype1i b)
{
double aa[]=new double[NVARS+1];
for(int i=0;i < NVARS;i++)
 {
 aa[i]=b.getGene(i);
            }
            aa[NVARS]=b.getFitness();
return aa;
}

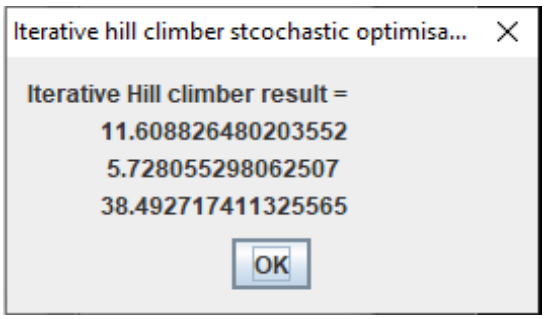
public double[] calculate(double low[],double up[],if_xj fi,boolean report)
{
 setVc(low,up,fi);// new set of random selection in each time
 setBest(Vc,fi);
            do
 {
            local=true;
 do
 {
 setVn(fi);
 evaluate(fi);
 if(Vn[POPSIZE].getFitness() > Vc.getFitness())
 {
               setVc(Vn[POPSIZE], fi);
               if(Vn[POPSIZE].getFitness() > best.getFitness()) setBest(Vn[POPSIZE],fi);
 }
 else local=false;
 } while(local);
 t++;
 setVc(low,up,fi); //new random value
 if(report)
   System.out.println(Matrix.toString(getBest(best)));
 } while(t<MAX);
return getBest(best);
}
```



Program 5.20-2 Iterative hill climbing with binary gene example program

```
 //iterated hillclimber
//iterasyonlu en dik yamaç tırmanıcı
import java.io.*;
import javax.swing.JOptionPane;

class NA112
{
   public static void main(String args[]) throws IOException
   {
   int N=18;
            int POPSIZE=400;
   int MAXGENS=400;
   int NVARS=2;
   if_xj fax=(double x[])->21.5+x[0]*Math.sin(4.0*Math.PI*x[0])+x[1]*Math.sin(20.0*Math.PI*x[1]);
   iteratedhillclimberi xx=new iteratedhillclimberi(POPSIZE,NVARS,N);
   double low[]={-3,4.1};
   double high[]={12.1,5.8};
   double a[]=xx.calculate(low,high,fax,false);
   double x0[]=new double[NVARS];
   for(int i=0;i<NVARS;i++) x0[i]=a[i];
   String s="Iterative Hill climber result =\n"+Matrix.toStringT(a);
    String s2="Iterative hill climber stcochastic optimisation:";
    JOptionPane.showMessageDialog(null,s,s2,JOptionPane.PLAIN_MESSAGE);
   }
}
```



Program 5.20-3 Iterative hill climbing with Gauss random gene

```
 import java.io.*;
```

```
import java.text.*;
import java.util.Locale;
public class IHCi
{
                // Maximizing a function
   // iterated Hill climber
   // a stochastic approach
   double T;
   double Vc[];
   double Vn[][];
   double best[];
   int MAX;//number
   int GLIMIT;//arka arkaya başarılı steplerin sayısı
   boolean local;
   int NVARS;//numberof variables
   int N;//number of bit representation for each variable
   double T0;
   double k,k0;//time index
   int POPSIZE;
   double low[];
   double high[];
   int t;
   //if_xj f;  // function to be avaluated by genetic algorithm

   public IHCi(int iPOPSIZE,int iNVARS,int iN,double Ti)
   {
                T=Ti;
                T0=T;
   POPSIZE=iPOPSIZE;
   NVARS=iNVARS; // number of variables equation to solve
   MAX=100;
   N=iN;           // number of bits represent each variable for example 32
   local=false;

   t=0;
   Vc=new double[NVARS];
   Vn=new double[POPSIZE+1][NVARS];
   best=new double[NVARS];
   low=new double[NVARS];
   high=new double[NVARS];
   local=true;
                }
public static double  gaussrandom(double xort,double r)
{ int iset=0;
 double gset=0;
 double fac,rsq,v1,v2;
 double aret=0;
 if(iset==0)
 {
  do{
                v1=2.0*Math.random()-1.0;
                v2=2.0*Math.random()-1.0;
                rsq=v1*v1+v2*v2;
    }while(rsq>=1.0 || rsq==0.0);
  fac=Math.sqrt(-2.0*Math.log(rsq)/rsq);
  gset=v1*fac;
  iset=1;
  aret=v2*fac;
 }
 else
 {
 iset=0;
 aret=gset;
 }
```

```
return aret/4.0*r+xort;
}

            public void setVc(double ilow[],double ihigh[],if_xj fi)
            {

            //select original gene randomly
            for(int i=0;i<NVARS;i++)
              {low[i]=ilow[i];high[i]=ihigh[i];Vc[i]=(high[i]-low[i])*Math.random();}
  }

            public void setBest(double gi[],if_xj fi)
            {
            //select originalgene andomly
            for(int i=0;i<NVARS;i++)
              {best[i]=gi[i];}
            }

  public void setVc(double gi[],if_xj fi)
            {
            //select original gene from other gene
                        for(int i=0;i<NVARS;i++)
              {Vc[i]=gi[i];}
            }

            public void setVn(if_xj fi)
            {//select neighbouring genes similar to the original one
            for(int i=0;i<POPSIZE;i++)
              { for(int j=0;j<NVARS;j++)
               {Vn[i][j]=Vc[j];}}
               mutate(fi);
            }

  public void mutate(if_xj fi)
  {
  int i;
  double lbound,hbound;
  int nmutations;
  int member;
  int var;
  nmutations=(int)((double)(POPSIZE*NVARS*N));
  for(i=0;i<nmutations;i++)
    {
    member=(int)(Math.random()*POPSIZE);
    var=(int)(Math.random()*NVARS);
    //replace the old value with a new mutated one
    //neigboring factor
    double d1=high[var]-Vn[member][var];
    double d2=Vn[member][var]-low[var];
    double d=Math.abs(Math.min(d1,d2));

    double R=gaussrandom(Vn[member][var],d);
    Vn[member][var] = R;
    //System.out.println("R="+R+"d1="+d1+"d2="+d2+"d="+d+"var = "+var+" "+Vn[member][var]);
    if(Vn[member][var]>high[var]) Vn[member][var]=high[var]-d*Math.random()*0.122561462;
    else if (Vn[member][var]<low[var]) Vn[member][var]=low[var]+d*Math.random()*0.122561462;;
              //after mutation recalculate the function value
    }
  }


            public void evaluate(if_xj fi)
            {
              int mem;
```

```
                int j;
                double x[]=new double[NVARS];
                for(mem=0;mem<POPSIZE;mem++)
                {
                //System.out.println(Vn[mem].toString());
                if(getFitness(Vn[mem],fi) > getFitness(Vn[POPSIZE],fi))
                  {
                          //Vn[POPSIZE]=Vn[mem];
                  for(j=0;j<NVARS;j++)
                    {
                    Vn[POPSIZE][j]=Vn[mem][j];
                          }//end of for(j=0..
                        }//end of if
                }//end of for(mem=0;..
               }



  public double[] getBest(double[] b,if_xj fi)
  {
  double aa[]=new double[NVARS+1];
  for(int i=0;i < NVARS;i++)
   {
   aa[i]=b[i];
               }
               aa[NVARS]=getFitness(b,fi);
  return aa;
  }

  public double getFitness(double []xi,if_xj fi)
  {
  return fi.func(xi);
  }

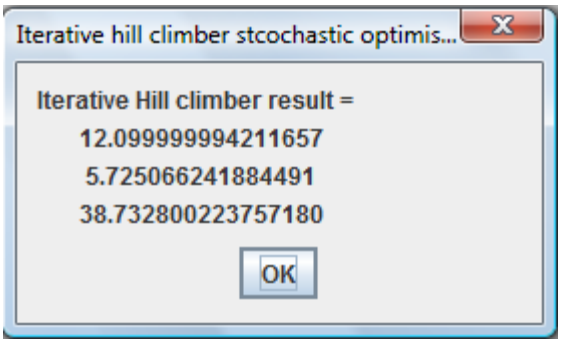
  public double[] calculate(double low[],double up[],if_xj fi,boolean report)
  {
   local=true;
   setVc(low,up,fi); //new random value
               t=0;
   do
   {
   do
   {   setVn(fi);
      evaluate(fi);
      if(getFitness(Vn[POPSIZE],fi) > getFitness(Vc,fi))
      {  setVc(Vn[POPSIZE], fi);
                  if(getFitness(Vn[POPSIZE],fi) > getFitness(best,fi)) setBest(Vn[POPSIZE],fi);
      }
      else local=false;
   } while(local);
   if(report)
     System.out.println(Matrix.toString(getBest(best,fi)));
   t++;
   } while(t<MAX);
  return getBest(best,fi);
   }
}
```

Program 5.20-4 Iterative hill climbing with Gauss random gene example problem

```
 // iterated hill climbing
//
import java.io.*;
```

```
import javax.swing.JOptionPane;

class NA113
{
   public static void main(String args[]) throws IOException
   {
   int N=18;
            int POPSIZE=30;
   int MAXGENS=400;
   int NVARS=2;
   if_xj fax=(double x[])->21.5+x[0]*Math.sin(4.0*Math.PI*x[0])+x[1]*Math.sin(20.0*Math.PI*x[1]);
   IHCi xx=new IHCi(POPSIZE,NVARS,N,20);
   double low[]={-3,4.1};
   double high[]={12.1,5.8};
   double a[]=xx.calculate(low,high,fax,false);
   double x0[]=new double[NVARS];
   for(int i=0;i<NVARS;i++) x0[i]=a[i];
   String s="Iterative Hill climber result =\n"+Matrix.toStringT(a);
   String s2="Iterative hill climber stcochastic optimisation";
   JOptionPane.showMessageDialog(null,s,s2,JOptionPane.PLAIN_MESSAGE);
   }
}
```



5.21 STOCHASTIC METHODS: SIMULATED ANNEALING

An alternative approach obtained from the thermodynamic analogy is the simulated annealing. If a metal is heated obove the melting point, atoms in the structure started randomly move. When the temperature drops movemets slow down, and slide to a lower energy level. At the end atoms self organise themselves in cristal structures with minimum energy levels. This process is known as annealing. The energy state of an annealed metal can be expressed by using Boltzman statistics.

$$p(e) = \alpha\, exp\left(\frac{-e}{kT}\right)$$

In this equation e>=0 is the energy level, T is the temperature, α=1/(kT), k is boltzman constant p(e) is the probability distribution function. Annealing can be taken as a natural optimization process. This phenemonen can be mimic in computer codes. Consider minimisation of function $\mathbf{f(x_1,x_2,x_3,\ldots,x_n)}$ fonksiyonunu minimize etme problemini göz önüne alalim. In order to establish the model an artificial Temperature value will be assumed. Optimization should be started with a big value of T. (In the starting of the annealing process the temperature of the metal is high) . As a starting point take an initial guess vector $x^0=\mathbf{(x_1,x_2,x_3,\ldots,x_n)^0}$ vektörünü alalim. If we change the initial vector a small amount Δx, distribution in Δx should follow the Gauss distribution. So in the neighbourhood of first point a second point can be obtain as $x^1=x^0+\Delta x$. The difference in the function value will be $\Delta f=f(x^1)-f(x^0)$). Gaussian distribution function is:

$p(\Delta x_j) = \frac{1}{T\sqrt{2\pi}}\ exp\left(\frac{-\Delta x_j}{2T^2}\right) \quad 1 \le j \le n$

If $\Delta f < 0$ we can accept the new point as a better approach to the minimum. But if $\Delta f \ge 0$ we do not automatically reject the second point. A Boltzmann distribution function is created and investigated.
$p(\Delta f) = \frac{1}{kT}\ exp\left(\frac{-\Delta f}{kT}\right)$

In order to decide accepting or rejecting the result a random number z is created in the range of [0,1/(kT)]. If p(Δf) > z accept x^1 as a new iteration value, If not reject it. Then a new Δx for a new vector is calculated. In time T temperature factor decreased as a some defined function of time. In the beginning of the iteration due to high temperatures the chance of accepting a new x value is high. In time when the T is getting smaller, the system became more conservative on accepting $\Delta f \ge 0$ cases. If this process is written as an algorithm

```
Simulated annealing algorithm
start
    t=0
    set a T valu.
     Select a vc value as random
     Calculate the function value for vc
    While
        while
            select a new point vn in the neighbourhood of vc
             if f(vc) < f(vn) then vc = vn (accepth the new point)
            else if tesadüfi sayi(0-1) < exp[(f(vn)-f(vc))/T] then vc = vn even if f(vc) >= f(vn) accept the new
point
        until (exit condition)
        t=t+1
    until (exit condition)
stop
```

Function T can be evaluated with different formulas, some examples are given below.

$T = \frac{T_0}{1+\log(1+t)}$

$T = T - \frac{\log(t_0)}{t\log(t)}$

$T = \frac{T_0}{1+t}$

As an actual program two version is developed. In the first version Gene1 and genotpe1 classes of genetic algorithms (binary genes) are used to define the neighbourhood area (only one bit is replaced in random-gene is mutated)

Program 5.21-1 Simulated annealing with binary gene

```
 import java.io.*;
import java.text.*;
import java.util.Locale;


public class simulatedannealingi
{
```

```
        // Maximizing a function
//simulated annealing
// a version of simulated annealing statistical approach
double T; //Temperature
genotype1i Vc;
genotype1i Vn[];
genotype1i best;
genotype1i oldbest;
int MAX;//number
int GLIMIT;//arka arkaya başarılı steplerin sayısı
boolean local;
int NVARS;//numberof variables
int N;//number of bit representation for each variable
double T0;
double k,k0;//time index
int POPSIZE;
int BESTLIMIT;
int bestcounter;

//if_xj f;  // function to be avaluated by genetic algorithm

public simulatedannealingi(int iPOPSIZE,int iNVARS,int iN,double Ti)
{
        T=Ti;
        T0=T;
POPSIZE=iPOPSIZE;
NVARS=iNVARS; // number of variables equation to solve
MAX=20;
N=iN;          // number of bits represent each variable for example 32
Vn=new genotype1i[POPSIZE+1];
local=false;
GLIMIT=10;
BESTLIMIT=3;
k=100.0;
k0=k;
bestcounter=0;
        }

        public void setVc(double low[],double up[],if_xj fi)
        {
        //select originalgene andomly
        Vc=new genotype1i(NVARS,low,up,N,fi);
        }
        public void setBest(genotype1i gi,if_xj fi)
        {
        //select originalgene andomly
        best=new genotype1i(gi,fi);
        }
        public void setoldBest(if_xj fi)
        {
        //select originalgene andomly
        oldbest=new genotype1i(best,fi);
        }

public void setVc(genotype1i gi,if_xj fi)
        {
        //select original gene from other gene
        Vc=new genotype1i(gi,fi);
        }
        public void setVn(if_xj fi)
        {//select neighbouring genes similar to the original one
        for(int i=0;i<POPSIZE+1;i++)
          { Vn[i]=new genotype1i(Vc,fi);}
           mutate(fi);
```

```
        }

        public void evaluate(if_xj fi)
        {
          int mem;
          int j;
          double x[]=new double[NVARS];
          for(mem=0;mem<POPSIZE;mem++)
          {
          //System.out.println(Vn[mem].toString());
          Vn[mem].setFitness(fi);
          if(Vn[mem].getFitness() > Vn[POPSIZE].getFitness())
            {
                    Vn[POPSIZE]=Vn[mem];
            for(j=0;j<NVARS;j++)
              {
              Vn[POPSIZE].G[j].copyGene(Vn[mem].G[j]);
                }//end of for(j=0..
              }//end of if
          }//end of for(mem=0;..
        }

        public void setRFCF(genotype1i Pi[])
        {
        //calculates relative and cumulative fitness functions
        int mem;
        int PN=Pi.length;
        double sum=0.0;
        //total fitness of Vn
        for(mem=0;mem<PN;mem++)
          {sum+=Pi[mem].Fitness;}
        //System.out.println("sum="+sum);
        //calculate relative fitness of each genotype1i
        for(mem=0;mem<PN;mem++)
          {Pi[mem].setRFitness(sum);}
        //calculate cumulative fitness
        Pi[0].setCFitness(Pi[0].getRFitness());
 for(mem=1;mem<PN;mem++)
          {Pi[mem].setCFitness(Pi[mem-1].getCFitness()+Pi[mem].getRFitness());
          }
        }

        public void toString(genotype1i Pi[])
        {
        int mem;
        int PN=Pi.length;
        for(mem=0;mem<PN;mem++)
        {
        //list them
        System.out.println("Vn["+mem+"]="+Pi[mem].toString());
        }
        }


public void mutate(if_xj fi)
{
int i;
double lbound,hbound;
int nmutations;
int member;
int var;
nmutations=(int)((double)(POPSIZE*NVARS*N));
for(i=0;i<nmutations;i++)
  {
```

```
     member=(int)(Math.random()*POPSIZE);
     var=(int)(Math.random()*NVARS);
     //replace the old value with a new mutated one
     Vn[member].G[var].mutate();
                //after mutation recalculate the function value
     Vn[member].setFitness(fi);
     }
   }

   public double[] getBest(genotype1i b)
   {
   double aa[]=new double[NVARS+1];
   for(int i=0;i < NVARS;i++)
    {
    aa[i]=b.getGene(i);
                }
                aa[NVARS]=b.getFitness();
   return aa;
   }
   public double[] calculate(double low[],double up[],if_xj fi,boolean report)
   {
    int good_swap=0;
                setVc(low,up,fi);// new set of random selection in each time
    setBest(Vc,fi);
                do
    {
                setoldBest(fi);
    do
    {   setVn(fi);
        evaluate(fi);
        if(Vn[POPSIZE].getFitness() > Vc.getFitness())
        {  setVc(Vn[POPSIZE], fi);
                    if(Vn[POPSIZE].getFitness() > best.getFitness())
                       {setBest(Vn[POPSIZE],fi); }
                          if(oldbest.getFitness() == best.getFitness()) bestcounter++;
                    if(bestcounter>BESTLIMIT) break;
                    good_swap++;
        }
        else if(Math.random()<Math.exp((Vn[0].getFitness()-Vc.getFitness())/T))
        {   setVc(Vn[POPSIZE],fi);}
    if(report)
       System.out.println(Matrix.toString(getBest(best)));
    } while(good_swap<GLIMIT);
    k+=1.0;
    double lk=Math.log(k);
    T=T-Math.log(k0)/(k*lk*lk);
    setVc(low,up,fi); //new random value

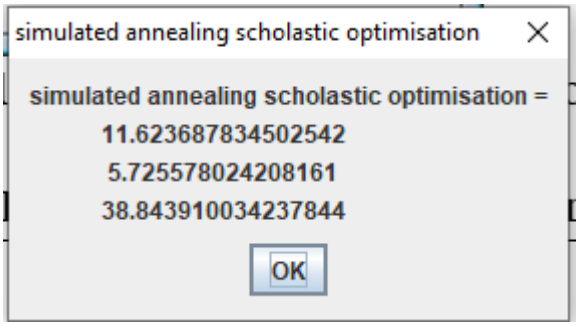
    } while(k<MAX);
   return getBest(best);
   }
}
```

Program 5.21-2 Simulated annealing with binary gene example program NA114.java

```
 // simulated annealing
//
import java.io.*;
import javax.swing.JOptionPane;

class NA114
{   public static void main(String args[]) throws IOException
    {
    int N=18;
```

```
    int POPSIZE=1;
    int MAXGENS=400;
    int NVARS=2;
    if_xj fax=(double x[])->21.5+x[0]*Math.sin(4.0*Math.PI*x[0])+x[1]*Math.sin(20.0*Math.PI*x[1]);;
    simulatedannealingi xx=new simulatedannealingi(POPSIZE,NVARS,N,20);
    double low[]={-3,4.1};
    double high[]={12.1,5.8};
    double a[]=xx.calculate(low,high,fax,false);
    double x0[]=new double[NVARS];
    for(int i=0;i<NVARS;i++) x0[i]=a[i];
    String s="simulated annealing scholastic optimisation =\n"+Matrix.toStringT(a);
     String s2="simulated annealing scholastic optimisation ";
     JOptionPane.showMessageDialog(null,s,s2,JOptionPane.PLAIN_MESSAGE);
    }
}
```



The second version will be using Gauss random gene to calculate neighbour point.

Program 5.21-3 Simulated annealing with Gauss random gene

```
 import java.io.*;
import java.text.*;
import java.util.Locale;

public class SAi
{   // Maximizing a function
    // simulated annealing
    // a version of simulated annealing statistical approach

    double T; //Temperature
    double Vc[];
    double Vn[][];
    double best[];
    int MAX;//number
    int GLIMIT;//arka arkaya başarılı steplerin SAiyısı
    boolean local;
    int NVARS;//numberof variables
    int N;//number of bit representation for each variable
    double T0;
    double k,k0;//time index
    int POPSIZE;
    double low[];
    double high[];

    //if_xj f;  // function to be avaluated by genetic algorithm

    public SAi(int iPOPSIZE,int iNVARS,int iN,double Ti)
    {
                T=Ti;
                T0=T;
    POPSIZE=iPOPSIZE;
    NVARS=iNVARS; // number of variables equation to solve
```

```
    MAX=20;
    N=iN;           // number of bits represent each variable for example 32
    local=false;
    GLIMIT=30;
    k=100.0;
    k0=k;
    Vc=new double[NVARS];
    Vn=new double[POPSIZE+1][NVARS];
    best=new double[NVARS];
    low=new double[NVARS];
    high=new double[NVARS];
                }

public static double  gaussrandom(double xort,double r)
{ int iset=0;
 double gset=0;
 double fac,rsq,v1,v2;
 double aret=0;
 if(iset==0)
 {
  do{
                v1=2.0*Math.random()-1.0;
                v2=2.0*Math.random()-1.0;
                rsq=v1*v1+v2*v2;
    }while(rsq>=1.0 || rsq==0.0);
  fac=Math.sqrt(-2.0*Math.log(rsq)/rsq);
  gset=v1*fac;
  iset=1;
  aret=v2*fac;
 }
 else
 {
 iset=0;
 aret=gset;
 }
 return aret/4.0*r+xort;
 }
                public void setVc(double ilow[],double ihigh[],if_xj fi)
                {
                //select original gene randomly
                for(int i=0;i<NVARS;i++)
                  {low[i]=ilow[i];high[i]=ihigh[i];Vc[i]=(high[i]-low[i])*Math.random();}
    }
                public void setBest(double gi[],if_xj fi)
                {
                for(int i=0;i<NVARS;i++)
                  {best[i]=gi[i];}
                }

    public void setVc(double gi[],if_xj fi)
                {
                //select original gene from other gene
                                for(int i=0;i<NVARS;i++)
                  {Vc[i]=gi[i];}
                }

                public void setVn(if_xj fi)
                {//select neighbouring genes similar to the original one
                for(int i=0;i<POPSIZE+1;i++)
                  { for(int j=0;j<NVARS;j++)
                    {Vn[i][j]=Vc[i];}}
                    mutate(fi);
                }
   public void mutate(if_xj fi)
```

```
{
int i;
double lbound,hbound;
int nmutations;
int member;
int var;
nmutations=(int)((double)(POPSIZE*NVARS*N));
for(i=0;i<nmutations;i++)
  {
  member=(int)(Math.random()*POPSIZE);
  var=(int)(Math.random()*NVARS);
  //replace the old value with a new mutated one
  //neigboring factor
  double d1=high[var]-Vn[member][var];
  double d2=Vn[member][var]-low[var];
  double d=Math.min(d1,d2);
  double R=gaussrandom(Vn[member][var],d*Math.random());
  Vn[member][var]= R;
  if(Vn[member][var]>high[var]) Vn[member][var]=high[var]-Math.random()*0.22561462;
  else if (Vn[member][var]<low[var]) Vn[member][var]=low[var]+Math.random()*0.22561462;;
            //after mutation recalculate the function value
  }
}
          public void evaluate(if_xj fi)
          {
            int mem;
            int j;
            double x[]=new double[NVARS];
            for(mem=0;mem<POPSIZE;mem++)
            {
            //System.out.println(Vn[mem].toString());
            if(getFitness(Vn[mem],fi) > getFitness(Vn[POPSIZE],fi))
              {
                    //Vn[POPSIZE]=Vn[mem];
              for(j=0;j<NVARS;j++)
                {
                Vn[POPSIZE][j]=Vn[mem][j];
                    }//end of for(j=0..
                  }//end of if
            }//end of for(mem=0;..
          }

public double[] getBest(double[] b,if_xj fi)
{
double aa[]=new double[NVARS+1];
for(int i=0;i < NVARS;i++)
 {
 aa[i]=b[i];
          }
          aa[NVARS]=getFitness(b,fi);
return aa;
}

public double getFitness(double []xi,if_xj fi)
{
return fi.func(xi);
}

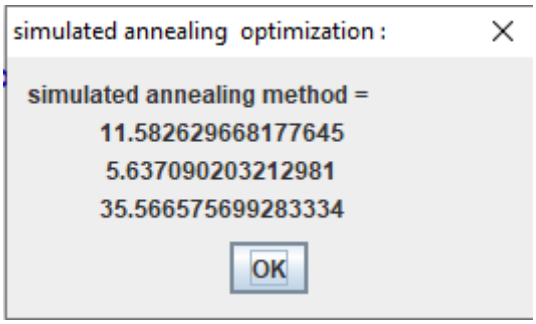
public double[] calculate(double low[],double up[],if_xj fi,boolean report)
{
 int good_swap=0;
          setVc(low,up,fi);// new set of random selection in each time
 setBest(Vc,fi);
          do
```

```
    {
    do
    {   setVn(fi);
        evaluate(fi);
        if(getFitness(Vn[POPSIZE],fi) > getFitness(Vc,fi))
        {  setVc(Vn[POPSIZE], fi);
                    if(getFitness(Vn[POPSIZE],fi) > getFitness(best,fi)) setBest(Vn[POPSIZE],fi);
                    good_swap++;
        }
        else if(Math.random()<Math.exp((getFitness(Vn[0],fi)-getFitness(Vc,fi))/(k*T)))
        {   setVc(Vn[POPSIZE],fi);}
    if(report)
       System.out.println(Matrix.toString(getBest(best,fi)));
    } while(good_swap<GLIMIT);
    k+=1.0;
    double lk=Math.log(k);
    T=T-Math.log(k0)/(k*lk*lk);
    setVc(low,up,fi); //new random value
    } while(k<MAX);
  return getBest(best,fi);
  }
}
```

Program 5.21-4 Simulated annealing with Gauss random gene example program NA115.java

```
 // simulated annealing
//
import java.io.*;
import javax.swing.JOptionPane;

class NA115
{
   public static void main(String args[]) throws IOException
   {
   int N=18;
              int POPSIZE=1;
   int MAXGENS=500;
   int NVARS=2;
   if_xj fax=(double x[])->21.5+x[0]*Math.sin(4.0*Math.PI*x[0])+x[1]*Math.sin(20.0*Math.PI*x[1]);
   SAi xx=new SAi(POPSIZE,NVARS,N,20);
   double low[]={-3,4.1};
   double high[]={12.1,5.8};
   double a[]=xx.calculate(low,high,fax,false);
   double x0[]=new double[NVARS];
   for(int i=0;i<NVARS;i++) x0[i]=a[i];
   String s="simulated annealing method =\n"+Matrix.toStringT(a);
   String s2="simulated annealing  optimization :";
   JOptionPane.showMessageDialog(null,s,s2,JOptionPane.PLAIN_MESSAGE);
   }
}
```



5.22 STOCHASTIC METHODS: FIREFLY ALGORITHM

You might notice amazing flashing lights of fireflys in the summer nights. The flashing light is produced by a process of bioluminescence. The true main purpose of such flashes are still on debate but one of the fundamental function is attracting mate. The rithmic flash, the rate of flashing and the amount of time form part of the signal system that brings both sexes together. We know that the light intensity at a particular distance from the light source obeys the inverse square law. That is to say, the light intensity decreases as the distance increases. Furthermore, the air absorbs light which becomes weaker and weaker as the distance increases. This physical factors are utilised by fireflies to find each other. The same principle can be utilised to create an artificial firefly for optimisation .The Firefly Algorithm was developed by Xin-She Yang [52], based on the idealization of the flashing characteristics of fireflies. In creating artificial firefly it is assumed that they are unisex, i.e. any firefly in the population can be attracted to any other firefly, Attractiveness is function of brightness. Brightness will decrease as a function of distance between. The less brighter one will move through the brighter one. If there are no brighter fire flies around, it will fly randomly. The brightness of the artificial firefly will directly be determined by objective function value. By using these principles, a pseudocode of the firefly algorithm can be written as:

Objective function $f(X)$, $X = (x_0, ...,x_n)$
Generate initial population of fireflies X^i $(i = 1,2, ...,m)$ *randomly*
Light intensity Ii at x≫ is determined by $f(X^i)$
Define light absorption coefficient γ
while $(t$ <MaxGeneration)
for $i = 1 : n$ all n fireflies
    for $j = 1 : n$ all n fireflies (inner loop)
        if $(Ii < Ij)$, Move firefly i towards j; end i f
        Vary attractiveness with distance r as
        $\beta = \beta_0 e^{-\gamma d^2}$ or $\beta = \frac{\beta_0}{1+\gamma d^2}$
        $d_{ij} = \sqrt{\sum_{k=1}^{n}(x_{i,k} - x_{i,k})^2}$
        Evaluate new solutions and update light intensity
        $X^i = (1-\beta)X^i + \beta X^j + \alpha(gaussrandom() - \frac{1}{2})$
    end for j
end for i
Rank the fireflies and find the current global best *gt*
end while
Postprocess results and visualization

A java program is developed for the above firefly algorithm . Program is extended also to use Nelder-Mead method so that the result can be tested by using Nelder_Mead

Program 5.22.1 FireFly optimisation firefly.java

```
 import java.io.*;
import java.text.*;
import java.util.Locale;

public class fireflyi
{
double gamma,alpha,beta;
double alpha0,alphan;
int m;
int n;
// m number of fireflies
// n number of variables for each fireflyi
// I0 light intensity at source
// gamma absorbtion coefficient
// alpha size of the random step
// beta attractiveness
double x[][];
double I0[];
double ymin;
double xmin[];
int imin;
int ngeneration;
```

```
public fireflyi(if_xj ff,int mi,int ni, double xxmin[],double xxmax[],double gammai,double alpha0i,double alphani,int ngenerationi)
{
// initial population set
n=ni;
m=mi;
x=new double[m+1][n];
I0=new double[m+1];
xmin=new double[n];
gamma=gammai;
alpha0=alpha0i;
alphan=alphani;
ngeneration=ngenerationi;
for(int j=0;j<n;j++)
  {xmin[j]=xxmin[j]+Math.random()*(xxmax[j]-xxmin[j]);}
ymin=ff.func(xmin);
  //end of init
int l=0;
double d=0;
double t=0;
while(l<ngeneration)
{
// find minimum of population
for(int i=0;i<m;i++)
{for(int j=0;j<n;j++)
  {x[i][j]=xxmin[j]+Math.random()*(xxmax[j]-xxmin[j]);}
  I0[i]=ff.func(x[i]);
  if(I0[i]<ymin)
    {ymin=I0[i];imin=i;
    for(int k=0;k<n;k++)
    {xmin[k]=x[i][k];}
    };
}
for(int j=0;j<n;j++)
  {x[m][j]=xmin[j];}
  I0[m]=ymin;
l++;
double fi=0,fj=0;
for(int i=0;i<=m;i++)
{ //fi=ff.func(x[i]);
  for(int j=0;j<n;j++)
  {
    //fj=ff.func(x[j]);
    if(I0[i]<I0[j])
    {
    beta=attractiveness(i,j);
    for(int k=0;k<n;k++)
    {   //The best one will remain the rest will change
                  if(i!=imin)
       { x[i][k]=(1.0-beta)*x[i][k]+beta*x[j][k]+alpha*(Math.random()-0.5);} }
    }
    else
    {
                  for(int k=0;k<n;k++)
      {  if(i!=imin)
                    { x[i][k]=x[i][k]+alpha*(Math.random()-0.5); }
                  }
    }
  }

}

    //alpha will change by time
    //(shrink to a smaller value)
    alpha=alphan+(alpha0-alphan)*Math.exp(-t);
    t=0.1*l;
}
}

public double distance(int i, int j)
{
double d=0;
for(int k=0;k<n;k++)
{d+=(x[i][k]-x[j][k])*(x[i][k]-x[j][k]);}
d=Math.sqrt(d);
```

```
return d;
}

public double attractiveness(int i,int j)
{   double d=distance(i,j);
                return I0[i]/(1+gamma*d*d);
                //return I0[i]*Math.exp(-gamma*d*d);
}

public String toString()
{ String s="ymin = "+ymin+"\nxmin = ";
  for(int k=0;k<n;k++)
  {s+=xmin[k]+" ";}
  return s;
}
public String toString(if_xj ff,double xx[])
{ String s="ymin = "+ff.func(xx)+"\nxmin = ";
  for(int k=0;k<n;k++)
  {s+=xx[k]+" ";}
  return s;
}
public double[] xmin()
{
  return xmin;
}

public static double[] nelder(if_xj fnelder,double a[],double da[],int maxiteration,double tolerance,int printlist)
{
                int i,j;
double x[][]=new double[a.length+1][a.length];
double p[][]=new double[a.length+1][a.length+1];
for(i=0;i<x.length;i++)
   {for(j=0;j<x[0].length;j++)
     {if(i==j){x[i][j]=a[j]+da[j];p[i][j]=x[i][j];}
      else    {x[i][j]=a[j];p[i][j]=x[i][j]; }
     }
     p[i][j] = fnelder.func(p[i]);
   }


   // Inlet variable definitions
   // fnelder : abstract multivariable function f(x)
   // x : independent variable set of n+1 simplex elements
   // maxiteration : maximum iteration number
   // tolerance :
        int NDIMS = x.length-1;
        int NPTS = x.length;
        int FUNC = NDIMS;
        int ncalls = 0;
          ////// construct the starting simplex //////////////////
          //double p[][]=new double[NPTS][NPTS]; // [row][col] = [whichvx][coord,FUNC]
          double z[]=new double[NDIMS];
          double best = 1E99;
          ///////////////// calculate the first function values for the simplex /////////////////
          int iter=0;
          for (iter=1; iter<maxiteration; iter++)
          {
                        ///////////  define lo, nhi, hi (low high next_to_high //////////////
             int  ilo=0, ihi=0, inhi = -1; // -1 means missing
             double flo = p[0][FUNC];
             double fhi = flo;
             double pavg,sterr;
             for (i=1; i<NPTS; i++)
             {
               if (p[i][FUNC] < flo)
                 {flo=p[i][FUNC]; ilo=i;}
               if (p[i][FUNC] > fhi)
                 {fhi=p[i][FUNC]; ihi=i;}
             }
             double fnhi = flo;
             inhi = ilo;
             for (i=0; i<NPTS; i++)
               if ((i != ihi) && (p[i][FUNC] > fnhi))
                 {fnhi=p[i][FUNC]; inhi=i;}
```

```
////////// exit criteria //////////////
if ((iter % 4*NDIMS) == 0)
{
   // calculate the avarage (including maximum value)
   pavg=0;
   for(i=0;i<NPTS;i++)
     pavg+=p[i][FUNC];
   pavg/=NPTS;
   double tot=0;
   if(printlist!=0)
   {  System.out.print(iter);
      for (j=0; j<=NDIMS; j++)
         { System.out.print(p[ilo][j]+" ");}
      System.out.println("");
   }
   for(i=0;i<NPTS;i++)
   { tot=(p[i][FUNC]-pavg)*(p[i][FUNC]-pavg);}
   sterr=Math.sqrt(tot/NPTS);
   //if(sterr < tolerance)
     { for (j=0; j<NDIMS; j++)
          { z[j]=p[ilo][j];}
       //break;
     }
   best = p[ilo][FUNC];
}

///// calculate avarage without maximum value //////

double ave[] = new double[NDIMS];
for (j=0; j<NDIMS; j++)
  ave[j] = 0;
for (i=0; i<NPTS; i++)
  if (i != ihi)
    for (j=0; j<NDIMS; j++)
      ave[j] += p[i][j];
for (j=0; j<NDIMS; j++)
   ave[j] /= (NPTS-1);


///////// reflect ////////////////

double r[] = new double[NDIMS];
for (j=0; j<NDIMS; j++)
  r[j] = 2*ave[j] - p[ihi][j];
double fr = fnelder.func(r);

if ((flo <= fr) && (fr < fnhi))  // in zone: accept
{
   for (j=0; j<NDIMS; j++)
     p[ihi][j] = r[j];
   p[ihi][FUNC] = fr;
   continue;
}

if (fr < flo)  //// expand
{
   double e[] = new double[NDIMS];
   for (j=0; j<NDIMS; j++)
     e[j] = 3*ave[j] - 2*p[ihi][j];
   double fe = fnelder.func(e);
   if (fe < fr)
   {
      for (j=0; j<NDIMS; j++)
        p[ihi][j] = e[j];
      p[ihi][FUNC] = fe;
      continue;
   }
   else
   {
      for (j=0; j<NDIMS; j++)
        p[ihi][j] = r[j];
      p[ihi][FUNC] = fr;
      continue;
   }
```

```
            }

            ////////////// shrink:

            if (fr < fhi)
            {
               double c[] = new double[NDIMS];
               for (j=0; j<NDIMS; j++)
                 c[j] = 1.5*ave[j] - 0.5*p[ihi][j];
               double fc = fnelder.func(c);
               if (fc <= fr)
               {
                  for (j=0; j<NDIMS; j++)
                    p[ihi][j] = c[j];
                  p[ihi][FUNC] = fc;
                  continue;
               }
               else   /////// daralt
               {
                  for (i=0; i<NPTS; i++)
                    if (i != ilo)
                    {
                       for (j=0; j<NDIMS; j++)
                         p[i][j] = 0.5*p[ilo][j] + 0.5*p[i][j];
                       p[i][FUNC] = fnelder.func(p[i]);
                    }
                  continue;
               }
            }

            if (fr >= fhi)   ///
            {
               double cc[] = new double[NDIMS];
               for (j=0; j<NDIMS; j++)
                 cc[j] = 0.5*ave[j] + 0.5*p[ihi][j];
               double fcc = fnelder.func(cc);
               if (fcc < fhi)
               {
                  for (j=0; j<NDIMS; j++)
                    p[ihi][j] = cc[j];
                  p[ihi][FUNC] = fcc;
                  continue;
               }
               else    /////////
               {
                  for (i=0; i<NPTS; i++)
                    if (i != ilo)
                    {
                       for (j=0; j<NDIMS; j++)
                         p[i][j] = 0.5*p[ilo][j] + 0.5*p[i][j];
                       p[i][FUNC] = fnelder.func(p[i]);
                    }
               }
            }
         }
      return z;
}
    public static double[] nelder(if_xj fnelder,double a[],double da[],double tolerance)
    {return nelder(fnelder,a,da,500,tolerance,0);}

    public static double[] nelder(if_xj fnelder,double a[],double da[])
    {return nelder(fnelder,a,da,500,1.0e-10,0);}

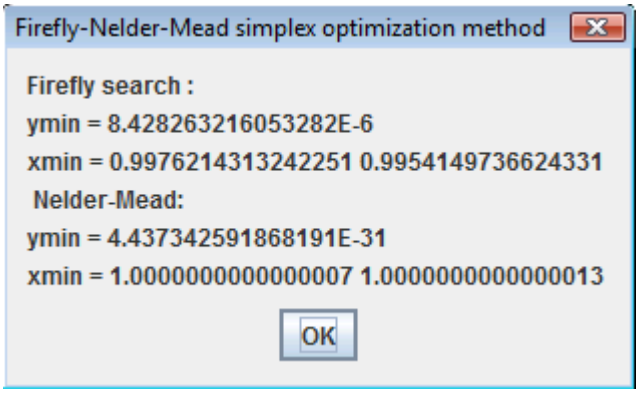
    public static double[] nelder(if_xj fnelder,double a[])
    {
                    double [] da=new double[a.length];
          for(int i=0;i<a.length;i++) da[i]=0.1*a[i];
          return nelder(fnelder,a,da);
    }
}
```

Program 5.22.2 FireFly optimisation Example test case fireflytest.java

```
import java.util.*;
```

```
import java.awt.*;
import java.applet.Applet;
import java.awt.event.*;
import javax.swing.*;

public class fireflytest
{
public static void main(String arg[])
{
if_xj ff=(double x[])->(1.0-x[0])*(1.0-x[0])+100*(x[1]-x[0]*x[0])*(x[1]-x[0]*x[0]);
double xmin[]={0,0};
double xmax[]={2.0,2.0};
double gamma=1.0;
double alpha0=0.5;
double alphan=0.01;
fireflyi  ffly=new fireflyi(ff,500,2,xmin,xmax,gamma,alpha0,alphan,1000);
double x0[]=ffly.xmin();
double r1[]= fireflyi.nelder(ff,x0);
String s="Firefly search : \n"+ffly.toString()+"\n Nelder-Mead: \n"+ffly.toString(ff,r1);
    String s2="Firefly-Nelder-Mead simplex optimization method ";
    JOptionPane.showMessageDialog(null,s,s2,JOptionPane.PLAIN_MESSAGE);
}
}
```



As a second example case Michaelewicz function with n=2 and m=10 is taken

$$f(x) = -\sum_{i=0}^{n}\left[sin(x_i)\left[sin\left(\frac{(i+1)x_i^2}{\pi}\right)\right]^{2m}\right]$$

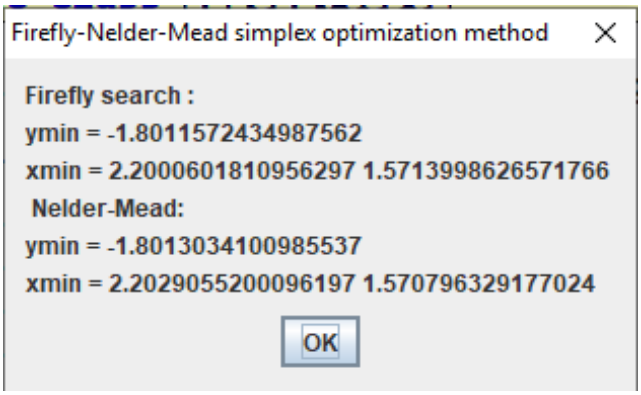
Program 5.22.3 FireFly optimisation Example test case fireflytest.java

```
 import java.util.*;
import java.awt.*;
import java.applet.Applet;
import java.awt.event.*;
import javax.swing.*;

class fa implements if_xj
{
int n;
public double func(double x[])
{
//çözümü istenen fonksiyon
double ff=0;
int n=x.length;
for(int i=0;i<n;i++)
{ff+=-Math.sin(x[i])*Math.pow(Math.sin((i+1)*x[i]*x[i]/Math.PI),20);}
return ff; //maksimum testi
}
}

public class fireflytest1
{
public static void main(String arg[])
{
```

```
fa ff=new fa();
double xmin[]={0,0};
double xmax[]={2.5,2.5};
double gamma=1.0;
double alpha0=0.5;
double alphan=0.01;
fireflyi  ffly=new fireflyi(ff,500,2,xmin,xmax,gamma,alpha0,alphan,1000);
double x0[]=ffly.xmin();
double r1[]= fireflyi.nelder(ff,x0);
String s="Firefly search : \n"+ffly.toString()+"\n Nelder-Mead: \n"+ffly.toString(ff,r1);
    String s2="Firefly-Nelder-Mead simplex optimization method ";
    JOptionPane.showMessageDialog(null,s,s2,JOptionPane.PLAIN_MESSAGE);
}
}
```

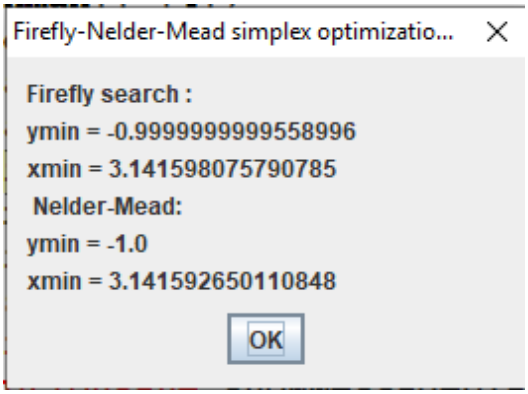


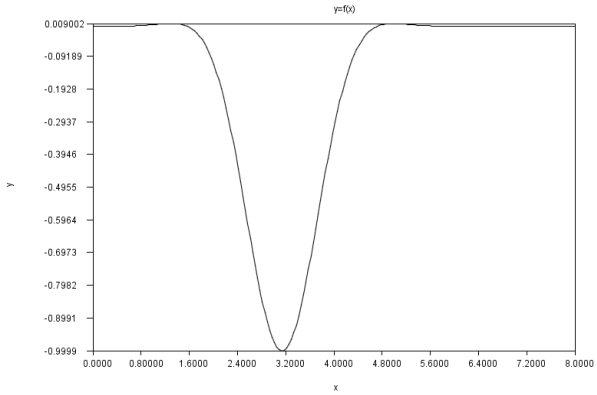
Another interesting test function is Easom's function: $f(x) = cos(x)e^{-(x-\pi)^2}$ which has a solution x=π, y=f(x)=1

Program 5.22.4 FireFly optimisation Example test case fireflytest.java

```
 import java.util.*;
import java.awt.*;
import java.applet.Applet;
import java.awt.event.*;
import javax.swing.*;

public class fireflytest
{
public static void main(String arg[])
{
if_xj ff=(double x[])->Math.cos(x[0])*Math.exp(-(x[0]-Math.PI)*(x[0]-Math.PI));
double xmin[]={0};
double xmax[]={4};
double gamma=1.0;
double alpha0=0.5;
double alphan=0.01;
fireflyi  ffly=new fireflyi(ff,500,1,xmin,xmax,gamma,alpha0,alphan,1000);
double x0[]=ffly.xmin();
double r1[]= fireflyi.nelder(ff,x0);
String s="Firefly search : \n"+ffly.toString()+"\n Nelder-Mead: \n"+ffly.toString(ff,r1);
    String s2="Firefly-Nelder-Mead simplex optimization method ";
    JOptionPane.showMessageDialog(null,s,s2,JOptionPane.PLAIN_MESSAGE);
}
}
```





5.23 STOCHASTIC METHODS: PARTICLE SWARM ALGORITHM

Particle swarm optimization (PSO) was developed by Kennedy and Eberhart in 1995, based on swarm behaviour such as fish and bird schooling in nature. PSO searches the space of an objective function by adjusting the trajectories of individual agents, called particles. The movement of the particles has both stochastic and deterministic elements. Each particle is attracted toward the best location of the given time and the best location of the near past (a current global). It has also tendency to move randomly. When a particle finds a location that is better than any previously found locations, then it updates it as the new current best for particle *i*.It also remembers all the currents best values to update a current global value. The aim is to find the global best among all the current best values available until no more improvements are available.

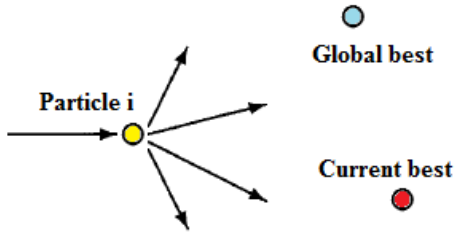


Figure 5.23.1 Particle will attract toward the global best, current best and also move randomly

The method is given in Table 5.23.1 as pseodocode. If x_i and v_i position vector and velocity for particle i, the new velocity vector is determined as:

$v_i^{t+1} = v_i^t + \alpha^* \varepsilon_1^* (g^* - x_i^t) + \beta^* \varepsilon_2^* (x_i^* - x_i^t)$

Where ε_1 and ε_1 are two random vectors between 0 and 1, g^* is the current global best x_i^* is local best. The parameters α and β are the learning parameters or acceleration constants. The initial velocity of the particles can be taken as zero, $v_i^{t=0} = 0$. The new position can then be updated by $x_i^{t+1} = x_i^t + v_i^{t+1}$. Although v_i can be any values, it is bounded in some range $[0, v_{max}]$.

There are many variants of the standart PSO algorithm. The most noticiable one is the version in which previous velocity function multiplied with an inertia function. This is called an accelerated PSO.

$v_i^{t+1} = \theta v_i^t + \alpha^* \varepsilon_1^* (g^* - x_i^t) + \beta^* \varepsilon_2^* (x_i^* - x_i^t)$

This is equivalent to introduce a virtual mass to stabilise the motion of the particles.

Table 5.23.1 Particle swarm algorithm

Objective function $f(P_i)$, $P_i = (x_0, ..., x_m)_i$ for particle i and m space dimension variable
Initialize locations P_i and velocity V_i of n particles by random selection Use as velocity vector: $V_i^{t+1} = \theta V_i^t + \alpha^* random()[GPmin - x_i^t] + \beta^* random()[Pmin - x_i^t]$

Where θ is inertia function(an iterative variable between 0 and 1)
α and β are learning parameters or acceleration constants
$\alpha = \alpha_0 \gamma^t$ where $0 \leq \gamma \leq 1$ $\alpha_0 \cong 0.5..1$ for t is the virtual time
Find Global best GPmin from min{ $f(P_0)\ldots f(P_n)$} (at t = 0)

while (criterion)

t = t + 1 (pseudo time or iteration counter)
for loop over all n particles and all m dimensions
Generate new velocity V_i^{t+1}
Calculate new locations $P_i^{t+1} = P_i^t + V_i^{t+1}$
Evaluate objective functions at new locations
Find the current best for each particle Pmin
Find the global best for each particle GPmin
end for
end while
Output the final results Pmin and GPmin

An example java code is developed. The java code and an example problem is given below:

Program 5.23.1 Particle swarm algorithm: particle_swarm1.java

```
 import java.io.*;
import java.text.*;
import java.util.Locale;

public class particle_swarm1i
{
int m;
int n;
double alpha,beta,beta0,teta,gamma;

// m number of population
// n number of variables for each individual
double x[][];  //location for each population member
double y[];    //function value of each population member
double yxmin;  //current best minimum
double ygmin;  //global best minimum
double xmin[]; //current best x
double gmin[]; //global best x
double v[][];  //velocity vector
int ngeneration;  //number of generations in iteration
if_xj ff;

public particle_swarm1i(if_xj ffi,int mi, double xxmin[],double xxmax[],double alphai,double betai,int ngenerationi)
{
teta=0.5;
gamma=0.5;
ff=ffi;
//initial population set
n=xxmin.length;
m=mi;
alpha=alphai;
beta0=betai;
beta=beta0;
ngeneration=ngenerationi;
x=new double[m][n];
v=new double[m][n];
y=new double[m];
xmin=new double[n];
gmin=new double[n];

ngeneration=ngenerationi;
yxmin=1.0e50;
for(int i=0;i<m;i++)
{for(int j=0;j<n;j++)
  {x[i][j]=xxmin[j]+Math.random()*(xxmax[j]-xxmin[j]);
   v[i][j]=Math.random();
```

```
  }

  y[i]=ff.func(x[i]);
  if(y[i]<yxmin)
    {yxmin=y[i];
     ygmin=y[i];
    for(int k=0;k<n;k++)
    {xmin[k]=x[i][k];
     gmin[k]=x[i][k];
    } //end for(int k=0;k<n;k++)
    }//end if(y[i]<yxmin)
}
//end of init
int t=0;

while(t<ngeneration)
{
yxmin=1.0e50;
for(int i=0;i<m;i++)
{  for(int j=0;j<n;j++)
   {x[i][j]=x[i][j]+v[i][j];
   if(x[i][j]<xxmin[j]) x[i][j]=xxmin[j];
   if(x[i][j]>xxmax[j]) x[i][j]=xxmax[j];
   }
   y[i]=ff.func(x[i]);
   if(y[i]<yxmin)
   {yxmin=y[i];
    for(int k=0;k<n;k++)
    {xmin[k]=x[i][k];}
   }
}
for(int i=0;i<m;i++)
{
   if(y[i]<ygmin)
   {ygmin=y[i];
   for(int k=0;k<n;k++)
   {gmin[k]=x[i][k];}
   }
}
for(int i=0;i<m;i++)
{
   for(int j=0;j<n;j++)
   {v[i][j]=teta*v[i][j]+alpha*Math.random()*(gmin[j]-xmin[j])+beta*Math.random()*(xmin[j]-x[i][j]);}
}//end for int i=..
t++;
beta=beta0*Math.pow(gamma,0.01*t);
}

}


public String toString()
{ String s="ymin = "+ff.func(xmin)+"\nxmin = ";
  for(int k=0;k<n;k++)
  {s+=xmin[k]+" ";}
  s+="\ny global min = "+ff.func(gmin)+"\nxmin = ";
  for(int k=0;k<n;k++)
  {s+=gmin[k]+" ";}
  return s;
}

public String toString(double x[])
{ String s="ymin = "+ff.func(x)+"\nxmin = ";
  for(int k=0;k<n;k++)
  {s+=x[k]+" ";}
  return s;
}

public double[] xmin()
{
  return xmin;
}
public double[] gmin()
{
```

```
  return gmin;
}


public static double[] nelder(if_xj fnelder,double a[],double da[],int maxiteration,double tolerance,int printlist)
{
                int i,j;
double x[][]=new double[a.length+1][a.length];
double p[][]=new double[a.length+1][a.length+1];
for(i=0;i<x.length;i++)
  {for(j=0;j<x[0].length;j++)
    {if(i==j){x[i][j]=a[j]+da[j];p[i][j]=x[i][j];}
     else    {x[i][j]=a[j];p[i][j]=x[i][j]; }
    }
    p[i][j] = fnelder.func(p[i]);
  }


  // Inlet variable definitions
  // fnelder : abstract multivariable function f(x)
  // x : independent variable set of n+1 simplex elements
  // maxiteration : maximum iteration number
  // tolerance :
       int NDIMS = x.length-1;
       int NPTS = x.length;
       int FUNC = NDIMS;
       int ncalls = 0;
         ////// construct the starting simplex ///////////////////
         //double p[][]=new double[NPTS][NPTS]; // [row][col] = [whichvx][coord,FUNC]
         double z[]=new double[NDIMS];
         double best = 1E99;
         ///////////////// calculate the first function values for the simplex /////////////////

         int iter=0;

         for (iter=1; iter<maxiteration; iter++)
         {
                    ////////////  define lo, nhi, hi (low high next_to_high //////////////
           int  ilo=0, ihi=0, inhi = -1; // -1 means missing
           double flo = p[0][FUNC];
           double fhi = flo;
           double pavg,sterr;
           for (i=1; i<NPTS; i++)
           {
             if (p[i][FUNC] < flo)
               {flo=p[i][FUNC]; ilo=i;}
             if (p[i][FUNC] > fhi)
               {fhi=p[i][FUNC]; ihi=i;}
           }
           double fnhi = flo;
           inhi = ilo;
           for (i=0; i<NPTS; i++)
            if ((i != ihi) && (p[i][FUNC] > fnhi))
              {fnhi=p[i][FUNC]; inhi=i;}
           ////////// exit criteria //////////////
           if ((iter % 4*NDIMS) == 0)
           {
              // calculate the avarage (including maximum value)
              pavg=0;
              for(i=0;i<NPTS;i++)
                pavg+=p[i][FUNC];
              pavg/=NPTS;
              double tot=0;
              if(printlist!=0)
              {  System.out.print(iter);
                 for (j=0; j<=NDIMS; j++)
                   { System.out.print(p[ilo][j]+" ");}
                 System.out.println("");
              }
              for(i=0;i<NPTS;i++)
              { tot=(p[i][FUNC]-pavg)*(p[i][FUNC]-pavg);}
              sterr=Math.sqrt(tot/NPTS);
              //if(sterr < tolerance)
                { for (j=0; j<NDIMS; j++)
```

```
            { z[j]=p[ilo][j];}
          //break;
        }
    best = p[ilo][FUNC];
}

///// calculate avarage without maximum value //////

double ave[] = new double[NDIMS];
for (j=0; j<NDIMS; j++)
  ave[j] = 0;
for (i=0; i<NPTS; i++)
  if (i != ihi)
    for (j=0; j<NDIMS; j++)
      ave[j] += p[i][j];
for (j=0; j<NDIMS; j++)
   ave[j] /= (NPTS-1);


///////// reflect ////////////////

double r[] = new double[NDIMS];
for (j=0; j<NDIMS; j++)
  r[j] = 2*ave[j] - p[ihi][j];
double fr = fnelder.func(r);

if ((flo <= fr) && (fr < fnhi))  // in zone: accept
{
    for (j=0; j<NDIMS; j++)
      p[ihi][j] = r[j];
    p[ihi][FUNC] = fr;
    continue;
}

if (fr < flo)  //// expand
{
    double e[] = new double[NDIMS];
    for (j=0; j<NDIMS; j++)
      e[j] = 3*ave[j] - 2*p[ihi][j];
    double fe = fnelder.func(e);
    if (fe < fr)
    {
       for (j=0; j<NDIMS; j++)
         p[ihi][j] = e[j];
       p[ihi][FUNC] = fe;
       continue;
    }
    else
    {
       for (j=0; j<NDIMS; j++)
         p[ihi][j] = r[j];
       p[ihi][FUNC] = fr;
       continue;
    }
}

////////////// shrink:

if (fr < fhi)
{
    double c[] = new double[NDIMS];
    for (j=0; j<NDIMS; j++)
      c[j] = 1.5*ave[j] - 0.5*p[ihi][j];
    double fc = fnelder.func(c);
    if (fc <= fr)
    {
        for (j=0; j<NDIMS; j++)
          p[ihi][j] = c[j];
        p[ihi][FUNC] = fc;
        continue;
    }
    else   /////// daralt
    {
        for (i=0; i<NPTS; i++)
```

```
                    if (i != ilo)
                    {
                       for (j=0; j<NDIMS; j++)
                         p[i][j] = 0.5*p[ilo][j] + 0.5*p[i][j];
                       p[i][FUNC] = fnelder.func(p[i]);
                    }
                  continue;
               }
            }

            if (fr >= fhi)   ///
            {
               double cc[] = new double[NDIMS];
               for (j=0; j<NDIMS; j++)
                 cc[j] = 0.5*ave[j] + 0.5*p[ihi][j];
               double fcc = fnelder.func(cc);
               if (fcc < fhi)
               {
                  for (j=0; j<NDIMS; j++)
                    p[ihi][j] = cc[j];
                  p[ihi][FUNC] = fcc;
                  continue;
               }
               else    /////////
               {
                  for (i=0; i<NPTS; i++)
                    if (i != ilo)
                    {
                       for (j=0; j<NDIMS; j++)
                         p[i][j] = 0.5*p[ilo][j] + 0.5*p[i][j];
                       p[i][FUNC] = fnelder.func(p[i]);
                    }
               }
            }
         }
      return z;
}
    public static double[] nelder(if_xj fnelder,double a[],double da[],double tolerance)
    {return nelder(fnelder,a,da,500,tolerance,0);}

    public static double[] nelder(if_xj fnelder,double a[],double da[])
    {return nelder(fnelder,a,da,500,1.0e-10,0);}

    public static double[] nelder(if_xj fnelder,double a[])
    {
                    double [] da=new double[a.length];
         for(int i=0;i<a.length;i++) da[i]=0.1*a[i];
         return nelder(fnelder,a,da);
    }
}
```

Program 5.23.2 Particle swarm test: particleswarmtest1.java

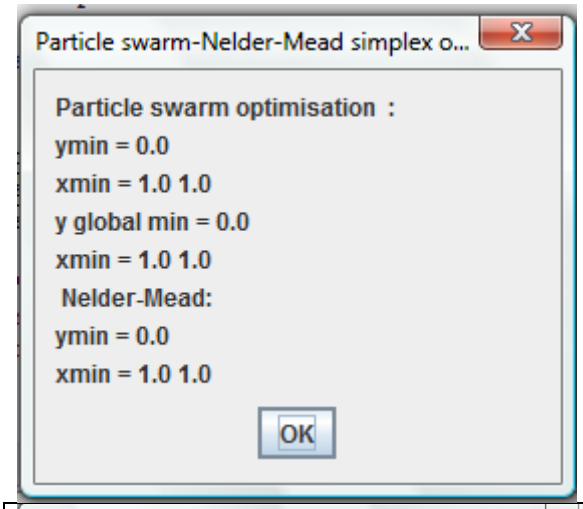
```
 import java.util.*;
import java.awt.*;
import java.applet.Applet;
import java.awt.event.*;
import javax.swing.*;

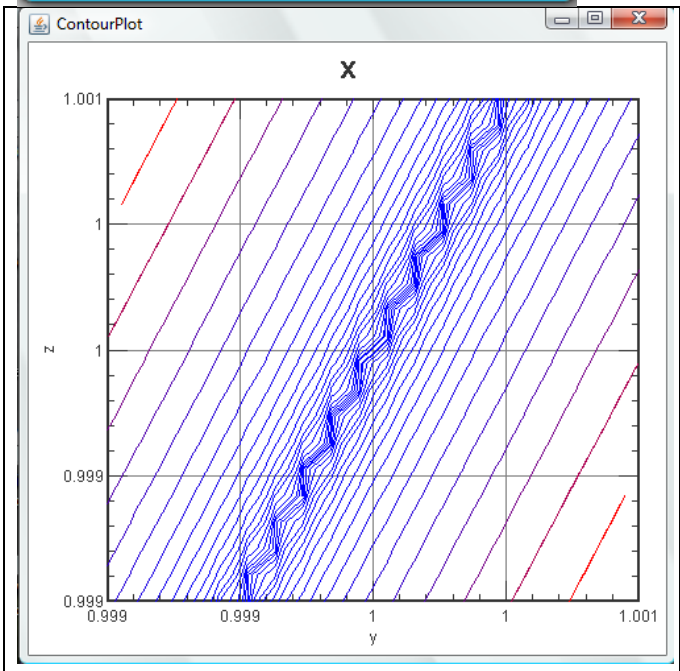
class fa implements if_xj
{
int n;
public double func(double x[])
{
double ff=(1.0-x[0])*(1.0-x[0])+100.0*(x[1]-x[0]*x[0])*(x[1]-x[0]*x[0]);
return ff; //maksimum testi
}}

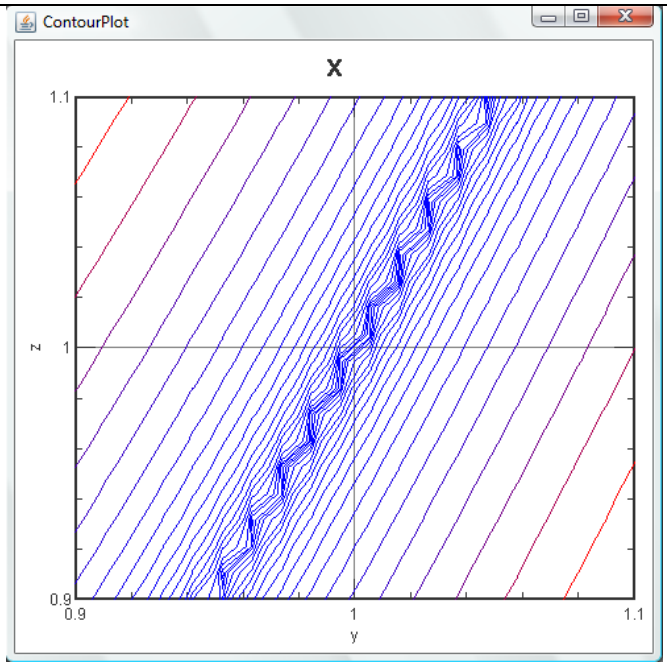
public class particleswarmtest1
{
public static void main(String arg[])
{
fa ff=new fa();
double xmin[]={0,0};
```

```
double xmax[]={2.5,2.5};
particle_swarm1i ps=new particle_swarm1i(ff,500,xmin,xmax,2.0,0.7,500);
double r1[]= ps.nelder(ff,ps.gmin());
String s="Particle swarm optimisation  : \n"+ps.toString()+"\n Nelder-Mead: \n"+ps.toString(r1);
    String s2="Particle swarm-Nelder-Mead simplex optimization method ";
    JOptionPane.showMessageDialog(null,s,s2,JOptionPane.PLAIN_MESSAGE);

}
}
```







Another example function:

$$f(x) = \left(\sum_{i=0}^{d} |x_i|\right) exp\left(-\sum_{i=0}^{d} sin(x_i^2)\right) - 2\pi \leq x_i \leq 2\pi$$

This function has a global minimum at (0,0) and the function value is also 0. It has a singularity at (0,0)

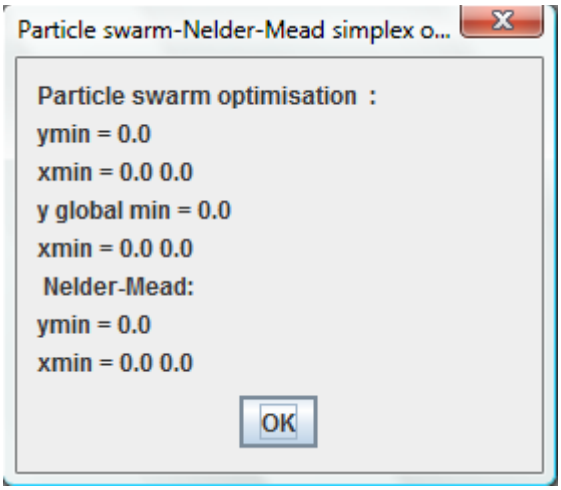
Program 5.23.3 Particle swarm test: particleswarmtest1.java

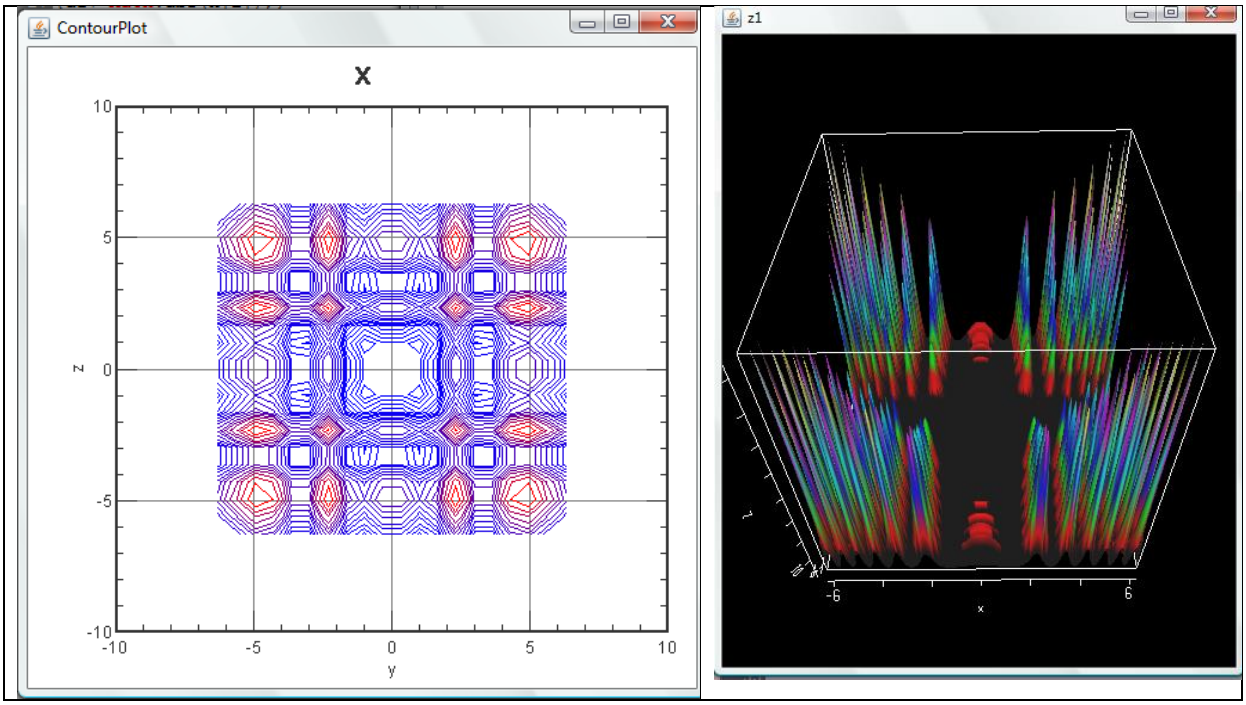
```
import java.util.*;
```

```
import java.awt.*;
import java.applet.Applet;
import java.awt.event.*;
import javax.swing.*;

class fa implements if_xj
{
int n;
public double func(double x[])
{
//çözümü istenen fonksiyon
double ff=(1.0-x[0])*(1.0-x[0])+100.0*(x[1]-x[0]*x[0])*(x[1]-x[0]*x[0]);
return ff; //maksimum testi
}}


public class particleswarmtest1
{
public static void main(String arg[])
{
fa ff=new fa();
double xmin[]={0,0};
double xmax[]={2.5,2.5};
particle_swarm1i ps=new particle_swarm1i(ff,500,xmin,xmax,2.0,0.7,500);
double r1[]= ps.nelder(ff,ps.gmin());
String s="Particle swarm optimisation  : \n"+ps.toString()+"\n Nelder-Mead: \n"+ps.toString(r1);
    String s2="Particle swarm-Nelder-Mead simplex optimization method ";
    JOptionPane.showMessageDialog(null,s,s2,JOptionPane.PLAIN_MESSAGE);
}}
```





As another test function Michaelewicz function is given

$$f(x) = -\sum_{i=0}^{n}\left[sin(x_i)\left[sin\left(\frac{(i+1)x_i^2}{\pi}\right)\right]^{2m}\right]$$

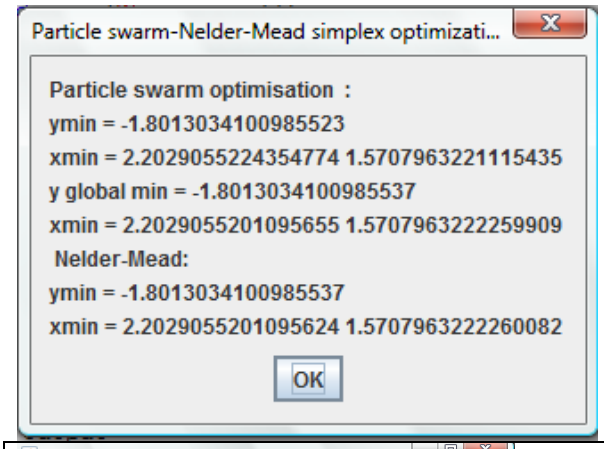
The minimum of the function f = -1.801 is located at (2.2029,1.5704796)

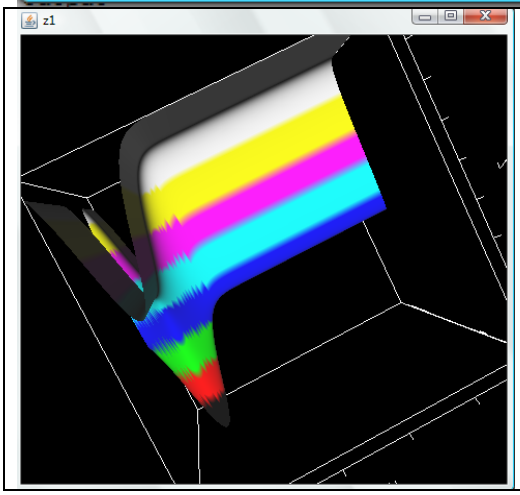
Program 5.23.4 Particle swarm test: particleswarmtest.java

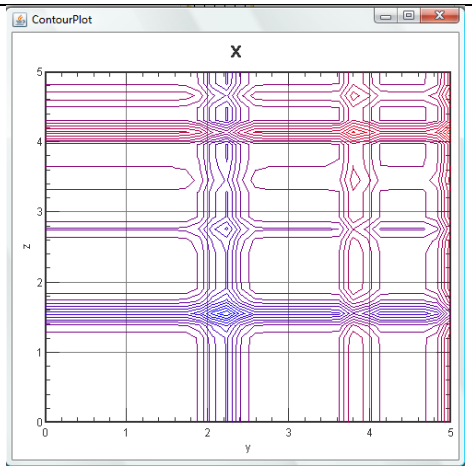
```
import java.util.*;
import java.awt.*;
import java.applet.Applet;
import java.awt.event.*;
import javax.swing.*;

class fa extends f_xj
{int n;
public double func(double x[])
{double ff=0;
int n=x.length;
for(int i=0;i<n;i++)
{ff+=-Math.sin(x[i])*Math.pow(Math.sin((i+1)*x[i]*x[i]/Math.PI),20);}
return ff; //maksimum testi
}}

public class particleswarmtest1
{
public static void main(String arg[])
{
fa ff=new fa();
double xmin[]={0,0};
double xmax[]={2.5,2.5};
particle_swarm1 ps=new particle_swarm1(ff,500,xmin,xmax,2.0,0.7,500);
double r1[]= ps.nelder(ff,ps.gmin());
String s="Particle swarm optimisation  : \n"+ps.toString()+"\n Nelder-Mead: \n"+ps.toString(r1);
    String s2="Particle swarm-Nelder-Mead simplex optimization method ";
    JOptionPane.showMessageDialog(null,s,s2,JOptionPane.PLAIN_MESSAGE);
}
}
```







Python version of Particle swarm algorithm

Program 5.23.5 Particle swarm algorith python version

```
 # -*- coding: utf-8 -*-
"""
Created on Tue Sep  4 11:40:02 2018

@author: Mustafa Turhan Çoban
"""
from gauss import *;
from math import *;
from f_xj import *;
import numpy as np;
import matplotlib.pyplot as plt;
from mpl_toolkits.mplot3d import Axes3D
from random import *;

class f1(f_xj):
  def func(self,x):
      y=(1.0-x[0])*(1.0-x[0])+100.0*(x[1]-x[0]*x[0])*(x[1]-x[0]*x[0]);
      return y;

class particle_swarm():
    def __init__(self,ffi,mi,xxmin,xxmax,alphai,betai,ngenerationi):
        self.teta=0.5647645;
        self.gamma=0.6564537;
        self.ff=ffi;
        self.alpha=alphai;
        self.beta0=betai;
        self.beta=betai;
        self.n=len(xxmin);
        self.m=mi;  #number of species
        self.ngeneration=ngenerationi; #number of generations
        self.x= [[0 for i in range(self.n)] for j in range(self.m)];
        self.v= [[0 for i in range(self.n)] for j in range(self.m)];
```

```
        self.y= [0 for j in range(self.m)];
        self.xmin=[0 for j in range(self.n)];
        self.gmin=[0 for j in range(self.n)];
        yxmin=1.0e50;
        for i in range(self.m):
            for j in range(self.n):
                self.x[i][j]=xxmin[j]+random()*(xxmax[j]-xxmin[j]);
                self.v[i][j]=random();
                #print("vi",self.v[i][j],"   i",i,"j",j)
            self.y[i]=self.ff.func(self.x[i]);
            if self.y[i]<yxmin:
                yxmin=self.y[i];
                ygmin=self.y[i];
                for k in range(self.n):
                    self.xmin[k]=self.x[i][k];
                    self.gmin[k]=self.x[i][k];
        t=0;
        while t<self.ngeneration:
            yxmin=1.0e50;
            for i in range(self.m):
                for j in range(self.n):
                    self.x[i][j]=self.x[i][j]+self.v[i][j];
                    if self.x[i][j]<xxmin[j]: self.x[i][j]=xxmin[j];
                    if self.x[i][j]>xxmax[j]: self.x[i][j]=xxmax[j];
                self.y[i]=self.ff.func(self.x[i]);
                if self.y[i]<yxmin:
                    yxmin=self.y[i];
                    for k in range(self.n):
                        self.xmin[k]=self.x[i][k];
            for i in range(self.m):
                if self.y[i]<ygmin: ygmin=self.y[i];
                for k in range(self.n):
                    self.gmin[k]=self.x[i][k];
            for i in range(self.m):
                for j in range(self.n):
                    self.v[i][j]=self.teta*self.v[i][j]+self.alpha*random()*(self.gmin[j]-self.x[i][j])+self.beta*random()*(self.xmin[j]-self.x[i][j]);
            t=t+1;
            self.beta=self.beta0*pow(self.gamma,0.1*t);

f=f1();
xmin=[0.0,0.0];
xmax=[2.0,2.0];
r1= particle_swarm(f,500,xmin,xmax,0.511545,0.771354,1000);
z=r1.xmin;
print("xmin = ",z,"fmin=",f.func(z));
```

```
runfile('D:/okul/SCO1/particle_swarm.py', wdir='D:/okul/SCO1')
Reloaded modules: gauss, f_xj
xmin =  [0.9999967950335541, 0.9999936653609952] fmin= 1.0838572191578976e-11
```

5.24 STOCHASTIC METHODS: BAT SEARCH ALGORITHM

Xin-She Yang (2010) [51] propose the Bat Algorithm (BA), BA is inspired by the research on the social behavior of bats. The BA is based on the echolocation behaviour of bats. Microbats use a type of sonar (echolocation) to detect prey, avoid obstacles, and locate their roosting crevices in the dark. These bats emit a very loud sound pulse and listen for the echo that bounces back from the surrounding objects. Their pulses vary in properties and can be correlated with their hunting strategies, depending on the species. Based on the above description of bat process, Xin-She Yang proposes the Bat algorithm. The structure of the pseudo code of the Bat Algorithm is as follows :

Table 5.24.1 Bat Search algorithm

Objective function $f(x_1, x_2, \ldots, x_n)$
Initialize the bat population x_i (i = 1, 2, …, n) and v_i
Define pulse frequency f_i at x_i

Initialize pulse rates r_i and the loudness A_i
while (t < Max Number of iterations)
{
Generate new solutions by adjusting frequency
And updating velocities and locations/solutions:
$f_i = f_{min} + (f_{max} - f_{min})\beta$
$v_i^t = v_i^{t-1} + (x_i^t - x_*)f_i$
$x_i^t = x_i^{t-1} + v_i^t$
If (rand >*ri*)
{ Select a solution among the best solutions
Generate a local solution around the selected best solution
}
End if
Generate a new solution by flying randomly
If (rand < A_i and $f(x_i) < f(x_*)$)
{ Accept the new solutions
Increase r_i and reduce A_i
}
End if
Rank the bats and find the current best *x**
} **End while**
Postprocess results

Bats fly randomly with velocity *vi* at position *xi* with a fixed frequency f_{min}, varying wavelength λ and loudness *A0* to search the prey, they can automatically adjust the wavelength (or frequency) of their emitted pulses and adjust the rate of pulse emission $r \in [0,1]$, depending on the proximity of their target. Although the loudness can vary from a large positive *A*0 to a minimum constant value *Amin*. $\beta \in [0,1]$is a random vector drawn from a uniform random number generator. A program using above algorithm is developed. Code are listed below:

Program 5.24.1 bat search algorithm

```
import java.io.*;
import java.text.*;
import java.util.Locale;

//Program to find minimum of the function
//for maximum finding change the sign of the function

public class bati
{
int m,n;
// n number of bat
// m number of variables (space dimension) for each bat
double x[][]; //bat population
double y[]; // function value;
double f[];  //pulse frequency
double v[][]; //velocity
double ri[];  //rate of pulse emmission
public double ymin;
public double xmin[];
double A[];
double beta;
double fmin,fmax;//minimum and maximum velocities
double A0,Amin,r0;
double alpha,gamma;
double t;
int ngeneration;
int imin;
```

```
double xxmin[];
double xxmax[];

public bati(if_xj ff,int ni, double xxmini[],double xxmaxi[],int ngenerationi)
{
// initial population set
xxmin=xxmini;
xxmax=xxmaxi;
fmin=0;
fmax=2;
A0=0.5;
Amin=0;
r0=0.5;
alpha=0.9;
gamma=0.9;
n=ni;
m=xxmin.length;
x=new double[n][m];
v=new double[n][m];
y=new double[n];
ri=new double[n];
f=new double[n];
xmin=new double[m];
A=new double[n];
double eps;
double AA=0;
double xold;
ngeneration=ngenerationi;
// initilize bat population
for(int i=0;i<n;i++)
{ ri[i]=Math.random();
for(int j=0;j<m;j++)
{
x[i][j]=xxmin[j]+Math.random()*(xxmax[j]-xxmin[j]);
}
y[i]=ff.func(x[i]);
f[i]=fmin+(fmax-fmin)*Math.random();
checkmin(i);
}
//initilize bat velocities
for(int i=0;i<n;i++)
{for(int j=0;j<m;j++)
{
v[i][j]= v[i][j]+(x[i][j]-xmin[j])*(f[i]-fmin);
}
}
//initilize Ai and ri
for(int i=0;i<n;i++)
{ri[i]=r0*Math.random();
A[i]=A0;
}
int l;
t=0;l=0;
while(l<ngeneration)
{
for(int i=0;i<n;i++)
{for(int j=0;j<m;j++)
```

```
{xold=x[i][j];
x[i][j]= x[i][j]+v[i][j];
//if out of range
int ic1=0;
int ic2=0;
while(x[i][j]<xxmin[j]&&ic1<5)  {x[i][j]= xold+Math.random()*v[i][j];ic1++;}
while(x[i][j]>xxmax[j]&& ic2<5) {x[i][j]= xold-Math.random()*v[i][j];ic2++;}
if(x[i][j]<xxmin[j])  x[i][j]=xxmin[j]+(xxmax[j]-xxmin[j])*Math.random();
if(x[i][j]>xxmax[j])  x[i][j]=xxmin[j]+(xxmax[j]-xxmin[j])*Math.random();
}
y[i]=ff.func(x[i]);
f[i]=fmin+(fmax-fmin)*Math.random();
checkmin(i);
}
for(int i=0;i<n;i++)
{for(int j=0;j<m;j++)
{v[i][j]= v[i][j]+(x[i][j]-xmin[j])*f[i];}
}
for(int i=0;i<n;i++)
{ri[i]=r0*Math.random()*(1-Math.exp(-gamma*t));
A[i]=alpha*A[i];
}
//random walk
for(int i=0;i<n;i++)
{if(Math.random()>ri[i])
{x[i]=local_solution(xmin,xxmin,xxmax,0.05);
y[i]=ff.func(x[i]);
checkmin(i);
}
}

for(int i=0;i<n;i++)
{ if(Math.random()<A[i] && y[i]<ymin)
AA=avg(A);
for(int j=0;j<m;j++)
{eps=-1+2.0*Math.random();
x[i][j]=x[i][j]+eps*AA;}
y[i]=ff.func(x[i]);
checkmin(i);
ri[i]=r0*Math.random()*(1-Math.exp(-gamma*t));
A[i]=alpha*A[i];
}

for(int i=0;i<n;i++)
{  checkmin(i);}
l++;
t=l;
}//end of while

}

// if minimum is found replace ymin and xmin
// with the new founded value
public void checkmin(int i)
{
if(y[i]<ymin)
{ymin=y[i];imin=i;
```

```
for(int j=0;j<m;j++)
{xmin[j]=x[i][j];}
}
}

public static double avg(double xx[])
{ double toplam=0;
int nn=xx.length;
for(int i=0;i<nn;i++)
{toplam+=xx[i];}
toplam/=nn;
return toplam;
}

//Gauss random number distributed around xavg within a diameter of r
public static double  gaussrandom(double xavg,double r)
{ int iset=0;
double gset=0;
double fac,rsq,v1,v2;
double aret=0;
if(iset==0)
{
do{
        v1=2.0*Math.random()-1.0;
        v2=2.0*Math.random()-1.0;
        rsq=v1*v1+v2*v2;
}while(rsq>=1.0 || rsq==0.0);
fac=Math.sqrt(-2.0*Math.log(rsq)/rsq);
gset=v1*fac;
iset=1;
aret=v2*fac;
}
else
{
iset=0;
aret=gset;
}
return aret/8.0*r+xavg;
}

//Finds local solutions in the neighbourhood of the selected best soluton
// by using Gauss random number
public static double[] local_solution(double selected_best[],double xxmin[],double xxmax[],double ratio)
{
int m=xxmin.length;
double dx=1;
double xx[]=new double[m];
for(int j=0;j<m;j++)
{
dx=(xxmax[j]-xxmin[j])*ratio;
xx[j]=gaussrandom(selected_best[j],dx);
//we will not permit data out of range
int ic1=0;
int ic2=0;
while(xx[j]<xxmin[j]) {xx[j]=xxmin[j]+0.001*Math.random()*dx;ic1++;}
while(xx[j]>xxmax[j]) {xx[j]=xxmax[j]-0.001*Math.random()*dx;ic2++;}
}
```

```
return xx;
}

//output function
public double[] xmin()
{ return xmin;}

//output function
public String toString()
{ String s="ymin = "+ymin+"\nxmin = ";
for(int k=0;k<xmin.length;k++)
{s+=xmin[k]+" ";}
return s;
}

public void plot()
{
// m number of variables (space dimension) for each bat
if(m==2)
{   double z[][]=Matrix.T(x);
Plot pp=new Plot(z[0],z[1]);
pp.setPlotType(0,35);
pp.setMinMax(xxmin[0],xxmax[0],xxmin[1],xxmax[1]);
pp.plot();
}
}

}
```

Program 5.24.2 bat search algorithm test program

```
import java.util.*;
import java.awt.*;
import java.applet.Applet;
import java.awt.event.*;
import javax.swing.*;

public class battest
{
public static void main(String arg[])
{
if_xj ff=(double x[])->(1.0-x[0])*(1.0-x[0])+100.0*(x[1]-x[0]*x[0])*(x[1]-x[0]*x[0]);
double xmin[]={0,0};
double xmax[]={4,4};
bati  b=new bati(ff,40,xmin,xmax,1000);
double x0[]=b.xmin();
String s="bat search : \n"+b.toString()  ;
String s2="Bat search optimization method ";
JOptionPane.showMessageDialog(null,s,s2,JOptionPane.PLAIN_MESSAGE);
b.plot();
}}
```

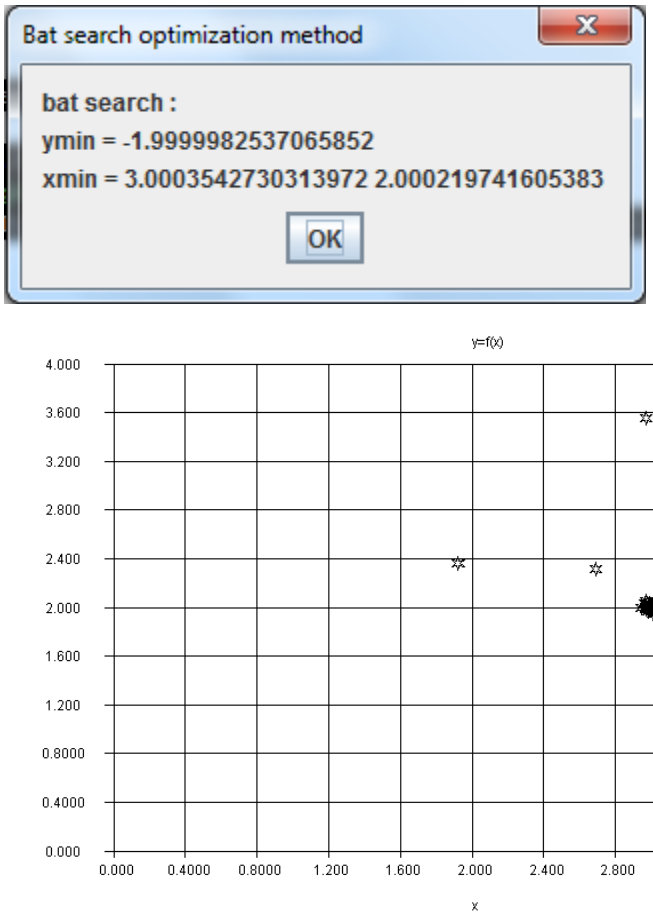


Figure 5.24.1 distribution of the bats at the solution space

5.25 STOCHASTIC METHODS: HARMONY AND MELODY SEARCH ALGORITHMS

Harmony Search (HS) is a relatively new heuristic optimization algorithm and it was first developed by Z. W. Geem *et cd.* in 2001[53]. Harmony search is a music-inspired metaheuristic optimization algorithm. It is inspired by the observation that the aim of music is to search for a perfect state of harmony. This harmony in music is analogous to find the optimum solution in an optimization process. The search process in optimization can be compared to a musician's improvisation process. This perfectly pleasing harmony is determined by the audio aesthetic standard. The aesthetic quality of a musical instrument is essentially determined by its pitch (or frequency), timbre (or sound quality), and amplitude (or loudness). Timbre is largely determined by the harmonic content which is in turn determined by the waveforms or modulations of the sound signal. However, the harmonics it can generate will largely depend on the pitch or frequency range of the particular instrument. In music theory, pitch p is often represented as a function of frequency, f, as

$$p = 69 + 12 * log_2 \left[\frac{f}{440Hz}\right]$$

or the inverse equation is

$$f = 440\left[2^{(p-69)/12}\right]$$

According to this a note A has a frequency of f_0=440 Hz or pitch p=69

Table 5.25.1 Harmony Search algorithm

Harmony Search
Objective function $f(x_1, x_2, \dots, x_n)$
Generate initial harmonics (real number arrays)
Define pitch adjusting rate (r_{pa}) and pitch limits

Define harmony memory accepting rate (T_{accept})
while (t <Max number of iterations)
Generate new harmonics by accepting best harmonics
Adjust pitch to get new harmonics (solutions)
if (rand> r_{accept}),
Choose an existing harmonic randomly
else i f (rand> r_{pa}),
Adjust the pitch randomly within a bandwidth
$x_{new} = x_{old} + b_p(2 * random - 1)$
else
Generate new harmonics via randomization
$p_a = p_{lower\ limit} + p_{range} random$
end if
Accept the new harmonics (solutions) if better
end while
Find the current best estimates

Program 5.25.1 harmony search algorithm

```
 import java.io.*;
import java.text.*;
import java.util.Locale;


public class HarmonySearch1i {
        public int m;       // number of variables
        public int HMS;     // Harmony memory size ( population size)
        public int maxIter;  // maximum number of iterations
        public double PAR;   // pitch adjusting rate
        public double BW;    // Bandwith
        public double HMCR;  // Harmony Memory consideration rate 0< HMCR< 1
        public double xmin[]; // minimum limits of independent variable
        public double xmax[]; //maximum limits of independent variable
        public double NCHV[];
        public double bestFitHistory[];  // best fit history
        public double bestHarmony[];     // Best Harmony
        public double worstFitHistory[]; // worst fit history
        public double HM[][];            //Harmony memory
        static int generation = 0;
        public boolean terminationCriteria = true;
        if_xj ff;

   public HarmonySearch1i(if_xj ffi,double xxmin[],double xxmax[],int maxIteration,double BWi,double HMCRi,int
HMSi,double PARi)
   {        ff=ffi;
            maxIter=maxIteration;
            m=xxmin.length;
            HMCR=HMCRi;
            HMS=HMSi;
            BW=BWi;
            xmin = xxmin;
                    xmax = xxmax;
                    NCHV = new double[m];
                    bestHarmony = new double[m + 1];
                    bestFitHistory = new double[maxIter + 1];
                    worstFitHistory = new double[maxIter + 1];
                    HM = new double[HMS][m + 1];
            xmin=xxmin;
            xmax=xxmax;
   }

   public double random(double low,double high)
           {return low+Math.random()*(high-low);}

           public int random(int low, int high)
           {return (int)(Math.floor(Math.random()*(high-low)+low+.5));}
```

```
public void initiator()
{
	int i;
	double curFit;
	for (i = 0; i < HMS; i++)
	{
		for (int j = 0; j < m; j++) {HM[i][j] = random(xmin[j], xmax[j]);}
		curFit = ff.func(HM[i]);  // fitness(HM[i]);
		HM[i][m] = curFit;	// the fitness is stored in the last column of HM
	}
}

public boolean stopCondition() {
	if (generation > maxIter)
		terminationCriteria = false;
	return terminationCriteria;
}

public void updateHarmonyMemory(double newFitness) {

	// find worst harmony
	double worst = HM[0][m];
	int worstIndex = 0;
	for (int i = 0; i < HMS; i++)
		if (HM[i][m] > worst) {
			worst = HM[i][m];
			worstIndex = i;
		}
	worstFitHistory[generation] = worst;
	// update harmony
	if (newFitness < worst) {
		for (int k = 0; k < m; k++)
			HM[worstIndex][k] = NCHV[k];
		HM[worstIndex][m] = newFitness;
	}
	// find best harmony
	double best = HM[0][m];
	int bestIndex = 0;
	for (int i = 0; i < HMS; i++)
		if (HM[i][m] < best) {
			best = HM[i][m];
			bestIndex = i;
		}
	bestFitHistory[generation] = best;
	if (generation > 0 && best != bestFitHistory[generation - 1]) {
		for (int k = 0; k < m; k++)
			bestHarmony[k] = HM[bestIndex][k];
		bestHarmony[m] = best;
	}
}

public void memoryConsideration(int varIndex) { NCHV[varIndex] = HM[random(0, HMS - 1)][varIndex];}

public void pitchAdjustment(int varIndex) {

	double rand = Math.random();
	double temp = NCHV[varIndex];
	if (rand < 0.5) {
		temp += rand * BW;
		if (temp < xmax[varIndex])
			NCHV[varIndex] = temp;
	} else {
		temp -= rand * BW;
		if (temp > xmin[varIndex])
			NCHV[varIndex] = temp;
	}
```

```
	}

	public void randomSelection(int varIndex) {NCHV[varIndex] = random(xmin[varIndex], xmax[varIndex]);}

	public double[]  xmin()
	{
		initiator();
		while (stopCondition()) {
			for (int i = 0; i < m; i++) {
				if (Math.random() < HMCR) {
					memoryConsideration(i);
					if (Math.random() < PAR)
						pitchAdjustment(i);
				} else
					randomSelection(i);
			}
			double currentFit;
			currentFit = ff.func(NCHV);//fitness(NCHV);
			updateHarmonyMemory(currentFit);
			generation++;
		}
		return bestHarmony;
	}

public String toString()
{ double x[]=xmin();
  double y=x[x.length-1];
	String s="ymin = "+y+"\nxmin = ";
  for(int k=0;k<x.length-1;k++)
  {s+=x[k]+" ";}
  return s;
}

public void plot()
{
if(m==2)
{   double z[][]=Matrix.T(HM);
    for(int i=0;i<z[0].length;i++)
    {System.out.println(z[0][i]+"   "+z[1][i]);}
	Plot pp=new Plot(z[0],z[1]);
	pp.setPlotType(0,35);
	pp.setMinMax(xmin[0],xmax[0],xmin[1],xmax[1]);
	pp.plot();
}
}

}
```

Program 5.25.2 harmony search algorithm test program

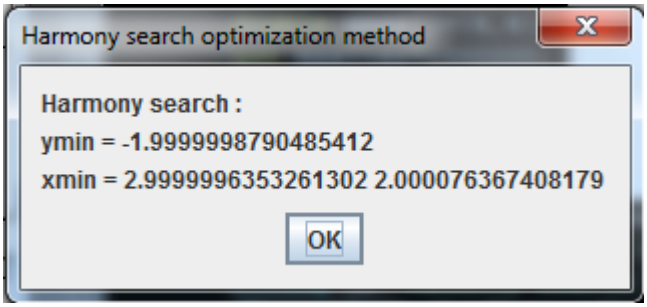
```
 import java.util.*;
import java.awt.*;
import java.applet.Applet;
import java.awt.event.*;
import javax.swing.*;

public class Harmonytest {

	public static void main(String[] args) {

		if_xj ff=(double x[])->-(1.0+Math.cos(Math.PI*(x[0]-3.0))*Math.cos(2.0*Math.PI*(x[1]-2.0))/
   (1+(x[0]-3.0)*(x[0]-3.0)+(x[1]-2.0)*(x[1]-2.0)));
		double xmin[]={0,0};
       double xmax[]={4,4};
       int maxIter=1000000;
```

```
        double BW=0.2;
        double HMCR=0.9;
        int HMS=10;
        double PAR=0.4;
                        HarmonySearch1i hs=new HarmonySearch1i(ff,xmin,xmax,maxIter,BW,HMCR,HMS,PAR);
                        String s="Harmony search : \n"+hs.toString()  ;
        String s2="Harmony search optimization method ";
        JOptionPane.showMessageDialog(null,s,s2,JOptionPane.PLAIN_MESSAGE);
        ContourPlot pp = new ContourPlot(ff,xmin[0],xmax[0],100,xmin[1],xmax[1],100,40, false,"Harmony search
optimisation", "x", "y");
                        pp.plot();
                }
}
```



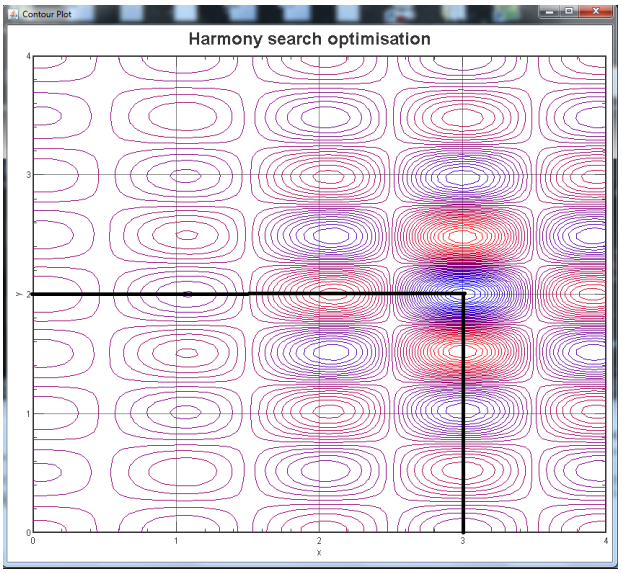


Figure 5.25.1 Contour plot of the harmony search example program

In order to enhance the efficiency of HS algorithm, a novel improved optimization technique, called Melody Search (MS) was proposed by Ashrafi and Dariane[54]. MS algorithm mimics the musical performance processes and interactive relations occurred between members of a musicians group; while, they are looking for the best series of pitches within a melodic line. In such a group, the existing several music players—with different tastes, ideas and experiences—can lead in achieving the best subsequence of pitches faster. Although the new algorithm adopts the basic concepts of HS, the structure is quite different. Unlike HS that uses a single harmony memory, MS algorithm employs several memories named Player Memory (PM). Memories interact to each other like musicians performance in a group. Furthermore, in the MS algorithm the possible range of each pitch for random selection can be changed going through different iterations.
MS algorithm is given as a pseudocode below:

Table 5.25.2 MELODY Search algorithm

Step 1. Initializing the optimization problem and adopting algorithm parameters
Step 2. Initial Phase;
2.1. Initialize player memories (PM)

2.2. Improvise a new melody from each PM
2.3. Update each PM
2.4. Repeat sub-steps 2.2 and 2.3 until the criterion for stopping the initial phase is satisfied
Step 3. Second phase;
3.1. Improvise a new melody from each PM according to the possible range of pitches
3.2. Update each PM
3.3. Determine the possible ranges of pitches for next improvisation (just for randomization)
3.4. Repeat sub-steps 3.1, 3.2 and 3.3 until the criterion for stopping the algorithm is satisfied (i.e. maximum number of iterations: NI)

Computer codes developed accoding to this algorithm.

Program 5.25.3 Melody search algorithm

```
 ////////////////////////////////////////////////////////////////////////////////////////////////
//Performance evaluation of an improved harmony search algorithm for numerical optimization : Melody Search (MS)
////////////////////////////////////////////////////////////////////////////////////////////////
//S.M. Ashrafi , A.B.Dariane////////////////////////////////////////////////////////////////////////////
// Programmed by : Oğuz Emrah Turgut
////////////////////////////////////////////////////////////////////////////////////////////////


public class melody_searchi
{
int D;// Dimension(Number of pitch)
int PMN;//number of player memory
int PMS;//player memory size
int Maxiter1;//maximum of iteration for  first phase
int Maxiter2;//maximum number of iteration for second phase
double PMCR;//player memory considering rate
double PARmax;// Maximum value of pitch adjustment rate
double PARmin;// Minimum value of pitch adjustment rate
double PAR;//Pitch adjustment rate(chenges within every iteration)
double Xnew[][];//Holds new pitches for stage 1
double Xnew2[];//Holds new pitches for stage 2
double[][] fitness;// Holds fitness values for stage 1
double[][] fitness2;// Holds fitness value for stage 2
double[] fitness3;
double[][][] MM;//Melody memory first stage
double[][] MM2;//Melody memory second stage
double[] Lower1;// Lower bound of first stage
double[] Upper1;// Upper bound of first stage
double[] Lower2;// Lower bound of second stage (determined by first stage)
double[] Upper2;// Upper bound of second stage (determined by first stage)
double[] maxval;
int[] maxindex;
double[] minval;
int[] minindex;
double[] maxval2;
int[] maxindex2;
double[] minval2;
int[] minindex2;
double[] bw;//bandwith
double[][] worst;
double[][] best;
double[][] best2;
double[] best3;
double[] worst3;
int L;//random integer between 1 and PMS
int H;//random integer between 1 and D
double min2;
int mindex2;
double max2;
int maxdex2;
if_xj ff;

	public melody_searchi(if_xj iff,int iPMN,int iPMS,int iMaxiter1,int iMaxiter2,double iPMCR,double[] iLower,double[] iUpper)
	{
		ff=iff;
		PMN=iPMN;
```

```
PMS=iPMS;
Maxiter1=iMaxiter1;
Maxiter2=iMaxiter2;
PMCR=iPMCR;
Lower1=iLower;
Upper1=iUpper;
D=Upper1.length;
Lower2=new double[D];
Upper2=new double[D];
maxval=new double[PMN];
maxindex=new int[PMN];
minval=new double[PMN];
minindex=new int[PMN];
maxval2=new double[PMN];
maxindex2=new int[PMN];
minval2=new double[PMN];
minindex2=new int[PMN];
fitness=new double[PMN][PMS];
fitness2=new double[PMN][PMS];
fitness3=new double[PMN];
worst=new double[PMN][D];
best=new double[PMN][D];
best2=new double[PMN][D];
best3=new double[D];
worst3=new double[D];
MM=new double[PMN][PMS][D];
MM2=new double[PMN][D];
Xnew=new double[PMN][D];
Xnew2=new double[D];
PARmax=2.0;
PARmin=0.0;

}

void initialize()
{
        // Improvisation a new melody
        for(int i=0;i<PMN;i++)
        {for(int j=0;j<PMS;j++)
         {for(int k=0;k<D;k++)
          {MM[i][j][k]=Lower1[k]+(Math.random()*(Upper1[k]-Lower1[k]));}}}

          //Calculate fitness of each player
          for(int i=0;i<PMN;i++)
          {for(int j=0;j<PMS;j++)
           {fitness[i][j]=ff.func(MM[i][j]);}}

          // Calculate best and worst melody for each player
           for(int i=0;i<PMN;i++)
           {
                double dd[]=getmaxval_index(fitness[i]);
                maxval[i]=dd[0];
                maxindex[i]=(int)dd[1];
                }

               for(int i=0;i<PMN;i++)
          {
                double dd2[]=getminval_index(fitness[i]);
                minval[i]=dd2[0];
                minindex[i]=(int)dd2[1];
                }
                for(int i=0;i<PMN;i++)
                {
                  for(int k=0;k<D;k++)
                  {
                        best[i][k]=MM[i][minindex[i]][k];
                        }
                }
                for(int i=0;i<PMN;i++)
                {
                  for(int k=0;k<D;k++)
                  {
                        worst[i][k]=MM[i][maxindex[i]][k];
                        }
```

```
                    }
                }


                double[][] solution()
                {
                initialize();

                /// FIRST STAGE /// Determine best melody for each music player

                int iter1=1;
                        while(iter1<Maxiter1)
                        {
                            for(int i=0;i<PMN;i++)
                            {
                                    for(int k=0;k<D;k++)
                                    {
                                            if(Math.random()<=PMCR) // memory consideration
                                    {
                            L=(int)(((double)PMS)*Math.random());
                                            H=(int)(((double)D)*Math.random());
                            PAR=PARmax-((PARmax-PARmin)*((double)iter1)/((double)Maxiter1));
                                            if((iter1%2)==1)
                                            {
                                                    Xnew[i][k]=MM[i][L][k]+(((Math.random()-0.5)*2.0)*((Upper1[k]-
Lower1[k])/200.0));
                                                    }
                                            else
                                            {
                                                    Xnew[i][k]=MM[i][L][H]+(((Math.random()-0.5)*2.0)*((Upper1[k]-
Lower1[k])/200.0));
                                                    }
                                            if(Math.random()<=PAR) // pitch adjustment
                                            {
                                                    Xnew[i][k]=best[i][k];
                                                    }

                                        }
                                                else // randomization
                                                {
                                                        Xnew[i][k]=Lower1[k]+(Math.random()*(Upper1[k]-Lower1[k]));
                                                        }
                                    }
                                                }

                                                for(int i=0;i<PMN;i++)   // Xnew is better than worst , exclude worst
                                                {if(ff.func(Xnew[i])<ff.func(worst[i]))
                                                   {for(int k=0;k<D;k++)
                                                           {worst[i][k]=Xnew[i][k];}
                                                      }
                                                }

                                                // Update  worst and best melody
                                                for(int i=0;i<PMN;i++)
                                          {for(int j=0;j<PMS;j++)
                                            {fitness[i][j]=ff.func(MM[i][j]);}}


                                    for(int i=0;i<PMN;i++)
                                    {
                                            double dd3[]=getmaxval_index(fitness[i]);
                                            maxval[i]=dd3[0];
                                            maxindex[i]=(int)dd3[1];
                                            }

                                            for(int i=0;i<PMN;i++)
                                    {
                                            double dd4[]=getminval_index(fitness[i]);
                                            minval[i]=dd4[0];
                                            minindex[i]=(int)dd4[1];
                                            }

                                            for(int i=0;i<PMN;i++)
                                            {
```

```
                    for(int k=0;k<D;k++)
                    {
                              best[i][k]=MM[i][minindex[i]][k];
                             }
                   }

                          for(int i=0;i<PMN;i++)
                   {
                    for(int k=0;k<D;k++)
                    {
          MM[i][maxindex[i]][k]=worst[i][k]; // worst is now  new maxindexed MM
                             }
                   }

                   iter1++;
                 }


                /// SECOND STAGE /// Determine best harmony from selected melodies



                // Determine new upper and lower bound
                for(int i=0;i<PMN;i++)
                {
                  for(int j=0;j<PMS;j++)
                   {fitness2[i][j]=ff.func(MM[i][j]);}
}



                 for(int i=0;i<PMN;i++)
{
                 double dd6[]=getminval_index(fitness2[i]);
                 minval2[i]=dd6[0];
                 minindex2[i]=(int)dd6[1];
                }
                for(int i=0;i<PMN;i++)
                {
                  for(int k=0;k<D;k++)
                  {
                          best2[i][k]=MM[i][minindex[i]][k];
                          }
                }

                for(int i=0;i<PMN;i++)
                {
                   for(int k=0;k<D;k++)
                   {
                              MM2[i][k]=best[i][k];
                            }
                }

                // MM2 now is new harmony matrix


                // Procedure for determining lower and upper bounds for every column
                double min=0.0;
                for(int k=0;k<D;k++)
                {
                          min=MM2[0][k];
                   for(int i=0;i<PMN;i++)
                   {
                            if(MM2[i][k]<min)
                            {min=MM2[i][k];}
                           }
                           Lower2[k]=min;
                }



                double max=0.0;
                for(int k=0;k<D;k++)
                {
```

```
                                max=MM2[0][k];
                            for(int i=0;i<PMN;i++)
                            {
                                if(MM2[i][k]>max)
                                {max=MM2[i][k];}
                                }
                                Upper2[k]=max;
                        }


                        // determine best and worst vector in MM2
                        for(int i=0;i<PMS;i++)
                        {fitness3[i]=ff.func(MM2[i]);}

                        double dd7[]=getminval_index(fitness3);
                        min2=dd7[0];
                        mindex2=(int)dd7[1];

                        double dd8[]=getmaxval_index(fitness3);
                        max2=dd8[0];
                        maxdex2=(int)dd8[1];

                        for(int i=0;i<D;i++)
                        {best3[i]=MM2[mindex2][i];
                         worst3[i]=MM2[maxdex2][i];}



                        int iter2=1;
                        while(iter2<Maxiter2)
                        {

                                for(int k=0;k<D;k++)
                                {
                                        if(Math.random()<PMCR)//Memory consideration
                                        {
                                                L=(int)(((double)PMS)*Math.random());
                                    H=(int)(((double)D)*Math.random());
                                    PAR=PARmax-((PARmax-PARmin)*((double)iter2)/((double)Maxiter2));
                                                if((iter2%2)==1)
                                                {
                                                Xnew2[k]=MM2[L][k]+(((Math.random()-0.5)*2.0)*((Upper2[k]-
Lower2[k])/200.0));
                                                }
                                                else
                                                {
                                                Xnew2[k]=MM2[L][H]+(((Math.random()-0.5)*2.0)*((Upper2[k]-
Lower2[k])/200.0));
                                                }
                                                 if(Math.random()<PAR)// pitch adjustment
                                                 {
                                                        for(int kk=0;kk<D;kk++)
                                                   {Xnew2[kk]=best3[kk];}

                                                 }

                                    }
                                        else // Randomization
                                        {
                                          Xnew2[k]=Lower2[k]+(Math.random()*(Upper2[k]-Lower2[k]));
                                        }

                                    }

                        if(ff.func(Xnew2)<ff.func(worst3))
                        {
                           for(int i=0;i<D;i++)
                           {worst3[i]=Xnew2[i];}

                        }

                        for(int i=0;i<PMN;i++)
                    {fitness3[i]=ff.func(MM2[i]);}
```

```
                // get minimum index and best solution
                double[] dd9=getminval_index(fitness3);
                min2=dd9[0];
                mindex2=(int)dd9[1];
                for(int i=0;i<D;i++)
                {best3[i]=MM2[mindex2][i];}

                // get maximum index and replace new worst value
                double[] dd10=getmaxval_index(fitness3);
                max2=dd10[0];
                maxdex2=(int)dd10[1];
                for(int i=0;i<D;i++)
                {MM2[maxdex2][i]=worst3[i];}

                        // Procedure for determining lower and upper bounds for every column
                         min=0.0;
                        for(int k=0;k<D;k++)
                        {
                                min=MM2[0][k];
                           for(int i=0;i<PMN;i++)
                           {
                                 if(MM2[i][k]<min)
                                 {min=MM2[i][k];}
                                }
                                Lower2[k]=min;
                        }



                         max=0.0;
                        for(int k=0;k<D;k++)
                        {
                                max=MM2[0][k];
                           for(int i=0;i<PMN;i++)
                           {
                                 if(MM2[i][k]>max)
                                 {max=MM2[i][k];}
                                }
                                Upper2[k]=max;
                        }

                        iter2++;

                       }
                       double[][] dp=new double[2][D];
                       dp[0][0]=min2;
                       for(int i=0;i<D;i++)
                       {dp[1][i]=best3[i];}

                       return dp;

          }

          double[] getmaxval_index(double a[])
{
double m=0.0;
double b[]=new double[a.length];
for(int i=0;i<a.length;i++)
{b[i]=a[i];}
double maxval=a[0];
for(int j=0;j<a.length;j++)
{if(a[j]>maxval){maxval=a[j];}}
for(int i=0;i<b.length;i++)
{if(b[i]==maxval){m=i;break;}}
double dep2[]=new double[2];
dep2[0]=maxval;
dep2[1]=m;
return dep2;
}



double[] getminval_index(double[] a)
```

```
        {
        double m=0.0;
        double b[]=new double[a.length];
        for(int i=0;i<a.length;i++)
        {b[i]=a[i];}
        double minval=a[0];
        for(int i=0;i<a.length;i++)
        {if(a[i]<minval){minval=a[i];}}
        for(int i=0;i<a.length;i++)
        {if(b[i]==minval){m=i;break;}};
        double[] dep=new double[2];
        dep[0]=minval;
        dep[1]=m;
        return dep;
        }

        void toStringnew()
        {
                double out[][]=solution();
                System.out.println("Optimized value = "+out[0][0]);
                for(int i=0;i<D;i++)
                {System.out.println("x["+i+"] = "+out[1][i]);}

              }

}
```

Program 5.25.4 Melody search algorithm test program

```
 public class melody_search_test1
{
public static void main(String args[])
        {   int PMN=100;// Number of player
    int PMS=100;// Player Memory
                int Maxiter1=5000;// Maximum iteration for first stage
                int Maxiter2=1000;// Maximum iteration for second stage
    double PMCR=0.99;// Player memory consideration
    double[] xmin={-100.0,-100.0};
    double[] xmax={100.0,100.0};
    if_xj ff=(double x[])->-Math.cos(x[0])*Math.cos(x[1])*Math.exp(-(x[0]-Math.PI)*(x[0]-Math.PI)-(x[1]-Math.PI)*(x[1]-Math.PI));;
                melody_searchi ms=new melody_searchi(ff,PMN,PMS,Maxiter1,Maxiter2,PMCR,xmin,xmax);
                ms.toStringnew();
              }
}
```

```
---------- Capture Output ----------
> "C:\turhan\java\bin\javaw.exe" melody_search_test
Optimized value = -0.9999999997776523
x[0] = 3.1416013343921128
x[1] = 3.1416011903038434

> Terminated with exit code 0.
```

In recent years, quite a variation of HS is shown in the papers. Another good variant of the program is the Inteligent tuned Harmony Search Algorithm[55]. Pseudocode of this more complicated algorithm is given below. Please refer to the reference paper for further details.

Table 5.25.3 Intelligent tuned Harmony Search algorithm

IHTS Procedure

Step 1: Initılise IHTS parameters

F(x) objective function

N: Number of variables

x_{i_lower} : Lower bound

x_{i_upper} : Upper bound

HMS: Harmony memory size=10

HMCR: Harmony Memory Consideration Rate=0.99

PAR_{min}=0

PAR_{max}=1

Maxiter=Maximum number of iterations

Step 2: Initilise Harmony Memory(HM)

Find best and worst harmony vectors for the HM

Step3: Improvise Harmony Memory (HM)

while (not Maxiter)
for i = 1 to number of decision variables N do
{
if rand [0, 1] <=HMCR
{ // (memory consideration) //
d = int(1 + (HMS _ 1) rand [0, 1])
x_i'=HM(d,i)

$$PAR(\text{iter}) = \text{PAR}_{\max} - (\text{PAR}_{\max} - \text{PAR}_{\min}) \left[\frac{iter}{max\ iter}\right]$$

if rand [0, 1] $\leq$ PAR(iter)
{// (pitch adjustment) //
HM ^{mean}=mean(HM(N+1));
if (HM(d, N + 1)) $\leq$ mean(HM(N + 1))
{
/ / (Group A, Both Intensification and Diversification) //
if rand [0, 1] $\leq$ 0.5

$$y_i = x_i^{Best} - (x_i^{Best} - x_i)rand[0,1]$$

else

$$y_i = x_i^{Best} - (x_i^{Worst} - x_i)rand[0,1]$$

Endif
}
Else
{
/ / (Group B, Only Diversification) //
m = int(1 + (N _ 1)/ rand [0, 1])

$$x_m^{Best} = x_m^{Best}(x_{i_upper}/x_{m_upper})$$

$$y_i = x_i' - (x_m^{Best} - x_i')rand[0,1]$$

}
Endif

$$if(x_{i_lower} \leq y_i \leq x_{i_upper})$$

x_i'= y_i
endif
}Endif // (pitch adjustment) //
Else
{// (memory consideration) //

$$x_i' = x_{i_lower} + (x_{i_upper} - x_{i_lower})rand[0,1]$$

}Endif
}endfor
Step.4: Update Harmony Memory
Calculate the fitness of new harmony vector.

$$if(f(x') \leq f(x^{worst})$$

X^{worst}=x'
Solution=f(x^{best})
End procedure

A program is written according to this algorithm is given below:

Program 5.25.5 Intelligent Tune Harmony Search search algorithm

```
////////////////////////////////////////////////////////////////////////////////////////////////////
//Referance : An Intelligent Tuned Harmony Search algorithm for optimisation
//Parikshit Yadav, Rajesh Kumar, S.K. Panda, C.S. Chang
// Programmed by : Oğuz Emrah Turgut
////////////////////////////////////////////////////////////////////////////////////////////////////

public class intelligent_tuned_harmony_searchi
{
int N;
double Lower[];
double Upper[];
int HMS;
double HMCR;
double PARmin;
double PARmax;
double PAR;
```

```
int Maxiter;
double best[];
double worst[];
double y[];
int d;
int m;
double[][] HM;
double[] fitness;
double[] xdot;
int indexmin;
double minval;
int indexmax;
double maxval;
double fitnessmean;
if_xj ff;
        public intelligent_tuned_harmony_searchi(if_xj iff,double[] iLower,double[] iUpper,int iHMS,double iHMCR,int iMaxiter)
        {
                ff=iff;
                Maxiter=iMaxiter;
                Lower=iLower;
                Upper=iUpper;
                N=Upper.length;
                HMS=iHMS;
                HMCR=iHMCR;
                PARmin=0.0;
                PARmax=2.0;
                HM=new double[HMS][N];
                fitness=new double[HMS];
                best=new double[N];
                worst=new double[N];
                y=new double[N];
                xdot=new double[N];
                }

                void initialize()
                {
                        // introduce Harmony(improvisation)
                 for(int i=0;i<HMS;i++)
                   {for(int j=0;j<N;j++)
                                {HM[i][j]=Lower[j]+(Upper[j]-Lower[j])*Math.random();}}

            // calculate fitness
                                 for(int i=0;i<HMS;i++)
                                {fitness[i]=ff.func(HM[i]);}

                                // get minimum index
                                double[] dd=getminval_index(fitness);
                                minval=dd[0];
                                indexmin=(int)dd[1];
                                for(int i=0;i<N;i++)
                                {best[i]=HM[indexmin][i];}

                                // get maximum index
                                 double[] dd2=getmaxval_index(fitness);
                                 maxval=dd2[0];
                                 indexmax=(int)dd2[1];
                                 for(int i=0;i<N;i++)
                                 {worst[i]=HM[indexmax][i];}
                        }


                double[] getminval_index(double[] a)
                {
                double m=0.0;
                double b[]=new double[a.length];
                for(int i=0;i<a.length;i++)
                {b[i]=a[i];}
                double minval=a[0];
                for(int i=0;i<a.length;i++)
                {if(a[i]<minval){minval=a[i];}}
                for(int i=0;i<a.length;i++)
                {if(b[i]==minval){m=i;break;}};
                double[] dep=new double[2];
                dep[0]=minval;
```

```
dep[1]=m;
return dep;
}

double[] getmaxval_index(double a[])
{
double m=0.0;
double b[]=new double[a.length];
for(int i=0;i<a.length;i++)
{b[i]=a[i];}
double maxval=a[0];
for(int j=0;j<a.length;j++)
{if(a[j]>maxval){maxval=a[j];}}
for(int i=0;i<b.length;i++)
{if(b[i]==maxval){m=i;break;}}
double dep2[]=new double[2];
dep2[0]=maxval;
dep2[1]=m;
return dep2;
}
        double mean(double x[]) // mean of a vector
        {
                int n=x.length;double s1=0.0;
                for(int i=0;i<n;i++)
                {s1+=x[i];}
                return s1/(double)n;
                }
        double[][] solution()
        {
                 initialize() ;
                 int iter=0;
                 while(iter<Maxiter)
                 {
                         for(int i=0;i<N;i++)
                         {
                                 if(Math.random()<HMCR)
                                 {
                                         d=(int)(((double)HMS)*Math.random());
                     xdot[i]=HM[d][i];
                                         PAR=PARmax-((PARmax-PARmin)*((double)iter)/((double)Maxiter));
                                     if(Math.random()<PAR)
                                     {
                                             fitnessmean=mean(fitness);
                                             if(fitness[d]<fitnessmean)
                                             {
                                                     // Group A, Both intensification and
Diversification
                                                     if(Math.random()<=0.5)
                                                     {
                                                             y[i]=best[i]-(best[i]-HM[d][i])*
Math.random();
                                                             }
                                                     else
                                                     {
                                                             y[i]=best[i]-(worst[i]-
HM[d][i])*Math.random();
                                                     }
                                                    }
                                                   else
                                             { // Group B only diversification
                                                      m=(int)(((double)N)*Math.random());
                                                      y[i]=xdot[i]+(best[m]-xdot[i])*Math.random();

                                                     }

                                                     if((Lower[i]<=y[i])&&(Upper[i]>y[i]))
                                                     {xdot[i]=y[i];}

                                                    }
                                 }
                                 else
                                 {
                                 xdot[i]=Lower[i]+(Upper[i]-Lower[i])*Math.random();
```

```
                                                    }
                                        }
                                        if(ff.func(xdot)<=ff.func(worst))
                                        {
                                                for(int i=0;i<N;i++)
                                                {worst[i]=xdot[i];}
                                        }
                                        for(int i=0;i<HMS;i++)
                        {fitness[i]=ff.func(HM[i]);}

                        // get minimum index and best solution
                        double[] dd=getminval_index(fitness);
                        minval=dd[0];
                        indexmin=(int)dd[1];
                        for(int i=0;i<N;i++)
                        {best[i]=HM[indexmin][i];}

                        // get maximum index and replace new worst value
                         double[] dd2=getmaxval_index(fitness);
                         maxval=dd2[0];
                         indexmax=(int)dd2[1];
                         for(int i=0;i<N;i++)
                         {HM[indexmax][i]=worst[i];}

                         iter++;
                }
                double[][] dep=new double[2][N];
                dep[0][0]=minval;
                for(int i=0;i<N;i++)
                {dep[1][i]=best[i];}
        return dep;
}

                        void toStringnew()
                        {
                         double[][] out=solution();
                         System.out.println("Optimized value = "+out[0][0]);
                         for(int i=0;i<N;i++)
                         {System.out.println("x["+i+"] = "+out[1][i]);}
                        }
                        }
```

Program 5.25.6 Intelligent Tune Harmony Search search algorithm test program

```
 public class intelligent_tuned_harmony_search_test
{
        public static void main(String args[])
                {
                        double[] xmin={-10.0,-10.0};
                        double[] xmax={10.0,10.0};
                        int HMS=100;
                        double HMCR=0.99;
                        int Maxiter=50000;
                        if_xj ff=(double x[])->-Math.cos(x[0])*Math.cos(x[1])*Math.exp(-(x[0]-Math.PI)*(x[0]-Math.PI)-
(x[1]-Math.PI)*(x[1]-Math.PI));
                        intelligent_tuned_harmony_searchi iths=new
intelligent_tuned_harmony_searchi(ff,xmin,xmax,HMS,HMCR,Maxiter);
                        iths.toStringnew();
                }
}
```

```
---------- Capture Output ----------
> "C:\turhan\java\bin\javaw.exe" intelligent_tuned_harmony_search_test
Optimized value = -1.0
x[0] = 3.1415926533665637
x[1] = 3.141592659303584

> Terminated with exit code 0.
```

5.26 STOCHASTIC METHODS: CUCKOO SEARCH ALGORITHM

This algorithm which is introduced by Yang and Deb[56] is constructed on brood parasitism of the cuckoos. Brood parasitism is a evoltionary strategy for cuckoos which is accomplished by laying their eggs in other kind of birds. If this alien egg is found by host bird, the nest(a place where eggs are hold) will be rejected or this egg will be thrown away. This behaviour is named as a evolutionary phenomena to mimic the egg apperance of local host birds. Regarding to this concept, following rules are employed .

- Cuckoos are allowed to lay one egg at a specific time, and drops it in a random nest
- Nest that contains high quality of eggs (nets with best fitness values) will carry over to next generation.
- The number of nest is fixed and a host bird can find an alien egg with a specified probability $p_a \in [0,1]$. If it happens, eggs which are dropped in nests will be thrown away or present nest will be built another place.

With this assumptions Cuckoo Search algorithm is developed . In order to carry out this algorithm effectively, Levy Flights are used to perform local and global searches in solution domain. Levy flights are kind of a random walk and simulated as foraging path of an animal and its next movement is based on its previous one. This pattern is also appliciable for natural phenomenas as biology, astronomy and physics. Behaviour of Levy Flight is proposed by

$$x_i^{t+1} = x_i^t + \alpha\theta Levy(\lambda)$$

$\theta$is donated by entery-wise multiplication, α is a step size and Lévy(λ) is Levy distribution. With the help of Levy distribution, displaced egg's position will be updated if this new position is better than the former one. According to literature survey, α value can be set to 1.0 . And also p_a parameter is suggested to be used as 0.25. Cuckoo Search algorithm pseudocode is given below

Program 5.26.1 Cuckoo Search algorithm

```
 import java.util.Arrays;
import java.util.Collections;
import java.util.List;
import java.util.Random;
import java.util.ArrayList;

////////////////////////////////////////////////////////////////////////
// Enginering Optimisation by Cuckoo Search      //
//                                                                    //
//          Xin-She Yang          Suash Deb              //
////////////////////////////////////////////////////////////////////////
// Programmed by : Oğuz Emrah Turgut

//REFERANCE
//Papers -- Citation Details:
// 1) X.-S. Yang, S. Deb, Cuckoo search via Levy flights,
// in: Proc. of World Congress on Nature & Biologically Inspired
// Computing (NaBIC 2009), December 2009, India,
// IEEE Publications, USA, pp. 210-214 (2009).
//http://arxiv.org/PS_cache/arxiv/pdf/1003/1003.1594v1.pdf
// 2) X.-S. Yang, S. Deb, Engineering optimization by cuckoo search,
//Int. J. Mathematical Modelling and Numerical Optimisation,
//Vol. 1, No. 4, 330-343 (2010).
//http://arxiv.org/PS_cache/arxiv/pdf/1005/1005.2908v2.pdf

public class cuckoo_search1i
{
int n;
int nd;
double pa;
double tol;
double nest[][];
double xmin[];
double xmax[];
if_xj ff;
```

```
public cuckoo_search1i(if_xj iff,int in,double ipa,double itol,double ixmin[],double ixmax[])
{
        n=in;
        nd=ixmin.length;
        pa=ipa;
        tol=itol;
        nest=new double[n][nd];
        xmin=new double[nd];
        xmax=new double[nd];
        ff=iff;
        double[] one=ones();
        for(int i=0;i<nd;i++)
        {xmin[i]=ixmin[i];xmax[i]=ixmax[i];}
        }

        double[] randn()
        {
        double a[]=new double[nd];
        for(int i=0;i<nd;i++)
        {a[i]=Math.random();}
        return a;
        }

        double[] ones()
{
        double v[]=new double[nd];
        for(int i=0;i<nd;i++)
        {v[i]=1.0;}
        return v;
        }

        double [][][] initialize()
        {
        for(int i=0;i<n;i++)
        {for(int j=0;j<nd;j++)
           {nest[i][j]=xmin[j]+((xmax[j]-xmin[j])*Math.random());}
        }
        double[] fitness=new double[n];
        for(int i=0;i<n;i++)
        {fitness[i]=Math.pow(10,7);}
        double dp [][][]=new double [2][n][nd];
        for(int i=0;i<n;i++)
        {  for(int j=0;j<nd;j++)
          {dp[0][i][j]=nest[i][j];}
        }
        for(int i=0;i<n;i++)
        {dp[1][i][0]=fitness[i];}
        return dp;
        }

        double logGamma(double x)
        {
double tmp =(x-0.5)*Math.log(x+4.5)-(x+4.5);
double ser = 1.0 + 76.18009173    / (x + 0)   - 86.50532033    / (x + 1)+ 24.01409822    / (x + 2)   -  1.231739516   / (x + 3)+
0.00120858003 / (x + 4)   -  0.00000536382 / (x + 5);
return tmp + Math.log(ser * Math.sqrt(2 * Math.PI));
}
double gamma(double x) { return Math.exp(logGamma(x)); }


double[] getminval_index(double[] a)
        {
        double m=0.0;
```

```
double b[]=new double[a.length];
for(int i=0;i<a.length;i++)
{b[i]=a[i];}
double minval=a[0];
for(int i=0;i<a.length;i++)
{if(a[i]<minval){minval=a[i];}}
for(int i=0;i<a.length;i++)
{if(b[i]==minval){m=i;break;}};
double[] dep=new double[2];
dep[0]=minval;
dep[1]=m;
return dep;
}

double[][][] get_best_nest(double[][] nest,double[][] newnest,double[] fitness)
{
double fnew=0.0;
 for(int j=0;j<n;j++)
 {
   fnew=ff.func(newnest[j]);
   if(fnew<fitness[j])
   {fitness[j]=fnew;nest[j]=newnest[j];}
 }
 double[] dep=getminval_index(fitness);
 double fmin=dep[0];
 int index=(int)dep[1];
 double[] best=new double[nd];
 for(int i=0;i<nd;i++)
 {best[i]=nest[index][i];}
 double depp[][][]=new double[4][n][nd];
 depp[0][0][0]=fmin;
 for(int i=0;i<n;i++)
 {depp[1][i][0]=fitness[i];}
 for(int i=0;i<nd;i++)
 {depp[2][0][i]=best[i];}
 for(int i=0;i<n;i++)
 {for(int j=0;j<nd;j++)
    {depp[3][i][j]=nest[i][j];}
 }
 return depp;
}
// Random permutation of matrix
double[][] randperm(double a[][])
{
double y[][]=new double[a.length][a[0].length];
double[] sw=new double[a[0].length];
for(int i=0;i<a.length;i++)
{y[i]=a[i];}
int m=a.length;
int rp=0;
for(int i=0;i<m-1;i++)
        {rp=(int)((m-i)*Math.random()+i);sw=y[i];
        y[i]=y[rp];y[rp]=sw;
        }
return y;
}
double[][] get_cuckoos(double[][] nest,double best[])
{
double[] s=new double[nd];
double[] u=new double[nd];
double[] v=new double[nd];
double[] step=new double[nd];
double[] stepsize=new double[nd];
double beta=1.5;
```

```
        double sigma=Math.pow((gamma(1+beta)*Math.sin(3.1415*beta/2.0)/(gamma((1.0+beta)/2.0)*beta*Math.pow(2.0,(beta-
1.0)/2.0))),(1.0/beta));
        double[] sk=new double[nd];
        Random rndm=new Random();
        for(int j=0;j<n;j++)
        {
        s=nest[j];
          for(int i=0;i<nd;i++)
          {u[i]=(rndm.nextGaussian())*sigma;
                    v[i]=(rndm.nextGaussian());
                    step[i]=u[i]/Math.pow((Math.abs(v[i])),(1/beta) );
                    stepsize[i]=0.01*step[i]*(s[i]-best[i]);
                    }
                    for(int k=0;k<nd;k++)
                    {sk[k]=stepsize[k]*rndm.nextGaussian();
                     s[k]=sk[k]+s[k];
                    }
                    nest[j]=simplebounds(s);
        }
        return nest;
        }
        double[] simplebounds(double s[])
        {
          for(int i=0;i<nd;i++)
          {if(s[i]<xmin[i])
           {s[i]=xmin[i];}
           if(s[i]>xmax[i])
           {s[i]=xmax[i];}
          }
          return s;
        }
        double[][] emptynests(double[][] nest)
        {
                    double[][] K=new double[n][nd];
           double[][] randmat=new double[n][nd];
                    double[] perm1=new double[n];
                    double[] perm2=new double[n];
                    double[][] nest1=new double[n][nd];
                    double[][] nest2=new double[n][nd];
                    double[][] nest3=new double[n][nd];
                    double[][] nest4=new double[n][nd];
                    double[][] stepsize=new double[n][nd];
           double[][] randmat2=new double[n][nd];
                    double[][] newnest=new double[n][nd];
           for(int i=0;i<n;i++)
           {for(int j=0;j<nd;j++)
                      {randmat[i][j]=Math.random();
                      if(randmat[i][j]<pa){K[i][j]=0.0;}
                      if(randmat[i][j]>pa){K[i][j]=1.0;}
                      }
                    }
                    for(int i=0;i<n;i++)
                    {for(int j=0;j<nd;j++)
                      {nest1[i][j]=nest[i][j];nest2[i][j]=nest[i][j];}}
                    nest3=Matrix.substract(randperm(nest1),randperm(nest2));
                    for(int i=0;i<n;i++)
                    {for(int j=0;j<nd;j++)
                      {stepsize[i][j]=nest3[i][j]*Math.random();}}
                      nest4=Matrix.dotproduct(stepsize,K);
                      newnest=Matrix.add(nest,nest4);
                      return newnest;
        }
        public void solution()
        {
```

```
        double nestt[][]=new double[n][nd];
        double fitness[]=new double[n];
        double bestnest[]=new double[nd];
        double best[]=new double[nd];
        double newnest[][]=new double[n][nd];
        double d1[][][]=initialize();
        for(int i=0;i<n;i++)
        {for(int j=0;j<nd;j++)
          {nestt[i][j]=d1[0][i][j];}
        }
        for(int i=0;i<n;i++)
        {fitness[i]=d1[1][i][0];}
        double d2[][][]=get_best_nest(nestt,nestt,fitness);
        double fmin=d2[0][0][0];
          for(int i=0;i<n;i++)
          {fitness[i]=d2[1][i][0];}
          for(int i=0;i<nd;i++)
          {bestnest[i]=d2[2][0][i];}
                  for(int i=0;i<n;i++)
                  {   for(int j=0;j<nd;j++)
                    {nestt[i][j]=d2[3][i][j];}}
                  int iter=0;
                  double fnew=0.0;
                  nestt=emptynests(nestt);
                  //System.out.println(Matrix.toString(nestt));
                  while(fmin>tol)
                  {
                          newnest=get_cuckoos(nest,bestnest);
                          double d3[][][]=get_best_nest(nestt,newnest,fitness);
             fnew=d3[0][0][0];
             for(int i=0;i<n;i++)
             {fitness[i]=d3[1][i][0];}
             for(int i=0;i<nd;i++)
             {best[i]=d3[2][0][i];}
                    for(int i=0;i<n;i++)
                    {for(int j=0;j<nd;j++)
                      {nestt[i][j]=d3[3][i][j];}
                    }
                    iter+=n;
                          newnest=emptynests(nestt);
                          //System.out.println(Matrix.toString(nestt));
                          double d4[][][]=get_best_nest(nestt,newnest,fitness);
              fnew=d4[0][0][0];
             for(int i=0;i<n;i++)
             {fitness[i]=d4[1][i][0];}
             for(int i=0;i<nd;i++)
             {best[i]=d4[2][0][i];}
                    for(int i=0;i<n;i++)
                    {for(int j=0;j<nd;j++)
                      {nestt[i][j]=d4[3][i][j];}
                    }
                          iter+=n;
                          if(fnew<fmin)
                          {fmin=fnew;bestnest=best;}
                  }
                  System.out.println("Optimum value of a function = "+fmin);
                  for(int i=0;i<bestnest.length;i++)
                  {System.out.println("x["+i+"] = "+bestnest[i]);}
        }
}
```

Program 5.26.2 Cuckoo Search algorithm test program

```
public class cuckoosearchtest1
{
public static void main(String args[])
```

```
{
if_xj ff=(double x[])->3.0*x[0]*x[0]-4.0*x[0]*x[1]+2.0*x[1]*x[1]-x[0]-x[1];
int n=250;//Number of nests
double pa=0.25;// Discovery rate of alien eggs/solutions
double tol=1e-5;//minimum values of tolerance leads to better results
double xmin[]={0.0,0.0};//Lower bounds
double xmax[]={3.0,3.0};//Upper bounds
cuckoo_search1i cc=new cuckoo_search1i(ff,n,pa,tol,xmin,xmax);
cc.solution();
}
}
```

```
---------- Capture Output ----------
> "E:\co\java\bin\javaw.exe" cuckoosearchtest1
Optimum value of a function = -1.120827819699341
x[0] = 0.9863193554487094
x[1] = 1.2809568971894936

> Terminated with exit code 0.
```

5.27 STOCHASTIC METHODS: HUNTING SEARCH ALGORITHM

This method is a meta-heuristic algorithm which simulates the behavior of animals hunting in a group. Animals have different attributes to catch their preys. Wolfs and lions make a circle around its prey and try to hunt it. The proposed algorithm reveals that the hunters which are expressed before is a solution vector for a problem. These simulated hunters come together and catch the prey which, in essence, is a optimum point for this problem.

As it is said before , each “artificial hunter” corresponds to solution of the problem.But in actual hunting of animals there is a significant difference with this algorithm. In actual hunting, hunters see or smell their prey and set its position. But contrary, in optimization problems, solution is not indicated and optimum solution is found by several iterations. In group hunting of animals solution(prey) is dynamic and the hunters (design variables) must change its location. In optimization problems , solution is static and its position does not change during iteration process. To resolve this complexity, artificial hunters(design variables) move towards leader(design variables which has the best solution). Leader is assumed to be the best solution so far and other hunters move towards it. If artificial hunter’s new postion is better than leader , selected artificial hunter is became a new leader. After moving toward the leader, artificial hunters must correct its positions. This is established on “Hunting Group Consideration Rate(HGCR)”. And if they escape out of the ring , prey is surrounded by hunters with random distribition.Finally if the positions of hunters are too close to each other , they reorganize and try to find optimum point. Algorithm mainly consists of six main steps that follows
Step 1 : Initializing parameters of the algorithm
Step 2 : Determining and hunting group[HG]
Step 3 : Moving towards the leader
Step 4 : Changing current positions (if necessary)
Step 5 : Reorganizing the hunting group
Step 6 : Repeat Step 3,4,5 until maximum number of iteration is met
1.Step
First upper and lower limits of design elements are contructed to make a circle around the prey(to satisfy non out of bounds condition). Hunting Group Size (HGS), Maximum Movement toward the Leader (MML) and Hunting Group Consideration Rate(HGCR) are defined to start iterations in this step.
2.Step
Hunting group is set up with random numbers respect to submitted limiting conditions. Cost function (objective function=fitness) is calculated for each hunter and leader hunter is defined by its fitness in this step.
3.Step

With moving towards to leader , new solution vector with N design variables $x_i^{new} = (x_1, x_2, .., x_i, .. x_N)$ is constructed. Movements to leader is established by

$x_i^{new} = x_i + random * MML * (x_i^{Leader} - x_i)$

Where "rand" is a random number between 0 and 1, x_i^{Leader} is the *i*th position of the leader. For each hunting group element if new position of hunter is better than previous position, hunter will change its new position, but if not , hunter remains its position. MML parameter can have different values for various proplems. İts value is dependent of "epochs" which will be described in latter steps. For epochs with large number of iterations MML value is suggested as 0.05 while with small number of iterations its value is proposed as 0.4 .

4.Step

Position correction is built up to organize hunt process in this step.After moving towards the leader , hunters chooses their new solution in two ways: (1) real value correction (2) digital value correction. In this paper, real value correction is taken into account. New hunter position is selected from HG or real value correction by HGCR parameter which varies from 0.0 to 1.0(proposed value lies between 0.1 – 0.4). Value updating process is carried out as

$$x_i^j \leftarrow \begin{Bmatrix} x_i^j \in (x_i^1, x_i^2, x_i^3, \dots, x_i^{HGS}) \; with \; probability \; HGCR \; i = 1..N, j = 1..HGS \\ x_i^j = x_i^j \mp Ra \quad with \; probability \; (1 - HGCR) \end{Bmatrix}$$

Ra is a artificial distance between design variables and is updated during optimization process. Ra is an exponantial funtion and helps to find optimum value and served as

$$Ra = Ra_{min}[\max(x_i) - \min(x_i)] exp\left(\frac{ln\left[\frac{Ra_{min}}{Ra_{max}}\right] iter}{maxiter}\right)$$

Where "iter" is a number of iteration, $\max(x_i)$ and $\min(x_i)$ are maximum and minimum value of hunter vector (x_i). "Ra_{min}" and "Ra_{max}" are the artificial relative search radius of the hunter and "maxiter" is the maximum number of iteration.

Step 5.

During iteration process hunters may be trapped in a local extremums. In order to avoid this situation hunters are rearranged to find another optimum point. After several number of iterations, hunter reorganizes themselves. A new term is emerged as "epoch" which is certain number of search to prevent local extremum trapping. Rearragement process is proposed as

$x_i^{new} = x_i^{Leader} \mp rand[\max(x_i) - \min(x_i)] \alpha \exp(-\beta \; EN)$

Where α and β are real values to determine convergence of algorithm. Large values of α and small values of β are for the problems which has several local optimums. Contrary small values of α and large values of β are for small optimization problems with a small number of design varibles. EN is a counter which numerates epochs concluded so far. EN helps hunters to search more locally into optimum point.

Program 5.27.1 Hunting Search algorithm

```
 public class Hunting_searchi
{
int N;
int HGS;//Number of solution vectores in group
double MML;//Maximum movement toward the leader
double HGCR;//Hunting group consideration rate
double HG[][];
double Lower[];
double Upper[];
double dummyHG[][];
double fitness[];
double best[];
double worst[];
double alfa;
double beta;
double Ramax;
double Ramin;
int maxiter;
int epochiter;
double min;
double max;
int indexmin;
```

```
int indexmax;
int d;
double EN;
double[] ENcount;
if_xj ff;


    public Hunting_searchi(if_xj iff,int iHGS,double iMML,double iHGCR,double iLower[],double[] iUpper,double ialfa,double
ibeta,double iRamax,double iRamin,int imaxiter,int iepochiter)
    {
                ff=iff;
                HGS=iHGS;
                HGCR=iHGCR;
                MML=iMML;
                Ramax=iRamax;
                Ramin=iRamin;
                Lower=iLower;
                Upper=iUpper;
                epochiter=iepochiter;
                maxiter=imaxiter;
                N=Upper.length;
                HG=new double[HGS][N];
                dummyHG=new double[HGS][N];
                fitness=new double[HGS];
                best=new double[N];
                worst=new double[N];
                alfa=ialfa;
                beta=ibeta;
                EN=(double)maxiter/((double)HGS*(double)epochiter);
                EN=Math.ceil(EN);

                 //System.out.println(EN);
                 ENcount=new double[(int)EN];
                 for(int i=0;i<(int)EN;i++)
                 {ENcount[i]=i+1;}
                // System.out.println(Matrix.toString(ENcount));
                }



                double[] getminval_index(double[] a)
                {
                double m=0.0;
                double b[]=new double[a.length];
                for(int i=0;i<a.length;i++)
                {b[i]=a[i];}
                double minval=a[0];
                for(int i=0;i<a.length;i++)
                {if(a[i]<minval){minval=a[i];}}
                for(int i=0;i<a.length;i++)
                {if(b[i]==minval){m=i;break;}};
                double[] dep=new double[2];
                dep[0]=minval;
                dep[1]=m;
                return dep;
                }

                double[] getmaxval_index(double a[])
                {
                double m=0.0;
                double b[]=new double[a.length];
                for(int i=0;i<a.length;i++)
                {b[i]=a[i];}
                double maxval=a[0];
                for(int j=0;j<a.length;j++)
                {if(a[j]>maxval){maxval=a[j];}}
                for(int i=0;i<b.length;i++)
                {if(b[i]==maxval){m=i;break;}}
                double dep2[]=new double[2];
                dep2[0]=maxval;
                dep2[1]=m;
                return dep2;
                }
```

```
void init()
{
   for(int i=0;i<HGS;i++)
   {
                for(int j=0;j<N;j++)
                {HG[i][j]=Lower[j]+(Math.random()*(Upper[j]-Lower[j]));}
               }

               //Calculate fitness
               for(int i=0;i<HGS;i++)
               {fitness[i]=ff.func(HG[i]);}

               // get minimum index and find best
               double[] dd=getminval_index(fitness);
               indexmin=(int)dd[1];
               for(int i=0;i<N;i++)
               {best[i]=HG[indexmin][i];}

               // get maximum index
               double[] dd2=getmaxval_index(fitness);
               indexmax=(int)dd2[1];
               for(int i=0;i<N;i++)
               {worst[i]=HG[indexmax][i];}
}

double max(double x[])
{
 int n=x.length;
 double maxv=x[0];
 for(int i=0;i<n;i++)
 {if(x[i]>maxv)
               {maxv=x[i];}}
 return maxv;
}

double min(double x[])
{
 int n=x.length;
 double minv=x[0];
 for(int i=0;i<n;i++)
 {if(x[i]<minv)
   {minv=x[i];}}
 return minv;

}


double[][] iterations()
{
               init();
               //System.out.println(Matrix.toString(HG));
               int iter=1;
               while(iter<maxiter)
               {

               for(int i=0;i<HGS;i++)
               {for(int j=0;j<N;j++)
                 {dummyHG[i][j]=HG[i][j]+(2.0*(Math.random()-0.5)*MML*(best[j]-HG[i][j]));}}

               for(int i=0;i<HGS;i++)
               {
                 if(ff.func(dummyHG[i])<ff.func(HG[i]))
                 {
                              for(int j=0;j<N;j++)
                              {HG[i][j]=dummyHG[i][j];}
                            }
              }

              //Calculate fitness
               for(int i=0;i<HGS;i++)
               {fitness[i]=ff.func(HG[i]);}

               // get minimum index and find best
```

```
                        double[] dd=getminval_index(fitness);
                        indexmin=(int)dd[1];
                        for(int i=0;i<N;i++)
                        {best[i]=HG[indexmin][i];}


                        for(int i=0;i<HGS;i++)
                        {
                          if(Math.random()<HGCR)
                          {
                            d=(int)(((double)HGS)*Math.random());
                            for(int j=0;j<N;j++)
                            {HG[i][j]=HG[d][j];}
                          }
                          else
                          {
                                    for(int j=0;j<N;j++)
                                    {
                                    HG[i][j]=HG[i][j]+(2.0*(Math.random()-0.5))*Ramin*(max(HG[i])-
min(HG[i]))*Math.exp(Math.log(Ramin/Ramax)*(double)iter/(double)maxiter);
                            }
                                  }
                        }


                        int tt=(int)((double)iter/((double)HGS*(double)epochiter));
                        for(int i=0;i<HGS;i++)
                        {
                           for(int j=0;j<N;j++)
                           {HG[i][j]=best[j]+(2.0*(Math.random()-0.5))*(max(HG[i])-min(HG[i]))*alfa*Math.exp(-beta*ENcount[tt]);
}
                        }




                        //Calculate fitness
                        for(int i=0;i<HGS;i++)
                        {fitness[i]=ff.func(HG[i]);}

                        // get minimum index and find best
                        double[] dd3=getminval_index(fitness);
                        min=dd3[0];
                        indexmin=(int)dd3[1];
                        for(int i=0;i<N;i++)
                        {best[i]=HG[indexmin][i];}

                        iter++;
                        }
                        double[][] out=new double[2][N];
                        for(int i=0;i<N;i++)
                        {out[0][i]=best[i];}
                        out[1][0]=min;
                        return out;

}

                void toStringnew()
                {
                          double[][] in=iterations();
                          System.out.println("Optimized value = "+in[1][0]);
                          for(int i=0;i<N;i++)
                          {System.out.println("x["+i+"] = "+in[0][i]);}
                        }

}
```

Program 5.27.2 Hunting Search algorithm test program

```
 public class Hunting_search_test
{
     public static void main(String args[])
              {
                        int maxiter=20000;
```

```
                        int epochiter=40;
                        int HGS=100;
                        double MML=0.3;
                        double HGCR=0.2;
                        double[] Lower={-5.0,-5.0};
                        double[] Upper={5.0,5.0};
                        double alfa=-0.1;
                        double beta=1.0;
                        double Ramax=1.0e-2;
                        double Ramin=1.0e-7;
                        if_xj ff=(double x[])->-Math.cos(x[0])*Math.cos(x[1])*Math.exp(-(x[0]-Math.PI)*(x[0]-Math.PI)-(x[1]-
Math.PI)*(x[1]-Math.PI));

                         Hunting_searchi hunt=new
Hunting_searchi(ff,HGS,MML,HGCR,Lower,Upper,alfa,beta,Ramax,Ramin,maxiter,epochiter);
                        hunt.toStringnew();
                }
}
```

```
---------- Capture Output ----------
> "D:\java\bin\javaw.exe" Hunting_search_test
Optimized value = -1.0
x[0] = 3.14159264888846
x[1] = 3.141592648868062

> Terminated with exit code 0.
```

5.28 STOCHASTIC METHODS: HARRIS' HAWKS ALGORITHM

Harris Hawks algorithm is a stachastic method based on hunting behavior of harris hawks[93].

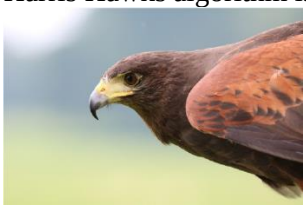


Figure 5.28-1 Harris Hawk

Harris hawk, unlike other hawk species develop a group cooperative hunting behavior. Harris hawks shows evolved innovative team chasing capabilities in tracing, encircling, flushing out, and eventually attacking and catching their pray.

Their group behavior has Exploration and exploitation phases. In the exploration phase Haris' hawks track and detect their prey. But prey can not be seen directly all the time, in this case they fly randomly and perch to some position to detect the prays. Perching point will be selected on two different strategy, based on the position of the other members of the team and base on the possible position of the prey. As computer simulation of this can be given with the following formulations: Assume a random number q, and if q≥0.5 perch in a random position, if q<0.5 select a position based on position of the other members and possible prey location.

$$X(t+1) = \begin{cases} X_{rand}(t) - r_1[X_{rand}(t) - 2r_2X(t)] & if \quad q \geq 0.5 \\ \left(X_{prey}(t) - X_m(t)\right) - r_1[LB + r_4(UB - LB)] & if \quad q \geq 0.5 \end{cases}$$

Where $X(t+1)$ is the position vector of the hawks in the next iteration. $X_{prey}(t)$ is the position of the prey, $X(t)$ is the current position of hawks, r_1, r_2, r_3, r_4 and q are random numbers inside (0,1) range, which are updated in each iteration. LB and UB are search limits (upper and lower boundaries of variables). $X_{rand}(t)$ is a randomly selected hawk from the hawk group. $X_m(t)$ is the average position of the hawks. The average position of the hawks group is calculated by using mathematical averahe formula $X_m(t) = \frac{1}{N}\sum_{i=1}^{N} X_i(t)$ where $X_i(t)$ is the location of the each hawk fort he iteration t, N is the total number of hawks.

In order to translate from Exploration to exploitation phase prey energy is observed. Energy of the prey is modelled as:
$E = 2E_0\left(1 - \frac{t}{T}\right)$ where E indicates escaping energy of the prey, T is yhe maximum number of iterations, and E_0 is the initial state of its energy. When E is positive, prey is stenthening, when E is negative the prey is weakening. Exploration happens when $|E| \geq 1$ and explotation happens when $|E| < 1$. In exploitation phase, attacks the intented prey. Howewer prey often attemps to escape from dangerous situation. Hence, different chasing styles occur in real situation. Suppose that r is the chance of a prey successfully escaping(r<0.5) or not successfully escaping(r≥0.5) before the surprise attack. The hawk behave two different mechanism to respond these, a hard or soft besiege.
When r≥ 0 and $|E| \geq 0.5$ soft besiege happens. In this case the prey has enough energy , and try to escape by some random movements. The Harris' hawks encircle it softly to make rabbit more exhausted the performs the surprise pounce. It is modelled as the following equations:

$$X(t+1) = \Delta X(t) - E\left[JX_{prey}(t) - X(t)\right]$$
$$\Delta X(t) = JX_{prey}(t) - X(t)$$
$$J = 2(1 - r_5)$$

Where $\Delta X(t)$ is the difference between the position vector of the rabbit and the current location in the iteration t, and r_5 is a random number. The hawks behave according to following pattern:

$$Y = X_{prey}(t) - E\left[JX_{prey}(t) - X(t)\right]$$
$$Z = Y + S * LF(D)$$

Where D is dimension of the problem aans S is a random vector of size 1xD and LF is the levy flight function.

$$LF(x) = 0.01\frac{u\sigma}{|v|^{\frac{1}{\beta}}}$$
$$\sigma = \left(\frac{\Gamma(1+\beta)sin\left(\frac{\beta\pi}{2}\right)}{\Gamma(\frac{1+\beta}{2})\beta\, 2^{\left(\frac{\beta-1}{2}\right)}}\right)^{\frac{1}{\beta}}$$

Where u,v are random numbers in (0,1) range, Γ(z) is gamma function which is defined as:

$\Gamma(z) = \int_{z=0}^{\infty} t^{z-1}e^{-t}dt = \frac{e^{-\gamma z}}{z}\prod_{n=1}^{\infty}\left(1 + \frac{z}{n}\right)^{-1} e^{t/n}$

where $\gamma = 0.57721566490153286\ldots$ is the Euler-Mascheroni constant

Hence the final strategy becomes:

$$X(t+1) = \begin{cases} Y & if\ F(Y) < F(X(t)) \\ Z & if\ F(Z) < F(X(t)) \end{cases}$$

When r< 0.5 and $|E| < 0.5$ the prey does not have enough energy so hard besiege happens. Hawks directly attempts the surprise pounce to kill the prey.

$$Y = X_{prey}(t) - E\left[JX_{prey}(t) - X_m(t)\right]$$
$$Z = Y + S * LF(D)$$
$$X(t+1) = \begin{cases} Y & if\ F(Y) < F(X(t)) \\ Z & if\ F(Z) < F(X(t)) \end{cases}$$

Program 5.28-1 Harris-Hawk optimisation algorithm, function definition through if_xj interface

```
import java.util.*;
import java.io.*;
Written by: Oğuz Emrah Turgut
public class harris_hawks_optimization1
{
```

```
double[] Lower;
double[] Upper;
int N;
int D;
int Maxiter;
if_xj ff;
int iter;

double[] Rabbit_Location;
double Rabbit_Energy;
double[] CNVG;
double[][] X;
double[] X1;
double[] X2;
double[] fitnessX;
double[] mean2Dmat;
double[] X_rand;
double E1;
double E0;
double Escaping_Energy;

double q;
double r;
int rand_Hawk_index;

        public harris_hawks_optimization1(if_xj iff,int iN,double[] iLower,double[] iUpper, int iMaxiter)
        {
        Lower=iLower;
        Upper=iUpper;
        ff=iff;
        N=iN;
        D=Lower.length;
        Maxiter=iMaxiter;
        Rabbit_Location=new double[D];
        CNVG=new double[Maxiter];
        fitnessX=new double[N];
        X=new double[N][D];
        mean2Dmat=new double[D];
        X1=new double[D];
        X2=new double[D];
        X_rand=new double[D];
        Rabbit_Energy=1e80;
        }

         static double mean(double X[])
  {
           int m=X.length;
           double sum=0.0;
           for(int i=0;i<m;i++)
           {sum+=X[i];}
           return sum/(double)m;
        }

        static double[] mean2D(double XX[][])
        {
           int m=XX.length;
           int n=XX[0].length;
           double[] X=new double[m];
           double[] Y=new double[n];

            for(int j=0;j<n;j++)
            {
                              for(int i=0;i<m;i++)
                {X[i]=XX[i][j];}
                  Y[j]=mean(X);
            }
            return Y;
  }

   double logGamma(double x) {
   double tmp = (x - 0.5) * Math.log(x + 4.5) - (x + 4.5);
   double ser = 1.0 + 76.18009173    / (x + 0)   - 86.50532033    / (x + 1)
             + 24.01409822    / (x + 2)   -  1.231739516   / (x + 3)
```

```
                + 0.00120858003 / (x + 4)   -  0.00000536382 / (x + 5);
    return tmp + Math.log(ser * Math.sqrt(2 * Math.PI));
  }
    double gamma(double x) { return Math.exp(logGamma(x)); }

    double[] Levy()
    {
              double beta=1.5;
     double sigma=Math.pow((gamma(1+beta)*Math.sin(Math.PI*beta/2)/(gamma((1+beta)/2)*beta*Math.pow(2,((beta-
1)/2)))),(1/beta));
              Random rnd=new Random();
              double[] u=new double[D];
              double[] v=new double[D];
              double[] step=new double[D];
              for(int j=0;j<D;j++)
              {u[j]=rnd.nextGaussian()*sigma;}
              for(int j=0;j<D;j++)
              {v[j]=rnd.nextGaussian();}
              for(int j=0;j<D;j++)
              {step[j]=u[j]/(Math.pow(Math.abs(v[j]),(1.0/beta)));}
              return step;
             }


    double[][] boundary(double[][] XX)
    {
                        for(int i=0;i<N;i++)
                                {for(int j=0;j<D;j++)
                                {if((XX[i][j]<Lower[j])||(XX[i][j]>Upper[j]))
                        {XX[i][j]=Lower[j]+((Upper[j]-Lower[j])*Math.random());}}}
                        return XX;
              }

              void init()
              {
                 for(int i=0;i<N;i++)
                 {for(int j=0;j<D;j++)
                          {X[i][j]=(Lower[j]+((Upper[j]-Lower[j])*Math.random()));}}
              }

    double[][] solution()
    {
              init();
              iter=0;

              while(iter<=Maxiter)
              {

                        X=boundary(X);
                        for(int i=0;i<N;i++)
                        {fitnessX[i]=ff.func(X[i]);
                           if(fitnessX[i]<Rabbit_Energy)
                                   {Rabbit_Energy=fitnessX[i];
                                             for(int j=0;j<D;j++)
                                             {Rabbit_Location[j]=X[i][j];}}}

                                 E1=2.0*(1.0-((double)iter/(double)Maxiter));

                                 for(int i=0;i<N;i++)
                                 {
                                 E0=2.0*Math.random()-1.0;
                                 Escaping_Energy=E1*E0;

                                   if(Math.abs(Escaping_Energy)>=1.0)
                                   {
                                   q=Math.random();
                                   rand_Hawk_index=(int)Math.floor((double)N*Math.random());
                                     for(int j=0;j<D;j++)
                                     {X_rand[j]=X[rand_Hawk_index][j];}
                                              if(q<0.5)
                                              {
                                                for(int j=0;j<D;j++)
                                                {X[i][j]=X_rand[j]-Math.random()*Math.abs(X_rand[j]-2.0*Math.random()*X[i][j]);}
                                              }
```

```
                                        else if (q>=0.5)
                                        {
                                                mean2Dmat=mean2D(X);
                                                for(int j=0;j<D;j++)
                                                {X[i][j]=(Rabbit_Location[j]-mean2Dmat[j])-
(Math.random()*(Lower[j]+((Upper[j]-Lower[j])*Math.random())));}

                                        }

                                }
                                else if(Math.abs(Escaping_Energy)<1.0)
                                {
                                        r=Math.random();
                                        if((r>=0.5)&&(Math.abs(Escaping_Energy)<0.5))
                                        {
                                          for(int j=0;j<D;j++)
                                          {X[i][j]=Rabbit_Location[j]-Escaping_Energy*Math.abs(Rabbit_Location[j]-X[i][j]);}

                                        }
                                  if((r>=0.5)&&(Math.abs(Escaping_Energy)>=0.5))
                                        {
                                          for(int j=0;j<D;j++)
                                          {X[i][j]=(Rabbit_Location[j]-X[i][j])-Escaping_Energy*Math.abs(2.0*(1.0-
Math.random())*Rabbit_Location[j]-X[i][j]);}
                                        }
                                        if((r<0.5)&&(Math.abs(Escaping_Energy)>=0.5))
                                        {
                                          for(int j=0;j<D;j++)
                                          {X1[j]=Rabbit_Location[j]-Escaping_Energy*Math.abs(2.0*(1.0-
Math.random())*Rabbit_Location[j]-X[i][j]);}

                                          if( ff.func(X1)< ff.func(X[i]))
                                          {
                                                  for(int j=0;j<D;j++)
                                                  {X[i][j]=X1[j];}

                                                }
                                                else
                                                {
                                                  double[] Levyout=Levy();
                                                  for(int j=0;j<D;j++)
                                                  {X2[j]=Rabbit_Location[j]-Escaping_Energy*Math.abs( 2.0*(1.0-
Math.random())*Rabbit_Location[j]-X[i][j]) + Math.random()*Levyout[j];  }

                                                   if( ff.func(X2)<ff.func(X[i]) )
                                                   {for(int j=0;j<D;j++)
                                                           {X[i][j]=X2[j];}}
                                                }


                                        }
                                        if((r<0.5)&&(Math.abs(Escaping_Energy)<0.5))
                                        {
                                                mean2Dmat=mean2D(X);
                                          for(int j=0;j<D;j++)
                                          {X1[j]=Rabbit_Location[j]-Escaping_Energy*Math.abs(2.0*(1.0-
Math.random())*Rabbit_Location[j]-mean2Dmat[j]);}

                                          if( ff.func(X1)< ff.func(X[i]))
                                          {
                                                  for(int j=0;j<D;j++)
                                                  {X[i][j]=X1[j];}

                                                }
                                                else
                                                {
                                                  double[] Levyout=Levy();
                                                  mean2Dmat=mean2D(X);
                                                  for(int j=0;j<D;j++)
                                                  {X2[j]=Rabbit_Location[j]-Escaping_Energy*Math.abs( 2.0*(1.0-
Math.random())*Rabbit_Location[j]-mean2Dmat[j]) + Math.random()*Levyout[j];  }

                                                   if( ff.func(X2)<ff.func(X[i]) )
                                                   {for(int j=0;j<D;j++)
```

```
                                                        {X[i][j]=X2[j];}}
                                                    }


                                                }
                                            }

                                }
                                iter++;
                }

                double[][] out=new double[2][D];
                for(int j=0;j<D;j++)
                {out[0][j]=Rabbit_Location[j];}
                out[1][0]=Rabbit_Energy;
                return out;

               }

void toStringnew()
{
  double[][] out=solution();
  System.out.println("Optimized value = "+out[1][0]);
  for(int i=0;i<D;i++)
  {System.out.println("x["+i+"] = "+out[0][i]);}
}
public static void main(String arg[])
{//harris_hawks_optimization hho=new harris_hawks_optimization();
 //hho.toStringnew();
}

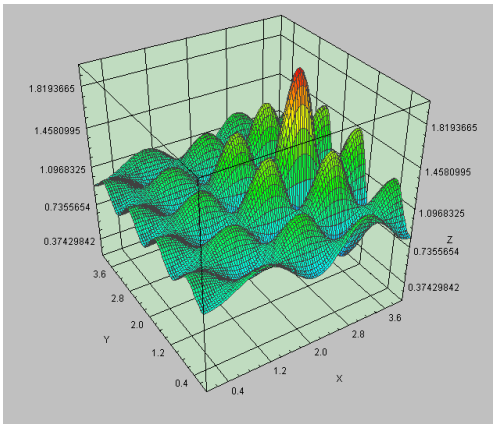
}
```

Program 5.28-2 Harris-Hawks optimisation algorithm rtest program , function definition through if_xj interface lambda variable

```
 import java.util.*;
import java.io.*;

public class harris_hawks_optimization_test3
{
public static void main(String arg[])
   {
   int iN=1000;
   if_xj f=(double x[])->1.0+Math.cos(Math.PI*(x[0]-3.0))*Math.cos(2.0*Math.PI*(x[1]-2.0))/
   (1+(x[0]-3.0)*(x[0]-3.0)+(x[1]-2.0)*(x[1]-2.0));
   double[] iLower={0.0,0.0};
   double[] iUpper={4,4};
   int iMaxiter=200;
   //harris_hawks_optimization(f_xj iff,int iN,double[] iLower,double[] iUpper, int iMaxiter)
            harris_hawks_optimization1 hho=new harris_hawks_optimization1(f,iN,iLower,iUpper,iMaxiter);
   double[][] x0=hho.solution();
   String s="";
   s+="harris-hawks optimisation : \n"+Matrix.toStringT(x0[0]);
            double [] r2=NA41_i.nelder(f,x0[0]);
            s+="Nelder-Mead solution : \n"+Matrix.toStringT(r2);
            System.out.println(s);
            surfacePlot p=new surfacePlot(f,iLower[0],iUpper[0],iLower[1],iUpper[1],100);
            p.plot();
}
}
```

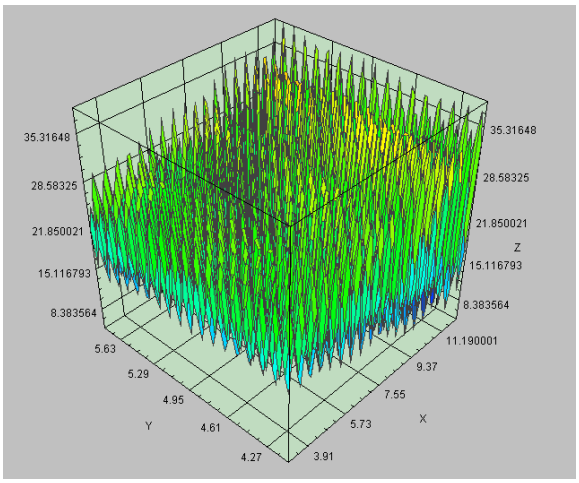
```
---------- Capture Output ----------
> "E:\co\java\bin\javaw.exe" harris_hawks_optimization_test3
harris-hawks optimisation :
        3.000000002599889
        2.480328761142911
Nelder-Mead solution :
        3.000000002599889
        2.480328761142911
```



Program 5.28-3 Harris-Hawks optimisation algorithm test program , function definition through if_xj interface lambda variable

```
import java.util.*;
import java.io.*;
public class harris_hawks_optimization_test4
{
public static void main(String arg[])
   {
   int iN=300;
   if_xj f=(double x[])->21.5+x[0]*Math.sin(4.0*Math.PI*x[0])+x[1]*Math.sin(20.0*Math.PI*x[1]);
   double[] iLower={3.0,4.1};
   double[] iUpper={12.1,5.8};
   int iMaxiter=200;
   //harris_hawks_optimization(f_xj iff,int iN,double[] iLower,double[] iUpper, int iMaxiter)
            harris_hawks_optimization1 hho=new harris_hawks_optimization1(f,iN,iLower,iUpper,iMaxiter);
   double[][] x0=hho.solution();
   String s="";
   s+="harris-hawks optimisation : \n"+Matrix.toStringT(x0[0]);
            double [] r2=NA41_i.nelder(f,x0[0]);
            s+="Nelder-Mead solution : \n"+Matrix.toStringT(r2);
            System.out.println(s);
            surfacePlot p=new surfacePlot(f,iLower[0],iUpper[0],iLower[1],iUpper[1],70);
            p.plot();
}
```

```
---------- Capture Output ----------
> "E:\co\java\bin\javaw.exe" harris_hawks_optimization_test4
harris-hawks optimisation :
          11.875533237082800
           5.775043861484454
Nelder-Mead solution :
          11.875533237082784
           5.775043861484401
```



5.29 STOCHASTIC METHODS: THERMAL EXCHANGE ALGORITHM

Thermal Exchange model[94] is based on cooling of the small (zero dimensional) object. In heat transfer this mechanism is called time dependent lumped solution. Consider a small body with volume V(m^3), surface area A(m^2), thermal convectivity at fluid-surface interaction h(W/m^2K), specific heat C_p (J/kgK), initial (time t=0 s) temperature T_0 (degree K) , environmental temperature T_b (degree K), and temperature of the body at time t is T(degree K), then time dependent variation of temperature of the object will be:

$\frac{T-T_b}{T_0-T_b} = exp\left(-\frac{hA}{V\rho C_p}t\right)$

Defining $\beta = \frac{hA}{V\rho C_p}$ equation becomes

$\frac{T-T_b}{T_0-T_b} = exp(-\beta t)$ or $T = T_b + (T_0 - T_b)exp(-\beta t)$

Based on these TEO is defined as follows:

Step 1: Initilization

The initial temperature of all the objects are determined in an *m* -dimensional search space.

$$T_i^0 = T_{min} + random(T_{max} - T_{min})$$

Where T_{min} and T_{max} are min and maximum bounds of the search region. Random is a ramdom number (0..1) region.

Step 2: Evaluation

Function values will be evaluated

Step 3: Saving

The group of best T vectors and their objective function value will be saved into memory.

Step 4: Group creating (for body and environment)

Agents will be divided into two equal groups. The first group will be treated as environment and the second group will be treated as cooling bodies. For example T_0 to $T_{n/2}$ will be environmental temperatures and $T_{n/2+1}$ to T_n will be cooling bodies.

Step 5: Defining β

Definition of β was given as $\beta = \frac{hA}{V\rho C_p}$ in this definition h is thermal conductivity coefficients and when h is bigger heat transfer will be higher so cooling effect will be higher, so the change of the β will determine cooling rate. In the algorithm, β will be defined as the cost of the object to cost of the wors object ratio.

$\beta = \frac{Cost(object)}{Cost(worst\ object)}$

Step 6: Defining t

$t = \frac{iteration}{Max\ iteration}$

Step 7: Escaping from local optima i

Environmental temperature is changed. Where c_1 and c_2 are controlling variables.

$T_i^{env} = (1 - \left(c_1 + c_2(1-t)\right) * random)T_i^{env}$

c_1 and c_2 are chosen either 0 or 1

Step 8: Updating the agent

The temperature of the each object is updated.

$T_i^{new} = T_i^{env} + (T_i^{old} - T_i^{new})\exp(-\beta t)$

Step 9: Escaping from local optima ii

The randomly selected parameter Pro (0,1) is introduced and it is specify whether a component of each cooling object must be changed. With each agent Pro is compared with a random number. If random(i)<Pro ith agent is selected randomly and its value is regenerated as

$T_{i,j} = T_{j,min} + random(T_{j,max} - T_{j,min})$

Where $T_{i,j}$ is the jth variable of agent i. $T_{j,max}$ and $T_{j,min}$ are lower and upper bounds of jth variable.

Step 10: Checking terminating conditions

The optimisation process will be terminated after a ficed number of iterations. If the termination does not satisfied it repeat fixed number of iterations.

Java code and example test cases are given below.

Program 5.29-1 Thermal exhange optimisation algorithm, function definition through if_xj interface

```
import java.util.*;
import java.io.*;

public class Thermal_Exchange_Optimizationi
{
int N;
int D;
int Maxiter;
if_xj ff;
double[] Lower;
double[] Upper;
double[][] Therm_Object;
double[][] MEM_Therm_Object;
double[][] TRIAL_TO;
double[][] TRIAL;
double[]   TRIAL_fit;
double[] fitness_Therm_Object;
double[] sorted_fitness_Therm_Object;
double[] MEM_fitness_Therm_Object;
double[][] ENV_TO;
double[][] UPD_TO;
int MEM_TO;
double c1;
double c2;
double c;
double beta;
double p;
int index;
int NN;
int iter;
    public Thermal_Exchange_Optimizationi(if_xj iff,int iN,double[] iLower,double[] iUpper, int iMaxiter)
    {
            N=iN;
      ff=iff;
      Lower=iLower;
      Upper=iUpper;
      Maxiter=iMaxiter;
      D=Lower.length;
      MEM_TO=5;
      p=0.3;
      Therm_Object=new double[N][D];

      fitness_Therm_Object=new double[N];
      sorted_fitness_Therm_Object=new double[N];
      ENV_TO=new double[N][D];
      UPD_TO=new double[N][D];
      MEM_Therm_Object=new double[MEM_TO][D];
      MEM_fitness_Therm_Object=new double[MEM_TO];
      TRIAL=new double[N+MEM_TO][D];
      TRIAL_TO=new double[N+MEM_TO][D];
      TRIAL_fit=new double[N+MEM_TO];
      NN=N+MEM_TO;
    }

    double[][] sort_and_index(double[] A)
            {
                    ArrayList<Double> B=new ArrayList<Double>();
                    for(int i=0;i<A.length;i++)
                    {B.add(A[i]);}
                    ArrayList<Double> nstore=new ArrayList<Double>(B);
                    Collections.sort(B);
                    double[] ret=new double[B.size()];
                    Iterator<Double> iterator=B.iterator();
                    int ii=0;
                    while(iterator.hasNext())
                    {ret[ii]=iterator.next().doubleValue();ii++;}
                    int[] indexes=new int[B.size()];
                    for(int n=0;n<B.size();n++)
                    {indexes[n]=nstore.indexOf(B.get(n));}
                    double[][] outt=new double[2][B.size()];
                    for(int i=0;i<B.size();i++)
                    {outt[0][i]=ret[i];outt[1][i]=indexes[i];}
                    return outt;
```

```
        }

        double[] getminval_index(double[] a)
        {
        double m=0.0;
        double b[]=new double[a.length];
        for(int i=0;i<a.length;i++)
        {b[i]=a[i];}
        double minval=a[0];
        for(int i=0;i<a.length;i++)
        {if(a[i]<minval){minval=a[i];}}
        for(int i=0;i<a.length;i++)
        {if(b[i]==minval){m=i;break;}};
        double[] dep=new double[2];
        dep[0]=minval;
        dep[1]=m;
        return dep;
}


        void init()
{
           for(int i=0;i<N;i++)
           {for(int j=0;j<D;j++)
                        {Therm_Object[i][j]=Lower[j]+((Upper[j]-Lower[j])*Math.random());}}

                        for(int i=0;i<N;i++)
                        {fitness_Therm_Object[i]=ff.func(Therm_Object[i]);}

                        double[][] BB=sort_and_index(fitness_Therm_Object);
                        double[] I=new double[N];
                        for(int i=0;i<N;i++)
                         {I[i]=BB[1][i];}
                         for(int i=0;i<N;i++)
                         {sorted_fitness_Therm_Object[i]=BB[0][i];}

                        for(int i=0;i<N;i++)
                         {for(int j=0;j<D;j++)
                                        {Therm_Object[i][j]=Therm_Object[(int)I[i]][j];}}

                                        for(int i=0;i<MEM_TO;i++)
                                        {MEM_fitness_Therm_Object[i]=ff.func(Therm_Object[i]);
                                                for(int j=0;j<D;j++)
                                                {MEM_Therm_Object[i][j]=Therm_Object[i][j];}}

           }

           double[][] environment(double[][] T, int iterr)
           {

                        int mm=T.length;
                        int nn=T[0].length;
                        double[][] Env_TO=new double[mm][nn];
                        double t=(double)iterr/(double)Maxiter;
                        for(int i=0;i<N;i++)
                        {
                         c1=Math.round(Math.random());
                         c2=Math.round(Math.random());
                                c=c1+c2*(1-t);
                         for(int j=0;j<D;j++)
                         {Env_TO[i][j]=(1.0-c*Math.random())*T[i][j];}
                        }
                       return Env_TO;
                       }

                       double[][] Newtonslaw(double[][] T,double[][] ENVTO,int iterr)
                       {
                                double t=(double)iter/(double)Maxiter;
                                for(int i=0;i<N;i++)
                        {fitness_Therm_Object[i]=ff.func(T[i]);}
                        double[][] BB=sort_and_index(fitness_Therm_Object);
                        double[] I=new double[N];
                        for(int i=0;i<N;i++)
                         {I[i]=BB[1][i];}
```

```
                    for(int i=0;i<N;i++)
                    {sorted_fitness_Therm_Object[i]=BB[0][i];}
                    for(int i=0;i<N;i++)
                    {for(int j=0;j<D;j++)
                              {T[i][j]=T[(int)I[i]][j];}}
                          for(int i=0;i<N;i++)
                    {for(int j=0;j<D;j++)
                              {ENVTO[i][j]=ENVTO[(int)I[i]][j];}}

                              double[][] TO_NEW=new double[N][D];

                          for(int i=0;i<N;i++)
                          {
                            beta=sorted_fitness_Therm_Object[i]/sorted_fitness_Therm_Object[N-1];
                                    if(i<N/2)
                                    {
                                     for(int j=0;j<D;j++)
                                     {TO_NEW[i][j]=ENVTO[(N/2)-1+i][j]+(T[i][j]-ENVTO[(N/2)-1+i][j])*Math.exp(-
beta*t);}
                                                          if(Math.random()<p)
                                                          {index=(int)Math.floor(Math.random()*(double)D);
                                                           TO_NEW[i][index]=Lower[index]+((Upper[index]-
Lower[index])*Math.random());
                                                          }

                                    }
                                    else
                                    {
                                     for(int j=0;j<D;j++)
                                     {TO_NEW[i][j]=ENVTO[i-(N/2)][j]+(T[i][j]-ENVTO[i-(N/2)][j])*Math.exp(-beta*t);}

                                         if(Math.random()<p)
                                                          {index=(int)Math.floor(Math.random()*(double)D);
                                                           TO_NEW[i][index]=Lower[index]+((Upper[index]-
Lower[index])*Math.random());
                                                          }

                              }
        }

                    return TO_NEW;
              }

              double[][] Update(double[][] T,double[][] T_M)
              {

                    double[][] TT=new double[N][D];
                    for(int i=0;i<N;i++)
                    {for(int j=0;j<D;j++)
                              {TRIAL[i][j]=T[i][j];}}

                   for(int i=N;i<NN;i++)
                    {for(int j=0;j<D;j++)
                              {TRIAL[i][j]=T_M[i-N][j];}}

                    for(int i=0;i<N;i++)
                    {TRIAL_fit[i]=ff.func(TRIAL[i]);}


                    double[][] AA=sort_and_index(TRIAL_fit);


                    for(int i=0;i<NN;i++)
                    {for(int j=0;j<D;j++)
                              {TRIAL_TO[i][j]=TRIAL[(int)AA[1][i]][j];}}

                   for(int i=0;i<N;i++)
                    {for(int j=0;j<D;j++)
                              {TT[i][j]=TRIAL_TO[i][j];}}

                    return TT;
                  }

                  double[][] sorted(double[][] TT)
```

```
        {
                int mm=TT.length;
                int nn=TT[0].length;
                double[][] TNEW=new double[mm][nn];
                double[] fitnessTT=new double[mm];
                for(int i=0;i<mm;i++)
                {fitnessTT[i]=ff.func(TT[i]);}
                double[][] FF=sort_and_index(fitnessTT);
                for(int i=0;i<mm;i++)
                {for(int j=0;j<nn;j++)
                  {TNEW[i][j]=TT[(int)FF[1][i]][j];}}

                return TNEW;
}



        double[][] Memory(double[][] TO, double[][] TO_M)
        {

                double[] fit_TO_M=new double[MEM_TO];
                double[] fit_TO=new double[N];
                for(int i=0;i<MEM_TO;i++)
                {fit_TO_M[i]=ff.func(TO_M[i]);}
          for(int i=0;i<N;i++)
                {fit_TO[i]=ff.func(TO[i]);}

                for(int i=0;i<N;i++)
                {
                   for(int j=0;j<MEM_TO;j++)
                   {
                           if(fit_TO[i]<fit_TO_M[j])
                           {
                                   for(int jj=0;jj<D;jj++)
                                   {TO_M[j][jj]=TO[i][jj];}
                                    fit_TO_M[j]=fit_TO[i];

                                  }
                        }


            }

                return TO_M;
}

double[][] solution()
{
            init();
            iter=0;

            while(iter<Maxiter)
            {
                    ENV_TO=environment(Therm_Object,iter);

                    UPD_TO=Newtonslaw(Therm_Object,ENV_TO,iter);

                    for(int i=0;i<N;i++)
                    {if( ff.func(UPD_TO[i])<ff.func(Therm_Object[i]))
                     {for(int j=0;j<D;j++)
                       {Therm_Object[i][j]=UPD_TO[i][j];}}}

                    MEM_Therm_Object=sorted(MEM_Therm_Object);
                    Therm_Object=sorted(Therm_Object);
                    Therm_Object=Update(Therm_Object,MEM_Therm_Object);

                    MEM_Therm_Object=Memory(Therm_Object,MEM_Therm_Object);
                    MEM_Therm_Object=sorted(MEM_Therm_Object);

                    iter++;

            }

double[][] out=new double[2][D];
```

```
                for(int j=0;j<D;j++)
                {out[0][j]=MEM_Therm_Object[0][j];}
                 out[1][0]=ff.func(MEM_Therm_Object[0]);
                return out;



                        }


 void toStringnew()
{
  double[][] out=solution();
  System.out.println("Optimized value = "+out[1][0]);
  for(int i=0;i<D;i++)
  {System.out.println("x["+i+"] = "+out[0][i]);}
}

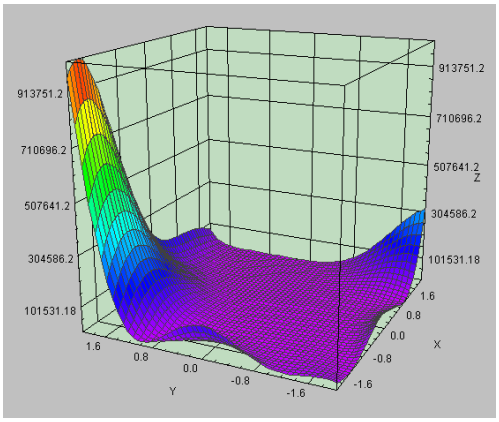

}
```

Test function: $f(x) = \left[1 + (x_0 + x_1 + 1)^2\left(19 - 14x_0 + 3x_0^2 - 14x_1 + 6x_0x_1 + 3x_1^2\right)\right] * \left[30 + (2x_0 - 3x_1)^2\left(18 - 32x_0 + 12x_0^2 + 48x_1 - 36x_0x_1 + 27x_1^2\right)\right]$

Program 5.29-2 Thermal exhange optimisation test, function definition through if_xj interface

```
public class Thermal_Exchange_Optimization_test1
{
public static void main(String args[])
{   //Golstein & price function
          if_xj ff=(double x[])->(1.0+(x[0]+x[1]+1.0)*(x[0]+x[1]+1.0)*(19.0-14.0*x[0]+3.0*x[0]*x[0]-
14.0*x[1]+6.0*x[0]*x[1]+3.0*x[1]*x[1]))*
          (30.0+(2.0*x[0]-3.0*x[1])*(2.0*x[0]-3.0*x[1])*(18.0-32.0*x[0]+12.0*x[0]*x[0]+48.0*x[1]-36.0*x[0]*x[1]+27.0*x[1]*x[1]));
 double[] Lower={-2,-2};
 double[] Upper={2,2};
 int Maxiter=600;
 int N=48;
 Thermal_Exchange_Optimizationi wo=new   Thermal_Exchange_Optimizationi(ff,N,Lower,Upper,Maxiter);
 double[][] out=wo.solution();
 System.out.println("Optimized value = "+out[1][0]);
 for(int i=0;i<wo.D;i++)
 {System.out.println("x["+i+"] = "+out[0][i]);}
 surfacePlot pp=new surfacePlot(ff,Lower,Upper);
 pp.plot();
}
}
```

```
---------- Capture Output ----------
> "E:\co\java\bin\javaw.exe" Thermal_Exchange_Optimization_test1
Optimized value = 3.0014879299264376
x[0] = 0.002243055184328878
x[1] = -0.998527680225803
```



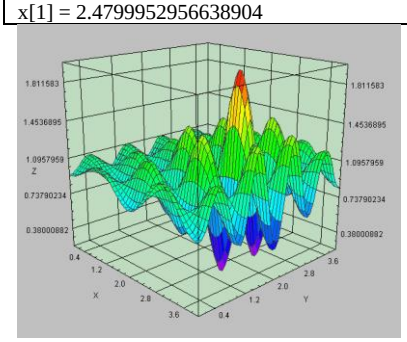
Program 5.29-3 Thermal exhange optimisation test, function definition through if_xj interface

```
public class Thermal_Exchange_Optimization_test2
{
```

```
public static void main(String args[])
{
            if_xj ff=(double x[])->1.0+Math.cos(Math.PI*(x[0]-3.0))*Math.cos(2.0*Math.PI*(x[1]-2.0))/
   (1+(x[0]-3.0)*(x[0]-3.0)+(x[1]-2.0)*(x[1]-2.0));
 double[] Lower={0,0};
 double[] Upper={4,4};
 int Maxiter=600;
 int N=48;
 Thermal_Exchange_Optimizationi wo=new   Thermal_Exchange_Optimizationi(ff,N,Lower,Upper,Maxiter);
 double[][] out=wo.solution();
 System.out.println("Optimized value = "+out[1][0]);
 for(int i=0;i<wo.D;i++)
 {System.out.println("x["+i+"] = "+out[0][i]);}

 surfacePlot pp=new surfacePlot(ff,Lower,Upper);
 pp.plot();
}
}
```

```
---------- Capture Output ----------
> "E:\co\java\bin\javaw.exe" Thermal_Exchange_Optimization_test2
Optimized value = 0.1936801694238951
x[0] = 2.998188195764863
x[1] = 2.4799952956638904
```



5.30 NON LINEAR OPTIMIZATION WITH BOUNDARY RESTRICTIONS

A non-linear optimization problem with boundary restrictions can be defined as

Minimum f(x)

Boundary restrictions : p(x)=0

q(x) * 0

In general these type of boundary restrictions are added up into the general equation as

For the first equation p(x)=0 define

$$P(x) = P(x_0, x_1, ..) = \sum_{k=0}^{r} p_k^2\,(x)$$

For the second equation q(x) ≥ 0 için

$$Q(x) = Q(x_0, x_1, ..) = \sum_{k=0}^{r} min^2\{\,0, q_k\,(x)\}$$

Function min selects the minimum of 0 and $q_k(x)$.

The general function then becomes

$$F(x) = f(x) + \mu[P(x) + Q(x)]\ .$$

μ in the equation is called punishment coefficient. For the second equation for q(x) ≥ 0 equation Q(x) can be also written as

$$Q(x) = Q(x_0, x_1, ..) = \textstyle\sum_{k=0}^{r} min^2\{\,0, [q_k\,(x) - \varepsilon]\}\ .$$

In this equation ε >0 is a small number. With this small change refusal of the fiunction when the value is exactly equal to the restriction limit can be evoided. In the sample problem Fletcher – Reves Optimization method is used, but any unrestricted optimization methods can be used for this purpose.

Program 5.29-1 Non-Restricted boundary fletcher reeves optimization

```
import java.util.*;
import java.awt.*;
import java.applet.Applet;
import java.awt.event.*;
import javax.swing.*;

//OPTIMIZATION WITH BOUNDARY RESTRICTION
class f1 extends f_xj
{
public double func(double x[])
{
double ff;
double mu=0.0;
ff=(x[0]-1.0)*(x[0]-1.0)+(x[1]-2.0)*(x[1]-2.0)+(x[2]-3.0)*(x[2]-3.0)+mu*(P(x)+Q(x));
return ff;
}
public double P(double x[])
{// p(x)=0 boundary restriction
int n=x.length;
 return (x[0]+x[1]-2.0)*(x[0]+x[1]-2.0);
}

public double Q(double x[])
{// q(x)>=0 boundary restriction
double eps=1.0e-6;
int n=x.length;
double ax=x[1]-x[2]-3.0-eps;
double bx=0;
if(ax<0) bx=ax*ax;
 return bx;
}
}

public class NA116
{
public static double VT_V(double [] left)
{
//multiplys a vector transpose with a vector
int n=left.length;
double tot=0;
for(int i=0;i<n;i++)
  {
  tot+=left[i]*left[i];
  }
return tot;
}

public static double[][] V_VT(double [] left)
{
//multiplys a vector transpose with a vector
int n=left.length;
double aa[][]=new double[n][n];
for(int i=0;i<n;i++)
  {
  for(int j=0;j<n;j++)
    {
```

```
            aa[i][j]=left[i]*left[j];
            }
  }
return aa;
}

public static double norm(double v[])
{
// vector norm
double total=0;;
for(int i=0;i<v.length;i++)
  {total+=v[i]*v[i];}
return Math.sqrt(total);
}

public static double turev(f_xj f,double x[],int denklem_ref,int x_ref)
{
double h0=0.256808;
int i,m;
int n=7;
double f1[];
f1=new double[x.length];
double f2[];
f2=new double[x.length];
double x1[];
x1=new double[x.length];
double x2[];
x2=new double[x.length];
for(i=0;i<x.length;i++)
{
x1[i]=x[i];
x2[i]=x[i];
}
double T[][];
T=new double[n][n];
double h[];
h=new double[n];
for(i=0;i<n;i++)
{
  h[i]=0;
  for(int j=0;j<n;j++)
    T[i][j]=0;
}
h[0]=h0;
double r=0.5;
for( i=1;i<n;i++)
{
h[i]=h0*Math.pow(r,i);
}

for(i=0;i<n;i++)
{
x1[x_ref]+=h[i];
x2[x_ref]-=h[i]*1.0094847;
f1=df(f,x1);
f2=df(f,x2);
T[i][0]=( f1[denklem_ref] - f2[denklem_ref])/(2.0*h[i]);
```

```
x1[x_ref]=x[x_ref];
x2[x_ref]=x[x_ref];
}
for(m=1;m<n;m++)
{
  for(i=0;i<n-m;i++)
  {
  T[i][m]=(h[i]*h[i]*T[i+1][m-1] - h[i+m]*h[i+m]*T[i][m-1])/(h[i]*h[i]
  - h[i+m]*h[i+m]);
  }
}
double xx=T[0][n-1];
return xx;
}
public static double[] df(f_xj f,double xxi[])
{
//Gredyen vektor
// f(x0,x1,x2,...xn) founksiyonunun türev vektörü
// [df/dx0, df/dx1,df/dx2,.....,df/dxn]
// df/dxj  j=0...x.length
double a[]=new double[xxi.length];
for(int i=0;i<xxi.length;i++)
   {
   a[i]=turev(f,xxi,i);
   }
return a;
}
public static double turev(f_xj f,double xxi[],int x_ref)
{
// df/dxj
double h0=0.256808;
int i,m;
int n=7;
double x1[];
x1=new double[xxi.length];
double x2[];
x2=new double[xxi.length];
for(i=0;i<xxi.length;i++)
{
x1[i]=xxi[i];
x2[i]=xxi[i];
}
double T[][];
T=new double[n][n];
double h[];
h=new double[n];
for(i=0;i<n;i++)
{
  h[i]=0;
  for(int j=0;j<n;j++)
    T[i][j]=0;
}
h[0]=h0;
double r=0.5;
for( i=1;i<n;i++)
{
//h[i]=h0*Math.pow(r,i);
```

```
h0/=2.0;
h[i]=h0;
}
//first turev (difference formula)
for(i=0;i<n;i++)
{
x1[x_ref]+=h[i];
x2[x_ref]-=h[i];
T[i][0]=(f.func(x1)-f.func(x2))/(2.0*h[i]);
x1[x_ref]=xxi[x_ref];
x2[x_ref]=xxi[x_ref];
}

for(m=1;m<n;m++)
{
  for(i=0;i<n-m;i++)
  {
  T[i][m]=(h[i]*h[i]*T[i+1][m-1] -
  h[i+m]*h[i+m]*T[i][m-1])/(h[i]*h[i] - h[i+m]*h[i+m]);
  }
}
double xx=T[0][n-1];
return xx;
}
public static double[] turev(f_xj f_deriv,double x[])
{
// df/dxj  j=0...x.length
// This method calculates turev of a function with more than one independent variable.
// Accuracy of method can be adjusted by changing variables h0 and n
// function input should be in the form given in abstract class
// f_xj,j=0...x.length = df/dx(x_ref)
double a[]=new double[x.length];
for(int x_ref=0;x_ref<x.length;x_ref++)
   {
   a[x_ref]=turev(f_deriv,x,x_ref);
   }
return a;
}

public static double SIGN(double a,double b) {return (b > 0.0 ? Math.abs(a) : -Math.abs(a));}
public static double MAX(double a,double b) {return (a > b ? a : b);}


public static double linmin(f_xj f,double p[],double xi[])
{
            double tol=2.0e-4;
    int n=p.length;
    double pcom[],xicom[];
            int j;
            double xx,xmin,bx,ax;
            f1dim f1=new f1dim(f,p,xi);
            ax=0.0;
            xx=1.0;
            bx=2.0;
            double aa[]=mnbrak(f1,ax,xx,bx);
            ax=aa[0];
            xx=aa[1];
```

```
            bx=aa[2];
            xmin=brent(f1,ax,xx,bx,tol);
            return xmin;
}

public static double[][] linminiter(f_xj f,double p[],double xi[])
{   // yeni iterasyon noktasini verir
   int n=p.length;
   double xmin=linmin(f,p,xi);
   double aa[][]=new double[2][n];
            for (int j=0;j<n;j++)
            {   xi[j] *= xmin;
                        p[j] += xi[j];
                        aa[0][j]=p[j];
                        aa[1][j]=xi[j];
            }
   return aa;
}
public  static double[] brentf(f_x f,double ax,double bx,double cx,double tol)
{
// ax,bx,cx üç adet tahmin noktasi (bxax ile cx arasinda yer almalidir)
// f : fonksiyon (f_x abstract sinifinda tanimlanmiş)
// tol tolerans
int ITMAX=100;
double CGOLD=(3.0-Math.sqrt(5))/2.0; //altin oran
double ZEPS=1.0e-10;
double xmin;
double aa[]=new double[2];
// SHFT(double a,double b,double c,d) (a)=(b);(b)=(c);(c)=(d);
            int iter;
            double a,b,d,etemp,fu,fv,fw,fx,p,q,r,tol1,tol2,u,v,w,x,xm;
            double e=0.0;
            d=0.0;
            a=((ax < cx) ? ax : cx);
            b=((ax > cx) ? ax : cx);
            x=w=v=bx;
            fw=fv=fx=f.func(x);
            for (iter=1;iter<=ITMAX;iter++) {
                        xm=0.5*(a+b);
                        tol2=2.0*(tol1=tol*Math.abs(x)+ZEPS);
                        if (Math.abs(x-xm) <= (tol2-0.5*(b-a))) {
                                    xmin=x;
                  aa[0]=xmin;
                  aa[1]=fx;
                  return aa;
                        }
                        if (Math.abs(e) > tol1) {
                                    r=(x-w)*(fx-fv);
                                    q=(x-v)*(fx-fw);
                                    p=(x-v)*q-(x-w)*r;
                                    q=2.0*(q-r);
                                    if (q > 0.0) p = -p;
                                    q=Math.abs(q);
                                    etemp=e;
                                    e=d;
                                    if (Math.abs(p) >= Math.abs(0.5*q*etemp) || p <= q*(a-x) || p >= q*(b-x))
                                                d=CGOLD*(e=(x >= xm ? a-x : b-x));
```

```
                        else {
                                d=p/q;
                                u=x+d;
                                if (u-a < tol2 || b-u < tol2)
                                        d=SIGN(tol1,xm-x);
                        }
                } else {
                        d=CGOLD*(e=(x >= xm ? a-x : b-x));
                }
                u=(Math.abs(d) >= tol1 ? x+d : x+SIGN(tol1,d));
                fu=f.func(u);
                if (fu <= fx) {
                        if (u >= x) a=x; else b=x;
                        {v=w;w=x;x=u;}
                        {fv=fw;fw=fx;fx=fu;}
                } else {
                        if (u < x) a=u; else b=u;
                        if (fu <= fw || w == x) {
                                v=w;
                                w=u;
                                fv=fw;
                                fw=fu;
                        } else if (fu <= fv || v == x || v == w) {
                                v=u;
                                fv=fu;
                        }
                }
        }
        System.out.println("BRENT metodunda maksimum iterasyon sayisi aşildi");
        xmin=x;
        //minumum değer a[0] ile fonksiyon değeri a[1] ile geridöndürülmektedir
        aa[0]=xmin;
        aa[1]=fx;
        return aa;
}
public  static double[] mnbrak(f_x f,double ax,double dx)
{
// dx ikinci derece polinom için gerekli iki ek noktayi oluşturmak için
// ana noktadan seçilen uzaklik
return mnbrak(f,ax,(ax-dx),(ax+dx));
}

public  static double[] mnbrak(f_x f,double ax)
{
// dx ikinci derece polinom için gerekli iki ek noktayi oluşturmak için
// ana noktadan seçiln uzaklik
return mnbrak(f,ax,(ax-0.1),(ax+0.1));
}

public  static double[] mnbrak(f_x f,double ax,double bx,double cx)
{
// ax,bx,cx üç adet tahmin noktasi verildiğinde içinde minimum bulunan ax,bx,cx üç adetnoktayi bize arama
// sonucu bulur
// f : fonksiyon (f_x abstract sinifinda tanimlanmiş)
// tol tolerans
int ITMAX=100;
double xmin;
```

```
double GOLD=(Math.sqrt(5.0)+1.0)/2.0;;
double GLIMIT=100.0;
double TINY=1.0e-20;
double fa,fb,fc;
fa=0;
fb=0;
fc=0;
        double ulim,u,r,q,fu,dum;
        double aa[]=new double[3];
        fa=f.func(ax);
        fb=f.func(bx);
        if (fb > fa) {
                //SHFT(dum,*ax,*bx,dum)
                {dum=ax;ax=bx;bx=dum;}
                //SHFT(dum,*fb,*fa,dum)
                {dum=fb;fb=fa;fa=dum;}
        }
        cx=(bx)+GOLD*(bx-ax);
        fc=f.func(cx);
                aa[0]=ax;
                aa[1]=bx;
                aa[2]=cx;
        while (fb > fc) {
                r=(bx-ax)*(fb-fc);
                q=(bx-cx)*(fb-fa);
                u=(bx)-((bx-cx)*q-(bx-ax)*r)/
                        (2.0*SIGN(MAX(Math.abs(q-r),TINY),q-r));
                ulim=(bx)+GLIMIT*(cx-bx);
                if ((bx-u)*(u-cx) > 0.0) {
                        fu=f.func(u);
                        if (fu < fc) {
                                ax=bx;
                                bx=u;
                                fa=fb;
                                fb=fu;
                                return aa;
                        } else if (fu > fb) {
                                cx=u;
                                fc=fu;
                                return aa;
                        }
                        u=(cx)+GOLD*(cx-bx);
                        fu=f.func(u);
                } else if ((cx-u)*(u-ulim) > 0.0) {
                        fu=f.func(u);
                        if (fu < fc) {
                                //SHFT(bx,cx,u,cx+GOLD*(cx-bx))
                                {bx=cx;cx=u;u=cx+GOLD*(cx-bx);}
                                //SHFT(fb,fc,fu,f.func(u))
                                {fb=fc;fc=fu;fu=f.func(u);}
                        }
                } else if ((u-ulim)*(ulim-cx) >= 0.0) {
                        u=ulim;
                        fu=f.func(u);
                } else {
                        u=cx+GOLD*(cx-bx);
                        fu=f.func(u);
```

```
                        }
                        //SHFT(ax,bx,cx,u)
                        {ax=bx;bx=cx;cx=u;}
                        //SHFT(fa,fb,fc,fu)
                        {fa=fb;fb=fc;fc=fu;}
                        aa[0]=ax;
                        aa[1]=bx;
                        aa[2]=cx;
                }
                        return aa;
}
public  static double brent(f_x f,double ax,double bx,double cx,double tol)
{
 double aa[]=brentf(f,ax,bx,cx,tol);
 return aa[0];
}
// 4. fletcher Reeves metodu
//_____________________________________________________________________
public static double[] fletcher_reeves( f_xj f,double x[])
{
int i,j,ii,jj,k;
int nmax=500;
double gamma=1.0;
double tolerance=1.0e-4;
int n=x.length;
double alpha;
double g[]=new double[n];
double ge[]=new double[n];
double d[]=new double[n];
double beta;
double alpha_d;
double Q[]=new double[n];
int nn=15;
i=0;
j=0;k=0;
while( j++ < nmax && gamma > tolerance )
{
   g=turev(f,x);
   gamma=norm(g);
   if(k<1) {for(i=0;i<n;i++){d[i]=-g[i];};ge=g;}
   else
   {beta=VT_V(g)/VT_V(ge);
         for(i=0;i<n;i++){Q[i]=beta*d[i];};
   }
   for(i=0;i<n;i++){d[i]=-g[i]+Q[i];}
   alpha=linmin(f,x,d);
   for(i=0;i<n;i++){x[i]+=alpha*d[i];}
j++;k++;
if(k>=nn) k=0;
}
if(j >= nmax) JOptionPane.showMessageDialog(null,"Uyari maximum iterasyon sayisi aşildi \n"+
  " çözüm geçerli olmiyabilir","MAKSİMUM ITERASYON SAYISI UYARISI",JOptionPane.WARNING_MESSAGE);
return x;
}
              public static double[] verigir(String s)
    {
     String s1=JOptionPane.showInputDialog(s);
```

```
    StringTokenizer token=new StringTokenizer(s1);
    int n=token.countTokens()-1;
    int m=n+1;
    double a[]=new double[m];
    int j=0;
    while(token.hasMoreTokens())
    {
    Double ax=new Double(token.nextToken());
    a[j++]=ax.doubleValue();
    }
    return a;
    }

    public static void main (String args[])
    {
   String s="enter initial root estimations with one space n between: ";
    double [] x0=verigir(s);
    f1 f_x=new f1();
    double [] r1= fletcher_reeves(f_x,x0);
    s=" çözüm seti  : \n"+Matrix.toStringT(r1);
    String s2="Fletcher-Reeves multidimensional restricted boundary optimization:";
    JOptionPane.showMessageDialog(null,s,s2,JOptionPane.PLAIN_MESSAGE);
    System.out.println("y="+f_x.func(r1));
    System.exit(0);
    }
}
```

Our first example is without any restriction our equation is :

$\mathbf{f(x)=(x_0-1.0)^2+(x_1-2.0)^2+(x_2-3.0)^2}$



Now two boundary restriction will apply to the same function

$\mathbf{p(x) = x_0 + x_1 - 2.0=0}$

$\mathbf{q(x) = x_1 - x_2 - 3.0 - \varepsilon \geq 0}$

with this boundary restrictions equation becomes

$\mathbf{f(x)=(x_0-1.0)^2+(x_1-2.0)^2+(x_2-3.0)^2+\mu(p^2(x)+q^2(x))}$

For $\mathbf{\mu=10}$



For **μ=100**



As it is seen from the result for different values of **μ** the results are differs. **μ** is the weight function and the effects of restrictions increased when a bigger value of **μ** is used.

5.30 TRAVELLING SALESMAN PROBLEM (TSP)

Travelling salesman problem is a special kind of optimization problem. Problem satement is relatively simple. A travelling salesman will have to visit n cities in order to sell his products. The coordinates of the cities are set. How salesman can make the shortest travel? Problem can be considered as two basic form. In the first form starting and ending cities can be set. In the second form nothing is set.
The simplest way of the definitions of the function:

$$\text{Distance}=\sum_{i=1}^{n}[(x_i - x_{i-1})^2 + (y_i - y_{i-1})^2]^{1/2}$$

In this equation x_i and y_i is coordinates of nth city . If the same Formula is written for a constant starting and ending cities

$$Dis\,tan\,c\,e = [(x_0 - x_{start})^2 + (y_0 - y_{start})^2]^{1/2} + \sum_{i=1}^{n}[(x_i - x_{i-1})^2 + (y_i - y_{i-1})^2]^{1/2} + \left[(x_{final} - x_n)^2 + (y_{final} - y_n)^2\right]^{1/2}$$

In this equation (x_{start}, y_{start}) is the starting city coordinate and (x_{final}, y_{final}) is the destination city coordinate. In the normal version of the TSP problem every city must be travelled only once, but in some real problem this restriction does not have to be applied. The logical exact solution seems to be measured the distance from first city to the distance of all remaining cities travel there and continue on the process, but when the number of possibilities are concerned it is not practical. It should be also note that the distance between the cities does not have to fit the linear approach given in here (actual roads can be longer than the linear path). In this case a cross-reference table of the distances should be given to calculate it. Case equation will be:

$$\text{Distance}=\sum_{i=1}^{n}\left[d_{i,i-1}\right]^{1/2}$$

where $d_{i,(i-1)}$ given as a distance table (a matrix)

In here linear path will be assumed.

5.30.1 SOLUTION OF TSP BY GENETIC ALGORITHMS

Genetic algorithms can be applied to the TSP problem but, the definition of the gene structure and breeding procedures should be defined specifically to this problem, previous definition of Genetic algorithms was valid for a continious problem, but this of is a combination problem. The basic gene structure is formed by using ArrayList vector structure of Java. In order to do that a list of city corrdinates are needed as input. For example assume that the following city coordinates are given (it is stored in the file cities.txt)

```
100.0        0.0
100.0        0.0
95.10565162951535  30.901699437494738
80.90169943749474  58.778525229247315
58.778525229247315 80.90169943749474
30.901699437494745 95.10565162951535
6.123233995736766E-15          100.0
-30.901699437494734            95.10565162951536
-58.7785252292473   80.90169943749474
-80.90169943749473 58.77852522924732
-95.10565162951535 30.901699437494752
-100.0       1.2246467991473532E-14
-95.10565162951538 -30.90169943749469
-80.90169943749474 -58.7785252292473
-58.77852522924732 -80.90169943749473
-30.901699437494756            -95.10565162951535
-1.8369701987210297E-14        -100.0
30.901699437494724 -95.10565162951536
58.77852522924729  -80.90169943749476
80.90169943749473  -58.778525229247336
95.10565162951535  -30.901699437494763
```

In this list the first line is the starting cities coordines, the second line is destination cities coordinates. The remaining is the cities to be visited in between starting and destination. The cities will be given a coordinate code according to the spacing in the given list. For example index of the third city (95.10565162951535, 30.901699437494738) is 0, the fourth city (80.90169943749474,58.778525229247315) is 1 etc. In the Array list visiting order of the index is tha basic gene combination. For example assume that ArrayList contains the following list

2	7	3	8	1	4	6	5

The actual travel is

Starting city-2-7-3-8-1-4-6-5-destination city. List can be arranged so that a repetation of the same city is restricted. Program mainly consist of two basic program TSP_gene and TSP_genetic. TSP_gene defines the gene structure as given above, breeding and mutation processes aealso included. Each individual has only one gene. The second program TSP_Genetic defines the defines genetic processes (select parents, breed, mutate new generations etc.) . The best solution is kept and copied to the new generation. A plot method is added to treck the path of the travel. List of numbers does not have much meaning in a case like this. It should be noted that due to statistical nature of GA the best result or the same result in each run should not be expected from the method.

Program 5.30-1 solution of TSP by genetic algorithms, tsp_gene class

```
import java.util.*;
import java.awt.*;
import javax.swing.*;
import java.awt.event.*;
//çikiş ve variş noktalari belirlenmiş tur
public class TSP_gene
{
public int n;
public double xcikilan,ycikilan;
public double xvarilan,yvarilan;
public double x[],y[];
public double Fitness;    // uyum değeri
public double RFitness;   // relatif uyum
public double CFitness;   // kümülatif uyum
public boolean Breeder;
ArrayList<String> v;
String dizi;
public TSP_gene(double xi[],double yi[])
{n=xi.length-2;
v=new ArrayList<String>(n);
x=new double[n];
y=new double[n];
xcikilan=xi[0];
ycikilan=yi[0];
xvarilan=xi[1];
yvarilan=yi[1];
Breeder=false;
for(int i=0;i<n;i++) {x[i]=xi[i+2];y[i]=yi[i+2];v.add(""+i);}
Collections.shuffle(v);
//swap();
distance();
}

public void mutate()
{
int n1=(int)(n*Math.random());
int n2=(int)(n*Math.random());
//System.out.println("n="+n+"n1="+n1+"n2="+n2);
if(n1!=n2) Collections.swap(v, n1, n2);
}


public void swap()
{int pos[]=new int[n];
 int key[]=new int[n];
 int p,k;
 int n1=0;
for(int i=0;i<n;i++)
{ p=Collections.binarySearch(v,""+i);
  if(p==i) {pos[i]=p;n1++;}
}
for(int i=0;i<n1;i+=2)
{Collections.swap(v,pos[i],pos[i+1]);}
}

public TSP_gene(TSP_gene anne)
```

```
{n=anne.n;
 v=kopye(anne.v);
 xcikilan=anne.xcikilan;
 ycikilan=anne.ycikilan;
 xvarilan=anne.xvarilan;
 yvarilan=anne.yvarilan;
 x=new double[n];
 y=new double[n];
 for(int i=0;i<n;i++) {x[i]=anne.x[i];y[i]=anne.y[i];}
 distance();
}

public TSP_gene(int a[],double xi[],double yi[])
{
ArrayList<String> anne=add(a);
v=kopye(anne);
Collection torba=v;
n=xi.length-2;
xcikilan=xi[0];
ycikilan=yi[0];
xvarilan=xi[1];
yvarilan=yi[1];
x=new double[n];
y=new double[n];
for(int i=0;i<n;i++) {x[i]=xi[i+2];y[i]=yi[i+2];}
distance();
}

public TSP_gene(String dosya)
{double a[][]=Text.readDouble(dosya);
n=a[0].length-2;
v=new ArrayList<String>(n);
x=new double[n];
y=new double[n];
xcikilan=a[0][0];
ycikilan=a[1][0];
xvarilan=a[0][1];
yvarilan=a[1][1];
for(int i=0;i<n;i++) {x[i]=a[0][i+2];y[i]=a[1][i+2];v.add(""+i);}
Collections.shuffle(v);
distance();
}

   public TSP_gene(TSP_gene anne, TSP_gene baba)
   {
   n=anne.n;
   xcikilan=anne.xcikilan;
   ycikilan=anne.ycikilan;
   xvarilan=anne.xvarilan;
   yvarilan=anne.yvarilan;
   x=new double[n];
   y=new double[n];
   for(int i=0;i<n;i++)
   {x[i]=anne.x[i];y[i]=anne.y[i];}
   ArrayList<String> cocuk=s_cocuk(anne.v,baba.v);
            v=kopye(cocuk);
   distance();
```

```
        }

public TSP_gene(ArrayList<String> anne,double xi[],double yi[])
{
v=kopye(anne);
Collection torba=v;
n=xi.length-2;
xcikilan=xi[0];
ycikilan=yi[0];
xvarilan=xi[1];
yvarilan=yi[1];
x=new double[n];
y=new double[n];
for(int i=0;i<n;i++) {x[i]=xi[i+2];y[i]=yi[i+2];}
distance();
}

public void plot()
{
double xi[]=new double[n+2];
double yi[]=new double[n+2];
   int a=0;
   int b=0;
        String s1="";
        String s2="";
   Iterator i=v.iterator();
   int j=0;
   double distance;
   dizi="0";
   s1=(String)i.next();
   a=Integer.parseInt(s1);
   distance=Math.sqrt((x[a]-xcikilan)*(x[a]-xcikilan)+(y[a]-ycikilan)*(y[a]-ycikilan));
   xi[j]=xcikilan;
   yi[j]=ycikilan;
   j++;
   xi[j]=x[a];
   yi[j]=y[a];
   j++;
   while(i.hasNext())
   {
   s2=(String)i.next();
   b=Integer.parseInt(s2);

   distance+=Math.sqrt((x[a]-x[b])*(x[a]-x[b])+(y[a]-y[b])*(y[a]-y[b]));
   xi[j]=x[b];
   yi[j]=y[b];
   j++;
   a=b;
   }
   xi[j]=xvarilan;
   yi[j]=yvarilan;
   distance+=Math.sqrt((x[b]-xvarilan)*(x[b]-xvarilan)+(y[b]-yvarilan)*(y[b]-yvarilan));
   Fitness=1.0/distance;
   Plot pp=new Plot(xi,yi);
   double wx[]=new double[2];
   double wy[]=new double[2];
   wx[0]=xi[0];
```

```
    wy[0]=yi[0];
    wx[1]=xi[n+1];
    wy[1]=yi[n+1];
    pp.addData(wx,wy);
    pp.addData(xi,yi);
    pp.setPlotType(1,32);
    pp.setPlotType(2,22);
    pp.setPlabel("Travellin Salesman Problem distance = "+distance);
    pp.setXlabel("x, km");
    pp.setYlabel("y, km");
    pp.plot();
}

public double distance()
{
    int a=0;
    int b=0;
            String s1="";
            String s2="";
    Iterator i=v.iterator();
    int j=0;
    double distance;
    dizi="0";
    s1=(String)i.next();
    a=Integer.parseInt(s1);
    distance=Math.sqrt((x[a]-xcikilan)*(x[a]-xcikilan)+(y[a]-ycikilan)*(y[a]-ycikilan));
    while(i.hasNext())
    {
    s2=(String)i.next();
    b=Integer.parseInt(s2);
    distance+=Math.sqrt((x[a]-x[b])*(x[a]-x[b])+(y[a]-y[b])*(y[a]-y[b]));
    a=b;
    }
    distance+=Math.sqrt((x[b]-xvarilan)*(x[b]-xvarilan)+(y[b]-yvarilan)*(y[b]-yvarilan));
    Fitness=1.0/distance;
    return distance;
}

public void setFitness()
{Fitness=1.0/distance();}

public double getFitness()
{return Fitness;}

public void setRFitness(double RFitnessi)
{RFitness=RFitnessi;}

public double getRFitness()
{return RFitness;}

public void setCFitness(double CFitnessi)
{CFitness=CFitnessi;}

public double getCFitness()
{return CFitness;}
  public static String print(Collection c)
  {
```

```
 String s="";
 Iterator i=c.iterator();
 while(i.hasNext()) {s+=i.next()+"\n";}
 return s;
}

public String dizi()
{return print(v);}

public ArrayList<String>kopye(ArrayList<String>list1)
 {
    ArrayList<String> bilesimVectoru = new ArrayList<String>();
    String s1,s2;
    Iterator n1=list1.iterator();
    while(n1.hasNext())
    {
       s1=(String)n1.next();
       bilesimVectoru.add(s1);
    }
    return bilesimVectoru;
 }

 public static ArrayList<String> add(int a[])
 {
    ArrayList<String> bilesimVectoru = new ArrayList<String>();
    String s1;
    for(int i=0;i<a.length;i++)
    {
       s1=""+a[i];
       if(!bilesimVectoru.contains(s1))
          bilesimVectoru.add(s1);
    }
    return bilesimVectoru;
 }

 public ArrayList<String> bilesim(ArrayList<String>list1, int a)
 {
    ArrayList<String> bilesimVectoru = new ArrayList<String>();
    String s1,s2;
    Iterator n1=list1.iterator();
    String b=new String(""+a);
    while(n1.hasNext())
    {
       s1=(String)n1.next();
       bilesimVectoru.add(s1);
    }
    s2=(String)b;
       if(!bilesimVectoru.contains(s2))
          bilesimVectoru.add(s2);
    return bilesimVectoru;
 }

 public ArrayList<String> bilesim(ArrayList<String> list1, ArrayList<String> list2)
 {
    ArrayList<String> bilesimVectoru = new ArrayList<String>();
    String s1,s2;
    Iterator n1=list1.iterator();
```

```
    Iterator n2=list2.iterator();
    while(n1.hasNext())
    {
       s1=(String)n1.next();
       bilesimVectoru.add(s1);
    }
    while(n2.hasNext())
    {
       s2=(String)n2.next();
       if(!bilesimVectoru.contains(s2))
          bilesimVectoru.add(s2);
    }
    return bilesimVectoru;
}

public  ArrayList<String> altküme(ArrayList<String> list1, int nn)
{
    ArrayList<String> bilesimVectoru = new ArrayList<String>();
    String s1,s2;
    Iterator n1=list1.iterator();
    for(int i=0;i<nn;i++)
    {
       s1=(String)n1.next();
       bilesimVectoru.add(s1);
    }
    return bilesimVectoru;
}

public  ArrayList<String> kalanküme(ArrayList<String> list1, int nn)
{
    ArrayList<String> bilesimVectoru = new ArrayList<String>();
    String s1,s2;
    Iterator n1=list1.iterator();

    for(int i=0;i<nn;i++)
    { s1=(String)n1.next();}

    while(n1.hasNext())
    {
       s1=(String)n1.next();
       bilesimVectoru.add(s1);
    }
    return bilesimVectoru;
}

public  ArrayList<String> s_cocuk(ArrayList<String> anne, ArrayList<String> baba)
{int nn=(int)(n*Math.random())+1;
//System.out.println("nn="+nn);
          ArrayList<String> anne1=altküme(anne,nn);
          ArrayList<String> anne2=kalanküme(anne,nn);
          ArrayList<String> baba1=altküme(baba,nn);
          ArrayList<String> baba2=kalanküme(baba,nn);
          ArrayList<String> cocuk=bilesim(anne1,baba1);
          ArrayList<String> cocuk1=bilesim(cocuk,anne2);
          ArrayList<String> cocuk2=bilesim(cocuk1,baba2);
          return cocuk2;
          }
```

```
        public String ciktiS()
   {
      Iterator enm = v.iterator();
      String s="  ("+xcikilan+","+ycikilan+")\n";
      String s1="";
      int a=0;
      while(enm.hasNext())
      {  s1=(String)enm.next();
        a=Integer.parseInt(s1);
              s+=s1+" ("+x[a]+","+y[a]+") \n";
            }
            s+="  ("+xvarilan+","+yvarilan+") \n";
   return s;
   }

        public String[] ciktiSA()
   {
      Iterator enm = v.iterator();
      String s[]=new String[n];
      int i=0;
      while(enm.hasNext())
      {  s[i]= (String)enm.next()+" ";i++;}
   return s;
   }

   public String toString()
   {
        String s=ciktiS()+" distance = "+distance();
   return s;
   }
}
```

Program 5.30-2 solution of TSP by genetic algorithms, tsp_genetic class

```
import java.io.*;
import java.text.*;
import java.util.Locale;

class TSP_genetic
{
public TSP_gene G[];    // string of variables makes a genotype1
public boolean Breeder;   //
public int    POPSIZE;          // population size
int    MAXGENS;          // maximum number of generations;
double PMUTATION;        // proportion of mutated variables
double PXOVER;           // probability of crossover;
int    Generation;       // Current generation number
int    NBreeder;   //   Number of survivors in a generation
int    Best; //The best genotype1 in the population
double toplameniyi=0;
double nesileniyi=0;
double xi[],yi[]; //City coordinates

  public TSP_genetic(String name,int POPSİZEi,int MAXGENSi,double PXOVERi,double PMUTATIONi )
  {double a[][]=Text.readDoubleT(name);
   double xi[]=a[0];
```

```
  double yi[]=a[1];
  int best;
  POPSIZE=POPSİZEi;
  PXOVER=PXOVERi;
  MAXGENS=MAXGENSi;
  PMUTATION=PMUTATIONi;
  G=new TSP_gene[POPSIZE+1];
  double totalfitness=0;
  for(int i=0;i<(POPSIZE+1);i++)
  {G[i]=new TSP_gene(xi,yi);G[i].setFitness();totalfitness+=G[i].getFitness();}
  double xR,xC=0;
  for(int i=0;i<(POPSIZE+1);i++)
  {xR=G[i].getRFitness()/totalfitness;xC+=xR;G[i].setRFitness(xR);G[i].setCFitness(xC);}
 }

 public void setPopulation()
 {
  double totalfitness=0;
  for(int i=0;i<POPSIZE;i++)
  {G[i]=new TSP_gene(xi,yi);}
   evaluate();
 }

 public void evaluate()
 {
  nesileniyi=0;
  double totalfitness=0;
  for(int i=0;i<POPSIZE+1;i++)
  {G[i].setFitness();
   double distance=1.0/G[i].getFitness();
   if(distance>nesileniyi) nesileniyi=distance;
   if(distance>toplameniyi) toplameniyi=distance;
   totalfitness+=G[i].getFitness();
  }
  double xR=0,xC=0,xRP=G[POPSIZE].getFitness()/totalfitness;
  for(int i=0;i<POPSIZE;i++)
  {xR=G[i].getFitness()/totalfitness;xC+=xR;G[i].setRFitness(xR);G[i].setCFitness(xC);
              if(xR > xRP)
                 {
                          Best=i;
                          G[POPSIZE]=new TSP_gene(G[i]);
                          xRP=G[POPSIZE].getFitness()/totalfitness;
       }
  }
 }
 public TSP_gene[] copyPopulation()
 {
             TSP_gene Po[]=new TSP_gene[POPSIZE+1];
             for(int i=0;i<(POPSIZE+1);i++)
               { Po[i]=new TSP_gene(G[i]);
                 Po[i].Fitness=G[i].Fitness;
                          Po[i].RFitness=G[i].RFitness;
                 Po[i].CFitness=G[i].CFitness;
               }
             return Po;
 }
```

```
            public void select()
            {
            double r,r1,r2,gf0,gf1;
            evaluate();
            TSP_gene Po[]=new TSP_gene[POPSIZE+1];
            NBreeder=0;
            int i,j;
            for(i=0;i<(POPSIZE+1);i++)
   {   r=Math.random();
       gf0=0;
       for(j=0;j<POPSIZE;j++)
       {
       gf1=G[j].getCFitness();
       if((r>=gf0) && (r<=gf1))
       {Po[NBreeder]=new TSP_gene(G[j]);
       NBreeder++;break;}
       gf0=gf1;
       }
   }
     int breed=(int)(POPSIZE*PXOVER);
     for(i=0;i<breed;i++)
               {
                        int a=(int)(Math.random()*NBreeder);
          int b=(int)(Math.random()*NBreeder);
          G[i]=new TSP_gene(Po[a],Po[b]);
          r2=Math.random();
          //mutation probability : PMUTATION
                   if(r2 < PMUTATION) G[i].mutate();
               }
                  for(i=breed;i<POPSIZE;i++)
                  {  int a=(int)(Math.random()*NBreeder);
            G[i]=new TSP_gene(Po[a]);
                  }
   }

   public void generations()
   {
            for(int i=0;i<MAXGENS;i++) {select();}
            }

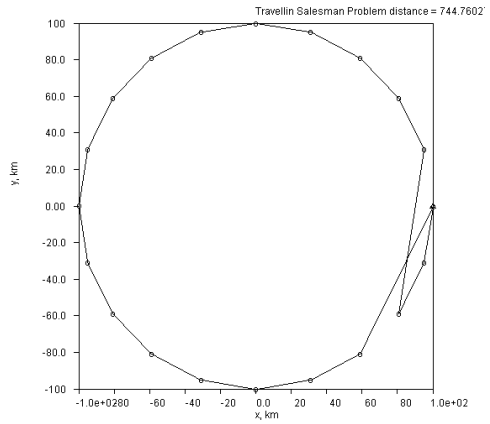
   public TSP_gene selected_generation()
   {
   generations();
   G[POPSIZE].plot();
   return G[POPSIZE];
            }


  public static void main(String arg[])
  {
   TSP_genetic sg=new TSP_genetic("cities.txt",1000,1000,1.0,0.05 );
   System.out.println("selected generation : "+sg.selected_generation().toString());
  }

}
```

```
---------- Capture Output ----------
```

```
> "C:\turhan\java\bin\javaw.exe" TSP_genetic
selected generation :   (100.0,0.0)
18 (95.10565162951535,-30.901699437494763)
17 (80.90169943749473,-58.778525229247336)
0 (95.10565162951535,30.901699437494738)
1 (80.90169943749474,58.778525229247315)
2 (58.778525229247315,80.90169943749474)
3 (30.901699437494745,95.10565162951535)
4 (6.123233995736766E-15,100.0)
5 (-30.901699437494734,95.10565162951536)
6 (-58.7785252292473,80.90169943749474)
7 (-80.90169943749473,58.77852522924732)
8 (-95.10565162951535,30.901699437494752)
9 (-100.0,1.2246467991473532E-14)
10 (-95.10565162951538,-30.90169943749469)
11 (-80.90169943749474,-58.7785252292473)
12 (-58.77852522924732,-80.90169943749473)
13 (-30.901699437494756,-95.10565162951535)
14 (-1.8369701987210297E-14,-100.0)
15 (30.901699437494724,-95.10565162951536)
16 (58.77852522924729,-80.90169943749476)
  (100.0,0.0)
 distance = 744.7602740406497
-142.35668789808918 122.48243559718966
```



Program 5.30-3 solution of TSP by genetic algorithms, tsp_gene1 class

```
   import java.util.*;
import java.awt.*;
import javax.swing.*;
import java.awt.event.*;
//çikiş ve variş noktalari belirlenmemiş tur
public class TSP_gene1
{
public int n;
public double x[],y[];
public double Fitness;    // uyum değeri
public double RFitness;   // relatif uyum
public double CFitness;   // kümülatif uyum
public boolean Breeder;
```

```
ArrayList<String> v;
String dizi;

public TSP_gene1(double xi[],double yi[])
{n=xi.length;
v=new ArrayList<String>(n);
x=new double[n];
y=new double[n];
Breeder=false;
for(int i=0;i<n;i++) {x[i]=xi[i];y[i]=yi[i];v.add(""+i);}
Collections.shuffle(v);
mesafe();
}

public void mutate()
{
int n1=(int)(n*Math.random());
int n2=(n1+1)%n;
//int n2=(int)(n*Math.random());
Collections.swap(v, n1, n2);
}


public void swap()
{int pos[]=new int[n];
 int key[]=new int[n];
 int p,k;
 int n1=0;
for(int i=0;i<n;i++)
{ p=Collections.binarySearch(v,""+i);
  if(p==i) {pos[i]=p;n1++;}
}
for(int i=0;i<n1;i+=2)
{Collections.swap(v,pos[i],pos[i+1]);}
}

public TSP_gene1(TSP_gene1 mother)
{n=mother.n;
 v=kopye(mother.v);
 x=new double[n];
 y=new double[n];
 for(int i=0;i<n;i++) {x[i]=mother.x[i];y[i]=mother.y[i];}
 mesafe();
}

public TSP_gene1(int a[],double xi[],double yi[])
{
ArrayList<String> mother=add(a);
v=kopye(mother);
Collection torba=v;
n=xi.length;
x=new double[n];
y=new double[n];
for(int i=0;i<n;i++) {x[i]=xi[i];y[i]=yi[i];}
mesafe();
}
```

```
public TSP_gene1(String dosya)
{double a[][]=Text.readDouble(dosya);
n=a[0].length;
v=new ArrayList<String>(n);
x=new double[n];
y=new double[n];
for(int i=0;i<n;i++) {x[i]=a[0][i];y[i]=a[1][i];v.add(""+i);}
Collections.shuffle(v);
mesafe();
}
        public TSP_gene1(TSP_gene1 mother, TSP_gene1 father)
   {
   n=mother.n;
   x=new double[n];
   y=new double[n];
   for(int i=0;i<n;i++)
   {x[i]=mother.x[i];y[i]=mother.y[i];}
   ArrayList<String> cocuk=s_cocuk(mother.v,father.v);
        v=kopye(cocuk);
   mesafe();
        }

public TSP_gene1(ArrayList<String> mother,double xi[],double yi[])
{
v=kopye(mother);
Collection torba=v;
n=xi.length;
x=new double[n];
y=new double[n];
for(int i=0;i<n;i++) {x[i]=xi[i];y[i]=yi[i];}
mesafe();
}

public void plot()
{
double xi[]=new double[n];
double yi[]=new double[n];
   int a=0;
   int b=0;
        String s1="";
        String s2="";
   Iterator i=v.iterator();
   int j=0;
   double mesafe=0;
   dizi="0";
   s1=(String)i.next();
   a=Integer.parseInt(s1);
   xi[j]=x[a];
   yi[j]=y[a];
   j++;
   while(i.hasNext())
   {
   s2=(String)i.next();
   b=Integer.parseInt(s2);
   mesafe+=Math.sqrt((x[a]-x[b])*(x[a]-x[b])+(y[a]-y[b])*(y[a]-y[b]));
   xi[j]=x[b];
```

```
    yi[j]=y[b];
    j++;
    a=b;
    }
    Plot pp=new Plot(xi,yi);
    double wx[]=new double[2];
    double wy[]=new double[2];
    wx[0]=xi[0];
    wy[0]=yi[0];
    wx[1]=xi[n-1];
    wy[1]=yi[n-1];
    pp.addData(wx,wy);
    pp.addData(xi,yi);
    pp.setPlotType(1,32);
    pp.setPlotType(2,22);
    pp.setPlabel("TSP problem distance = "+mesafe);
    pp.setXlabel("x, km");
    pp.setYlabel("y, km");
    pp.plot();
}

public double mesafe()
{
    int a=0;
    int b=0;
            String s1="";
            String s2="";
    Iterator i=v.iterator();
    int j=0;
    double mesafe;
    dizi="0";
    s1=(String)i.next();
    a=Integer.parseInt(s1);
    mesafe=0.0;
    while(i.hasNext())
    {
    s2=(String)i.next();
    b=Integer.parseInt(s2);
    mesafe+=Math.sqrt((x[a]-x[b])*(x[a]-x[b])+(y[a]-y[b])*(y[a]-y[b]));
    a=b;
    }
    Fitness=1.0/mesafe;
    return mesafe;
}

public void setFitness()
{Fitness=1.0/mesafe();}

public double getFitness()
{return Fitness;}

public void setRFitness(double RFitnessi)
{RFitness=RFitnessi;}

public double getRFitness()
{return RFitness;}
```

```
public void setCFitness(double CFitnessi)
{CFitness=CFitnessi;}

public double getCFitness()
{return CFitness;}
  public static String print(Collection c)
  {
    String s="";
    Iterator i=c.iterator();
    while(i.hasNext()) {s+=i.next()+"\n";}
    return s;
  }

  public String dizi()
  {return print(v);}

  public ArrayList<String>kopye(ArrayList<String>list1)
    {
       ArrayList<String> bilesimVectoru = new ArrayList<String>();
       String s1,s2;
       Iterator n1=list1.iterator();
       while(n1.hasNext())
       {
          s1=(String)n1.next();
          bilesimVectoru.add(s1);
       }
       return bilesimVectoru;
    }

    public static ArrayList<String> add(int a[])
    {
       ArrayList<String> bilesimVectoru = new ArrayList<String>();
       String s1;
       for(int i=0;i<a.length;i++)
       {
          s1=""+a[i];
          if(!bilesimVectoru.contains(s1))
             bilesimVectoru.add(s1);
       }
       return bilesimVectoru;
    }

    public ArrayList<String> bilesim(ArrayList<String>list1, int a)
    {
       ArrayList<String> bilesimVectoru = new ArrayList<String>();
       String s1,s2;
       Iterator n1=list1.iterator();
       String b=new String(""+a);
       while(n1.hasNext())
       {
          s1=(String)n1.next();
          bilesimVectoru.add(s1);
       }
       s2=(String)b;
          if(!bilesimVectoru.contains(s2))
             bilesimVectoru.add(s2);
       return bilesimVectoru;
```

```
}

public ArrayList<String> bilesim(ArrayList<String> list1, ArrayList<String> list2)
{
   ArrayList<String> bilesimVectoru = new ArrayList<String>();
   String s1,s2;
   Iterator n1=list1.iterator();
   Iterator n2=list2.iterator();
   while(n1.hasNext())
   {
      s1=(String)n1.next();
      bilesimVectoru.add(s1);
   }
   while(n2.hasNext())
   {
      s2=(String)n2.next();
      if(!bilesimVectoru.contains(s2))
         bilesimVectoru.add(s2);
   }
   return bilesimVectoru;
}

public  ArrayList<String> altküme(ArrayList<String> list1, int nn)
{
   ArrayList<String> bilesimVectoru = new ArrayList<String>();
   String s1,s2;
   Iterator n1=list1.iterator();
   for(int i=0;i<nn;i++)
   {
      s1=(String)n1.next();
      bilesimVectoru.add(s1);
   }
   return bilesimVectoru;
}

public  ArrayList<String> kalanküme(ArrayList<String> list1, int nn)
{
   ArrayList<String> bilesimVectoru = new ArrayList<String>();
   String s1,s2;
   Iterator n1=list1.iterator();

   for(int i=0;i<nn;i++)
   { s1=(String)n1.next();}

   while(n1.hasNext())
   {
      s1=(String)n1.next();
      bilesimVectoru.add(s1);
   }
   return bilesimVectoru;
}

public  ArrayList<String> s_cocuk(ArrayList<String> mother, ArrayList<String> father)
{int nn=(int)(n*Math.random())+1;
//System.out.println("nn="+nn);
         ArrayList<String> mother1=altküme(mother,nn);
         ArrayList<String> mother2=kalanküme(mother,nn);
```

```
            ArrayList<String> father1=altküme(father,nn);
            ArrayList<String> father2=kalanküme(father,nn);
            ArrayList<String> cocuk=bilesim(mother1,father1);
            ArrayList<String> cocuk1=bilesim(cocuk,mother2);
            ArrayList<String> cocuk2=bilesim(cocuk1,father2);
            return cocuk2;
           }

           public String ciktiS()
  {
    Iterator enm = v.iterator();
    String s="";
    int a=0;
    String s1="";
    while(enm.hasNext())
    {  s1=(String)enm.next();
      a=Integer.parseInt(s1);
              s+=s1+" ("+x[a]+","+y[a]+") \n";
            }
  return s;
  }

           public String[] ciktiSA()
  {
    Iterator enm = v.iterator();
    String s[]=new String[n];
    int i=0;
    while(enm.hasNext())
    {  s[i]= (String)enm.next()+" ";i++;}
  return s;
  }

  public String toString()
  {
           String s=ciktiS()+" mesafe = "+mesafe();
  return s;
  }
}
```

If starting and final destination is not important , TSP will change slightly

Program 5.25-3 solution of TSP by genetic algorithms, tsp_gene1 class

```
import java.io.*;
import java.text.*;
import java.util.Locale;
//
class TSP_genetic1
{
public TSP_gene1 G[];    // string of variables makes a genotype1
public boolean Breeder;   //
public int    POPSIZE;          // population size
int    MAXGENS;          // maximum number of generations;
double PMUTATION;        // proportion of mutated variables
double PXOVER;           // probability of crossover;
int    Generation;       // Current generation number
int    NBreeder;   //   Number of survivors in a generation
```

```
int    Best; //The best genotype1 in the population
double toplameniyi=0;
double nesileniyi=0;
double xi[],yi[]; //şehir adresleri

 public TSP_genetic1(String name,int POPSİZEi,int MAXGENSi,double PXOVERi,double PMUTATIONi )
 {double a[][]=Text.readDoubleT(name);
  double xi[]=a[0];
  double yi[]=a[1];
  int best;
  POPSIZE=POPSİZEi;
  PXOVER=PXOVERi;
  MAXGENS=MAXGENSi;
  PMUTATION=PMUTATIONi;
  G=new TSP_gene1[POPSIZE+1];
  double totalfitness=0;
  for(int i=0;i<(POPSIZE+1);i++)
  {G[i]=new TSP_gene1(xi,yi);G[i].setFitness();totalfitness+=G[i].getFitness();}
  double xR,xC=0;
  for(int i=0;i<(POPSIZE+1);i++)
  {xR=G[i].getRFitness()/totalfitness;xC+=xR;G[i].setRFitness(xR);G[i].setCFitness(xC);}
 }

 public void setPopulation()
 {
  double totalfitness=0;
  for(int i=0;i<POPSIZE;i++)
  {G[i]=new TSP_gene1(xi,yi);}
   evaluate();
 }

 public void evaluate()
 {
  nesileniyi=0;
  double totalfitness=0;
  for(int i=0;i<POPSIZE+1;i++)
  {G[i].setFitness();
   double mesafe=1.0/G[i].getFitness();
   if(mesafe>nesileniyi) nesileniyi=mesafe;
   if(mesafe>toplameniyi) toplameniyi=mesafe;
   totalfitness+=G[i].getFitness();
  }
  double xR=0,xC=0,xRP=G[POPSIZE].getFitness()/totalfitness;
  for(int i=0;i<POPSIZE;i++)
  {xR=G[i].getFitness()/totalfitness;xC+=xR;G[i].setRFitness(xR);G[i].setCFitness(xC);
            if(xR > xRP)
               {
                        Best=i;
                        G[POPSIZE]=new TSP_gene1(G[i]);
                        xRP=G[POPSIZE].getFitness()/totalfitness;
      }
 }
}
 public TSP_gene1[] copyPopulation()
 {
           TSP_gene1 Po[]=new TSP_gene1[POPSIZE+1];
           for(int i=0;i<(POPSIZE+1);i++)
```

```
            { Po[i]=new TSP_gene1(G[i]);
              Po[i].Fitness=G[i].Fitness;
                      Po[i].RFitness=G[i].RFitness;
              Po[i].CFitness=G[i].CFitness;
             }
           return Po;
}

           public void select()
           {
           double r,r1,r2,gf0,gf1;
           evaluate();
           //anne baba aday adaylarini seç
           TSP_gene1 Po[]=new TSP_gene1[POPSIZE+1];
           NBreeder=0;
           int i,j;
           for(i=0;i<(POPSIZE+1);i++)
  {   r=Math.random();
     gf0=0;
     for(j=0;j<POPSIZE;j++)
     {
     gf1=G[j].getCFitness();
     if((r>=gf0) && (r<=gf1))
     {Po[NBreeder]=new TSP_gene1(G[j]);
     //System.out.println("j= "+j+"gf0="+gf0+"gf1="+gf1+"r="+r+"NBreeder="+NBreeder);
     NBreeder++;break;}
     gf0=gf1;
     }
  }
  // seçilen anne, baba aday adaylarinin yeni çocuk yapmalarina olanak ver
   int breed=(int)(POPSIZE*PXOVER);
   for(i=0;i<breed;i++)
            {
                     int a=(int)(Math.random()*NBreeder);
       int b=(int)(Math.random()*NBreeder);
       G[i]=new TSP_gene1(Po[a],Po[b]);
       //System.out.println("a="+a+"b="+b+"cocuk="+G[i].toString());
       r2=Math.random();
       //mutasyon olasiliği : PMUTATION
              if(r2 < PMUTATION) G[i].mutate();
              //System.out.println(NBreeder+" "+G[POPSIZE].mesafe());
            }
           //listenin gerisini seçilen anne baba adaylariyla doldur
              for(i=breed;i<POPSIZE;i++)
              {  int a=(int)(Math.random()*NBreeder);
        G[i]=new TSP_gene1(Po[a]);
              }
  }

  public void generations()
  {
          for(int i=0;i<MAXGENS;i++) {select();}
          }

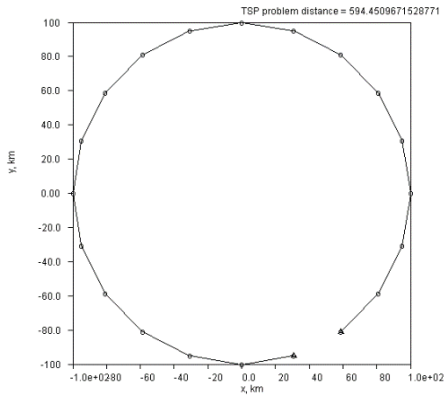
  public TSP_gene1 selected_generation()
  {
  generations();
```

```
   G[POPSIZE].plot();
   return G[POPSIZE];
            }

  public static void main(String arg[])
  {
    TSP_genetic1 sg=new TSP_genetic1("cities1.txt",1000,1000,1.0,0.03 );
    System.out.println("selected generation : "+sg.selected_generation().toString());
  }

}
```

```
---------- Capture Output ----------
> "C:\turhan\java\bin\javaw.exe" TSP_genetic1
Selected heneration : 16 (30.901699437494724,-95.10565162951536)
15 (-1.8369701987210297E-14,-100.0)
14 (-30.901699437494756,-95.10565162951535)
13 (-58.77852522924732,-80.90169943749473)
12 (-80.90169943749474,-58.7785252292473)
11 (-95.10565162951538,-30.90169943749469)
10 (-100.0,1.2246467991473532E-14)
9 (-95.10565162951535,30.901699437494752)
8 (-80.90169943749473,58.77852522924732)
7 (-58.7785252292473,80.90169943749474)
6 (-30.901699437494734,95.10565162951536)
5 (6.123233995736766E-15,100.0)
4 (30.901699437494745,95.10565162951535)
3 (58.778525229247315,80.90169943749474)
2 (80.90169943749474,58.778525229247315)
1 (95.10565162951535,30.901699437494738)
0 (100.0,0.0)
19 (95.10565162951535,-30.90169943749476)
18 (80.90169943749473,-58.778525229247336)
17 (58.77852522924729,-80.90169943749476)
 mesafe = 594.4509671528771
```



5.30-2 SOLUTION OF TSP BY MONTE-CARLO ALGORITHM

Because TSP is a combinatory problem Monte-Carlo Method, like direct combinatoral calculations, can not give us a good solution. In order to show that a Monte-Carlo algoritm is developed and tried

with very high number of iterations (100 million iteration). It is clear from the solution that, it is not enough to solve this problem only through randomness.

Program 5.30-4 solution of TSP by Monte-Carlo algorithms, tsp_gene1 class

```
import java.io.*;
import java.text.*;
import java.util.Locale;

class TSP_montecarlo
{
public TSP_gene G;    // string of variables makes a genotype1
public int    POPSIZE;          // population size
double xi[],yi[]; //şehir adresleri

  public TSP_montecarlo(String name,int POPSİZEi)
  {double a[][]=Text.readDoubleT(name);
   double xi[]=a[0];
   double yi[]=a[1];
   POPSIZE=POPSİZEi;
   TSP_gene G1=new TSP_gene(xi,yi);
   double toplameniyi=1e20;
   double distance;
   int counter;
   for(int i=0;i<POPSIZE;i++)
   {G1=new TSP_gene(xi,yi);
   distance=G1.distance();
   counter=i%1000000;
   if(counter==0) System.out.println("i="+i);
   if(distance < toplameniyi) {toplameniyi=distance;G=new TSP_gene(G1);System.out.println("i="+i+"\n"+G.toString());}
   }
   }

  public static void main(String arg[])
  {
    TSP_montecarlo sg=new TSP_montecarlo("cities.txt",100000000);
    //System.out.println("selected generations : "+sg.G.toString());
    sg.G.plot();
  }

}
```

```
---------- Capture Output ----------
> "C:\turhan\java\bin\javaw.exe" TSP_montecarlo
i=0
i=0
  (100.0,0.0)
17 (80.90169943749473,-58.778525229247336)
9 (-100.0,1.2246467991473532E-14)
10 (-95.10565162951538,-30.90169943749469)
7 (-80.90169943749473,58.77852522924732)
4 (6.123233995736766E-15,100.0)
15 (30.901699437494724,-95.10565162951536)
3 (30.901699437494745,95.10565162951535)
6 (-58.7785252292473,80.90169943749474)
1 (80.90169943749474,58.778525229247315)
```

2 (58.778525229247315,80.90169943749474)
8 (-95.10565162951535,30.901699437494752)
12 (-58.77852522924732,-80.90169943749473)
13 (-30.901699437494756,-95.10565162951535)
14 (-1.8369701987210297E-14,-100.0)
5 (-30.901699437494734,95.10565162951536)
16 (58.77852522924729,-80.90169943749476)
0 (95.10565162951535,30.901699437494738)
18 (95.10565162951535,-30.901699437494763)
11 (-80.90169943749474,-58.7785252292473)
 (100.0,0.0)
 distance = 2400.9357446270287
i=3
 (100.0,0.0)
17 (80.90169943749473,-58.778525229247336)
14 (-1.8369701987210297E-14,-100.0)
12 (-58.77852522924732,-80.90169943749473)
11 (-80.90169943749474,-58.7785252292473)
13 (-30.901699437494756,-95.10565162951535)
10 (-95.10565162951538,-30.90169943749469)
16 (58.77852522924729,-80.90169943749476)
6 (-58.7785252292473,80.90169943749474)
1 (80.90169943749474,58.778525229247315)
2 (58.778525229247315,80.90169943749474)
3 (30.901699437494745,95.10565162951535)
5 (-30.901699437494734,95.10565162951536)
9 (-100.0,1.2246467991473532E-14)
8 (-95.10565162951535,30.901699437494752)
0 (95.10565162951535,30.901699437494738)
18 (95.10565162951535,-30.901699437494763)
7 (-80.90169943749473,58.77852522924732)
4 (6.123233995736766E-15,100.0)
15 (30.901699437494724,-95.10565162951536)
 (100.0,0.0)
 distance = 2030.1843617772342
i=502
 (100.0,0.0)
0 (95.10565162951535,30.901699437494738)
2 (58.778525229247315,80.90169943749474)
12 (-58.77852522924732,-80.90169943749473)
13 (-30.901699437494756,-95.10565162951535)
10 (-95.10565162951538,-30.90169943749469)
7 (-80.90169943749473,58.77852522924732)
11 (-80.90169943749474,-58.7785252292473)
14 (-1.8369701987210297E-14,-100.0)
16 (58.77852522924729,-80.90169943749476)
15 (30.901699437494724,-95.10565162951536)
18 (95.10565162951535,-30.901699437494763)
4 (6.123233995736766E-15,100.0)
9 (-100.0,1.2246467991473532E-14)
6 (-58.7785252292473,80.90169943749474)
3 (30.901699437494745,95.10565162951535)
17 (80.90169943749473,-58.778525229247336)
8 (-95.10565162951535,30.901699437494752)
5 (-30.901699437494734,95.10565162951536)
1 (80.90169943749474,58.778525229247315)
 (100.0,0.0)

```
 distance = 2012.5374983077775
i=575
  (100.0,0.0)
2 (58.778525229247315,80.90169943749474)
0 (95.10565162951535,30.901699437494738)
1 (80.90169943749474,58.778525229247315)
3 (30.901699437494745,95.10565162951535)
5 (-30.901699437494734,95.10565162951536)
6 (-58.7785252292473,80.90169943749474)
4 (6.123233995736766E-15,100.0)
16 (58.77852522924729,-80.90169943749476)
7 (-80.90169943749473,58.77852522924732)
18 (95.10565162951535,-30.901699437494763)
8 (-95.10565162951535,30.901699437494752)
10 (-95.10565162951538,-30.90169943749469)
11 (-80.90169943749474,-58.7785252292473)
13 (-30.901699437494756,-95.10565162951535)
17 (80.90169943749473,-58.778525229247336)
9 (-100.0,1.2246467991473532E-14)
12 (-58.77852522924732,-80.90169943749473)
14 (-1.8369701987210297E-14,-100.0)
15 (30.901699437494724,-95.10565162951536)
  (100.0,0.0)
 distance = 1949.9796077260353
i=1148
  (100.0,0.0)
16 (58.77852522924729,-80.90169943749476)
17 (80.90169943749473,-58.778525229247336)
18 (95.10565162951535,-30.901699437494763)
2 (58.778525229247315,80.90169943749474)
11 (-80.90169943749474,-58.7785252292473)
12 (-58.77852522924732,-80.90169943749473)
10 (-95.10565162951538,-30.90169943749469)
8 (-95.10565162951535,30.901699437494752)
9 (-100.0,1.2246467991473532E-14)
13 (-30.901699437494756,-95.10565162951535)
5 (-30.901699437494734,95.10565162951536)
7 (-80.90169943749473,58.77852522924732)
6 (-58.7785252292473,80.90169943749474)
14 (-1.8369701987210297E-14,-100.0)
0 (95.10565162951535,30.901699437494738)
15 (30.901699437494724,-95.10565162951536)
1 (80.90169943749474,58.778525229247315)
3 (30.901699437494745,95.10565162951535)
4 (6.123233995736766E-15,100.0)
  (100.0,0.0)
 distance = 1945.2569392748203
i=1473
  (100.0,0.0)
18 (95.10565162951535,-30.901699437494763)
12 (-58.77852522924732,-80.90169943749473)
11 (-80.90169943749474,-58.7785252292473)
2 (58.778525229247315,80.90169943749474)
1 (80.90169943749474,58.778525229247315)
14 (-1.8369701987210297E-14,-100.0)
17 (80.90169943749473,-58.778525229247336)
16 (58.77852522924729,-80.90169943749476)
```

6 (-58.7785252292473,80.90169943749474)
9 (-100.0,1.2246467991473532E-14)
5 (-30.901699437494734,95.10565162951536)
7 (-80.90169943749473,58.77852522924732)
8 (-95.10565162951535,30.901699437494752)
3 (30.901699437494745,95.10565162951535)
4 (6.123233995736766E-15,100.0)
10 (-95.10565162951538,-30.90169943749469)
13 (-30.901699437494756,-95.10565162951535)
15 (30.901699437494724,-95.10565162951536)
0 (95.10565162951535,30.901699437494738)
 (100.0,0.0)
 distance = 1914.7548822898239
i=2518
 (100.0,0.0)
2 (58.778525229247315,80.90169943749474)
4 (6.123233995736766E-15,100.0)
3 (30.901699437494745,95.10565162951535)
5 (-30.901699437494734,95.10565162951536)
6 (-58.7785252292473,80.90169943749474)
7 (-80.90169943749473,58.77852522924732)
12 (-58.77852522924732,-80.90169943749473)
8 (-95.10565162951535,30.901699437494752)
9 (-100.0,1.2246467991473532E-14)
13 (-30.901699437494756,-95.10565162951535)
1 (80.90169943749474,58.778525229247315)
16 (58.77852522924729,-80.90169943749476)
15 (30.901699437494724,-95.10565162951536)
10 (-95.10565162951538,-30.90169943749469)
14 (-1.8369701987210297E-14,-100.0)
17 (80.90169943749473,-58.778525229247336)
18 (95.10565162951535,-30.901699437494763)
0 (95.10565162951535,30.901699437494738)
11 (-80.90169943749474,-58.7785252292473)
 (100.0,0.0)
 distance = 1909.6232492935656
i=4442
 (100.0,0.0)
0 (95.10565162951535,30.901699437494738)
2 (58.778525229247315,80.90169943749474)
1 (80.90169943749474,58.778525229247315)
8 (-95.10565162951535,30.901699437494752)
12 (-58.77852522924732,-80.90169943749473)
13 (-30.901699437494756,-95.10565162951535)
14 (-1.8369701987210297E-14,-100.0)
11 (-80.90169943749474,-58.7785252292473)
16 (58.77852522924729,-80.90169943749476)
18 (95.10565162951535,-30.901699437494763)
17 (80.90169943749473,-58.778525229247336)
9 (-100.0,1.2246467991473532E-14)
10 (-95.10565162951538,-30.90169943749469)
7 (-80.90169943749473,58.77852522924732)
3 (30.901699437494745,95.10565162951535)
5 (-30.901699437494734,95.10565162951536)
6 (-58.7785252292473,80.90169943749474)
4 (6.123233995736766E-15,100.0)
15 (30.901699437494724,-95.10565162951536)

(100.0,0.0)
distance = 1707.860830280625
i=9414
(100.0,0.0)
2 (58.778525229247315,80.90169943749474)
1 (80.90169943749474,58.778525229247315)
8 (-95.10565162951535,30.901699437494752)
7 (-80.90169943749473,58.77852522924732)
6 (-58.7785252292473,80.90169943749474)
3 (30.901699437494745,95.10565162951535)
4 (6.123233995736766E-15,100.0)
5 (-30.901699437494734,95.10565162951536)
15 (30.901699437494724,-95.10565162951536)
16 (58.77852522924729,-80.90169943749476)
18 (95.10565162951535,-30.901699437494763)
17 (80.90169943749473,-58.778525229247336)
9 (-100.0,1.2246467991473532E-14)
11 (-80.90169943749474,-58.7785252292473)
14 (-1.8369701987210297E-14,-100.0)
10 (-95.10565162951538,-30.90169943749469)
13 (-30.901699437494756,-95.10565162951535)
12 (-58.77852522924732,-80.90169943749473)
0 (95.10565162951535,30.901699437494738)
(100.0,0.0)
distance = 1644.5621964282616
i=14512
(100.0,0.0)
9 (-100.0,1.2246467991473532E-14)
13 (-30.901699437494756,-95.10565162951535)
14 (-1.8369701987210297E-14,-100.0)
11 (-80.90169943749474,-58.7785252292473)
12 (-58.77852522924732,-80.90169943749473)
10 (-95.10565162951538,-30.90169943749469)
8 (-95.10565162951535,30.901699437494752)
7 (-80.90169943749473,58.77852522924732)
4 (6.123233995736766E-15,100.0)
5 (-30.901699437494734,95.10565162951536)
1 (80.90169943749474,58.778525229247315)
2 (58.778525229247315,80.90169943749474)
0 (95.10565162951535,30.901699437494738)
18 (95.10565162951535,-30.901699437494763)
3 (30.901699437494745,95.10565162951535)
6 (-58.7785252292473,80.90169943749474)
17 (80.90169943749473,-58.778525229247336)
16 (58.77852522924729,-80.90169943749476)
15 (30.901699437494724,-95.10565162951536)
(100.0,0.0)
distance = 1630.24632213183
i=93035
(100.0,0.0)
4 (6.123233995736766E-15,100.0)
6 (-58.7785252292473,80.90169943749474)
5 (-30.901699437494734,95.10565162951536)
7 (-80.90169943749473,58.77852522924732)
12 (-58.77852522924732,-80.90169943749473)
8 (-95.10565162951535,30.901699437494752)
13 (-30.901699437494756,-95.10565162951535)

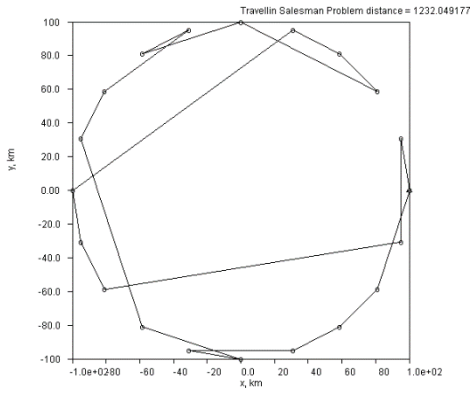
11 (-80.90169943749474,-58.7785252292473)
10 (-95.10565162951538,-30.90169943749469)
9 (-100.0,1.2246467991473532E-14)
16 (58.77852522924729,-80.90169943749476)
17 (80.90169943749473,-58.778525229247336)
15 (30.901699437494724,-95.10565162951536)
14 (-1.8369701987210297E-14,-100.0)
18 (95.10565162951535,-30.901699437494763)
2 (58.778525229247315,80.90169943749474)
3 (30.901699437494745,95.10565162951535)
1 (80.90169943749474,58.778525229247315)
0 (95.10565162951535,30.901699437494738)
 (100.0,0.0)
 distance = 1514.4486633644028
i=1000000
i=1399005
 (100.0,0.0)
16 (58.77852522924729,-80.90169943749476)
15 (30.901699437494724,-95.10565162951536)
14 (-1.8369701987210297E-14,-100.0)
13 (-30.901699437494756,-95.10565162951535)
11 (-80.90169943749474,-58.7785252292473)
12 (-58.77852522924732,-80.90169943749473)
1 (80.90169943749474,58.778525229247315)
2 (58.778525229247315,80.90169943749474)
8 (-95.10565162951535,30.901699437494752)
9 (-100.0,1.2246467991473532E-14)
7 (-80.90169943749473,58.77852522924732)
10 (-95.10565162951538,-30.90169943749469)
4 (6.123233995736766E-15,100.0)
5 (-30.901699437494734,95.10565162951536)
6 (-58.7785252292473,80.90169943749474)
3 (30.901699437494745,95.10565162951535)
17 (80.90169943749473,-58.778525229247336)
0 (95.10565162951535,30.901699437494738)
18 (95.10565162951535,-30.901699437494763)
 (100.0,0.0)
 distance = 1513.1324982330177
i=2000000
i=2171692
 (100.0,0.0)
18 (95.10565162951535,-30.901699437494763)
17 (80.90169943749473,-58.778525229247336)
13 (-30.901699437494756,-95.10565162951535)
9 (-100.0,1.2246467991473532E-14)
10 (-95.10565162951538,-30.90169943749469)
11 (-80.90169943749474,-58.7785252292473)
12 (-58.77852522924732,-80.90169943749473)
14 (-1.8369701987210297E-14,-100.0)
16 (58.77852522924729,-80.90169943749476)
1 (80.90169943749474,58.778525229247315)
6 (-58.7785252292473,80.90169943749474)
7 (-80.90169943749473,58.77852522924732)
8 (-95.10565162951535,30.901699437494752)
5 (-30.901699437494734,95.10565162951536)
2 (58.778525229247315,80.90169943749474)
4 (6.123233995736766E-15,100.0)

```
3 (30.901699437494745,95.10565162951535)
15 (30.901699437494724,-95.10565162951536)
0 (95.10565162951535,30.901699437494738)
  (100.0,0.0)
 distance = 1498.1779064811512
i=2419255
  (100.0,0.0)
0 (95.10565162951535,30.901699437494738)
1 (80.90169943749474,58.778525229247315)
3 (30.901699437494745,95.10565162951535)
6 (-58.7785252292473,80.90169943749474)
5 (-30.901699437494734,95.10565162951536)
2 (58.778525229247315,80.90169943749474)
4 (6.123233995736766E-15,100.0)
10 (-95.10565162951538,-30.90169943749469)
11 (-80.90169943749474,-58.7785252292473)
17 (80.90169943749473,-58.778525229247336)
18 (95.10565162951535,-30.901699437494763)
15 (30.901699437494724,-95.10565162951536)
14 (-1.8369701987210297E-14,-100.0)
13 (-30.901699437494756,-95.10565162951535)
12 (-58.77852522924732,-80.90169943749473)
8 (-95.10565162951535,30.901699437494752)
7 (-80.90169943749473,58.77852522924732)
9 (-100.0,1.2246467991473532E-14)
16 (58.77852522924729,-80.90169943749476)
  (100.0,0.0)
 distance = 1449.5497865351686
i=3000000
i=3436411
  (100.0,0.0)
0 (95.10565162951535,30.901699437494738)
1 (80.90169943749474,58.778525229247315)
2 (58.778525229247315,80.90169943749474)
5 (-30.901699437494734,95.10565162951536)
6 (-58.7785252292473,80.90169943749474)
11 (-80.90169943749474,-58.7785252292473)
9 (-100.0,1.2246467991473532E-14)
10 (-95.10565162951538,-30.90169943749469)
8 (-95.10565162951535,30.901699437494752)
3 (30.901699437494745,95.10565162951535)
4 (6.123233995736766E-15,100.0)
7 (-80.90169943749473,58.77852522924732)
14 (-1.8369701987210297E-14,-100.0)
13 (-30.901699437494756,-95.10565162951535)
12 (-58.77852522924732,-80.90169943749473)
15 (30.901699437494724,-95.10565162951536)
18 (95.10565162951535,-30.901699437494763)
16 (58.77852522924729,-80.90169943749476)
17 (80.90169943749473,-58.778525229247336)
  (100.0,0.0)
 distance = 1353.0320496763034
i=4000000
i=5000000
i=6000000
i=6469430
  (100.0,0.0)
```

```
18 (95.10565162951535,-30.901699437494763)
0 (95.10565162951535,30.901699437494738)
11 (-80.90169943749474,-58.7785252292473)
10 (-95.10565162951538,-30.90169943749469)
8 (-95.10565162951535,30.901699437494752)
6 (-58.7785252292473,80.90169943749474)
5 (-30.901699437494734,95.10565162951536)
4 (6.123233995736766E-15,100.0)
1 (80.90169943749474,58.778525229247315)
2 (58.778525229247315,80.90169943749474)
3 (30.901699437494745,95.10565162951535)
7 (-80.90169943749473,58.77852522924732)
9 (-100.0,1.2246467991473532E-14)
14 (-1.8369701987210297E-14,-100.0)
15 (30.901699437494724,-95.10565162951536)
12 (-58.77852522924732,-80.90169943749473)
13 (-30.901699437494756,-95.10565162951535)
16 (58.77852522924729,-80.90169943749476)
17 (80.90169943749473,-58.778525229247336)
  (100.0,0.0)
 distance = 1319.5094061059226
i=7000000
i=7827971
  (100.0,0.0)
2 (58.778525229247315,80.90169943749474)
4 (6.123233995736766E-15,100.0)
6 (-58.7785252292473,80.90169943749474)
8 (-95.10565162951535,30.901699437494752)
7 (-80.90169943749473,58.77852522924732)
5 (-30.901699437494734,95.10565162951536)
1 (80.90169943749474,58.778525229247315)
3 (30.901699437494745,95.10565162951535)
9 (-100.0,1.2246467991473532E-14)
12 (-58.77852522924732,-80.90169943749473)
10 (-95.10565162951538,-30.90169943749469)
11 (-80.90169943749474,-58.7785252292473)
13 (-30.901699437494756,-95.10565162951535)
14 (-1.8369701987210297E-14,-100.0)
16 (58.77852522924729,-80.90169943749476)
15 (30.901699437494724,-95.10565162951536)
17 (80.90169943749473,-58.778525229247336)
18 (95.10565162951535,-30.901699437494763)
0 (95.10565162951535,30.901699437494738)
  (100.0,0.0)
 distance = 1266.711996027475
i=8000000
i=9000000
i=10000000
i=11000000
i=12000000
i=13000000
i=14000000
i=15000000
i=16000000
i=17000000
i=18000000
i=19000000
```

```
i=20000000
i=21000000
i=22000000
i=23000000
i=24000000
i=25000000
i=26000000
i=27000000
i=27211995
  (100.0,0.0)
0 (95.10565162951535,30.901699437494738)
18 (95.10565162951535,-30.901699437494763)
11 (-80.90169943749474,-58.7785252292473)
10 (-95.10565162951538,-30.90169943749469)
9 (-100.0,1.2246467991473532E-14)
3 (30.901699437494745,95.10565162951535)
2 (58.778525229247315,80.90169943749474)
1 (80.90169943749474,58.778525229247315)
4 (6.123233995736766E-15,100.0)
6 (-58.7785252292473,80.90169943749474)
5 (-30.901699437494734,95.10565162951536)
7 (-80.90169943749473,58.77852522924732)
8 (-95.10565162951535,30.901699437494752)
12 (-58.77852522924732,-80.90169943749473)
14 (-1.8369701987210297E-14,-100.0)
13 (-30.901699437494756,-95.10565162951535)
15 (30.901699437494724,-95.10565162951536)
16 (58.77852522924729,-80.90169943749476)
17 (80.90169943749473,-58.778525229247336)
  (100.0,0.0)
 distance = 1232.0491774494658
i=28000000
i=29000000
i=30000000
i=31000000
i=32000000
i=33000000
i=34000000
i=35000000
i=36000000
i=37000000
i=38000000
i=39000000
i=40000000
i=41000000
i=42000000
i=43000000
i=44000000
i=45000000
i=46000000
i=47000000
i=48000000
i=49000000
i=50000000
i=51000000
i=52000000
i=53000000
```

```
i=54000000
i=55000000
i=56000000
i=57000000
i=58000000
i=59000000
i=60000000
i=61000000
i=62000000
i=63000000
i=64000000
i=65000000
i=66000000
i=67000000
i=68000000
i=69000000
i=70000000
i=71000000
i=72000000
i=73000000
i=74000000
i=75000000
i=76000000
i=77000000
i=78000000
i=79000000
i=80000000
i=81000000
i=82000000
i=83000000
i=84000000
i=85000000
i=86000000
i=87000000
i=88000000
i=89000000
i=90000000
i=91000000
i=92000000
i=93000000
i=94000000
i=95000000
i=96000000
i=97000000
i=98000000
i=99000000
```



5.30-3 SOLUTION OF TSP BY ANT ALGORITHM

In the natural world, ants (initially) wander randomly, and upon finding food return to their colony while laying down pheromone trails. If other ants find such a path, they are likely not to keep travelling at random, but to instead follow the trail, returning and reinforcing it if they eventually find food . Over time, however, the pheromone trail starts to evaporate, thus reducing its attractive strength. The more time it takes for an ant to travel down the path and back again, the more time the pheromones have to evaporate. A short path, by comparison, gets marched over more frequently, and thus the pheromone density becomes higher on shorter paths than longer ones. Pheromone evaporation also has the advantage of avoiding the convergence to a locally optimal solution. If there were no evaporation at all, the paths chosen by the first ants would tend to be excessively attractive to the following ones. In that case, the exploration of the solution space would be constrained. Thus, when one ant finds a good (i.e., short) path from the colony to a food source, other ants are more likely to follow that path, and positive feedback eventually leads to all the ants' following a single path. The idea of the ant colony algorithm is to mimic this behavior with "simulated ants" walking around the representing the problem to solve. In figure 1 it is shown the behavior of ants when a barrier faced.

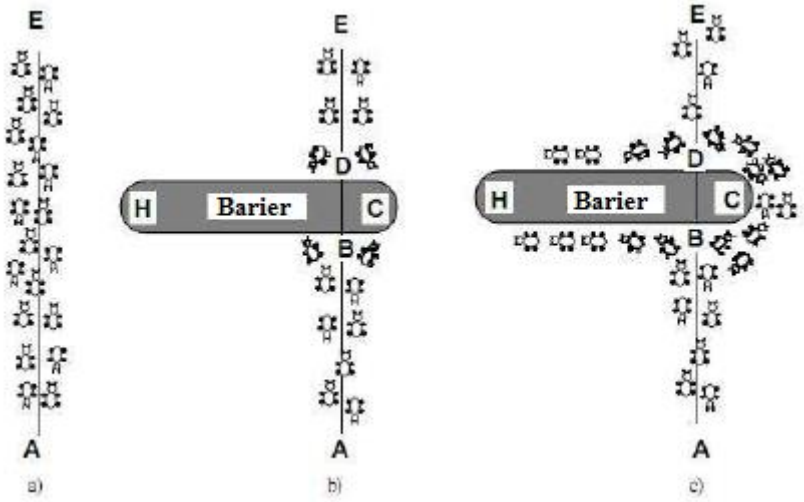


Figure 5.25-1 road selection by ants

Now let us investigate how to convert the basic principle into the computer code. Assume that ants can be reach to the food source by using two different paths. In this case probability of (m+1)th ant to select the first road can be written as:

$$P_{1(m+1)} = \frac{(m_1 + k)^h}{(m_1 + k)^h + (m_2 + k)^h}$$

In this equation m_1 and m_2 is the number of ant selected the first and second path. K and h in the equations are constants. The probability of (m+1)th ant to select the second road can be written as:

$P_{2(m+1)} = 1 - P_{1(m+1)}$

Simulated ants have some different properties that natural ants does not have

1. Simulated ants can have memory, for example in the TSP problems cities already visited can be memoried (keep in a variable)
2. They are npt completely blind, they can investigate the shortest road.
3. Their actual environment is much less time dependent.

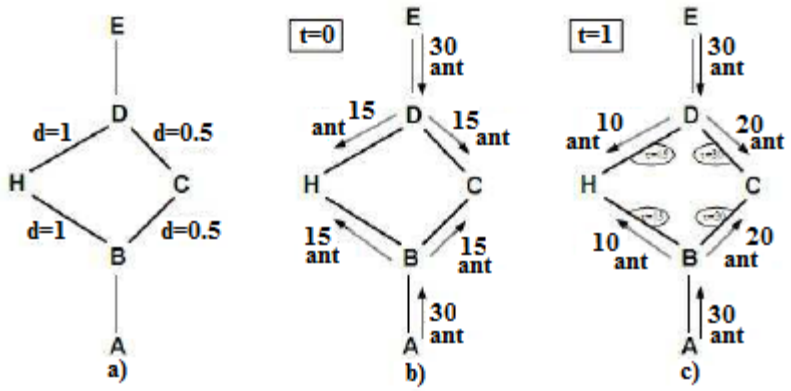


Figure 5.30-2 distribution of number of ants selected two different roads in time

When figüre 5.25-2 is investigated, ant behavior can be seen more closely.

- At the time t=0, 30 ant started to travel from A to E and another 30 ant started to travel from E to E
- As the first instance, ants travels in equal number to the cities H and C due to no previous pheromen trace on the road. Therefore each group will be consist of 15 ants
- Road B-C-D is shorter, therefore more pheromen will be collected on this path
- Art the time t=1 due to more pheromene in the B-C-D road more ants will prefer this road to the alternative one.

Now we have the basic idea to apply ant travel to the Travelling salesman problem. The following basic variables will be assumed

n : Number of the cities

d_{ij} : distance between cities i and j

$b_i(t)$: number of the ants in the city i, at the time t

m : total number of ants

$$m = \sum_{i=1}^{n} b_i(t)$$

$\tau_{ij}(t)$: pheromen density of the road ij at the time t

ρ : evaporation rate (0 to 1) In order to avoid infinite pheromen collection evaporation rate should ve selected smaller than 1. Note that 0 pheromen rate causes random movement also.

$\Delta\tau_{ij}^{k}$: the ration of pheromen density left by kth ant in the ij'th road to distance of this road.

$\Delta\tau_{ij}^{k} = \frac{Q}{L_k}$ If the kth ant used this particulare road

$\Delta\tau_{ij}^{k} = 0$ If the kth ant does not used this particulare road

L_k *;* total distance covered by the kth ant

Q; constant amount of pheromene left by each end per time

$Tabu_k$: The list of cities already travelled by kth ant

$$\eta_{ij} = \frac{1}{d_{ij}}$$

$p_{ij}^k(t)$: probability of going from city i to city j

If the city j is not in the list of already travelled cities;

$$p_{ij}^k(t) = \frac{[\tau_{ij}(t)]^\alpha \times [\eta_{ij}]^\beta}{\sum [\tau_{ik}(t)]^\alpha \times [\eta_{ik}]^\beta}$$

α ve β; are constant rate variables

If the city j is in the list of already travelled cities;

$p_{ij}^k(t) = 0$

Algorithm :

1. Start :
   a. t = 0 t : time counter
   b. NC = 0 NC :
   c. Starting pheromens for each road piece;
   d. Loacate m ants to the n cities.
2. s = 0 s : counter fort he list of the cities already travelled
   loop k = 1 to m
   for the kth ant place cities into the list of the cities already travelled
   repeat until list is full;
   a. s = s + 1
   b. k = 1 to m;
      i. select jth city to travel according to $p_{ij}^k(t)$ probability
      ii. move kth ant into the city j
      iii. add city j into the list of the cities already travelled
3. k = 1 to m
   a. let kth ant to travel from city $tabu_k(n)$ to $tabu_k(1)$
   b. calculate distance travelled by the ant k; L_k
   c. Record the shortest distance so far
   d. For each ij path;
      i. k = 1 to m
      If the kth ant used this road $\Delta\tau_{ij}^k = \frac{Q}{L_k}$
      If the kth ant does not used this road $\Delta\tau_{ij}^k = 0$

$$\Delta\tau_{ij} = \Delta\tau_{ij} + \Delta\tau_{ij}^k$$

4. for each ij path calculate; $\tau_{ij}(t+n)$;
   a. $\tau_{ij}(t+n) = \rho\tau_{ij}(t) + \Delta\tau_{ij}$
   b. t = t + n
   c. NC = NC + 1
   d. for each ij path $\Delta\tau_{ij} = 0$
5.
   a. If NC < NC_{max} ; clear the lists of travelled cities and g oto item 2;
      If NC >=NC_{max} write shotest distance and stop*(stagnation behaviour)*

Several different versions of the ant colony TSP problems can be found in the literatüre.
For the TSP problems ant algorithm is given in the following program. As the example problem case, again cities around a circle is taken.

Program 5.30-5 travelling salesman problem by using ants algorithm

```
public class antorj {
```

```
int id; //ant id number
int ssayi; //number of cities
int[] gyol; //gidilen güzergah
int asayi=0; //number of steps
double kmesafe; //total distance
double omesafe;
double feramon,guncel;

public antorj(int x,int y,double z) {
id=x;
ssayi=y;
feramon=z;
gyol=new int[ssayi];
kmesafe=0;
omesafe=400;
}

public void yuru(map w,double[][] x,int y,int z){
   double tolasilik;
   int isaretci;
   int a;
   int b;
   int yolsayisi=z;
   int bsehir,isaret,devam,devam2;
   int usehir;
   double of;

   a=0;
   b=0;
   isaretci=0;
   tolasilik=0;
   asayi=y; //atilmis adim sayisi
   usehir=gyol[asayi-1];
   b=ssayi-asayi;
   double[][] imkanliyol=new double[ssayi-asayi][2];
   for(int i=0;i<yolsayisi;i++){
      if(a>=b){
         break;
      }

      isaretci=0;
      if(x[i][0]==usehir){
         for(int j=0;j<asayi;j++){
            if (x[i][1]==gyol[j]){
               isaretci=1;
                 }
         }
         if (isaretci==0){
         imkanliyol[a][1]=x[i][1];
         imkanliyol[a][0]=x[i][2];
         a++;
         }
      } else
      {
      if(x[i][1]==usehir){
         for(int j=0;j<asayi;j++){
            if (x[i][0]==gyol[j]){
```

```
                isaretci=1;
                  }
            }
             if(isaretci==0){
            imkanliyol[a][1]=x[i][0];
            imkanliyol[a][0]=x[i][2];
             a++;
             }

        }
        }


    }

    for(int i=0;i<ssayi-asayi;i++){
        tolasilik=imkanliyol[i][0]+tolasilik;
        //System.out.print(" imkanli yol : ");
        //System.out.print(gyol[0]);
        //System.out.print(" imkanli yollar : ");
        //System.out.print(imkanliyol[i][1]);
        //System.out.println(" ");
        //System.out.print(" imkanli yol : ");
        //System.out.println(imkanliyol[i][0]);

    }

       of=Math.random();

        for(int i=0;i<ssayi-asayi;){
         if(of>imkanliyol[i][0]){
            i++;
            if(i>=(ssayi-asayi)){
                gyol[asayi]=(int) imkanliyol[i-1][1];
            }
         }
         else{
            //System.out.print(" imkanli yolda sorun var bence : ");
            //System.out.println(imkanliyol[i][1]);

            gyol[asayi]=(int) imkanliyol[i][1];
            break;
         }
        }
    kmesafe=kmesafe+w.cities[gyol[asayi-1]][(gyol[asayi])+3];
}

public double feramonsal(){
  return feramon;
}
public void ferguncel(double[][] x){
    guncel=feramon/kmesafe;
    for(int i=1;i<ssayi;i++){
        x[gyol[i-1]][gyol[i]]=x[gyol[i-1]][gyol[i]]+guncel;
    }
}
 public void buharlastir(double[][] x,double y){
```

```
      for(int i=0;i<ssayi;i++){
         for(int j=0;j<ssayi;j++){
           x[i][j]=x[i][j]*y;
         }
               }
   }

}
```

```
import java.io.*;
import java.math.*;

class map { double[][] cities;
            public double mesafe(double x1,double x2,double y1,double y2)
            { double mesafe;
              mesafe=Math.pow((Math.pow((x2-x1),2)+Math.pow((y2-y1),2)),0.5);
              return mesafe;
            }
            public map(readcity x) {
            cities=new double[x.numberofcities][x.numberofcities+3];
            for(int i=0;i<x.numberofcities;i++) {
            cities[i][0]=i; //?ehrin ID si
            cities[i][1]=x.cities[i][0]; //?ehrin x koordinat?
            cities[i][2]=x.cities[i][1]; //?ehrin y koordinat?
            for(int k=3;k<(x.numberofcities+3);k++)
            {cities[i][k]=mesafe(x.cities[i][0],x.cities[k-3][0],x.cities[i][1],x.cities[k-3][1]);}}
                                }
          }

class readcity
{   String dosyaismi=new String(" ");
   double[][] cities;
   int numberofcities;
   public readcity(String x)
   {   dosyaismi=x;
      cities=Text.readDouble(x);
      numberofcities=cities.length;
   }
}

public class ant{
                double mindistance;

      public static void quicksort(double[][] x,int y,int z){
      double[] mid=new double[3];
      double[] temp=new double[3];
      int k,j;
      k=y;
      j=z;
      mid[0]=x[(y+z)/2][0];
      mid[1]=x[(y+z)/2][1];
      mid[2]=x[(y+z)/2][2];
      do {while(x[k][2] < mid[2] & k < z) {k++;}
          while(x[j][2] > mid[2] & j > y) {j--;}
         if(k<=j){
            temp[2]=x[k][2];
            temp[1]=x[k][1];
            temp[0]=x[k][0];
```

```
            x[k][2]=x[j][2];
            x[k][1]=x[j][1];
            x[k][0]=x[j][0];
            x[j][2]=temp[2];
            x[j][1]=temp[1];
            x[j][0]=temp[0];
            k++;
            j--;
          }

      }while(k<=j);
      if(y<j) {
         quicksort(x,y,j);
      }
      if(k<z) {quicksort(x,k,z);}
      }

   public double readDistance()
   {return mindistance;}

            public  int[] ant_optimization(String dosya,double vaporizarion,double feramoni,int nant )
            {   double tolerans=1.0e-5;
      double dx=0.1;
      //vaporization 0-1
      double buharlasmaorani=vaporizarion; //feramonun buharlasma oranı
      //feramoni=5 (example)
      double feramon=feramoni; //feramon
      double beta=3; //
      double alfa=1; //
      double baslangicferamonu=1; //t=0 anindaki yoldaki feramon miktar?
      //int sakant=10; //iterasyon aras? korunacak maksimum kar?nca say?s?
      //int sakis; //saklanacak karınca işareti
      readcity sehir=new readcity(dosya); //şehirlerin okunması
      map bolge=new map(sehir); //mapnın hazırlanması
      double[][] feramondurumu=new double[sehir.numberofcities][sehir.numberofcities]; //yolların feramon durumunun
kaydedileceği
      double[][] deltaferamon=new double[sehir.numberofcities][sehir.numberofcities];
      //int antsayisi=1000; //koloni büyüklüğü
      int antsayisi=nant;
      int it=0;
      int maxit=10; //maksimum iterasyon sayisi
      int ilksehir;
      int numberofroads;
      double toplamprobability=0;
      double[][] probabilities=new double[sehir.numberofcities][sehir.numberofcities];
      double minprobability=0;
      double maxprobability=0;
      double tprobability=0;
      double mesafe;
      mindistance=0;
      double mindistanceeski=0.1;
      dx=Math.abs(mindistance-mindistanceeski);
      int minid=0;
      int[] minguzergah=new int[sehir.numberofcities];
      antorj[] antordusu=new antorj[antsayisi];
      //antorj[] sakkar=new antorj[sakant];
      numberofroads=0;
```

```
        //numberofroadsnin hesaplanması
        while (it<maxit  &&  dx>tolerans){
         mindistanceeski=mindistance;
         numberofroads=0;
        for(int i=1;i<sehir.numberofcities-1;i++){
           numberofroads=numberofroads+sehir.numberofcities-i;
        }
        numberofroads=numberofroads+1;
        //System.out.println(numberofroads);
        double[][] probabilitypasta=new double[numberofroads][3];

        //t=0 anındaki feramonun yollara konulması
       if(it==0){
        for(int i=0;i<(sehir.numberofcities-1);i++) {
           for(int j=i+1;j<(sehir.numberofcities);j++) {
              if(i==j){
                 feramondurumu[i][j]=0;
              }else{
                 feramondurumu[i][j]=baslangicferamonu;
              }
                    }
        }
       }
        //probability pastasinin hesaplanmasi
           //toplam olasiligin hesaplanmasi
        for(int i=0;i<sehir.numberofcities-1;i++){
           for(int j=i+1;j<(sehir.numberofcities);j++){
              if(i!=j){
                 toplamprobability=Math.pow((feramondurumu[i][j]),alfa)*Math.pow(bolge.cities[i][j+3],-beta);
                           }
           }
        }
        //toplamprobability=toplamprobability/2;
           //pastanin hesaplanması
        for(int i=0;i<sehir.numberofcities-1;i++){
           for(int j=i+1;j<(sehir.numberofcities);j++){
              if(i!=j){
                 probabilities[i][j]=(Math.pow((feramondurumu[i][j]),alfa)*Math.pow(bolge.cities[i][j+3],-
beta))/toplamprobability;
                 if(bolge.cities[i][j+3]==0){
                 }
                 if(i==0 & j==1){
                    maxprobability=probabilities[i][j];
                    minprobability=probabilities[i][j];
                    probabilitypasta[0][2]=minprobability;
                    probabilitypasta[0][0]=i;
                    probabilitypasta[0][1]=j;
                    probabilitypasta[numberofroads-1][2]=maxprobability;
                    probabilitypasta[numberofroads-1][0]=i;
                    probabilitypasta[numberofroads-1][1]=j;
                 } else {
                    if(maxprobability<probabilities[i][j]) {
                       maxprobability=probabilities[i][j];
                       probabilitypasta[numberofroads-1][2]=maxprobability;
                       probabilitypasta[numberofroads-1][0]=i;
                       probabilitypasta[numberofroads-1][1]=j;
                    } else {
```

```
                if(minprobability>probabilities[i][j]){
                   minprobability=probabilities[i][j];
                   probabilitypasta[0][0]=i;
                   probabilitypasta[0][1]=j;
                }
                }
            }
                        }
        else {
           probabilities[i][j]=0;
        }
    }
}
for(int i=1;i<numberofroads-1;i++){
   probabilitypasta[i][2]=minprobability;
}
   //pastanin siralanmasi
int l=0;
 for(int i=0;i<sehir.numberofcities-1;i++){
    for(int j=i+1;j<(sehir.numberofcities);j++){
      if(i!=j){
        probabilitypasta[l][2]=probabilities[i][j];
        probabilitypasta[l][0]=i;
        probabilitypasta[l][1]=j;
       l++;
        }
    }
 }
//probabilitiesi siralama
quicksort(probabilitypasta,0,numberofroads-1);
//t=0 anında ilk antların yerleştirilmesi
for(int i=0;i<antsayisi;i++){
   double x;
   antordusu[i]=new antorj(i,sehir.numberofcities,feramon);
   x=Math.random()*sehir.numberofcities;
   ilksehir=(int) x;
   antordusu[i].gyol[0]=ilksehir;
}
//antların yürümesi
for(int i=1;i<sehir.numberofcities;i++){
   for(int j=0;j<antsayisi;j++){
      antordusu[j].yuru(bolge,probabilitypasta,i,numberofroads);
   }
}
mindistance=antordusu[0].kmesafe;
minguzergah=antordusu[0].gyol;
for(int j=0;j<antsayisi;j++){
   if(antordusu[j].kmesafe<mindistance){
      mindistance=antordusu[j].kmesafe;
      minguzergah=antordusu[j].gyol;
   }
}

for(int j=0;j<antsayisi;j++){
   antordusu[j].ferguncel(deltaferamon);
}
 for(int i=0;i<(sehir.numberofcities-1);i++) {
```

```
          for(int j=i+1;j<(sehir.numberofcities);j++) {
             if(i==j){
                feramondurumu[i][j]=0;
             }else{
                feramondurumu[i][j]=feramondurumu[i][j]*buharlasmaorani+deltaferamon[i][j];
             }
                    }
       }
          it++;
          dx=Math.abs(mindistance-mindistanceeski);
       }
       return minguzergah;
       }
}
```

```
import java.io.*;
import java.math.*;

class harita { double[][] sehirler;
             public double mesafe(double x1,double x2,double y1,double y2)
             { double mesafe;
               mesafe=Math.pow((Math.pow((x2-x1),2)+Math.pow((y2-y1),2)),0.5);
               return mesafe;
             }
             public harita(sehiroku x) {
             sehirler=new double[x.sehirsayisi][x.sehirsayisi+3];
             for(int i=0;i<x.sehirsayisi;i++) {
             sehirler[i][0]=i; //?ehrin ID si
             sehirler[i][1]=x.sehirler[i][0]; //?ehrin x koordinat?
             sehirler[i][2]=x.sehirler[i][1]; //?ehrin y koordinat?
             for(int k=3;k<(x.sehirsayisi+3);k++)
             {sehirler[i][k]=mesafe(x.sehirler[i][0],x.sehirler[k-3][0],x.sehirler[i][1],x.sehirler[k-3][1]);}}
                                  }
          }

class sehiroku
{   String dosyaismi=new String(" ");
   double[][] sehirler;
   int sehirsayisi;
   public sehiroku(String x)
   {   dosyaismi=x;
      sehirler=Text.readDouble(x);
      sehirsayisi=sehirler.length;
   }
}

public class ant{
                  double minmesafe;

       public static void quicksort(double[][] x,int y,int z){
       double[] mid=new double[3];
       double[] temp=new double[3];
       int k,j;
       k=y;
       j=z;
       mid[0]=x[(y+z)/2][0];
       mid[1]=x[(y+z)/2][1];
       mid[2]=x[(y+z)/2][2];
       do {while(x[k][2] < mid[2] & k < z) {k++;}
           while(x[j][2] > mid[2] & j > y) {j--;}
          if(k<=j){
```

```
            temp[2]=x[k][2];
            temp[1]=x[k][1];
            temp[0]=x[k][0];
            x[k][2]=x[j][2];
            x[k][1]=x[j][1];
            x[k][0]=x[j][0];
            x[j][2]=temp[2];
            x[j][1]=temp[1];
            x[j][0]=temp[0];
            k++;
            j--;
        }

    }while(k<=j);
    if(y<j) {
       quicksort(x,y,j);
    }
    if(k<z) {quicksort(x,k,z);}
    }

public double readMesafe()
{return minmesafe;}

            public  int[] ant_optimization(String dosya,double vaporizarion,double feramoni,int nkarinca )
            {   double tolerans=1.0e-5;
    double dx=0.1;
    //vaporization 0-1
    double buharlasmaorani=vaporizarion; //feramonun buharlasma oranı
    //feramoni=5 (örneğin)
    double feramon=feramoni; //feramon
    double beta=3; //
    double alfa=1; //
    double baslangicferamonu=1; //t=0 anindaki yoldaki feramon miktar?
    //int sakkarinca=10; //iterasyon aras? korunacak maksimum kar?nca say?s?
    //int sakis; //saklanacak karınca işareti
    sehiroku sehir=new sehiroku(dosya); //şehirlerin okunması
    harita bolge=new harita(sehir); //haritanın hazırlanması
    double[][] feramondurumu=new double[sehir.sehirsayisi][sehir.sehirsayisi]; //yolların feramon durumunun kaydedileceği
    double[][] deltaferamon=new double[sehir.sehirsayisi][sehir.sehirsayisi];
    //int karincasayisi=1000; //koloni büyüklüğü
    int karincasayisi=nkarinca;
    int it=0;
    int maxit=10; //maksimum iterasyon sayisi
    int ilksehir;
    int yolsayisi;
    double toplamolasilik=0;
    double[][] olasiliklar=new double[sehir.sehirsayisi][sehir.sehirsayisi];
    double minolasilik=0;
    double maxolasilik=0;
    double tolasilik=0;
    double mesafe;
    minmesafe=0;
    double minmesafeeski=0.1;
    dx=Math.abs(minmesafe-minmesafeeski);
    int minid=0;
    int[] minguzergah=new int[sehir.sehirsayisi];
    karincaorj[] karincaordusu=new karincaorj[karincasayisi];
    //karincaorj[] sakkar=new karincaorj[sakkarinca];
    yolsayisi=0;
    //yolsayisinin hesaplanması
    while (it<maxit  &&  dx>tolerans){
     minmesafeeski=minmesafe;
     yolsayisi=0;
    for(int i=1;i<sehir.sehirsayisi-1;i++){
```

```
        yolsayisi=yolsayisi+sehir.sehirsayisi-i;
    }
    yolsayisi=yolsayisi+1;
    //System.out.println(yolsayisi);
    double[][] olasilikpasta=new double[yolsayisi][3];

    //t=0 anındaki feramonun yollara konulması
   if(it==0){
    for(int i=0;i<(sehir.sehirsayisi-1);i++) {
        for(int j=i+1;j<(sehir.sehirsayisi);j++) {
            if(i==j){
                feramondurumu[i][j]=0;
            }else{
                feramondurumu[i][j]=baslangicferamonu;
            }
                    }
    }
   }
    //olasilik pastasinin hesaplanmasi
        //toplam olasiligin hesaplanmasi
    for(int i=0;i<sehir.sehirsayisi-1;i++){
        for(int j=i+1;j<(sehir.sehirsayisi);j++){
            if(i!=j){
                toplamolasilik=Math.pow((feramondurumu[i][j]),alfa)*Math.pow(bolge.sehirler[i][j+3],-beta);
                            }
        }
    }
    //toplamolasilik=toplamolasilik/2;
        //pastanin hesaplanması
    for(int i=0;i<sehir.sehirsayisi-1;i++){
        for(int j=i+1;j<(sehir.sehirsayisi);j++){
            if(i!=j){
                olasiliklar[i][j]=(Math.pow((feramondurumu[i][j]),alfa)*Math.pow(bolge.sehirler[i][j+3],-beta))/toplamolasilik;
                if(bolge.sehirler[i][j+3]==0){
                }
                if(i==0 & j==1){
                    maxolasilik=olasiliklar[i][j];
                    minolasilik=olasiliklar[i][j];
                    olasilikpasta[0][2]=minolasilik;
                    olasilikpasta[0][0]=i;
                    olasilikpasta[0][1]=j;
                    olasilikpasta[yolsayisi-1][2]=maxolasilik;
                    olasilikpasta[yolsayisi-1][0]=i;
                    olasilikpasta[yolsayisi-1][1]=j;
                } else {
                    if(maxolasilik<olasiliklar[i][j]) {
                        maxolasilik=olasiliklar[i][j];
                        olasilikpasta[yolsayisi-1][2]=maxolasilik;
                        olasilikpasta[yolsayisi-1][0]=i;
                        olasilikpasta[yolsayisi-1][1]=j;
                    } else {
                    if(minolasilik>olasiliklar[i][j]){
                        minolasilik=olasiliklar[i][j];
                        olasilikpasta[0][0]=i;
                        olasilikpasta[0][1]=j;
                    }
                    }
                }
                            }
            else {
                olasiliklar[i][j]=0;
            }
        }
    }
```

```
    for(int i=1;i<yolsayisi-1;i++){
       olasilikpasta[i][2]=minolasilik;
    }
       //pastanin siralanmasi
    int l=0;
     for(int i=0;i<sehir.sehirsayisi-1;i++){
       for(int j=i+1;j<(sehir.sehirsayisi);j++){
         if(i!=j){
           olasilikpasta[l][2]=olasiliklar[i][j];
           olasilikpasta[l][0]=i;
           olasilikpasta[l][1]=j;
          l++;
           }
       }
     }
    //olasiliklari siralama
    quicksort(olasilikpasta,0,yolsayisi-1);
    //t=0 anında ilk karincaların yerleştirilmesi
    for(int i=0;i<karincasayisi;i++){
       double x;
       karincaordusu[i]=new karincaorj(i,sehir.sehirsayisi,feramon);
       x=Math.random()*sehir.sehirsayisi;
       ilksehir=(int) x;
       karincaordusu[i].gyol[0]=ilksehir;
    }
    //Karincaların yürümesi
    for(int i=1;i<sehir.sehirsayisi;i++){
       for(int j=0;j<karincasayisi;j++){
          karincaordusu[j].yuru(bolge,olasilikpasta,i,yolsayisi);
       }
    }
    minmesafe=karincaordusu[0].kmesafe;
    minguzergah=karincaordusu[0].gyol;
    for(int j=0;j<karincasayisi;j++){
       if(karincaordusu[j].kmesafe<minmesafe){
          minmesafe=karincaordusu[j].kmesafe;
          minguzergah=karincaordusu[j].gyol;
       }
    }

    for(int j=0;j<karincasayisi;j++){
       karincaordusu[j].ferguncel(deltaferamon);
    }
     for(int i=0;i<(sehir.sehirsayisi-1);i++) {
       for(int j=i+1;j<(sehir.sehirsayisi);j++) {
          if(i==j){
             feramondurumu[i][j]=0;
          }else{
             feramondurumu[i][j]=feramondurumu[i][j]*buharlasmaorani+deltaferamon[i][j];
          }
                }
    }
       it++;
       dx=Math.abs(minmesafe-minmesafeeski);
    }
    return minguzergah;
    }
}
```

Program 5.256 ant algorithm saple testing program

```
public class anttest
```

```
{
      public static void main(String args[])
      {
      double evaporationrate=0.90; //pheromoneun buharlasma oran?
      double pheromone=5; //pheromone
      int totalant=1000;
      ant x=new ant();
      int[] tur=x.ant_optimization("cities.txt",evaporationrate,pheromone,totalant);
      for(int i=0;i<tur.length;i++){System.out.print(tur[i]+" ");}
      System.out.println("\nminimum mesafe = "+x.readMesafe());
      }
}
```

Sample test data cities.txt

```
100.0       0.0
100.0       0.0
95.10565162951535           30.901699437494738
80.90169943749474           58.778525229247315
58.778525229247315          80.90169943749474
30.901699437494745          95.10565162951535
6.123233995736766E-15       100.0
-30.901699437494734         95.10565162951536
-58.7785252292473           80.90169943749474
-80.90169943749473          58.77852522924732
-95.10565162951535          30.901699437494752
-100.0      1.2246467991473532E-14
-95.10565162951538          -30.90169943749469
-80.90169943749474          -58.7785252292473
-58.77852522924732          -80.90169943749473
-30.901699437494756         -95.10565162951535
-1.8369701987210297E-14     -100.0
30.901699437494724          -95.10565162951536
58.77852522924729           -80.90169943749476
80.90169943749473           -58.778525229247336
95.10565162951535           -30.901699437494763
```

Program output

```
---------- Capture Output ----------
> "C:\turhan\java\bin\javaw.exe" anttest
20 19 18 17 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 0 1
minimum mesafe = 594.4509671528771

> Terminated with exit code 0.
```

5.31 LİNEAR MULTIVARIABLE OPTİMİZATION, SİMPLEX METHOD

As the last topic in optimization, Linear optimization will be investigated. Linear problems are existed only with the boundary restrictions. The general definition of the problem :

Maximum $Z=a_0x_0+ a_1x_1+ a_2x_2+ a_3x_3+\ldots\ldots+ a_nx_n$

Subject to:

$b_{00}x_0+ b_{01}x_0+ b_{02}x_0+ b_{03}x_0+\ldots\ldots+ b_{0n}x_0 <= A_0$

$b_{10}x_0+ b_{11}x_0+ b_{12}x_0+ b_{13}x_0+\ldots\ldots+ b_{1n}x_0 <= A_1$

$b_{20}x_0+ b_{21}x_0+ b_{22}x_0+ b_{23}x_0+\ldots\ldots+ b_{2n}x_0 <= A_2$

.....

$b_{n0}x_0+ b_{n,1}x_0+ b_{n2}x_0+ b_{n3}x_0+\ldots\ldots+ b_{nn}x_0 <= A_n$

If it is minimum instead of maximum we can simply change the sign of the function. An example case can be given as

Maximum Z=60x+50y
Subject to :
2x+4y<=80
3x+2y<=60

In order solve equation in Simplex method, form of the equation is changed as follows:

$Z-a_0x_0- a_1x_1- a_2x_2- a_3x_3-\ldots\ldots- a_nx_n = 0$
Subject to :
$b_{00}x_0+ b_{01}x_1+ b_{02}x_2+ b_{03}x_3+\ldots\ldots+ b_{0n}x_n + W_0 = A_0$
$b_{10}x_0+ b_{11}x_1+ b_{12}x_2+ b_{13}x_3+\ldots\ldots+ b_{1n}x_n + W_1 = A_1$
$b_{20}x_0+ b_{21}x_1+ b_{22}x_2+ b_{23}x_3+\ldots\ldots+ b_{2n}x_n + W_2 = A_2$

.....
$b_{n0}x_0+ b_{n,1}x_1+ b_{n2}x_2+ b_{n3}x_3+\ldots\ldots+ b_{nn}x_n + W_n = A_n$

where W's are additional slack variables. If the equation is written in matrix format

$$\begin{bmatrix} -a_0 & -a_1 & -a_2 & \ldots. & -a_n & 0 & 0 & \ldots & 0 \\ b_{00} & b_{01} & b_{02} & \ldots. & b_{0n} & 1 & 0 & \ldots & 0 \\ b_{10} & b_{11} & b_{12} & \ldots. & b_{1n} & 0 & 1 & \ldots & 0 \\ \ldots & \ldots & \ldots & \ldots. & \ldots & \ldots & \ldots & \ldots & \ldots \\ b_{n0} & b_{n1} & b_{n2} & \ldots. & b_{nn} & 0 & 0 & \ldots & 1 \end{bmatrix} \begin{bmatrix} x_0 \\ x_1 \\ x_2 \\ \ldots. \\ x_n \\ W_0 \\ W_1 \\ W_2 \\ \ldots. \\ W_n \end{bmatrix} = \begin{bmatrix} 0 \\ A_0 \\ A_1 \\ \ldots. \\ A_n \end{bmatrix}$$

Due to inequality of the row and columns matrix can not be solved directly. Simplex method is used to tolve the system as follows:

1. In the main equation coefficient with the biggest negative number is selected. The column with this number is called critical column.
2. The value in the right hand side vector is divide by the value in the critical column. The row with the minimum value is selected as critical row.
3. Critical raw and column intersection point is called the critical value
4. Critical row is normalised by dividing to critical number
5. The following process applied to raws outside the critical row
   New value = old value – [value in the critical column and the same row]*[value in the critical raw in the same column] / critical number
6. Step 1 to 5 repeated until all values in the main equation becomes positive

We can look at the process through an example :
Maximum of Z=60x+50y
Subject to:
2x+4y<=80
3x+2y<=60

Simplex format :

$Z-60x-50y+0W_A+0W_B = 0$

$2x+4y + W_A+0W_B= 80$

$3x+2y +0W_A+ W_B=60$

matrix format:

$$\begin{bmatrix} -60 & -50 & 0 & 0 & 0 \\ 2 & 4 & 1 & 0 & 80 \\ 3 & 2 & 0 & 1 & 60 \end{bmatrix}$$

As the first iteration critical column is the column with number -60. In order to find the critical row t1=80/2=40 and t2=60/3=20 compared, since 20 is smaller raw 3 is selected as critical raw. Critical value is then selected as 3. Third row is diveded by 3 becomes

$1 \quad \frac{2}{3} \quad 0 \quad \frac{1}{3} \quad 20$

In the raws other than critical raw, a gauss elimination type process aplied to make column values 0 For example in order to make coefficient of x value in the first row equal to 0 following procedure is applied

3rd row(critical row)multiplied with 50 and Added to first row. In this case the values of the first row becomes:

$0 \quad -10 \quad 0 \quad 20 \quad 1200$ olacaktir.

At the end of these processes matrix becomes

$$\begin{bmatrix} 0 & -10 & 0 & 20 & 1200 \\ 0 & \frac{8}{3} & 1 & -\frac{2}{3} & 40 \\ 1 & \frac{2}{3} & 0 & \frac{1}{3} & 20 \end{bmatrix}$$

The same iterative process is repeated as a second time and result will be

$$\begin{bmatrix} 0 & 0 & \frac{15}{4} & \frac{35}{2} & 1350 \\ 0 & 1 & \frac{3}{8} & -\frac{1}{4} & 15 \\ 1 & 0 & -\frac{1}{4} & \frac{1}{2} & 10 \end{bmatrix}$$

If Wa and Wb values are replaced with 0 value x=10 and y=15 is found. The function value becomes 1350 . Now the same problem will be solved by using the computer program

Program 5.26-1 Linear simplex optimization

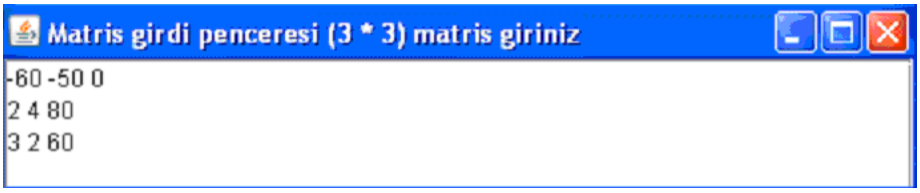
```
class simplex
{
    public static double[] simplex(double[][] a)
    {
    int n=a.length; //row
    int m=a[0].length; //column
    int m1=m-1;//
    int n1=n-1;
    int m2=m1+n1;
    int m3=m2+1;
    int keyrow=0;
    int keycolumn=0;
    double keynumber=0.0;
    double ratio=1.0;
    double b[][]=new double[n][m3];
    double d[]=new double[m];
    String s[]=new String[m];
```

```
for(int i=0;i<n;i++)
{ b[i][m2]=a[i][m1]; }
for(int i=0;i<n;i++)
{for(int j=0;j<m1;j++) {b[i][j]=a[i][j];}}
for(int i=0;i<n-1;i++)
  {
  for(int j=m1;j<m2;j++) {if(i==(j-m1)) b[i+1][j]=1.0;System.out.println("i="+i+"j="+j+"bij="+b[i+1][j]);}
  }
double max;
double min;
System.out.println(Matrix.toString(b));
for(int k=0;k<n1;k++)
{
        max=-9.999e30;
min=9.999e30;
for(int j=0;j<m1;j++)
{if( (b[0][j]<0) && Math.abs(b[0][j])>max) {max=Math.abs(b[0][j]);keycolumn=j;}}
for(int i=1;i<n;i++)
{ ratio=b[i][m2]/b[i][keycolumn];
  if(ratio<min)  {min=ratio;keyrow=i;}
}
keynumber=b[keyrow][keycolumn];
double c[][]=new double[n][m3];
for(int i=0;i<n;i++)
{ for(int j=0;j<m3;j++) {c[i][j]=b[i][j];}}
for(int i=0;i<n;i++)
{ if(i!=keyrow) {for(int j=0;j<m3;j++) {b[i][j]=c[i][j]-c[keyrow][j]*c[i][keycolumn]/keynumber;}}
  else {for(int j=0;j<m3;j++) {b[i][j]=c[i][j]/keynumber;}}
}
System.out.println("keynumber = "+keynumber);
System.out.println("keyrow = "+keyrow+" keycolumn = "+keycolumn);
System.out.println(Matrix.toString(b));
}

d[m1]=b[0][m2];
s[m1]="y=f(x)";
for(int j=0;j<m1;j++)
   { s[j]="x["+j+"]";}
for(int i=1;i<n;i++)
{
   for(int j=0;j<m1;j++)
   {
                if((b[i][j]==1.0)&& (b[0][j]==0.0))
        { d[j]=b[i][m2];}
   }

}
Text.print(d,s,"Simplex algorithm");
return d;
}
        public static void main(String arg[])
        {
        double a[][]=Text.readDouble_TextArea();
        //double a[][]={{-60,-50,0},{2,4,80},{3,2,60}};
        double d[]=simplex(a);
        System.out.println(Matrix.toString(d));
        }
```

```
}
```

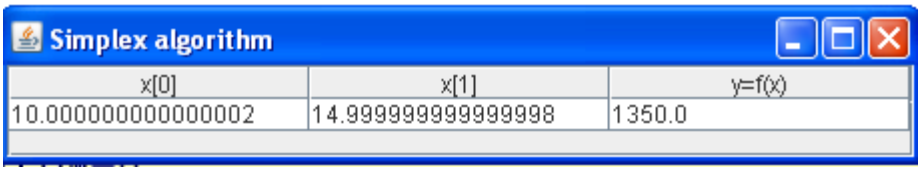



```
---------- Capture Output ----------
> "C:\co\java\bin\java.exe" OPO25
     -60.000000000000000      -50.000000000000000       0.000000000000000
0.000000000000000       0.000000000000000
       2.000000000000000       4.000000000000000       1.000000000000000
0.000000000000000         80.000000000000000
       3.000000000000000       2.000000000000000       0.000000000000000
1.000000000000000         60.000000000000000

keynumber = 3.0
keyrow = 2 keycolumn = 0
       0.000000000000000     -10.000000000000000       0.000000000000000
20.000000000000000     1200.000000000000000
       0.000000000000000       2.666666666666667       1.000000000000000      -
0.666666666666667       40.000000000000000
       1.000000000000000       0.666666666666667       0.000000000000000
0.333333333333333       20.000000000000000

keynumber = 2.666666666666667
keyrow = 1 keycolumn = 1
       0.000000000000000       0.000000000000000       3.750000000000000
17.500000000000000     1350.000000000000000
       0.000000000000000       1.000000000000000       0.375000000000000      -
0.250000000000000       14.999999999999998
       1.000000000000000       0.000000000000000      -0.250000000000000
0.500000000000000       10.000000000000002

     10.000000000000002              14.999999999999998              1350.000000000000000
```



5.32 SOLUTION OF THE NONLINEAR SYSTEM OF EQUATIONS BY USING OPTIMIZATION ALGORITHMS

Non-Linear system of equations can also be solved by using optimization methods through an adaptation function.

To solve function $f_i(x_j)=0$

$$g(x_j) = \sum_{i=0}^{n_equation} f_i(x_j) * f_i(x_j)$$

The derivative of this equation is the root of non-linear system of equation again became $f_i(x_j)=0$, therefore optimum of $g(x_j)$ is the same problem as solution of $f_i(x_j)=0$.
As the first adaptation algorithm Nelder-Mead Method will be given. Nelder_Mead method was defined at sub-chapter 5.10 as NA41:

Program 5.32-1 Solution of Non-Linear system of equations by using Nelder-Mead optimization method.

```
// Solution of system of equation by using Nelder-Mead Method
//
import java.util.*;
import java.awt.*;
import java.applet.Applet;
import java.awt.event.*;
import javax.swing.*;

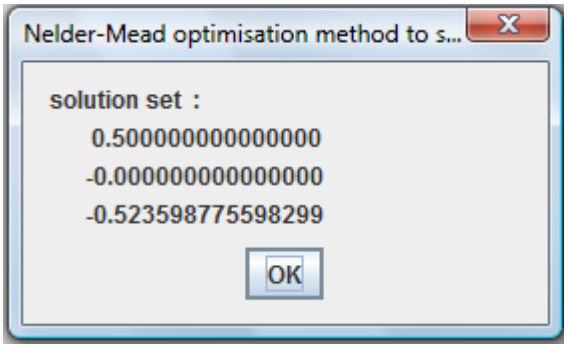

class f1 extends f_xj
{//adaptation function
fi_xi ff1;
public f1(fi_xi ffi) {ff1=ffi;}

public double func(double x[])
{
double ff=0.0;
double fa[]=ff1.func(x);
for(int i=0;i<fa.length;i++)  ff+=fa[i]*fa[i];
return ff;
}
}

class fa extends fi_xi
{ // Nonlinear system of equation
public double[] func(double x[])
{
double ff[]=new double[3];
ff[0]=3.0*x[0]-Math.cos(x[1]*x[2])-0.5;
ff[1]=x[0]*x[0]-81.0*(x[1]+0.1)*(x[1]+0.1)+Math.sin(x[2])+1.06;
ff[2]=Math.exp(-x[0]*x[1])+20.0*x[2]+(10.0*Math.PI-3.0)/3.0;
return ff;
}
}

public class NA117
{
     public static void main (String args[])
    {
    String s="leave a space between initial root guess values: ";
    double [] x0=NA41.inputdata(s);
    double [] dx0=new double[x0.length];
    for(int i=0;i<x0.length;i++) dx0[i]=0.2*x0[i];
```

```
    fa ff=new fa();
    f1 f=new f1(ff);
    double [] r1= NA41.nelder(f,x0,dx0);
    s=" solution set  : \n"+Matrix.toStringT(r1);
    String s2="Nelder-Mead optimization method to solve system of non-linear equation:";
    JOptionPane.showMessageDialog(null,s,s2,JOptionPane.PLAIN_MESSAGE);
    System.exit(0);
    }
}
```



The next example method to solve system of nonlinear equations will be Genetic algorithm method

Program 5.32-2 Solution of Non-Linear system of equations by using Genetic Algorithms optimization method.

```
//Genetic algorithms to solve system of equations
import java.io.*;

class f1 extends f_xj
{//adaptor class
fi_xi ff1;
public f1(fi_xi ffi) {ff1=ffi;}

public double func(double x[])
{
double ff=0.0;
double fa[]=ff1.func(x);
for(int i=0;i<fa.length;i++)  ff+=fa[i]*fa[i];
return 10-ff; //minimum değil maxsimum istendiğinden - ile çarptik
}
}

class fa extends fi_xi
{
public double[] func(double x[])
{
//çözümü istenen fonksiyon
double ff[]=new double[3];
ff[0]=3.0*x[0]-Math.cos(x[1]*x[2])-0.5;
ff[1]=x[0]*x[0]-81.0*(x[1]+0.1)*(x[1]+0.1)+Math.sin(x[2])+1.06;
ff[2]=Math.exp(-x[0]*x[1])+20.0*x[2]+(10.0*Math.PI-3.0)/3.0;
return ff;
}
}
```

```
class NA118
{
   public static void main(String args[]) throws IOException
   {
   int N=18;
            int POPSIZE=500;
   int MAXGENS=500;
   int NVARS=3;
   double PXOVER=0.2;
   double PMUTATION=0.05;
    fa ff=new fa();
    f1 f_x=new f1(ff);
   Genetic1 xx=new Genetic1(POPSIZE,MAXGENS,NVARS,N,PXOVER,PMUTATION);
   double low[]=new double[NVARS];
   double high[]=new double[NVARS];
   low[0]=-1.0;high[0]=1.0;
   low[1]=-1.0;high[1]=1.0;
   low[2]=-1.0;high[2]=1.0;
   xx.setPopulation(low,high,f_x);
   System.out.println("best=\n"+Matrix.toStringT(xx.calculate(f_x,false)));
   }
}
```

```
---------- Capture Output ----------
> "C:\turhan\java\bin\javaw.exe" NA118
best=
       0.499959945525915
      -0.000003814711818
      -0.523595899947739
       9.999999981442672


> Terminated with exit code 0.
```

As the next example steepest-descent method is used.

Program 5.32-3 Solution of Non-Linear system of equations by using Steepest descent optimization method.

```
// Solution of system of equation by using Steepest descent Method
//
import java.util.*;
import java.awt.*;
import java.applet.Applet;
import java.awt.event.*;
import javax.swing.*;

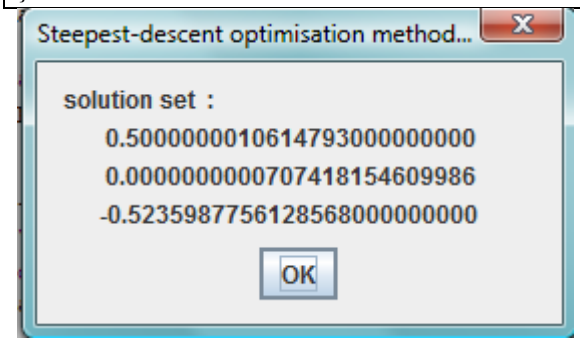

class f2 extends f_xj
{//adaptation function
fi_xi ff1;
public f2(fi_xi ffi) {ff1=ffi;}

public double func(double x[])
```

```
{
double ff=0.0;
double fa[]=ff1.func(x);
for(int i=0;i<fa.length;i++)  ff+=fa[i]*fa[i];
return ff;
}
}

class fa extends fi_xi
{ // Nonlinear system of equation
public double[] func(double x[])
{
double ff[]=new double[3];
ff[0]=3.0*x[0]-Math.cos(x[1]*x[2])-0.5;
ff[1]=x[0]*x[0]-81.0*(x[1]+0.1)*(x[1]+0.1)+Math.sin(x[2])+1.06;
ff[2]=Math.exp(-x[0]*x[1])+20.0*x[2]+(10.0*Math.PI-3.0)/3.0;
return ff;
}
}

public class NA119
{
        public static void main (String args[])
    {
        String s="leave a space between initial root guess values: ";
    double [] x0=NA43.inputdata(s);
    fa ff=new fa();
    f2 f=new f2(ff);
    double [] r1= NA43.steepestdescent(f,x0);
    s=" solution set  : \n"+Matrix.toStringT(r1);
    String s2="Steepest-descent optimization method to solve system of non-linear equation:";
    JOptionPane.showMessageDialog(null,s,s2,JOptionPane.PLAIN_MESSAGE);
    System.exit(0);
    }
}
```



Program 5.32-4 Solution of Non-Linear system of equations by using Davidon-Fletcher-Powell optimization method

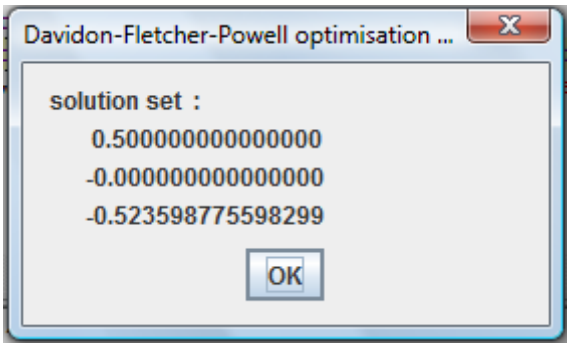
```
// Solution of system of equation by using Davidon-Fletcher-Powell Method
//
import java.util.*;
import java.awt.*;
import java.applet.Applet;
import java.awt.event.*;
import javax.swing.*;
```

```
class f2 extends f_xj
{//adaptation function
fi_xi ff1;
public f2(fi_xi ffi) {ff1=ffi;}

public double func(double x[])
{
double ff=0.0;
double fa[]=ff1.func(x);
for(int i=0;i<fa.length;i++)  ff+=fa[i]*fa[i];
return ff;
}
}

class fa extends fi_xi
{ // Nonlinear system of equation
public double[] func(double x[])
{
double ff[]=new double[3];
ff[0]=3.0*x[0]-Math.cos(x[1]*x[2])-0.5;
ff[1]=x[0]*x[0]-81.0*(x[1]+0.1)*(x[1]+0.1)+Math.sin(x[2])+1.06;
ff[2]=Math.exp(-x[0]*x[1])+20.0*x[2]+(10.0*Math.PI-3.0)/3.0;
return ff;
}
}

public class NA120
{
    public static void main (String args[])
    {  String s="leave a space between initial root guess values: ";
       double [] x0=NA43D.inputdata(s);
       fa ff=new fa();
       f2 f=new f2(ff);
       double [] r1= NA43D.davidon_fletcher_powell(f,x0);
       s=" solution set  : \n"+Matrix.toStringT(r1);
       String s2="Davidon-Fletcher-Powell optimization method to solve system of non-linear equation:";
       JOptionPane.showMessageDialog(null,s,s2,JOptionPane.PLAIN_MESSAGE);
       System.exit(0);
    }
}
```



As a last example particle swarm optimisation is applied to system of non-linear equation solving. The same example is used as the perivious problems.

Program 5.32-5 Solution of Non-Linear system of equations by using particle swarm optimization method

```
import java.util.*;
import java.awt.*;
import java.applet.Applet;
import java.awt.event.*;
import javax.swing.*;

class f2 extends f_xj
{//adaptation function
fi_xi ff1;
public f2(fi_xi ffi) {ff1=ffi;}

public double func(double x[])
{
double ff=0.0;
double fa[]=ff1.func(x);
for(int i=0;i<fa.length;i++)  ff+=fa[i]*fa[i];
return ff;
}
}

class fa extends fi_xi
{ // Nonlinear system of equation
public double[] func(double x[])
{
double ff[]=new double[3];
ff[0]=3.0*x[0]-Math.cos(x[1]*x[2])-0.5;
ff[1]=x[0]*x[0]-81.0*(x[1]+0.1)*(x[1]+0.1)+Math.sin(x[2])+1.06;
ff[2]=Math.exp(-x[0]*x[1])+20.0*x[2]+(10.0*Math.PI-3.0)/3.0;
return ff;
}
}

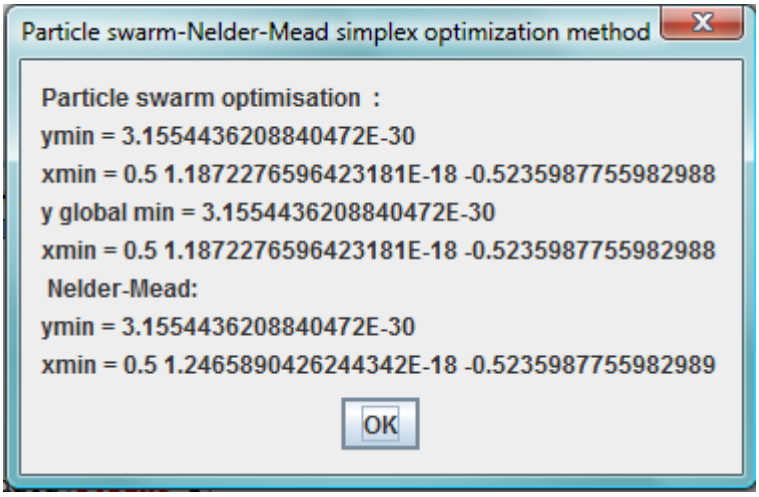

public class particleswarmtest3
{
                public static double[] inputdata(String s)
    {
    String s1=JOptionPane.showInputDialog(s);
    StringTokenizer token=new StringTokenizer(s1);
    int n=token.countTokens()-1;
    int m=n+1;
    double a[]=new double[m];
    int j=0;
    while(token.hasMoreTokens())
    {
    Double ax=new Double(token.nextToken());
    a[j++]=ax.doubleValue();
    }
    return a;
    }

public static void main(String arg[])
{
String s="leave a space between initial root guess values: xmin0 xmin1.. ";
        double xmin[]=inputdata(s);
        s="leave a space between initial root guess values: xmax0 xmax1.. ";
        double xmax[]=inputdata(s);
        fa ff=new fa();
        f2 f=new f2(ff);

particle_swarm1 ps=new particle_swarm1(f,2000,xmin,xmax,2.0,0.7,2000);
double r1[]= particle_swarm1.nelder(f,ps.gmin());
```

```
s="Particle swarm optimisation  : \n"+ps.toString()+"\n Nelder-Mead: \n"+ps.toString(r1);
    String s2="Particle swarm-Nelder-Mead simplex optimization method ";
    JOptionPane.showMessageDialog(null,s,s2,JOptionPane.PLAIN_MESSAGE);

}
}
```



5.27 OPTIMISATION BY USING OTHER COMPUTER ENVIRONMENTS

One variable optimisation problems can be solved by using excel spreadsheet solver (in turkish version çözücü)

5.28 PROBLEMS

PROBLEM 1

In a distillation tank operates with saturated steam benzen and toluen is seperated. In order to obtain purest toluen in liquid phase (maximize the toluen in the mixture) what would be the ideal temperature?

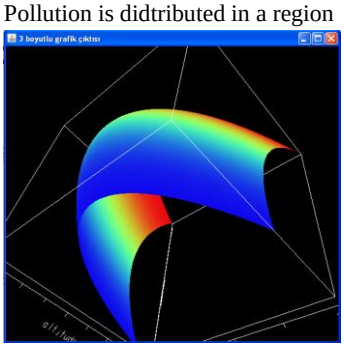
$x_{tolüen} P_{doyma_tolüen} + x_{benzen} P_{doyma_benzen} = P = 760$ mmHg

$\log_{10}(P_{doyma_benzen}) = 6.905 - 1211/(T+221)$
$\log_{10}(P_{doyma_tolüen}) = 6.953 - 1344/(T+219)$

temperature degree C, pressure mmHg

PROBLEM 2

Two dimensional distribution of pollution in a channel can be given by the equation
$C(x,y)=7.9+0.13x+0.21y-0.05x^2-0.016y^2-0.007xy$
Pollution is didtributed in a region $-10 <= x <= 10$ and $0 <= y <= 20$, find the place where the pollution is maximum



PROBLEM 3

A petrochemical company developing a new fuel additive for commercial airplanes. Additive is consist of three components X,Y and Z. Fort he best performance total additive in the fuel should be at least 6 mL/L. Due o security reasons the total of most flammable X and Y components should be less than 3 mL/L. Componnt X should always be equal or more than the

component Y, and componet Z should be more than half of the component Y. The cost of X,Y and Z components are 0.15,0.025 ve 0.05 TL respectively. Find the mixing component tha will give the minimum cost.

PROBLEM 4

A food company is trying to design a new container box for preserved food that will minimise the cost of material. Total 320 cm^3 volume is requested. Estetic concers limits diameter of the box in between 3 to 10 cm. and the height in between 2 to 10 cm. Box will be made of steel sheet of 0.4 mm thich with density of 7800 kg/m^3. The cost of box material is 37 TL. Assume that box including top and bottom covers are made of the same material, find the box size that will be given the minimum cost. And the cost per box.

PROBLEM 5

Air enters at 1 bar 300K to a 3 stage compressor system. After each stage air is cooled down again to 300 K before entering to the next stage. Calculate internal pressure for each stage to minimize work input to the compressors.

The work requirement for eah stage of the compressor:

$W=C_p(T_i - T_e)=C_pT_i(1-T_e/T_i)=kRT_i/(k-1)*[1-(P_e/P_i)^{(k-1)/k}]$

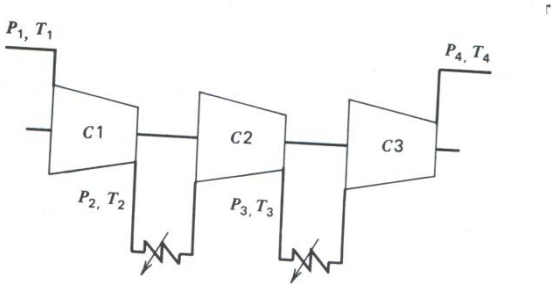
Total work fort he thre stages.

$W=kRT_i/(k-1)*[1-(P_2/P_1)^{(k-1)/k}]+ kRT_i/(k-1)*[1-(P_3/P_2)^{(k-1)/k}]+ kRT_i/(k-1)*[1-(P_4/P_3)^{(k-1)/k}]$

$W= kRT_i/(k-1)*[3-(P_2/P_1)^{(k-1)/k}-(P_3/P_2)^{(k-1)/k}-(P_4/P_3)^{(k-1)/k}]$

$(P_2/P_1) *(P_3/P_2)*(P_4/P_3)= (P_4/P_1)=27$

$k=1.4$ $C_p=1.005$ KJ/kg K



Work written as java function :

```
class f1 extends f_xj
{
public double func(double x[])
{
 double ff=1.4*8.3145*300/(1.4-1)*(1-x[0]*(1.4-1)/1.4)+ 1.4*8.3145*300/(1.4-1)*(1-(x[1]/x[0])*(1.4-1)/1.4)
 +1.4*8.3145*300/(1.4-1)*(1-(27/x[1])*(1.4-1)/1.4);
return ff; //minimum değil maxsimum istendiğinden - ile çarptik
}}
```

answer : 3 bar and 9 bar

PROBLEM 6

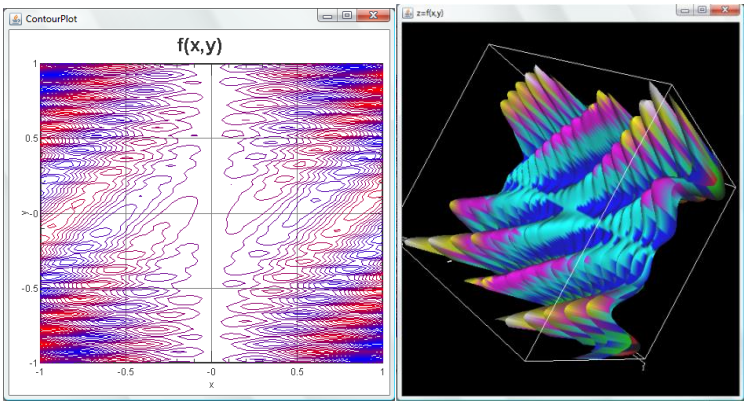
By using the data below find the aximum point. You can use an interpolation polynomial then mximise it.

X	Y
0	2
1	6
2	8
3	12
4	14
5	13
6	11
7	9
8	13
9	14.5
10	7

PROBLEM 7

Function below is given, find the maximum. -1<x [0]<1 -1<x [1]<1

```
double func(double x[])
 { double ff= 34+x[0]*(Math.sin(3.0*Math.PI*(x[0]-x[1]))+x[1]*Math.sin(20.0*Math.PI*x[1]));
   return ff;
}
```

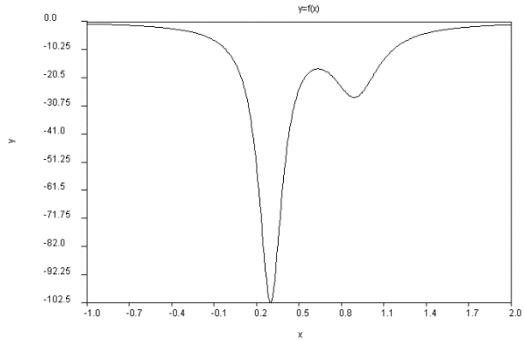


PROBLEM 8

An antibiotic producing microorganism has a specific growth rate(g) is function of food concentration (c)
If $g = 2*c/(4+0.8c+c^2+0.2c^3)$
Find the c that will give te maximum growth rate. In low food concentrations growth rate drops to zero, similarly in very high food concentrations growth rate will drop to zero due to food poisining effect. Because of this reason, values bigger than c = 10 mg/L should not be investigated.

PROBLEM 9

Find minimum of function
f(x)=-(1.0/((x-0.3)*(x-0.3)+0.01)+1.0/((x-0.9)*(x-0.9)+0.04))

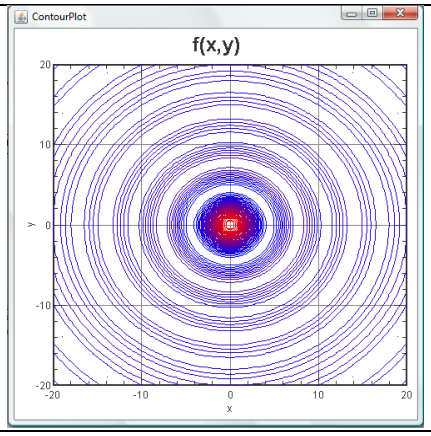


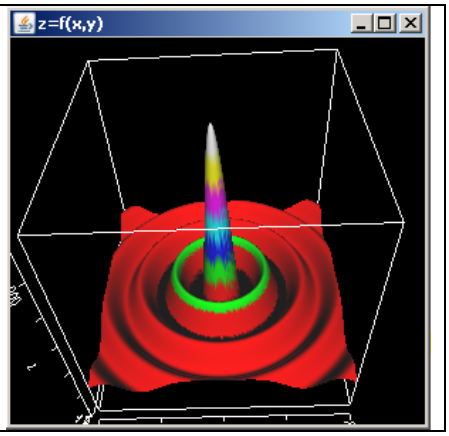
PROBLEM 10

Function

$$z(x,y) = \frac{sin(\sqrt{x^2+y^2+\varepsilon})}{\sqrt{x^2+y^2+\varepsilon}}$$

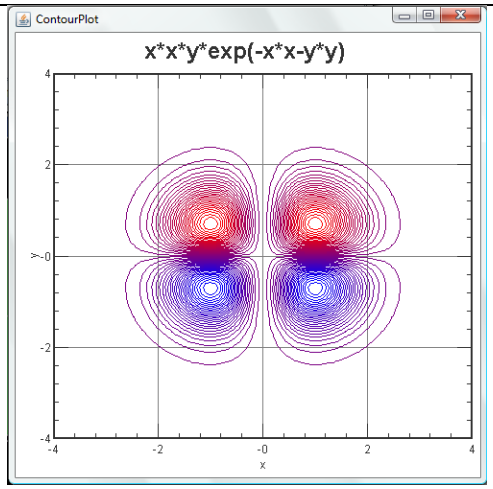
$-20 \le x \le 20$ ve $-20 \le y \le 20$ is given. Find the maximum of the function for ε=1e-2 starting from P(18,18)

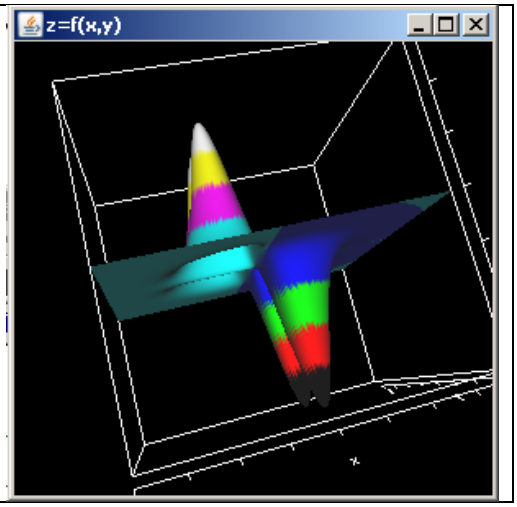




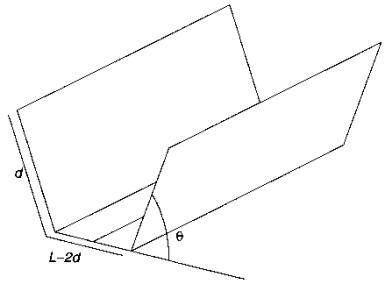
PROBLEM 11

Function
$z(x, y) = x^2 y e^{(-x^2-y^2)}$
With boundaries
$-4 \leq x \leq 4$ **ve**
$-4 \leq y \leq 4$
is given. Find the minimum of the function starting from P(3,3)



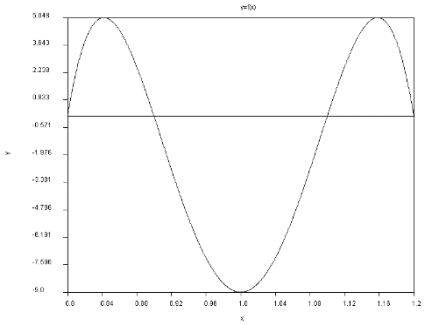


PROBLEM 12



In order to construct water channel in a house , a flat sheet will be banded as shown in the figure. Water carrying capacity of the channel can be written as
$V(d,\theta) = Z(d^2 \sin\theta \cos\theta + (L-2d)d\sin\theta)$
In this equation z is the length of the sheet (sheet area is zL). Find the d and θ to maximize water holding capacity of the channel.

PROBLEM 13
Function below is given
$f(x) = 2500x^6-15000x^5+14875x^4+40500x^3-97124x^2+73248x-19008$
find the minimum between x_0=0.8, x_1=1.2 values.



PROBLEM 14
Cities, A,B,C, and D located at corrdinates(0,0),(1,5),(5,2) and (7,3) respectively. The roads from all cities are merged at point P. Find point P that makes the total distance of the roads minimum.

PROBLEM 15
Function below is given
$f(x_0, x_1, x_2, x_3) = (x_0+10\ x_1)^2 +5(x_2-10\ x_3)^2+(x_1-2\ x_2)^4+10(x_1- x_4)^4$
find the minimum starting from P(1,-1,0,1)

PROBLEM 16
Function below is given
$f(x_0, x_1) = 3(x_0 - 1)^2 +2(x_1-1)\ (x_0-1)(x_1-2)+(x_1- 2)^4$

Minimum of the function is given at point (1,2). Located the minimum at 3D plot, then by using any 3D nonlinear optimization method find the minimum points starting at P (0,0)

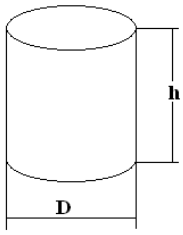
PROBLEM 17

Find theminimum of function

$$\boldsymbol{f(x) = \frac{15x}{(4x^2 - 3x + 4)}}$$

fin the range of 0 to10.

PROBLEM 18



One of the very basic optimization problem is the minimum cost of container problem. The cost of a box usually is a function of the surface area. Therefore we should minimize the area for a given volume For example if the volume of the container V=0.5 liter=0.5x10^{-3} m^3 :

Volume $V = \frac{\pi D^2}{4} h$ or from this equation h, height is obtained as $h = \frac{4V}{\pi D^2}$.

Surface area of the cylinder : $A = 2\frac{\pi D^2}{4} + \pi D h = \frac{\pi D^2}{2} + \frac{4V}{D}$.

Analytical solution of the minimization problem

$$\frac{dA}{dD} = \pi D - \frac{4V}{D^2} = 0$$

$$D = \sqrt[3]{\frac{4V}{\pi}}.$$

From here solution is
=0.086025401 m and .

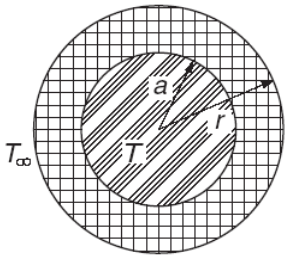
$$h = \frac{4*0.5x10^{-3}}{\pi D^2} = 0.086025401 \text{ m.}$$

Now obtain this results by using nuerical optimization methods.
For the range of 0.01<=D<=0.2

a) Graphic method
b) Golden ratio (Fibonnachi)
c) Newton-Raphson root finding
d) Bisection method
e) Newton-Raphson method
f) Secant method
g) Brent method
h) Second degree polynomials
i) Third degree polynomials

Note: Some of these methods are defined only as root finding method, is not defined as optimization methods.

PROBLEM 19

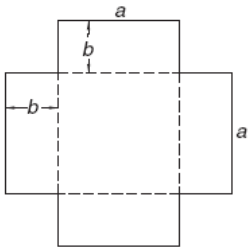


An isolation covered a wire carrying electrical current. The outlet diameter of electrical insulation is r. The electrical resistance on the wire caused heat. Thi heat crried through electrical insulation to the air around the wire with a temperature of T_{∞}. The temperature of the wire can be calculated as

$$T = \frac{q}{2\pi}\left(\frac{ln(r/a)}{k} + \frac{1}{hr}\right) + T_{\infty}$$

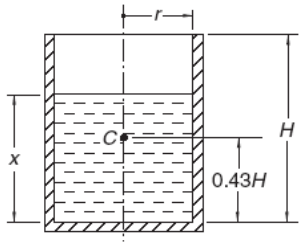
In this equation
q = heat generation in the wire = 50 W/m
a = diameter of the wire = 5 mm = $5x10^{-3}$ m
k = coeeficient of thermal conductivity of insulation = 0.2 W/(mK)
h=thermal convetivity coefficient for air = 30 W/(m^2K)
T_{∞}= environmental temperature = 300 K.
Find radius r that wil make temperature T minimum.

PROBLEM 20



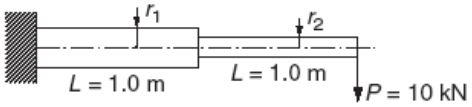
In order to make a cartoon box, the shape above should be used.After bending from the dotted linet he volume of the box will be 1 m^3. In order to spent the minimum amount of the cartoon, what a and b dimensions should be?

PROBLEM 21



A cylindrical container has a mass of M and has an empty center of gravity of C=0.43H. If the height of water in the cup is x, In order to have the common center of gravity of water and the cup as low as possible, what sholud be the value of x ?

PROBLEM 22



Two cylindrical beams with two different diameters is placed as in the figure. It is desired beam to have the minimum cross sectional area. Operational condions are:
$\sigma_1 \leq$180 MPa, $\sigma_2 \leq$180 MPa, and bending $\delta \leq$25 mm dir

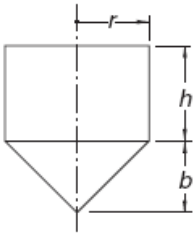
$\sigma_1 = \frac{8PL}{\pi r_1^3}$=maximum stress in the left beam

$\sigma_2 = \frac{8PL}{\pi r_2^3}$= maximum stress in the right beam

$\delta = \frac{4PL^3}{3\pi E}\left(\frac{7}{r_1^4} + \frac{1}{r_2^4}\right)$=bending at the end of the beam

ve E =Young modüulus=200 GPa. Calculate r_1 and r_2.

PROBLEM 23



It is desired to have volume of the cone base shape shown in the figure as 1 m^3. Calculate r,h and b dimensions to minimise the surface area.

$$V = \pi r^2 \left(\frac{b}{3} + h\right)$$
$$S = \pi r \left(2h + \sqrt{b^2 + r^2}\right)$$

PROBLEM 24

Lennard-Jones potential in between two molecules can be expressed as

$$V = 4\varepsilon \left[\left(\frac{\sigma}{r}\right)^{12} - \left(\frac{\sigma}{r}\right)^{6}\right]$$

In this equation ε and σ values are constants. Find $\left(\frac{\sigma}{r}\right)$ values that makes. V minimum

PROBLEM 25

Wave function for hydrogen atoms

$\psi = C(27 - 18\sigma + 2\sigma^2)e^{-\sigma/3}$ In this equation

$$\sigma = zr/a_0$$
$$C = \frac{1}{81\sqrt{3\pi}}\left(\frac{z}{a_0}\right)^{2/3}$$

z = nucleus electrical load

a_0 = Bohr diameter

r =radial distance.

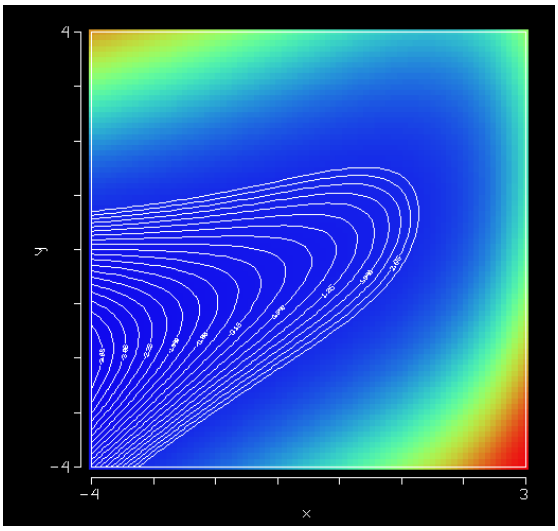
Find σ value that make ψ wave function minimum.

PROBLEM 26

Find optimum of function

$$f = e^{x_1} - x_1 x_2 + x_2^2$$
$$g = 2x_1 + x_2 - 2 \leq 0$$
$$h = x_1^2 + x_2^2 - 4 = 0$$

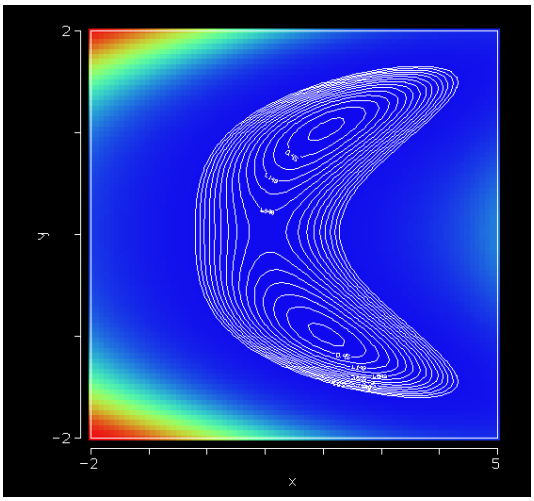
With -4<=x<=3 and -4<=y<=4 boundaries.



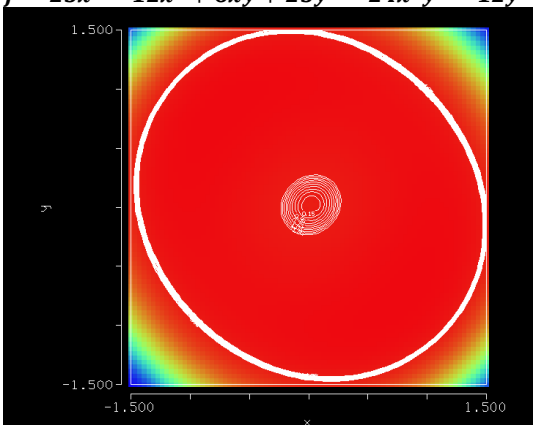
PROBLEM 27

Find the minimum of

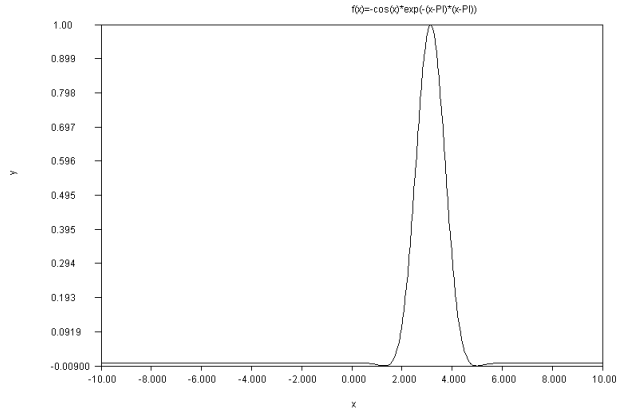
$f = (x - 2)^2 + (x - 2y^2)^2$.



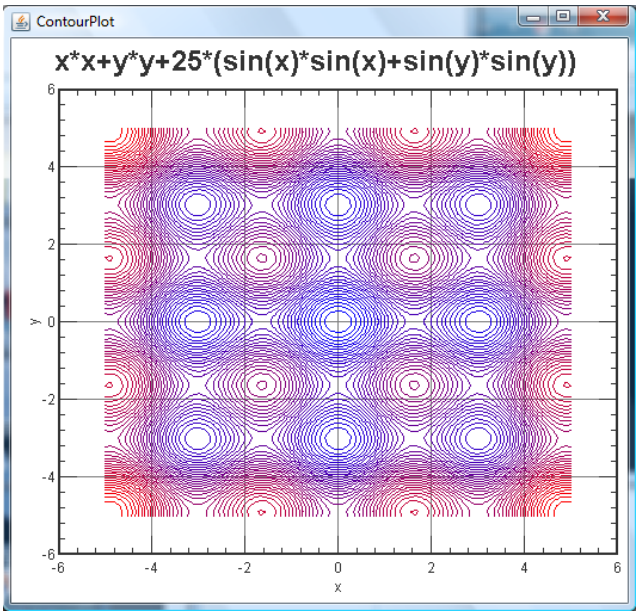
PROBLEM 28
Find the minimum of
$\boldsymbol{f = 25x^2 - 12x^4 + 6xy + 25y^2 - 24x^2y^2 - 12y^4}$.



PROBLEM 29 Find maximum of Easoms's function
$f = -cos(x)e^{-(x-\pi)^2}$between limits [-10,10] by using Genetic algorithms



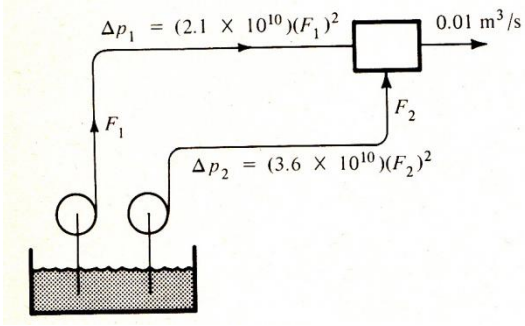
PROBLEM 30 Egg rate function
$f = x^2 + y^2 + 25[sin^2(x) + sin^2(y)]$ has the global minimum f=0 at (0,0) in the domain [-5,5]



PROBLEM 31 : Two parallel pump-pipe assemblies, shown in the figure, deliver water from a common source to a common destination. The total flow rate required at the destination is $F=F_1+F_2=0.01$ m^3/s. The drops in pressure in two lines are functions of the square of flow rates.

$\boldsymbol{\Delta p_1 = 2.1x10^{10}F_1^2}$ Pa and $\boldsymbol{\Delta p_2 = 3.6x10^{10}F_2^2}$ Pa where F_1 and F_2 are respective flow rates in Cubic meters per second. pumping power for a pump is given by the equation

$\boldsymbol{Power = \frac{F\Delta p}{\rho\eta}}$ where F is flow rate(m^3/s), Δp is pressure drop (Pa), ρ=1000 kg/m^3 is density and is pump efficiency. η_1=0.81 and η_2=0.86

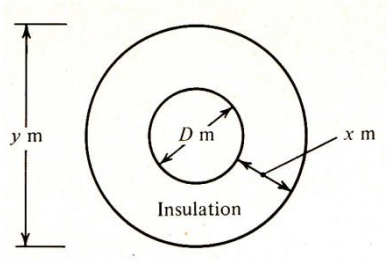


So pumping power is equal to: $Power = \frac{F_1\Delta p_1}{\rho\eta_1} + \frac{F_2\Delta p_2}{\rho\eta_2}$

$$Power = \frac{F_1\Delta p_1}{\rho\eta_1} + \frac{F_2\Delta p_2}{\rho\eta_2} = \frac{C_1F_1^3}{\rho\eta_1} + \frac{C_2F_2^3}{\rho\eta_2} = \frac{C_1F_1^3}{\rho\eta_1} + \frac{C_2}{\rho\eta_2}(F-F_1)^3$$

Solve the values of F_1 and F_2 for minimum power requirements.

PROBLEM 32: A pipe carrying high temperature water is to be insulated and then mounted in a restricted space, as shown in the figure. The choice of the pipe diameter D m and the insulation thickness x m are to be such that the OD of the insulation



Y is a minimum, but the total annual operation cost of the insulation is limited to 40000 TL. This annual operation cost has two components, the water pumping cost and the cost of the heat loss:

Pumping cost = $\frac{8}{D^5}$and Heat cost=$\frac{1500}{x}$

If an objective function is created for this problem:

$$y = D + 2x$$

Subject to:

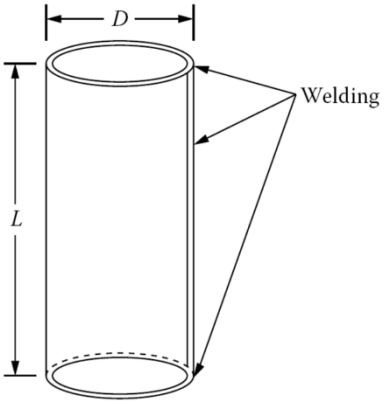
$$\frac{8}{D^5} + \frac{1500}{x} = 40000$$

Equation can be arranged as a single equation as:

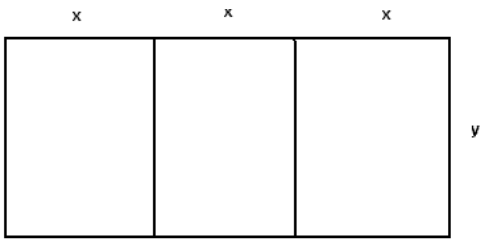
$$y = D + 3000\left[\frac{1}{40000 - \frac{8}{D^5}}\right]$$

Find the minimum y and corresponding x and D values.

PROBLEM 33: A manufacturer of steel cans wants to minimize costs. As a first approximation, the cost of making a can consists of the cost of the metalplus the cost of welding the longitudinal seam and the top and bottom. The can may have any diameter *D* and length *L*, for a given volume *V*. The wall thickness *d* is 1 mm. The cost of the material is \$0.50/kg and the cost of welding is \$0.1/m of the weld. The density of the material is 10^4 kg/m^3 Find the dimensions of the can that will minimize cost.



PROBLEM 33: A farmer needs 150 meters of fencing to fence three adjacent gardens. What would be maximum area of each garden, and what would be values of x and y



$$4y + 6x = 150$$
$$y = 37.5 - 1.5x$$
$$A(x) = 3(37.5 - 1.5x)x = 112.5x - 4.5x^2$$
$$\frac{dA(x)}{dx} = 112.5 - 9x = 0$$

$x = 12.5$ y=18.75

$A = 703.125$ total of 3 garden

Calculate by using a numerical method

6. CURVE FITTING AND FUNCTION APPROXIMATIONS

Data is obtained by measurements in the scientific world. Scientits are generalized the measurement results by using various formulas and theories. Creating functions by using experimental numerical data is called curve fitting. Formulas are generally formed from the theories of physical laws or freely selected by the researchers. In the fitted equation usually a good fit to the experimental data is a primary requirement. Fitting means assigning values to unknown coefficients in the equations. The coefficients can be linearly or non-linearly dependent the the variables of the function(s). Interpolation is also similar process, but in the interpolation a function is not always required. Interpolation could be just a repeatetive calculation process by following a certain rule. Different curve fitting methods will be investigated in this chapter. At the end of the chapter Approximation of the functions will also be covered.

6.1 LINEAR CURVE FITTING : LEAST SQUARE METHODS

Least square methods are one of the most used curve fitting methods. Polinomial equation version is widely used. Assuming having a linear function $f(a_j,x)$ where a_j are m linear coefficient and x are independent variable and $\phi_j(x)$ are m sub-funtions linearly multiplied with the coefficients

$$f(a_i,x)=\sum_{j=0}^{m} a_j^{(m)}\phi_j(x) \qquad (6.1\text{-}1)$$

It should be noted again that linearity is only for the coefficients, $\phi_j(x)$ functions does not have to be linear.

Sample functions: $f(a_i,x)=a_0\sin(x)+a_1\exp(x)+a_2\ln(x)+a_3x^2$

It is desired to fit data x_i, f_i, i=0..n into the equation so that the difference between data and fitted function dependent values for all points will be minimum. In order to establish this, the following function H will be minimized with respect to a_j

$$H(a_0^{(m)},..,a_n^{(m)})=\sum_{i=1}^{n} w(x_i)\left[f(x_i)-\sum_{j=0}^{m} a_j^{(m)}\phi_j(x_i)\right]^2 \qquad (6.1-1)$$

Where $w(x_i)$ values in the equation are called weight function. In order to minimiz the function root of the derivative of the function can be calculated.

$$\frac{\partial H(a_0^{(m)},..,a_n^{(m)})}{\partial a_k^{(m)}}=2\sum_{i=1}^{n} w(x_i)\left[f(x_i)-\sum_{j=0}^{m} a_j^{(m)}\phi_j(x)\right]\phi_k(x)=0 \quad k=0,\dots,m \qquad (6.1-2)$$

For weight function to be taken equal to unity ,$w(x_i)$ =1, equation in the open form can be written in the following form.

$$\begin{bmatrix} \sum_{i=1}^{n} w(x_i)\phi_0(x_i)\phi_0(x_i) & \sum_{i=1}^{n} w(x_i)\phi_0(x_i)\phi_1(x_i) & \sum_{i=1}^{n} w(x_i)\phi_0(x_i)\phi_2(x_i) & \cdots & \sum_{i=1}^{n} w(x_i)\phi_0(x_i)\phi_n(x_i) \\ \sum_{i=1}^{n} w(x_i)\phi_1(x_i)\phi_0(x_i) & \sum_{i=1}^{n} w(x_i)\phi_1(x_i)\phi_1(x_i) & \sum_{i=1}^{n} w(x_i)\phi_1(x_i)\phi_2(x_i) & \cdots & \sum_{i=1}^{n} w(x_i)\phi_2(x_i)\phi_n(x_i) \\ \sum_{i=1}^{n} w(x_i)\phi_2(x_i)\phi_0(x_i) & \sum_{i=1}^{n} w(x_i)\phi_2(x_i)\phi_1(x_i) & \sum_{i=1}^{n} w(x_i)\phi_2(x_i)\phi_2(x_i) & \cdots & \sum_{i=1}^{n} w(x_i)\phi_2(x_i)\phi_n(x_i) \\ \cdots & \cdots & \cdots & \cdots & \cdots \\ \sum_{i=1}^{n} w(x_i)\phi_n(x_i)\phi_0(x_i) & \sum_{i=1}^{n} w(x_i)\phi_n(x_i)\phi_1(x_i) & \sum_{i=1}^{n} w(x_i)\phi_n(x_i)\phi_2(x_i) & \cdots & \sum_{i=1}^{n} w(x_i)\phi_n(x_i)\phi_n(x_i) \end{bmatrix} \begin{Bmatrix} a_0 \\ a_1 \\ a_2 \\ \cdots \\ a_n \end{Bmatrix} =$$

$$\begin{Bmatrix} \sum_{i=1}^{n} w(x_i)\phi_0(x_i)f(x_i) \\ \sum_{i=1}^{n} w(x_i)\phi_1(x_i)f(x_i) \\ \sum_{i=1}^{n} w(x_i)\phi_2(x_i)f(x_i) \\ \cdots \\ \sum_{i=1}^{n} w(x_i)\phi_n(x_i)f(x_i) \end{Bmatrix} \quad (6.1\text{-}3)$$

For weight function to be taken equal to unity ,$w(x_i)$ =1, equation in the open form can be written in the following form.

If $w(x_i)=1$ matrix will be

$$\begin{bmatrix} \sum_{i=1}^{n}\phi_0(x_i)\phi_0(x_i) & \sum_{i=1}^{n}\phi_0(x_i)\phi_1(x_i) & \sum_{i=1}^{n}\phi_0(x_i)\phi_2(x_i) & \cdots & \sum_{i=1}^{n}\phi_0(x_i)\phi_n(x_i) \\ \sum_{i=1}^{n}\phi_1(x_i)\phi_0(x_i) & \sum_{i=1}^{n}\phi_1(x_i)\phi_1(x_i) & \sum_{i=1}^{n}\phi_1(x_i)\phi_2(x_i) & \cdots & \sum_{i=1}^{n}\phi_2(x_i)\phi_n(x_i) \\ \sum_{i=1}^{n}\phi_2(x_i)\phi_0(x_i) & \sum_{i=1}^{n}\phi_2(x_i)\phi_1(x_i) & \sum_{i=1}^{n}\phi_2(x_i)\phi_2(x_i) & \cdots & \sum_{i=1}^{n}\phi_2(x_i)\phi_n(x_i) \\ \cdots & \cdots & \cdots & \cdots & \cdots \\ \sum_{i=1}^{n}\phi_n(x_i)\phi_0(x_i) & \sum_{i=1}^{n}\phi_n(x_i)\phi_1(x_i) & \sum_{i=1}^{n}\phi_n(x_i)\phi_2(x_i) & \cdots & \sum_{i=1}^{n}\phi_n(x_i)\phi_n(x_i) \end{bmatrix} \begin{Bmatrix} a_0 \\ a_1 \\ a_2 \\ \cdots \\ a_n \end{Bmatrix} = \begin{Bmatrix} \sum_{i=1}^{n}\phi_0(x_i)f(x_i) \\ \sum_{i=1}^{n}\phi_1(x_i)f(x_i) \\ \sum_{i=1}^{n}\phi_2(x_i)f(x_i) \\ \cdots \\ \sum_{i=1}^{n}\phi_n(x_i)f(x_i) \end{Bmatrix} \quad (6.1\text{-}4)$$

This equation is an m+1 linear system of equation. It can easily be solved by using a system of equation solving method. A special form of the above equation a form of a polinomial can be assumed. If $\phi_j(x) = x^j$, equation becomes

$$f(a_i, x) = \sum_{j=0}^{m} a_j^{(m)} x^j \qquad (6.1-5)$$

$$f(a_i, x) = a_0 + a_1 x + a_2 x^2 + a_2 x^2 + \cdots + a_n x^n \quad (6.1.5a)$$

In this case the minimisation equation becomes

$$\frac{\partial H(a_0^{(m)}, \ldots, a_0^{(m)})}{\partial a_k^{(m)}} = 2\sum_{i=1}^{n} w(x_i)\left[f(x_i) - \sum_{j=0}^{m} a_j^{(m)} x^j\right] x^k = 0 \quad k = 0, \ldots, m \qquad (6.1-6)$$

$$\begin{bmatrix} \sum_{i=1}^{n} w(x_i) & \sum_{i=1}^{n} w(x_i)x_i & \sum_{i=1}^{n} w(x_i)x_i^2 & \cdots & \sum_{i=1}^{n} w(x_i)x_i^n \\ \sum_{i=1}^{n} w(x_i)x_i & \sum_{i=1}^{n} w(x_i)x_i^2 & \sum_{i=1}^{n} w(x_i)x_i^3 & \cdots & \sum_{i=1}^{n} w(x_i)x_i^{n+1} \\ \sum_{i=1}^{n} w(x_i)x_i^2 & \sum_{i=1}^{n} w(x_i)x_i^3 & \sum_{i=1}^{n} w(x_i)x_i^4 & \cdots & \sum_{i=1}^{n} w(x_i)x_i^{n+2} \\ \cdots & \cdots & \cdots & \cdots & \cdots \\ \sum_{i=1}^{n} w(x_i)x_i^n & \sum_{i=1}^{n} w(x_i)x_i^{n+1} & \sum_{i=1}^{n} w(x_i)x_i^{n+2} & \cdots & \sum_{i=1}^{n} w(x_i)x_i^{2n} \end{bmatrix} \begin{Bmatrix} a_0 \\ a_1 \\ a_2 \\ \cdots \\ a_n \end{Bmatrix} = \begin{Bmatrix} \sum_{i=1}^{n} w(x_i)f(x_i) \\ \sum_{i=1}^{n} w(x_i)x_i f(x_i) \\ \sum_{i=1}^{n} w(x_i)x_i^2 f(x_i) \\ \cdots \\ \sum_{i=1}^{n} w(x_i)x_i^n f(x_i) \end{Bmatrix} \quad (6.1-7)$$

Or if $w(x_i) = 1$

$$\begin{bmatrix} n = \sum_{i=1}^{n} x_i^0 & \sum_{i=1}^{n} x_i & \sum_{i=1}^{n} x_i^2 & \cdots & \sum_{i=1}^{n} x_i^n \\ \sum_{i=1}^{n} x_i & \sum_{i=1}^{n} x_i^2 & \sum_{i=1}^{n} x_i^3 & \cdots & \sum_{i=1}^{n} x_i^{n+1} \\ \sum_{i=1}^{n} x_i^2 & \sum_{i=1}^{n} x_i^3 & \sum_{i=1}^{n} x_i^4 & \cdots & \sum_{i=1}^{n} x_i^{n+2} \\ \cdots & \cdots & \cdots & \cdots & \cdots \\ \sum_{i=1}^{n} x_i^n & \sum_{i=1}^{n} x_i^{n+1} & \sum_{i=1}^{n} x_i^{n+2} & \cdots & \sum_{i=1}^{n} x_i^{2n} \end{bmatrix} \begin{Bmatrix} a_0 \\ a_1 \\ a_2 \\ \cdots \\ a_n \end{Bmatrix} = \begin{Bmatrix} \sum_{i=1}^{n} f(x_i) \\ \sum_{i=1}^{n} x_i f(x_i) \\ \sum_{i=1}^{n} x_i^2 f(x_i) \\ \cdots \\ \sum_{i=1}^{n} x_i^n f(x_i) \end{Bmatrix} \quad (6.1-8)$$

Program 6.1-1 Polynomial least square curve fitting

```
 import java.io.*;
import javax.swing.*;
import java.lang.*;
import java.util.Locale;

class NA44
{ // Polynomial least square
  public static double[] gausswithpartialpivot(double a[][],double b[])
  { //Gauss elimination with partial pivoting
  int n=b.length;
  double x[]=new double[n];
  double carpan=0;
  double toplam=0;
  double buyuk;
```

```
  double dummy=0;
  //gauss elimination
  int i,j,k,p,ii,jj;
  for(k=0;k<(n-1);k++)
  { //partial pivoting
                p=k;
                buyuk=Math.abs(a[k][k]);
                for(ii=k+1;ii<n;ii++)
                {  dummy=Math.abs(a[ii][k]);
                   if(dummy > buyuk) {buyuk=dummy;p=ii;}
                }
                if(p!=k)
                { for(jj=k;jj<n;jj++)
                  { dummy=a[p][jj];
                    a[p][jj]=a[k][jj];
                    a[k][jj]=dummy;
                  }
                  dummy=b[p];
                  b[p]=b[k];
                  b[k]=dummy;
                }
                //
    for(i=k+1;i<n;i++)
    {  carpan=a[i][k]/a[k][k];
       a[i][k]=0;
       for(j=k+1;j<n;j++)
       {   a[i][j]-=carpan*a[k][j]; }
          b[i]   =b[i]   -carpan*b[k];
                }
  }
  //backward substitution
  x[n-1]=b[n-1]/a[n-1][n-1];
  for(i=n-2;i>=0;i--)
  {
    toplam=0;
    for(j=i+1;j<n;j++)
    {  toplam+=a[i][j]*x[j];}
    x[i]=(b[i]-toplam)/a[i][i];
  }
  return x;
}

public static double[] PolynomialLSQ(double xi[],double yi[],int n)
{ //Polynomial least square
int l=xi.length;
int i,j,k;
int np1=n+1;
double A[][];
A=new double[np1][np1];
double B[];
B=new double[np1];
double X[];
X=new double[np1];
for(i=0;i<n+1;i++)
  {  for(j=0;j<n+1;j++)
     {if(i==0 && j==0) A[i][j]=l;
     else  for(k=0;k<l;k++) A[i][j] += Math.pow(xi[k],(i+j));
     }
     for(k=0;k<l;k++) { if(i==0)  B[i]+= yi[k];
                   else      B[i] += Math.pow(xi[k],i)*yi[k];}
  }
 System.out.println("A="+Matrix.toString(A));
 System.out.println("B="+Matrix.toString(B));
X=gausswithpartialpivot(A,B);
```

```
double max=0;
for(i=0;i<n+1;i++)
 if(Math.abs(X[i]) > max) max = Math.abs(X[i]);
for(i=0;i<n+1;i++)
 if((Math.abs(X[i]/max) > 0) && (Math.abs(X[i]/max) < 1.0e-100)) X[i]=0;
return X;
}

public static double funcPolynomialLSQ(double e[],double x)
{
// this function calculates the value of
// least square curve fitting function
int n=e.length;
double ff;
if(n!=0.0)
  { ff=e[n-1];
  for(int i=n-2;i>=0;i--)
   { ff=ff*x+e[i]; }
  }
else
  ff=0;
return ff;
}

public static double error(double x[],double y[],double e[])
{
//calculates absolute square root error of a least square approach
double n=x.length;
 int k;
 double total=0;
 for(k=0;k<n;k++)
 {
 total+=(y[k]-funcPolynomialLSQ(e,x[k]))*(y[k]-funcPolynomialLSQ(e,x[k]));
 }
total=Math.sqrt(total);
return total;
}

public static double[][]  funcPolynomialLSQ(double xi[],double yi[],int polinomkatsayisi,int aradegersayisi)
 {
              //aradegersayisi:  x--o--o--x--o--o--x zincirinde x deneysel noktalar ise
              // ara değer sayisi 2 dir
               int n=xi.length;
              int nn=(n-1)*(aradegersayisi+1)+1;
              double z[][]=new double[2][nn];
   double E[]=PolynomialLSQ(xi,yi,polinomkatsayisi);
   System.out.println("katsayilar :\n"+Matrix.toStringT(E));
   double dx=0;
   int k=0;
   int i;
   for(i=0;i<(n-1);i++)
     {z[0][k]=xi[i];z[1][k]=funcPolynomialLSQ(E,z[0][k]);k++;
     for(int j=0;j<aradegersayisi;j++)
     {dx=(xi[i+1]-xi[i])/((double)aradegersayisi+1.0);
     z[0][k]=z[0][k-1]+dx;z[1][k]=funcPolynomialLSQ(E,z[0][k]);k++;}
     }
     z[0][k]=xi[i];z[1][k]=funcPolynomialLSQ(E,z[0][k]);
   return z;
 }
 public static String toString(double z[][])
 { String s="";
   for(int i=0;i<z.length;i++)
   {  for(int j=0;j<z[0].length;j++)
                {s+=String.format("%f ",z[i][j]);}
```

```
                s+="\n";
    }
    return s;
  }
  public static String toStringT(double z[][])
  { String s="";
    for(int i=0;i<z[0].length;i++)
    {  for(int j=0;j<z.length;j++)
                {s+=String.format(Locale.US,"%f ",z[j][i]);}
                s+="\n";
    }
    return s;
  }
  public static void main(String args[]) throws IOException
  {
  double x[];
  double y[];
  String s1=JOptionPane.showInputDialog("Data file name (input1.txt)");
  // JFileChooser fc=new JFileChooser();
  // if (fc.showOpenDialog(null) == JFileChooser.APPROVE_OPTION) {File file = fc.getSelectedFile();s1=file.getName(); }
  double a[][]=Text.readDoubleT(s1);
  x=a[0];
  y=a[1];
  double z[][]=funcPolynomialLSQ(x,y,2,2);
  System.out.println("Polynomial Least Square Curve fitting\n"+toStringT(z));
  Plot pp=new Plot(a[0],a[1]);
  pp.setPlabel("Polynomial Least Square Curve fitting");
  pp.setXlabel("x");
  pp.setYlabel("y=f(x)");
  pp.setPlotType(0,22);
  pp.addData(z[0],z[1]);
  pp.setPlotType(1,0);
  pp.setGrid(1,1);
  pp.plot();
  }
}
```

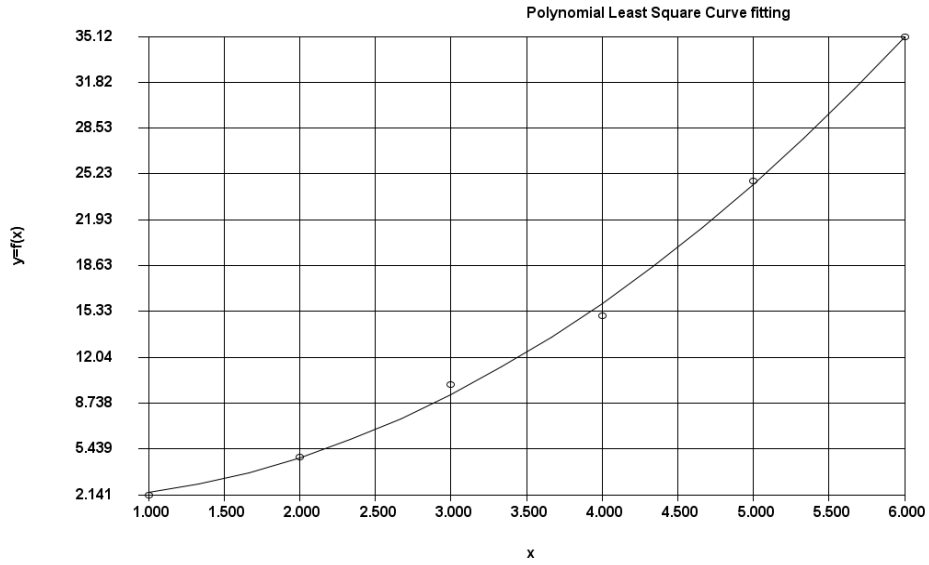
```
 ---------- Capture Output ----------
> "E:\co\java\bin\java.exe" NA44
A=       6.000000000000000       21.000000000000000       91.000000000000000
      21.000000000000000       91.000000000000000      441.000000000000000
      91.000000000000000      441.000000000000000     2275.000000000000000

B=       91.921799999999990            436.484110000000000            2235.058210000000000

katsayilar :
           1.770298000000101
          -0.493580857142935
           1.007310000000011

Polynomial Least Square Curve fitting
1.000000 2.284027
1.333333 2.902964
1.666667 3.745747
2.000000 4.812376
2.333333 6.102853
2.666667 7.617176
3.000000 9.355345
3.333333 11.317362
3.666667 13.503225
4.000000 15.912935
4.333333 18.546491
4.666667 21.403894
```

```
5.000000 24.485144
5.333333 27.790240
5.666667 31.319183
6.000000 35.071973
```



Program 6.1-2 Polynomial least square curve fitting (Python version)

```
 # -*- coding: utf-8 -*-
"""
Created on Sat Aug 25 09:31:06 2018

@author: Mustafa Turhan Çoban
"""
from math import *
from gauss import *;
import numpy as np;
import matplotlib.pyplot as plt;

def PolynomialLSQ(xi,yi,n):
    #Polynomial least square
    m=len(xi);
    print("l=",m);
    np1=n+1;
    print("np1=",np1);
    A = [[0 for x in range(np1)] for y in range(np1)];
    B =[0 for x in range(np1)];
    X =[0 for x in range(np1)];
    for i in range(np1):
        for j in range(np1):
            if i==0 and j==0:
                A[i][j]=m;
            else:
                for k in range(m):
                    pp=pow(xi[k],(i+j));
                    A[i][j]= A[i][j]+pow(xi[k],(i+j));
        for k in range(m):
            if i==0 and j==0:
                B[i]=  B[i]+yi[k];
            else:
                B[i]= B[i]+pow(xi[k],i)*yi[k];
    X=gauss(A,B);
    return X;
```

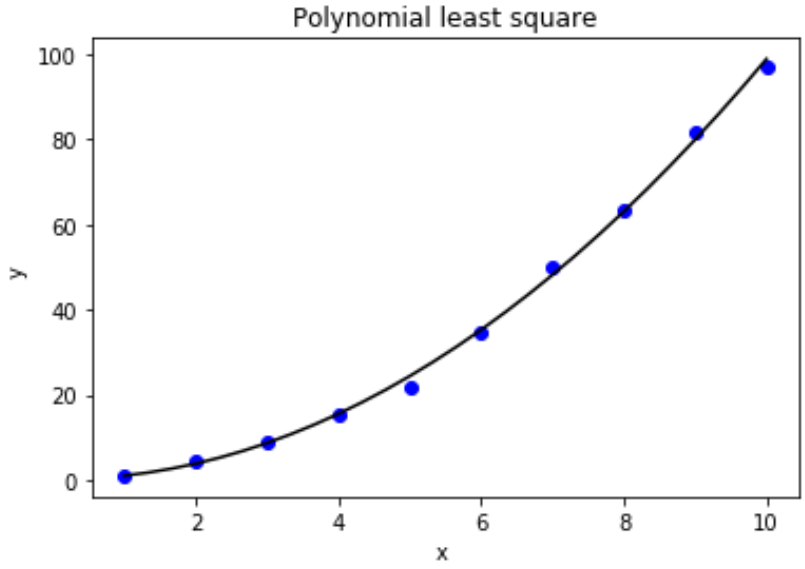
```
def funcPolynomialLSQ(e,x):
   # this function calculates the value of
   # least square curve fitting function
   n=len(e);
   if n!=0.0:
      ff=e[n-1];
      for i in range(n-2,-1,-1):
         ff=ff*x+e[i];
   else:
      ff=0;
   return ff;

def listPolynomialLSQ(E,xi,aradegersayisi):
    #aradegersayisi:  x--o--o--x--o--o--x zincirinde x deneysel noktalar ise
                # ara değer sayisi 2 dir
    n=len(xi);
    nn=(n-1)*(aradegersayisi+1)+1;
    z= [[0 for x in range(nn)] for y in range(2)];
    dx=0;
    k=0;
    for i in range(n-1):
       z[0][k]=xi[i];
       z[1][k]=funcPolynomialLSQ(E,z[0][k]);
       k=k+1;
       for j in range(aradegersayisi):
          dx=(xi[i+1]-xi[i])/(aradegersayisi+1.0);
          z[0][k]=z[0][k-1]+dx;
          z[1][k]=funcPolynomialLSQ(E,z[0][k]);
          k=k+1;
       z[0][k]=xi[i+1];z[1][k]=funcPolynomialLSQ(E,z[0][k]);
    return z;

a=np.loadtxt('a.txt');
print("a",a);
n=len(a);
x =[0 for x in range(n)];
y =[0 for x in range(n)];
for i in range(n):
   x[i]=a[i][0];
   y[i]=a[i][1];
print("x",x);
print("y",y);
E=PolynomialLSQ(x,y,2);
print("E=",E);
Z=listPolynomialLSQ(E,x,2);
print("Z=",Z[0]);
plt.title('Polynomial least square')
plt.xlabel('x ')
plt.ylabel('y ');
plt.plot(x,y,'bo',Z[0],Z[1],'k');
```

```
runfile('C:/Users/TURHAN/Desktop/turhan1/NA44.py', wdir='C:/Users/TURHAN/Desktop/turhan1')
Reloaded modules: gauss
a [[ 1.       1.2452 ]
 [ 2.       4.34534]
 [ 3.       8.9746 ]
 [ 4.      15.2    ]
 [ 5.      22.     ]
 [ 6.      34.8975 ]
 [ 7.      50.12378]
 [ 8.      63.61761]
 [ 9.      81.74675]
 [10.      97.     ]]
```

```
x [1.0, 2.0, 3.0, 4.0, 5.0, 6.0, 7.0, 8.0, 9.0, 10.0]
y [1.2452, 4.34534, 8.9746, 15.2, 22.0, 34.8975, 50.12378, 63.61761, 81.74675, 97.0]
l= 10
np1= 3
E= [0.2239984999998932, -0.1868977045453987, 1.005688749999995]
Z= [1.0, 1.3333333333333333, 1.6666666666666665, 2.0, 2.3333333333333335, 2.666666666666667, 3.0, 3.3333333333333335,
3.666666666666667, 4.0, 4.333333333333333, 4.666666666666666, 5.0, 5.333333333333333, 5.666666666666666, 6.0,
6.333333333333333, 6.666666666666666, 7.0, 7.333333333333333, 7.666666666666666, 8.0, 8.333333333333334, 8.666666666666668,
9.0, 9.333333333333334, 9.666666666666668, 10.0]
```



A general function version of the leat square program will look like :

Program 6.1-3 General least square curve fitting

```
import java.io.*;
import java.util.*;
import javax.swing.*;

//prototype function
abstract class f_xr
{
  abstract double func(double x,int equation_ref);
}

class fa extends f_xr
{
                double func(double T,int i)
                {
    double xx=1.0;
     if(i==0) xx=1.0;
     else if(i==1) xx=1.0/T;
     else if(i==2) xx=1.0/(T*T);
      else if(i==3) xx=1.0/(T*T*T);
     else if(i==4) xx=T;
     else if(i==5) xx=T*T;
     else if(i==6) xx=T*T*T;
  return xx;
                }
}

class NA45
{
//general least square curve fitting
```

```
  public static double[] gausswithpartialpivot(double a[][],double b[])
  { //Gauss elimination with partial pivoting
  int n=b.length;
  double x[]=new double[n];
  double carpan=0;
  double toplam=0;
  double buyuk;
  double dummy=0;
  //gauss elimination
  int i,j,k,p,ii,jj;
  for(k=0;k<(n-1);k++)
  { //partial pivoting
                p=k;
                buyuk=Math.abs(a[k][k]);
                for(ii=k+1;ii<n;ii++)
                {  dummy=Math.abs(a[ii][k]);
                  if(dummy > buyuk) {buyuk=dummy;p=ii;}
                }
                if(p!=k)
                { for(jj=k;jj<n;jj++)
                  { dummy=a[p][jj];
                    a[p][jj]=a[k][jj];
                    a[k][jj]=dummy;
                  }
                  dummy=b[p];
                  b[p]=b[k];
                  b[k]=dummy;
                }
                //
    for(i=k+1;i<n;i++)
    {  carpan=a[i][k]/a[k][k];
       a[i][k]=0;
       for(j=k+1;j<n;j++)
       {   a[i][j]-=carpan*a[k][j]; }
          b[i]   =b[i]   -carpan*b[k];
                }
  }
  //backward substitution
  x[n-1]=b[n-1]/a[n-1][n-1];
  for(i=n-2;i>=0;i--)
  {
    toplam=0;
    for(j=i+1;j<n;j++)
    {  toplam+=a[i][j]*x[j];}
    x[i]=(b[i]-toplam)/a[i][i];
  }
  return x;
}


public static double[] GeneralLeastSquare(double xi[],double yi[],int n)
{
fa f=new fa();
int l=xi.length;
int i,j,k;
int np1=n+1;
double A[][];
A=new double[np1][np1];
double B[];
B=new double[np1];
double X[];
X=new double[np1];
for(i=0;i<n+1;i++)
  {  B[i]=0;
```

```
            for(j=0;j<np1;j++)
    {
            A[i][j]=0.0;
    for(k=0;k<l;k++) A[i][j]+=f.func(xi[k],i)*f.func(xi[k],j);
    }
    for(k=0;k<l;k++) B[i]+= f.func(xi[k],i)*yi[k];
  }
//System.out.println("A = \n"+Matrix.toString(A));
//System.out.println("B = \n"+Matrix.toStringT(B));
X=gausswithpartialpivot(A,B);
//X=B/A;
double max=0;
for(i=0;i<n+1;i++)
 if(Math.abs(X[i]) > max) max = Math.abs(X[i]);
for(i=0;i<n+1;i++)
 if((Math.abs(X[i]/max) > 0) && (Math.abs(X[i]/max) < 1.0e-100)) X[i]=0;
Text.printT(X);
return X;
}

public static double funcGeneralLeastSquare(double e[],double x)
{
// this function calculates the value of
// least square curve fitting function
fa f=new fa();
int n=e.length;
double ff=0;
if(n!=0.0)
  {
  for(int i=n-1;i>=0;i--)
   {ff+=e[i]*f.func(x,i);}
  }
return ff;
}

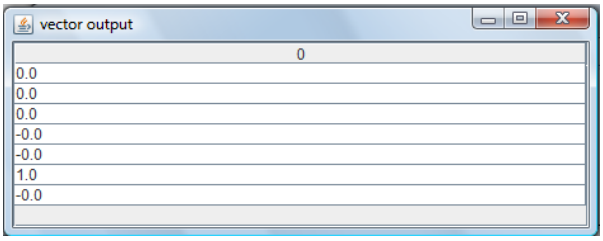
public static double hata(double x[],double y[],double e[])
{
//calculates absolute square root error of a least square approach
double n=x.length;
 int k;
 double total=0;
 for(k=0;k<n;k++)
 {
 total+=(y[k]-funcGeneralLeastSquare(e,x[k]))*(y[k]-funcGeneralLeastSquare(e,x[k]));
 }
total=Math.sqrt(total);
return total;
}

 public static double[][]  funcGeneralLeastSquare(double xi[],double yi[],int polinomkatsayisi,int aradegersayisi)
 {
            //aradegersayisi:  x--o--o--x--o--o--x zincirinde x deneysel noktalar ise
            // ara değer sayisi 2 dir
             int n=xi.length;
            int nn=(n-1)*(aradegersayisi+1)+1;
            double z[][]=new double[2][nn];
   double E[]=GeneralLeastSquare(xi,yi,polinomkatsayisi);
   double dx=0;
   int k=0;
   int i;
   for(i=0;i<(n-1);i++)
     {z[0][k]=xi[i];z[1][k]=funcGeneralLeastSquare(E,z[0][k]);k++;
     for(int j=0;j<aradegersayisi;j++)
     {dx=(xi[i+1]-xi[i])/((double)aradegersayisi+1.0);
```

```
      z[0][k]=z[0][k-1]+dx;z[1][k]=funcGeneralLeastSquare(E,z[0][k]);k++;}
      }
      z[0][k]=xi[i];z[1][k]=funcGeneralLeastSquare(E,z[0][k]);
    return z;
  }

  public static void main(String args[]) throws IOException
  {
  String s1=JOptionPane.showInputDialog("file name : ");
  double c[][]=Text.readDoubleT(s1);
  double b[]=GeneralLeastSquare(c[0],c[1],6);
  System.out.println("General Least Square Coefficient : \n"+Matrix.toStringT(b));
  double z[][]=funcGeneralLeastSquare(c[0],c[1],6,2);
  System.out.println("General Least Square Solution \n"+Matrix.toStringT(z));
  Plot pp=new Plot(c[0],c[1]);
  pp.setPlabel("General Least Square");
  pp.setXlabel("x");
  pp.setYlabel("y=f(x)");
  pp.addData(z[0],z[1]);
  pp.plot();
  }

}
```




```
---------- Capture Output ----------
> "C:\turhan\java\bin\javaw.exe" NA45
General Least Square Coefficient :
       0.000000000000000
       0.000000000000000
       0.000000000000000
      -0.000000000000000
      -0.000000000000000
       1.000000000000000
      -0.000000000000000

General Least Square Solution
   1.0000000000   1.0000000000
   1.3333333333   1.7777777778
   1.6666666667   2.7777777778
   2.0000000000   4.0000000000
   2.3333333333   5.4444444444
   2.6666666667   7.1111111111
   3.0000000000   9.0000000000
   3.3333333333  11.1111111111
   3.6666666667  13.4444444444
   4.0000000000  16.0000000000
   4.3333333333  18.7777777778
   4.6666666667  21.7777777778
   5.0000000000  25.0000000000
   5.3333333333  28.4444444444
```

```
 5.6666666667  32.1111111111
 6.0000000000  36.0000000000
 6.3333333333  40.1111111111
 6.6666666667  44.4444444444
 7.0000000000  49.0000000000
 7.3333333333  53.7777777778
 7.6666666667  58.7777777778
 8.0000000000  64.0000000000
```

Program 6.1-4 General least square curve fitting (Python version)

```
# -*- coding: utf-8 -*-
"""
Created on Sat Aug 25 09:31:06 2018

@author: Mustafa Turhan Çoban
"""
from math import *
from gauss import *;
import numpy as np;
import matplotlib.pyplot as plt;
from abc import ABC, abstractmethod
class f_xr(ABC):

    @abstractmethod
    def func(self,x,equation_ref):
        pass;

class f1(f_xr):
  def func(this,x,i):
      xx=0;
      if i==0: xx=1.0;
      elif i==1:xx=x;
      elif i==2:xx=x*x;
      elif i==3:xx=x*x*x;
      elif i==4:xx=x*x*x*x;
      elif i==5:xx=x*x*x*x*x;
      elif i==6:xx=x*x*x*x*x*x;
      return xx;

def PolynomialLSQ(f,xi,yi,n):
    #Polynomial least square
    m=len(xi);
    print("l=",m);
    np1=n+1;
    print("np1=",np1);
    A = [[0 for x in range(np1)] for y in range(np1)];
    B =[0 for x in range(np1)];
    X =[0 for x in range(np1)];
    for i in range(np1):
        for j in range(np1):
            for k in range(m):
                pp=pow(xi[k],(i+j));
                A[i][j]= A[i][j]+f.func(xi[k],i)*f.func(xi[k],j);
        for k in range(m):
            B[i]= B[i]+f.func(xi[k],i)*yi[k];
    X=gauss(A,B);
    return X;

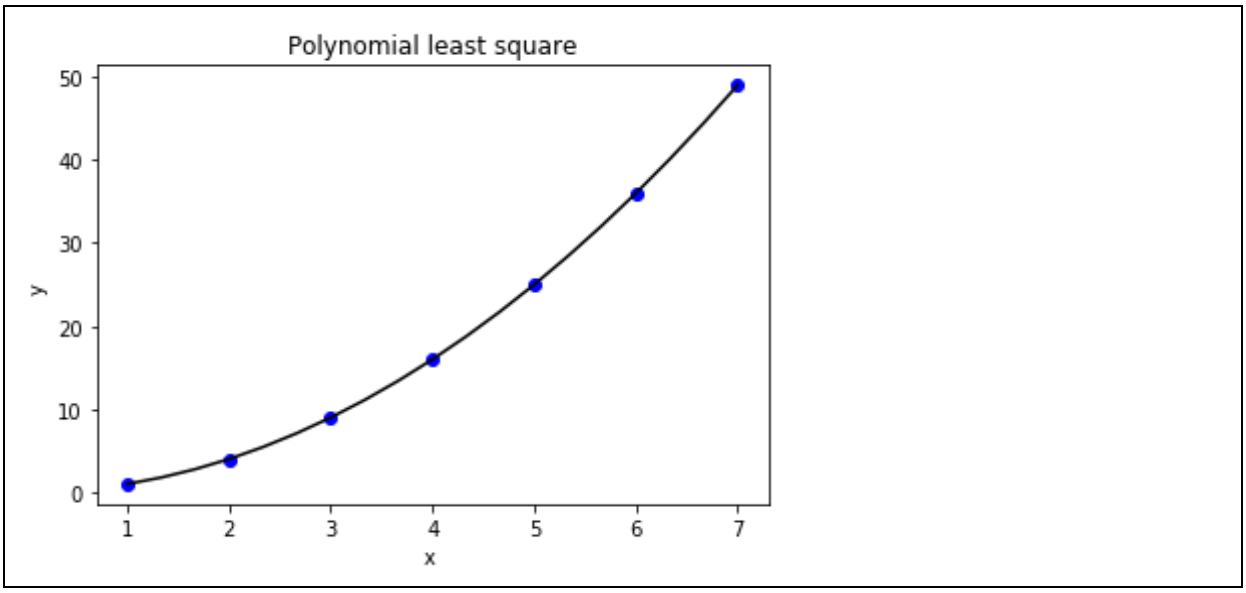
def funcPolynomialLSQ(f,e,x):
    # this function calculates the value of
    # least square curve fitting function
    n=len(e);
```

```
    ff=0;
    if n!=0.0:
        for i in range(n-1,-1,-1):
            ff=ff+e[i]*f.func(x,i);
    return ff;

def listPolynomialLSQ(f,E,xi,aradegersayisi):
    #aradegersayisi:  x--o--o--x--o--o--x zincirinde x deneysel noktalar ise
                # ara değer sayisi 2 dir
    n=len(xi);
    nn=(n-1)*(aradegersayisi+1)+1;
    z= [[0 for x in range(nn)] for y in range(2)];
    dx=0;
    k=0;
    for i in range(n-1):
        z[0][k]=xi[i];
        z[1][k]=funcPolynomialLSQ(f,E,z[0][k]);
        k=k+1;
        for j in range(aradegersayisi):
            dx=(xi[i+1]-xi[i])/(aradegersayisi+1.0);
            z[0][k]=z[0][k-1]+dx;
            z[1][k]=funcPolynomialLSQ(f,E,z[0][k]);
            k=k+1;
        z[0][k]=xi[i+1];z[1][k]=funcPolynomialLSQ(f,E,z[0][k]);
    return z;

a=np.loadtxt('a.txt');
print("a",a);
n=len(a);
x =[0 for x in range(n)];
y =[0 for x in range(n)];
for i in range(n):
    x[i]=a[i][0];
    y[i]=a[i][1];
print("x",x);
print("y",y);
f=f1();
E=PolynomialLSQ(f,x,y,2);
print("E=",E);
Z=listPolynomialLSQ(f,E,x,2);
print("Z=",Z[0]);
plt.title('Polynomial least square')
plt.xlabel('x ')
plt.ylabel('y ');
plt.plot(x,y,'bo',Z[0],Z[1],'k');
```

```
runfile('D:/okul/SCO1/gen_poly_least_sqr.py', wdir='D:/okul/SCO1')
Reloaded modules: gauss
a [[ 1.  1.]
 [ 2.  4.]
 [ 3.  9.]
 [ 4. 16.]
 [ 5. 25.]
 [ 6. 36.]
 [ 7. 49.]]
x [1.0, 2.0, 3.0, 4.0, 5.0, 6.0, 7.0]
y [1.0, 4.0, 9.0, 16.0, 25.0, 36.0, 49.0]
l= 7
np1= 3
E= [0.0, -0.0, 1.0]
Z= [1.0, 1.3333333333333333, 1.6666666666666665, 2.0, 2.3333333333333335, 2.666666666666667, 3.0, 3.3333333333333335,
3.666666666666667, 4.0, 4.333333333333333, 4.666666666666666, 5.0, 5.333333333333333, 5.666666666666666, 6.0,
6.333333333333333, 6.666666666666666, 7.0]
```



An excel or Octave version of polynomial least square curve fitting is given below. In order to carry out curve fitting, data (x,y) and x^2,x^3,x^4,.. should be defined and LINEST function (DOT in turkish version) should be used with ctrl-shift-enter
=DOT(G2:G9;B2:F9;DOĞRU;DOĞRU)
=LINEST(G2:G9;B2:F9;TRUE;TRUE)

x	x^2	x^3	x^4	x^5	y	f(x)=a0+a1*x+..
1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1
2.0000	4.0000	4.0000	16.0000	32.0000	4.0000	4
3.0000	9.0000	9.0000	81.0000	243.0000	9.0000	9
4.0000	16.0000	16.0000	256.0000	1024.0000	16.0000	16
5.0000	25.0000	25.0000	625.0000	3125.0000	25.0000	25
6.0000	36.0000	36.0000	1296.0000	7776.0000	36.0000	36
7.0000	49.0000	49.0000	2401.0000	16807.0000	49.0000	49
8.0000	64.0000	64.0000	4096.0000	32768.0000	64.0000	64

a0	a1	a2	a3	a4	a5	turkish	english	
6.91E-19	-9.81031E-18	1	0	0	-	DOT	LINEST	ctrl-shift-enter

An excel or Octave version of polynomial least square curve fitting with our own gauss elimination linear system of equation solution is given below:

n	x	y	x^2	x^3	x^4	x*y	x^2*y
1	1	1.2452	1	1	1	1.2452	1.2452
1	2	4.34534	4	8	16	8.69068	17.38136
1	3	8.9746	9	27	81	26.9238	80.7714
1	4	15.2	16	64	256	60.8	243.2
1	5	22	25	125	625	110	550
1	6	34.8975	36	216	1296	209.385	1256.31
1	7	50.12378	49	343	2401	350.86646	2456.0652
1	8	63.61761	64	512	4096	508.94088	4071.527
1	9	81.74675	81	729	6561	735.72075	6621.4868
1	10	97	100	1000	10000	970	9700
10	55	379.15078	385	3025	25333	2982.5728	24997.987

GAUSS ELIMINATION 3 UNKNOWN

	A			B
	10	55	385	379.15078
	55	385	3025	2982.5728
	385	3025	25333	24997.987

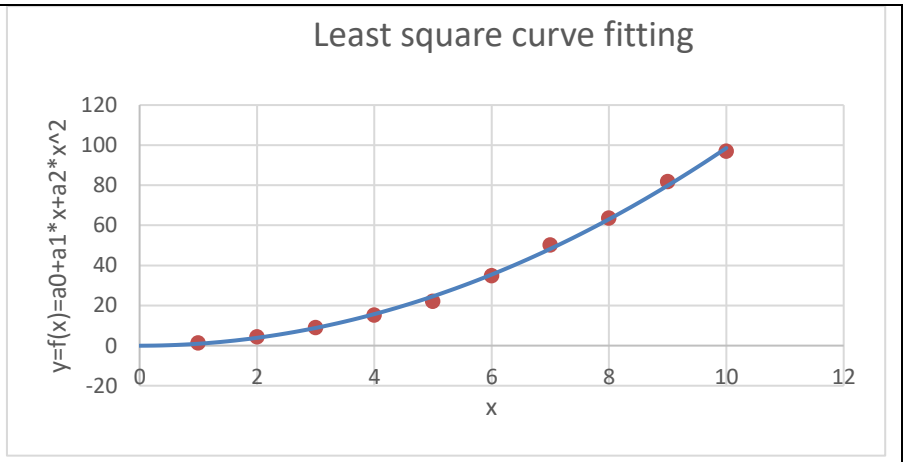
	10	55	385	379.15078
5.5	55	385	3025	2982.5728
38.5	385	3025	25333	24997.987

	10	55	385	379.15078
7.8571429	55	385	3025	2982.5728
	0	3025	25333	24997.987

x0	-0.0411622
x1	-0.0062948
x2	0.9867756

created data

x	y
0	-0.0411622
0.25	0.0189376
0.5	0.2023843
0.75	0.509178
1	0.9393186
1.25	1.4928062
1.5	2.1696408
1.75	2.9698223
2	3.8933507
2.25	4.9402261
2.5	6.1104485
2.75	7.4040178
3	8.820934
3.25	10.361197
3.5	12.024807
3.75	13.811765
4	15.722069
4.25	17.75572



6.2 LINEAR CURVE FITTING : ORTOGONAL POLYNOMIAL LEAST SQUARE

In the previous subsection , polynomial least square method is investigated. In order to find coefficient of polynomial least square a matrix solution process should be carried out. Furthermore, the equation tends to create ill matrices, so that chance of having an error in the coefficient while solving the matrix is high. Orthogonal polynomials has the following usefull property

$$\begin{Bmatrix} p_i(x)p_j(x) = 0 & if \;\; i \neq j \\ p_i(x)p_j(x) = A & if \;\; i = j \end{Bmatrix} \quad (6.2-1)$$

Now Assume to have a jth order orthogonal polynomial$p_j(x)$. And assume that least square polynomial equation is

$$y_m(x) = \sum_{j=0}^{m} b_j^{(m)} p_j(x) \quad (6.2-2)$$

If x_i, f_i i=1,…,n is given as a data to be curve fit, The minimization function can be defined as

$$H\left(b_0^{(m)},..,b_0^{(m)}\right)=\sum_{i=1}^{n}w(x_i)[f(x_i)-y_m(x)]^2=\sum_{i=1}^{n}w(x_i)\left[f(x_i)-\sum_{j=0}^{m}b_j^{(m)}p_j(x)\right]^2 \quad (6.2-3)$$

In order to minimize the function, derivative can be set as equal to zero

$$\frac{\partial H\left(b_0^{(m)},..,b_0^{(m)}\right)}{\partial b_k^{(m)}}=\sum_{i=1}^{n}w(x_i)[f(x_i)-y_m(x)]^2=\sum_{i=1}^{n}w(x_i)\left[f(x_i)-\sum_{j=0}^{m}b_j^{(m)}p_j(x)\right]p_k(x)=0 \;\; k=0,..,m \quad (6.2-3)$$

for a shorter notation new variables can be defined as:

$$d_{jk}\sum_{i=1}^{n}w(x_i)p_j(x_i)\,p_k(x_i) \quad (6.2-4)$$

$$\omega_{jk}=\sum_{i=1}^{n}w(x_i)p_n(x_i)f(x_i) \quad (6.2-5)$$

$$\sum_{j=0}^{m}d_{jk}\left[b_j^{(m)}\right]=\omega_{jk} \quad k=0,..,m \quad (6.2-6)$$

This equation came to the following matrix form

$$\begin{bmatrix}\sum_{i=1}^{n}w(x_i)p_0(x_i)\,p_0(x_i) & \sum_{i=1}^{n}w(x_i)p_0(x_i)\,p_1(x_i) & \sum_{i=1}^{n}w(x_i)p_0(x_i)\,p_2(x_i) & \cdots & \sum_{i=1}^{n}w(x_i)p_0(x_i)\,p_n(x_i)\\ \sum_{i=1}^{n}w(x_i)p_1(x_i)\,p_0(x_i) & \sum_{i=1}^{n}w(x_i)p_1(x_i)\,p_1(x_i) & \sum_{i=1}^{n}w(x_i)p_1(x_i)\,p_2(x_i) & \cdots & \sum_{i=1}^{n}w(x_i)p_1(x_i)\,p_n(x_i)\\ \sum_{i=1}^{n}w(x_i)p_2(x_i)\,p_0(x_i) & \sum_{i=1}^{n}w(x_i)p_2(x_i)\,p_1(x_i) & \sum_{i=1}^{n}w(x_i)p_2(x_i)\,p_2(x_i) & \cdots & \sum_{i=1}^{n}w(x_i)p_2(x_i)\,p_n(x_i)\\ \cdots & \cdots & \cdots & \cdots & \cdots\\ \sum_{i=1}^{n}w(x_i)p_n(x_i)\,p_0(x_i) & \sum_{i=1}^{n}w(x_i)p_n(x_i)\,p_1(x_i) & \sum_{i=1}^{n}w(x_i)p_n(x_i)\,p_2(x_i) & \cdots & \sum_{i=1}^{n}w(x_i)p_n(x_i)\,p_n(x_i)\end{bmatrix}\begin{Bmatrix}b_0\\b_1\\b_2\\\cdots\\b_n\end{Bmatrix}=$$

$$\begin{Bmatrix}\sum_{i=1}^{n}w(x_i)p_0(x_i)f(x_i)\\ \sum_{i=1}^{n}w(x_i)p_1(x_i)f(x_i)\\ \sum_{i=1}^{n}w(x_i)p_2(x_i)f(x_i)\\ \cdots\\ \sum_{i=1}^{n}w(x_i)p_n(x_i)f(x_i)\end{Bmatrix} \quad (6.2\text{-}6a)$$

Writing the equation this way does not give any advantage, But considering that $p_k(x_i)$ is an orthogonal function, due to the propertiy of the orthogonal functions. In this case the equation that formed a matrix converted to a simple multiplication equation becomes

$$\begin{bmatrix}\sum_{i=1}^{n}w(x_i)p_0(x_i)\,p_0(x_i) & & & & \\ & \sum_{i=1}^{n}w(x_i)p_1(x_i)\,p_1(x_i) & & & \\ & & \sum_{i=1}^{n}w(x_i)p_2(x_i)\,p_2(x_i) & \cdots & \\ & & & & \sum_{i=1}^{n}w(x_i)p_n(x_i)\,p_n(x_i)\end{bmatrix}\begin{Bmatrix}b_0\\b_1\\b_2\\\cdots\\b_n\end{Bmatrix}=$$

$$\begin{Bmatrix}\sum_{i=1}^{n}w(x_i)p_0(x_i)f(x_i)\\ \sum_{i=1}^{n}w(x_i)p_1(x_i)f(x_i)\\ \sum_{i=1}^{n}w(x_i)p_2(x_i)f(x_i)\\ \cdots\\ \sum_{i=1}^{n}w(x_i)p_n(x_i)f(x_i)\end{Bmatrix} \quad (6.2\text{-}7)$$

$$d_{kk}\left[b_j^{(m)}\right]=\omega_{kk} \quad k=0,\ldots,m \quad (6.2-7a)$$

To solve $b_j^{(m)}$

$$\left[b_j^{(m)}\right]=\frac{\omega_{kk}}{d_{kk}}=\frac{\sum_{i=1}^{n}w(x_i)p_k(x_i)f(x_i)}{\sum_{i=1}^{n}w(x_i)p_k(x_i)\,p_k(x_i)} \quad k=0,..,m \quad (6.2-8)$$

Several different orthogonal polynomials are available. The orthogonal polynomial that we will utilise in our first program has the following form. This equation is taken from "A First Course in Numerical Analysis, Anthony Ralston,Philip Rabinowitz, Mc Graw Hill ISBN 0-07-051158-6" [4]

$$p_{j+1}(x)=(x-\alpha_{j+1})p_j(x)-\beta_j p_{j-1}(x) \quad j=0,1,\ldots \quad (6.2-9)$$

$p_0(x)=1$ $p_{-1}(x)=0$

α_{j+1} and β_j are polynomial constants to be calculated.

$$\beta_k=\frac{\sum_{i=1}^{n}w(x_i)x_i p_{k-1}(x_i)\,p_k(x_i)}{\sum_{i=1}^{n}w(x_i)x_i p_{k-1}(x_i)\,p_{k-1}(x_i)}=\frac{\sum_{i=1}^{n}w(x_i)p_k(x_i)\,p_k(x_i)}{\sum_{i=1}^{n}w(x_i)p_{k-1}(x_i)\,p_{k-1}(x_i)} \quad (6.2-10)$$

$$\alpha_k = \frac{\sum_{i=1}^{n} w(x_i)x_i p_k(x_i)\, p_k(x_i)}{\sum_{i=1}^{n} w(x_i)p_{k-1}(x_i)\, p_{k-1}(x_i)} \quad (6.2-11)$$

Least square algorithm can be given as follows:

$$y_m(x) = \sum_{j=0}^{m} b_j^{(m)} p_j(x)$$

$$b_j = \frac{\omega_j}{\gamma_j}$$

$$\omega_j = \sum_{i=1}^{n} w(x_i)p_j(x_i)f(x_i)$$

$$\gamma_j = \sum_{i=1}^{n} w(x_i)p_j(x_i)p_j(x_i)$$

Because Orthogonal polynomial equations are not solving any system of equations, roundoff errors due to matrix solutions will not be existed. But the polynomial equation used has a more complex form compare to polynomial form.

Program 6.2-1 Orthogonal polynomial least square, Ralston & Rabinowitz equation

```
import java.io.*;

class NA121
{
public static double[][] OPEKK(double xi[],double fi[],int m)
//ortogonal polinomial least square method
//Referance : A First Course in Numerical Analysis
// Anthony Ralston,Philip Rabinowitz, Mc Graw Hill ISBN 0-07-051158-6
// m degree of polynomial xi fi input data
{
int i,j,k;
int n=xi.length;
int mp2=n+2;
int mp1=n+1;
double p[][]=new double[mp2][n];
double gamma[]=new double[mp1];
double beta[]=new double[mp1];
double omega[]=new double[mp1];
double alpha[]=new double[mp1];
double b[]=new double[mp1];
double wi[]=new double[n];
double a[][]=new double[3][mp1];
for(i=0;i<n;i++)
  {
   p[1][i]=1.0;
   p[0][i]=0.0;
   wi[i]=1.0;
  }
gamma[0]=0;
for(i=0;i<n;i++)
  {
  gamma[0]+=wi[i];
  }
beta[0]=0.0;
for(j=0;j<m+1;j++)
```

```
  {
  omega[j]=0;
  for(i=0;i<n;i++)
    {omega[j]+=wi[i]*fi[i]*p[j+1][i];}
  b[j]=omega[j]/gamma[j];
if( j != m)
    {
    alpha[j+1]=0;
    for(i=0;i<n;i++)
      { alpha[j+1]+=wi[i]*xi[i]*p[j+1][i]*p[j+1][i]/gamma[j];}
    for(i=0;i<n;i++)
      {p[j+2][i]=(xi[i]-alpha[j+1])*p[j+1][i]-beta[j]*p[j][i];}
      gamma[j+1]=0;
    for(i=0;i<n;i++)
      {gamma[j+1]+=wi[i]*p[j+2][i]*p[j+2][i];}
    beta[j+1]=gamma[j+1]/gamma[j];
    }
}//end of j
for(j=0;j<m+1;j++)
  {
  a[0][j]=b[j];
  a[1][j]=alpha[j];
  a[2][j]=beta[j];
  }
return a;
}

public static double funcOPEKK(double a[][],double x)
{
// polinom değerleri hesaplama fonksiyonu
double yy=0;
int k;
int m=a[0].length-1;
int mp2=m+2;
double q[];
q=new double[mp2];
//vector<double> q(m+2,0.0);
for(k=m-1;k>=0;k--)
  {
  q[k]=a[0][k]+(x-a[1][k+1])*q[k+1]-a[2][k+1]*q[k+2];
  yy=q[k];
  }
return yy;
}

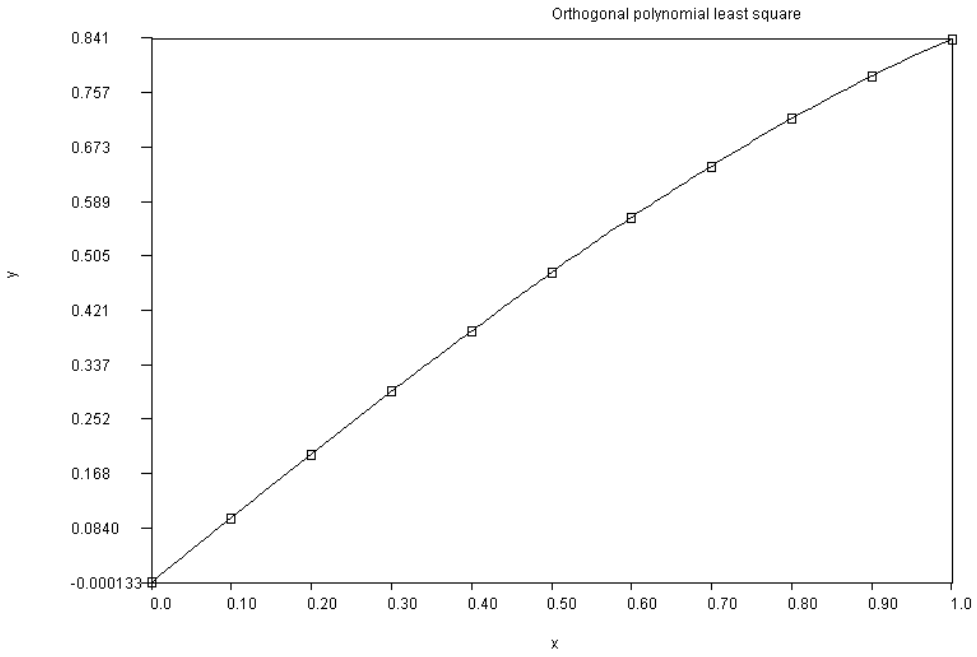
public static double[][]  funcOPEKK(double xi[],double yi[],int polynomialcoefficient,int numberofmidpoints)
 {
            //number of midpoints:  x--o--o--x--o--o--x  x data points midpoints are 2
            int n=xi.length;
            int nn=(n-1)*(numberofmidpoints+1)+1;
            double z[][]=new double[2][nn];
    double E[][]=OPEKK(xi,yi,polynomialcoefficient);
    double dx=0;
    int k=0;
    int i;
    for(i=0;i<(n-1);i++)
      {z[0][k]=xi[i];z[1][k]=funcOPEKK(E,z[0][k]);k++;
```

```
      for(int j=0;j<numberofmidpoints;j++)
      {dx=(xi[i+1]-xi[i])/((double)numberofmidpoints+1.0);
      z[0][k]=z[0][k-1]+dx;z[1][k]=funcOPEKK(E,z[0][k]);k++;}
      }
      z[0][k]=xi[i];z[1][k]=funcOPEKK(E,z[0][k]);
    return z;
  }

  public static void main(String args[]) throws IOException
  {
  double x[]={0.0,0.1,0.2,0.3,0.4,0.5,0.6,0.7,0.8,0.9,1.0};
  double y[]={0.0,0.099833,0.1986693,0.2955202,0.38941834,0.479425,0.5646424,0.6442176,0.717356,0.783326,0.84147};
  double z3[][]=funcOPEKK(x,y,3,8);
  Text.print(Text.T(z3),"Orthogonal polynomial least square ");
  Plot pp=new Plot(x,y);
  pp.setPlabel("Orthogonal polynomial least square ");
  pp.setPlotType(0,20);
  pp.addData(z3[0],z3[1]);
  pp.plot();
  }
}
```

x	y
0	-1.33E-04
0.011111111	0.011022274
0.022222222	0.022172031
0.033333333	0.033314774
0.044444444	0.044449318
0.055555556	0.055574478
0.066666667	0.066689069
0.077777778	0.077791906
0.088888889	0.088881804
0.1	0.099957578
0.111111111	0.111018044
0.122222222	0.122062016
0.133333333	0.13308831
0.144444444	0.14409574
0.155555556	0.155083123
0.166666667	0.166049271
0.177777778	0.176993002
0.188888889	0.18791313
0.2	0.19880847
0.211111111	0.209677837
0.222222222	0.220520046
0.233333333	0.231333912
0.244444444	0.242118251
0.255555556	0.252871877
0.266666667	0.263593606
0.277777778	0.274282253
0.288888889	0.284936632
0.3	0.295555559
0.311111111	0.306137849
0.322222222	0.316682317
0.333333333	0.327187778
0.344444444	0.337653048
0.355555556	0.34807694
0.366666667	0.358458271
0.377777778	0.368795855
0.388888889	0.379088508
0.4	0.389335044
0.411111111	0.399534279
0.422222222	0.409685028
0.433333333	0.419786106

0.444444444	0.429836327
0.455555556	0.439834508
0.466666667	0.449779463
0.477777778	0.459670007
0.488888889	0.469504955
0.5	0.479283123
0.511111111	0.489003325
0.522222222	0.498664376
0.533333333	0.508265092
0.544444444	0.517804288
0.555555556	0.527280779
0.566666667	0.53669338
0.577777778	0.546040905
0.588888889	0.555322171
0.6	0.564535991
0.611111111	0.573681182
0.622222222	0.582756559
0.633333333	0.591760935
0.644444444	0.600693128
0.655555556	0.60955195
0.666666667	0.618336219
0.677777778	0.627044748
0.688888889	0.635676353
0.7	0.644229849
0.711111111	0.652704051
0.722222222	0.661097774
0.733333333	0.669409833
0.744444444	0.677639043
0.755555556	0.68578422
0.766666667	0.693844178
0.777777778	0.701817733
0.788888889	0.709703699
0.8	0.717500892
0.811111111	0.725208127
0.822222222	0.732824219
0.833333333	0.740347982
0.844444444	0.747778233
0.855555556	0.755113785
0.866666667	0.762353455
0.877777778	0.769496058
0.888888889	0.776540407
0.9	0.783485319
0.911111111	0.790329609
0.922222222	0.797072091
0.933333333	0.803711581
0.944444444	0.810246894
0.955555556	0.816676844
0.966666667	0.823000248
0.977777778	0.82921592
0.988888889	0.835322674
1	0.841319327



A second example to orthogonal polynomial curve fitting will be **Chebyshev polynomial** curve fitting. Chebyshev polynomials defined at (-1, 1) region.General definition of Chebyshev polynomials

$T_n(x) = \cos(n.cos^{-1}(x)$ (6.2-12)

If this equation is written in open form

$T_0(x)=1$

$T_1(x)=x$

$T_2(x)=2x^2-x$ (6.2-12a)

…..

$T_{n+1}(x)=2x\ T_n(x)- T_{n-1}(x)$

It is clear that it is a serial equation. The weight function for the Chebchev polynomials

$w(x) = \frac{1}{\sqrt{1-x^2}}$ (6.2-13)

The root of Chebchev polynomials

$t_k = \cos\left(\pi \frac{2k+1}{2n+2}\right)$ k=0,…,n (6.2-13)

n in this equation indicates the degree of polynomial. In order to curve fit data y=f(x) using this set as an orthogonal polynomial data should be given at the root points of chebchev polynomials.

Furthermore due to regional limit of the data, to apply this equation to real data a conversion process is required. If the real data is in region (a,b) The convertion equations will be

$x_k = \left(\frac{b-a}{2}\right) t_k + \left(\frac{a+b}{2}\right)$ (6.2-14)

$t_k = \left(\frac{x_k-a}{b-1}\right) - 1$ (6.2-15)

If x_k and corresponding y_k ,is given

Coefficient of polynomial $P_n(x) = \sum_{k=0}^{n} C_k T_k(x)$ can be calculated by using least sqaure method. Due to orthogonality of the polynomials, matrix solution is not required, coefficients is found directly.

$C_0 = \frac{1}{n+1}\sum_{k=0}^{n} y_k T_0\,(t_k)$

$C_j = \frac{2}{n+1}\sum_{k=0}^{n} y_k T_j\,(t_k)\ \ j = 1,2,\dots,n$ (6.2-16)

As an example problem $f(x)=e^x$ function in the limit 0 to 2 will be investigated. If Chebchev polynomials are to be used in curve fitting, experiments should be design so that the data should be taken exactly in the roots of the Chebchev polynomials.

x_k	y_k
0.010178558	1.010231
0.090368005	1.094577
0.244250426	1.276664
0.459359183	1.583059
0.718267443	2.050877
1	2.718282
1.281732557	3.602877
1.540640817	4.66758
1.755749574	5.787784
1.909631995	6.750604
1.989821442	7.314228

Program 6.2-2 Orthogonal polynomial least square, Chebychev equation

```
import java.io.*;
//Chevychev orthogobnal polynomial least square

class NA122
{
        public static double Ti(double x,int equation_ref)
        {//Chebysev function
        double T[]=new double[equation_ref+1];
        T[0]=1.0;
        if(equation_ref>=1) T[1]=x;
        for(int i=2;i<=equation_ref;i++) {T[i]=2.0*x*T[i-1]-T[i-2];
   }
        return T[equation_ref];
        }
public static double[] xi(double a,double b,int N)
{
  int nn=N+1;
  double x[]=new double[nn];
  double t[]=new double[nn];
  for(int k=0;k<nn;k++)
    {t[k]=Math.cos((2.0*N+1-2.0*k)*Math.PI/(2.0*N+2.0));
     x[k]=(b-a)/2.0*t[k]+(a+b)/2.0;
    }
  return x;
}

public static double[] Chebyshev(double a,double b,double y[],int N)
{   //region a b
   int nn=N+1;
   System.out.println("N="+N);
   int k,j;
   double d=Math.PI/(2.0*N+2);
   double x[]=new double[nn];
   double t[]=new double[nn];
    // -1<=x<=1 region convertion
   for(k=0;k<nn;k++)
    {t[k]=Math.cos((2.0*N+1-2.0*k)*Math.PI/(2.0*N+2.0));
     x[k]=(b-a)/2.0*t[k]+(a+b)/2.0;
    }
    double C[]=new double[nn];
```

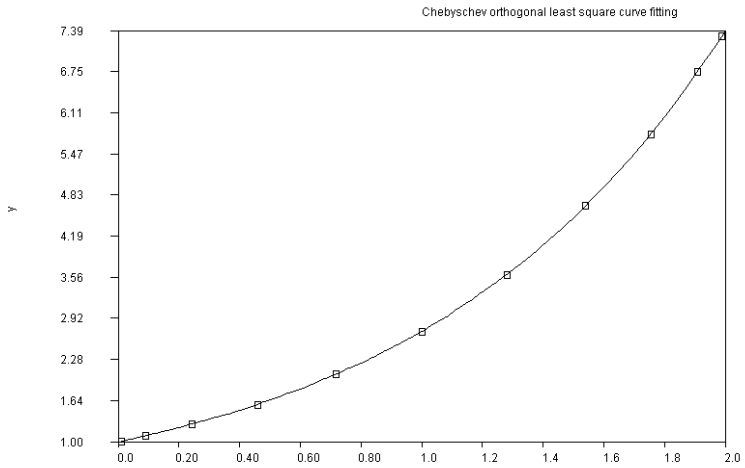
```
            double z=0;
    for(k=0;k<nn;k++)
      { z+=y[k];}
    C[0]=z/(N+1);
    for(j=1;j<=N;j++)
    {z=0;
     for(k=0;k<=N;k++)
     { z+=y[k]*Ti(t[k],j);}
    C[j]=2.0*z/(N+1);
    }
    return C;
}

public static double funcChebyshev(double C[],double a,double b,double x)
{
   double t=2.0*(x-a)/(b-a)-1;
   int n=C.length;
   double ff=0;
   if(n!=0.0)
   {  for(int i=0;i<n;i++)
     {ff+=C[i]*Ti(t,i);}
   }
   return ff;
}

public static double[][]  funcChebyshev(double a,double b,double y[],int polinomkatsayisi,int verisayisi)
 {
  int n=verisayisi+1;
  double z[][]=new double[2][n];
  double C[]=Chebyshev(a,b,y,polinomkatsayisi);
  double dx=(b-a)/(double)verisayisi;
  z[0][0]=a;
  for(int i=0;i<=verisayisi;i++)
    {if(i==0) z[0][i]=a;
     else z[0][i]=z[0][i-1]+dx;
                z[1][i]=funcChebyshev(C,a,b,z[0][i]);        }
   return z;
 }
  public static void main(String args[]) throws IOException
  {
  double a=0;
  double b=2.0;
  double y[]={1.010230536,1.094577019,1.276664,1.583059208,2.05087687,2.718281828,
           3.60287651,4.66758038,5.787784491,6.750604088,7.314227631};
  double x[]= xi(a,b,10);
  double z[][]=funcChebyshev(a,b,y,10,50);
  Text.print(Text.T(z),"Chebyschev orthogonal least square curve fitting");
  Plot pp=new Plot(x,y);
  pp.setPlabel("Chebyschev orthogonal least square curve fitting");
  pp.setPlotType(0,20);
  pp.addData(z[0],z[1]);
  pp.plot();
  }
}
```

x	y

0	1
0.04	1.040810774
0.08	1.083287068
0.12	1.127496852
0.16	1.173510871
0.2	1.221402758
0.24	1.27124915
0.28	1.323129812
0.32	1.377127764
0.36	1.433329415
0.4	1.491824698
0.44	1.552707219
0.48	1.616074402
0.52	1.68202765
0.56	1.750672501
0.6	1.822118801
0.64	1.896480879
0.68	1.973877732
0.72	2.054433211
0.76	2.13827622
0.8	2.225540928
0.84	2.316366976
0.88	2.410899706
0.92	2.509290389
0.96	2.611696473
1	2.718281828
1.04	2.829217014
1.08	2.944679551
1.12	3.064854203
1.16	3.189933276
1.2	3.320116923
1.24	3.455613465
1.28	3.596639726
1.32	3.743421378
1.36	3.896193302
1.4	4.055199967
1.44	4.220695817
1.48	4.392945681
1.52	4.572225195
1.56	4.758821245
1.6	4.953032424
1.64	5.155169512
1.68	5.365555971
1.72	5.584528464
1.76	5.812437394
1.8	6.049647464
1.84	6.296538261
1.88	6.553504862
1.92	6.82095847
1.96	7.099327065
2	7.389056099



6.3 LINEAR CURVE FITTING : LEAST SQUARE METHOD WITH MULTI VARIABLES

Curve fitting of a multivariable function basically the same of general curve fitting of the single variable function. The only basic difference is definition of the function to be used. Assume that

$$f(x_0, x_{1,}.., x_n) = \sum_{j=0}^{m} a_j^{(m)}\, \phi_j(x_0, x_{1,}.., x_n) \qquad (6.3-1)$$

$$f(x_0, x_{1,}.., x_n) = a_0\phi_0(x_0, x_{1,}.., x_n) + a_0\phi_1(x_0, x_{1,}.., x_n) + ..$$

$$\phi_0(x_0, x_1) = \sin(x_0) * \cos(x_1)$$

$$\phi_1(x_0, x_1) = x_0 * x_0 + x_1 * x_1$$

j th degree function is given. It is desired to fit x_0, $x_{1,}$ x_2,…, x_n ,$f_{i,}$ i=0...n data set into this function. The best fitted $a_j^{(m)}$ values to be found. For this purpose Minimum value of

$$H(x_0, x_{1,}.., x_n) = \sum_{i=1}^{n} w(x_0, x_{1,}.., x_n)\left[f_i - \sum_{j=0}^{m} a_j^{(m)}\, \phi_j(x_0, x_{1,}.., x_n)\right]^2 \qquad (6.3-2)$$

function should be found. $w(x_0, x_{1,}.., x_n)$ is called weight function and it should satisfy the condition $w(x_0, x_{1,}.., x_n) \geq 0 \quad i = 1, \dots, n.$ The minimum of the function is the root of the derivative of the function.

$$\frac{\partial H(x_0, x_{1,}.., x_n)}{\partial a_k^{(m)}} = 2\sum_{i=1}^{n} w(x_0, x_{1,}.., x_n)\left[f_i - \sum_{j=0}^{m} a_j^{(m)}\, \phi_j(x_0, x_{1,}.., x_n)\right] \phi_k(x_0, x_{1,}.., x_n) = 0 \quad k = 0,.., m \quad (6.3-2)$$

If unit value is taken as weight function, equation becomes:

$$2\sum_{i=1}^{n}\left[f_i - \sum_{j=0}^{m} a_j^{(m)}\, \phi_j(x_0, x_{1,}.., x_n)\right] \phi_k(x_0, x_{1,}.., x_n) = 0 \qquad k = 0,.., m \quad (6.3-2a)$$

As an example a surface polinomial can be defined as

$f(x,y) = a_0 + a_1x + a_2y + a_3x^2 + a_4y^2 + a_5xy + a_6x^3 + a_7y^3 + a_8x^2y + a_9xy^2 + a_{10}x^4 + a_{11}y^4 + a_{12}x^2y^2 + a_{13}x^3y + a_{14}xy^2$

This polynomial is defined in the example problem

Problem 6.3-1 Multidimensional general least square curve fitting

```
import java.io.*;
import java.util.*;
import javax.swing.*;
import java.awt.event.*;
```

```
// surface curve fitting
//f(x,y)=a0+a1*x+a2*y+a3*x^2+a4*y^2+a5*xy+a6*x^3+a7*y^3+a8*x^2y
//      +a9*xy^2+a10*x^4+a11*y^4+a12*x^2y^2+a13x^3y+a14*xy^3
abstract class f_xir
{
  abstract double func(double x[],int equation_ref);
}

class fa extends f_xir
{
            double func(double x[],int i)
            {//a general surface curve fitting model
    double xx=0.0;
    //density function
    if(i==0)       {xx=1.0;          }
    else if (i==1) {xx=x[0];         }
    else if (i==2) {xx=x[1];         }
    else if (i==3) {xx=x[0]*x[1];    }
    else if (i==4) {xx=x[1]*x[1];    }
    else if (i==5) {xx=x[0]*x[1];    }
    else if (i==6) {xx=x[0]*x[0]*x[0];}
    else if (i==7) {xx=x[1]*x[1]*x[1];}
    else if (i==8) {xx=x[0]*x[0]*x[1];}
    else if (i==9) {xx=x[0]*x[1]*x[1];}
    else if (i==10){xx=x[0]*x[0]*x[0]*x[0];}
    else if (i==11){xx=x[1]*x[1]*x[1]*x[1];}
    else if (i==12){xx=x[0]*x[0]*x[1]*x[1];}
    else if (i==13){xx=x[0]*x[0]*x[0]*x[1];}
    else if (i==14){xx=x[0]*x[1]*x[1]*x[1];}
    return xx;
            }
}
class NA46
{
//  General least square curve fitting with multivariable
            public static double[][] Transpose(double [][] left)
    { //transpose matrix (if A=a(i,j) Transpose(A)=a(j,i)
    int i,j;
    int n=left.length;
    int m=left[0].length;
    double b[][];
    b=new double[m][n];
    for(i=0;i<n;i++)
      {for(j=0;j<m;j++)  {b[j][i]=left[i][j];} }
    return b;
    }
  public static double[] gausswithpartialpivot(double a[][],double b[])
  { //Gauss elimination with partial pivoting
  int n=b.length;
  double x[]=new double[n];
  double carpan=0;
  double toplam=0;
  double buyuk;
  double dummy=0;
  //gauss elimination
  int i,j,k,p,ii,jj;
```

```
  for(k=0;k<(n-1);k++)
  { //partial pivoting
              p=k;
              buyuk=Math.abs(a[k][k]);
              for(ii=k+1;ii<n;ii++)
              {  dummy=Math.abs(a[ii][k]);
                if(dummy > buyuk) {buyuk=dummy;p=ii;}
              }
              if(p!=k)
              { for(jj=k;jj<n;jj++)
                { dummy=a[p][jj];
                  a[p][jj]=a[k][jj];
                  a[k][jj]=dummy;
                }
                dummy=b[p];
                b[p]=b[k];
                b[k]=dummy;
              }
              //
    for(i=k+1;i<n;i++)
    {  carpan=a[i][k]/a[k][k];
       a[i][k]=0;
       for(j=k+1;j<n;j++)
       {   a[i][j]-=carpan*a[k][j]; }
          b[i]   =b[i]   -carpan*b[k];
              }
  }
  //backward substitution
  x[n-1]=b[n-1]/a[n-1][n-1];
  for(i=n-2;i>=0;i--)
  {
    toplam=0;
    for(j=i+1;j<n;j++)
    {  toplam+=a[i][j]*x[j];}
    x[i]=(b[i]-toplam)/a[i][i];
  }
  return x;
}

public static double[] GeneralLeastSquare(double c[][],int n)
{
int n1=c.length;
  int n2=c[0].length-1;
  System.out.println("n1="+n1+"n2="+n2);
  double xi[][]=new double[n1][n2];
  double yi[]=new double[n1];
  for(int i=0;i<n1;i++){for(int j=0;j<n2;j++){xi[i][j]=c[i][j];};yi[i]=c[i][n2];}
  return GeneralLeastSquare(xi,yi,n);
}

public static double[] GeneralLeastSquare(double xi[][],double yi[],int n)
{ // n dimensional surface general least square curve fitting
fa f=new fa();
int l=xi.length;
int i,j,k;
int np1=n+1;
double A[][];
```

```
A=new double[np1][np1];
double B[];
B=new double[np1];
double X[];
X=new double[np1];
for(i=0;i<n+1;i++)
  {  B[i]=0;
            for(j=0;j<np1;j++)
    {
            A[i][j]=0.0;
    for(k=0;k<l;k++) A[i][j]+=f.func(xi[k],i)*f.func(xi[k],j);
    }
    for(k=0;k<l;k++) B[i]+= f.func(xi[k],i)*yi[k];
  }
X=gausswithpartialpivot(A,B);
//X=B/A;
double max=0;
for(i=0;i<n+1;i++)
 if(Math.abs(X[i]) > max) max = Math.abs(X[i]);
for(i=0;i<n+1;i++)
 if((Math.abs(X[i]/max) > 0) && (Math.abs(X[i]/max) < 1.0e-100)) X[i]=0;
Text.printT(X);
return X;
}

public static double funcGeneralLeastSquare(double e[],double x[])
{
// multidimensional function calculation
fa f=new fa();
int n=e.length;
double ff=0;
if(n!=0.0)
  {
  for(int i=n-1;i>=0;i--)
   {ff+=e[i]*f.func(x,i);}
  }
return ff;
}

public static double[] funcGeneralLeastSquare(double e[],double xi[][])
{
// multidimensional function calculation
fa f=new fa();
int n=e.length;
double ff[]=new double[xi.length];
for(int k=0;k<xi.length;k++)
{
if(n!=0.0)
  {
  for(int i=n-1;i>=0;i--)
   {ff[k]+=e[i]*f.func(xi[k],i);}
  }
}
return ff;
}

public static double[][] cikti(double c[][],int polinomkatsayisi,int aradegersayisi)
```

```
{ int n1=c.length;
  int n2=c[0].length-1;
  System.out.println("n1="+n1+"n2="+n2);
  double xi[][]=new double[n1][n2];
  double yi[]=new double[n1];
  for(int i=0;i<n1;i++){for(int j=0;j<n2;j++){xi[i][j]=c[i][j];};yi[i]=c[i][n2];
  }
  return cikti(xi,yi,polinomkatsayisi,aradegersayisi);
}


public static double[][] cikti(double xi[][],double yi[],int polinomkatsayisi,int aradegersayisi)
{
              int n=xi.length;
              int nk=xi[0].length;
              int nn=(n-1)*(aradegersayisi+1)+1;
    double E[]=GeneralLeastSquare(xi,yi,polinomkatsayisi);
    double x[][]=new double[nn][nk];
    double c[][]=new double[nn][nk+1];
    double yy[]=new double[nn];
    double dx[]=new double[nk];
    int k=0;
    int i,j,w;
    for(i=0;i<(n-1);i++)
      {   for(w=0;w<nk;w++){x[k][w]=xi[i][w];dx[w]=(xi[i+1][w]-xi[i][w])/((double)aradegersayisi+1.0);};k++;
         for(j=0;j<aradegersayisi;j++)
         { for(w=0;w<nk;w++){x[k][w]=x[k-1][w]+dx[w];};k++;}
      }
    for(w=0;w<nk;w++){x[k][w]=xi[i][w];}
    yy=funcGeneralLeastSquare(E,x);
    for(i=0;i<x.length;i++)
     {for(w=0;w<nk;w++) c[i][w]=x[i][w];
      c[i][nk]=yy[i];
     }
    return c;
}

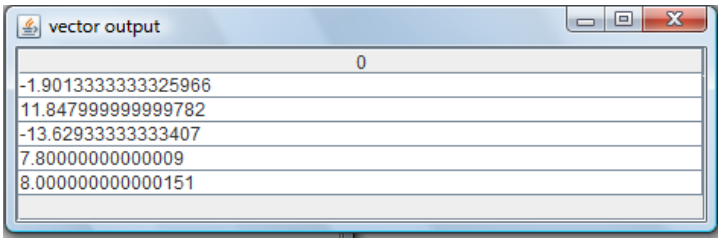
public static double funcGeneralLeastSquare(double e[],double x,double y)
{
// multidimensionaş surface fitting
double xx[]=new double[2];
xx[0]=x;
xx[1]=y;
return funcGeneralLeastSquare(e,xx);
}

public static double hata(double c[][],double e[])
{int n1=c.length;
  int n2=c[0].length-1;
  System.out.println("n1="+n1+"n2="+n2);
  double xi[][]=new double[n1][n2];
  double yi[]=new double[n1];
  for(int i=0;i<n1;i++){for(int j=0;j<n2;j++){xi[i][j]=c[i][j];};yi[i]=c[i][n2];
  }
return hata(xi,yi,e);
}
```

```
public static double hata(double x[][],double y[],double e[])
{
//calculates absolute square root error of a least square approach
double n=x.length;
 int k;
 double total=0;
 for(k=0;k<n;k++)
 {
 total+=(y[k]-funcGeneralLeastSquare(e,x[k]))*(y[k]-funcGeneralLeastSquare(e,x[k]));
 }
total=Math.sqrt(total);
return total;
}

  public static void main(String[] args)
  {
  String s1=JOptionPane.showInputDialog("file name : ");
  double c[][]=Text.readDouble(s1);
  double b[]=GeneralLeastSquare(c,4);
  double xi[][]=cikti(c,4,5);
  System.out.println("output \n"+Matrix.toString(xi));
  System.out.println("error \n"+Matrix.toString(hata(c,b)));

  }
}
```




```
---------- Capture Output ----------
> "C:\turhan\java\bin\javaw.exe" NA46
n1=90n2=2
n1=90n2=2
output
      0.000000000000000      0.000000000000000     -1.901333333332597
      0.000000000000000      0.066666666666667     -2.774399999999312
      0.000000000000000      0.133333333333333     -3.576355555554915
      0.000000000000000      0.200000000000000     -4.307199999999405
      0.000000000000000      0.266666666666667     -4.966933333332783
      0.000000000000000      0.333333333333333     -5.555555555555047
      0.000000000000000      0.400000000000000     -6.073066666666200
      0.000000000000000      0.466666666666667     -6.519466666666240
      0.000000000000000      0.533333333333333     -6.894755555555169
      0.000000000000000      0.600000000000000     -7.198933333332985
      0.000000000000000      0.666666666666667     -7.431999999999686
      0.000000000000000      0.733333333333333     -7.593955555555279
      0.000000000000000      0.800000000000000     -7.684799999999755
      0.000000000000000      0.866666666666667     -7.704533333333122
      0.000000000000000      0.933333333333333     -7.653155555555374
      0.000000000000000      1.000000000000000     -7.530666666666516
      0.000000000000000      1.066666666666667     -7.337066666666544
```

```
0.000000000000000   1.133333333333333   -7.072355555555461
0.000000000000000   1.200000000000000   -6.736533333333265
0.000000000000000   1.266666666666667   -6.329599999999955
0.000000000000000   1.333333333333333   -5.851555555555532
0.000000000000000   1.400000000000000   -5.302400000000000
0.000000000000000   1.466666666666667   -4.682133333333356
0.000000000000000   1.533333333333333   -3.990755555555593
0.000000000000000   1.600000000000000   -3.228266666666718
3.200000000000000   3.133333333333334   150.057244444444650
3.200000000000000   3.200000000000000   154.190400000000200
3.200000000000000   3.266666666666667   158.394666666666840
3.200000000000000   3.333333333333333   162.670044444444670
3.200000000000000   3.400000000000000   167.016533333333570
3.200000000000000   3.466666666666666   171.434133333333620
3.200000000000000   3.533333333333332   175.922844444444730
3.200000000000000   3.600000000000000   180.482666666667100

n1=90n2=2
error
    419.074496785476100
```

6.4 INTERPOLATION

We sometimes know the value of a function $f(x)$ at a set of points $x_1, x_2, \ldots, x_N$ (say, with $x_1 < \ldots. < x_N$), but we don't have an analytic expression for $f(x)$ that lets us calculate its value at an arbitrary point. For example, the $f(x_i)$'s might result from some physical measurement or from long numerical calculation that cannot be cast into a simple functional form. Often the x_i's are equally spaced, but not necessarily. The task now is to estimate $f(x)$ for arbitrary x by, in some sense, drawing a smooth curve through (and perhaps beyond) the x_i. If the desired x is in between the largest and smallest of the x_i's, the problem is called *interpolation*; if x is outside that range, it is called *extrapolation*,which is considerably more hazardous. Basic difference of interpolation compare to least square curve fitting is that interpolation is always passing through the data points. Some of the most common interpolation methods are given here:

6.4.1 Newton Interpolation Polynomial

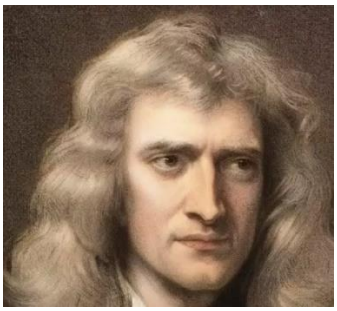


Figure 6.4.1 Sir Isaac Newton

Newton interpolation Formula of degree n+1 is given as:

$$f_n(x) = b_0 + b_1(x - x_0) + \cdots + b_n(x - x_0)(x - x_1)..(x - x_n))$$

$b_0 = f(x_0)$
$b_1 = f[x_1, x_0]$
$b_2 = f[x_2, x_1, x_0]$
…
$b_n = f[x_n, x_{n-1}, .., x_2, x_1, x_0]$ (6.4-1)
In this definition :

$f[x_i, x_j] = \frac{f(x_i)-f(x_j)}{x_i-x_i}$ (6.4.2a)

$f[x_i, x_j, x_k] = \frac{f[x_i,x_j]-f[x_j,x_k]}{x_i-x_k}$ (6.4.2b)

Similarly nth degree divided difference Formula can be given as:

$f[x_n, x_{n-1}, \dots, \dots, x_1, x_0] = \frac{f[x_n,x_{n-1},..,x_1]-f[x_{n-1},x_{n-2},...,..,x_1,x_0]}{x_n-x_0}$ (6.4.2c)

This equation called as Newton's interpolation polynomial.
An example problem :
Population of City A is given in the table below. Calculate population of the city for year 1945

year	1940	1950	1960	1970	1980	1990
Population	132165	151336	179323	203302	226542	249633

Newton's divided difference table :

Xi	yi	f[xi,xj]	f[xi,xj,xk]	f[xi,xj,xk,xl]	f[xi,xj,xk,xl,xn]	f[xi,xj,xk,xl,xn,xm]
1940	132165	1917.1	44.08	-2.137333333	0.067054167	-0.001564333
1950	151336	2798.7	-20.04	0.544833333	-0.0111625	
1960	179323	2397.9	-3.695	0.098333333		
1970	203302	2324	-0.745			
1980	226542	2309.1				
1990	249633					

Solution with increasing order can be given as:
132165+(1945-1940)*1917.1= 141750.5
141750.5+(1945-1940)* (1945-1950)* 44.08=140648.5
140648.5+(1945-1940)(1945-1950)*(1945-1960)* -2.137333333=139847
139847+(1945-1940)(1945-1950)*(1945-1960)*(1945-1970)* 0.067054167 =139218.4
139218.4+(1945-1940)(1945-1950)*(1945-1960)*(1945-1970)*(1945-1980)* -0.001564333=138705.1
The most important advantage of Newton intrepolation formula is the additive characters. In order to calculate (n+1) th degree of polynomial all you need is nth degree polynomial difference values. Example code is giving the result of the same problem

Problem 6.4-1 Newton Interpolation formula

```
import java.io.*;
class NA47
{
  public static double[][]  Newton(double xi[],double yi[])
 {
            int n=xi.length;
            double F[][]=new double[n+1][n];
            for(int i=0;i<n;i++)
              F[i][0]=yi[i];
   for(int i=1;i<n;i++)
   { for(int j=1;j<=i;j++)
     {F[i][j]=(F[i][j-1]-F[i-1][j-1])/(xi[i]-xi[i-j]);}
   }
```

```
    for(int i=0;i<n;i++)
    {F[n][i]=xi[i];}
    //System.out.println(Matrix.toString(F));
    return F;
  }

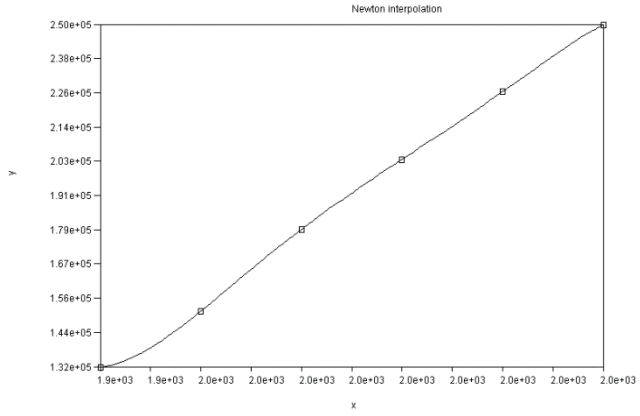
  public static double  funcNewton(double F[][],double x)
  {
              int n=F[0].length;
    double carpim=1.0;
    double toplam=0;
    for(int i=0;i<n;i++)
      {   carpim=F[i][i];
                  for(int j=0;j<=(i-1);j++)
        {carpim*=(x-F[n][j]);}
        toplam+=carpim;
      }
    return toplam;
  }

 public static double[][]  funcNewton(double xi[],double yi[],int numberofmidpoints)
 {
             //numberofmidpoints :  in x--o--o--x--o--o--x chain if x's are esxperimental points
             // numberofmidpoints is 2
             int n=xi.length;
             int nn=(n-1)*(numberofmidpoints+1)+1;
             double z[][]=new double[2][nn];
   double Q[][]=Newton(xi,yi);
   double dx=0;
   int k=0;
   int i;
   for(i=0;i<(n-1);i++)
      {  z[0][k]=xi[i];z[1][k]=funcNewton(Q,z[0][k]);k++;
        for(int j=0;j<numberofmidpoints;j++)
          {dx=(xi[i+1]-xi[i])/((double)numberofmidpoints+1.0);
        z[0][k]=z[0][k-1]+dx;z[1][k]=funcNewton(Q,z[0][k]);k++;}
      }
      z[0][k]=xi[i];z[1][k]=funcNewton(Q,z[0][k]);
   return z;
 }

  public static void main(String args[]) throws IOException
  {
  double x[]={1940,1950,1960,1970,1980,1990};
  double y[]={132165,151336,179323,203302,226542,249633};
  double z1[][]=funcNewton(x,y,9);
  System.out.println("Newton interpolation\n"+Matrix.toStringT(z1));
  Text.print(Text.T(z1),"Newton interpolation");
  Plot pp=new Plot(x,y);
  pp.setPlabel("Newton interpolation");
  pp.setPlotType(0,20);
  pp.addData(z1[0],z1[1]);
  pp.plot();
  }
}
```

Newton interpolation

0	1
1940.0	132165.0
1941.0	132684.8307565
1942.0	133658.281408
1943.0	135023.3483545
1944.0	136723.01625599997
1945.0	138705.0703125
1946.0	140921.908544
1947.0	143330.3540705
1948.0	145891.467392
1949.0	148570.3586685
1950.0	151336.0
1951.0	154161.0377065
1952.0	157021.60460800002
1953.0	159897.13230449997
1954.0	162770.16345599998
1955.0	165626.1640625
1956.0	168453.33574399998
1957.0	171242.42802049997
1958.0	173986.55059199999
1959.0	176680.98561849998
1960.0	179323.0
1961.0	181911.65765650003
1962.0	184447.631808
1963.0	186933.01725449998
1964.0	189371.142656
1965.0	191766.3828125
1966.0	194123.97094400003
1967.0	196449.8109705
1968.0	198750.289792
1969.0	201032.0895685
1970.0	203302.0
1971.0	205566.7306065
1972.0	207832.72300800003




```
---------- Capture Output ----------
> "C:\turhan\java\bin\javaw.exe" NA47
Newton interpolation
1940.0000000000132165.0000000000
1941.0000000000132684.8307565000
1942.0000000000133658.2814080000
1943.0000000000135023.3483545000
1944.0000000000136723.0162560000
1945.0000000000138705.0703125000
1946.0000000000140921.9085440000
1947.0000000000143330.3540705000
1948.0000000000145891.4673920000
```

```
1949.0000000000148570.3586685000
1950.0000000000151336.0000000000
1951.0000000000154161.0377065000
1952.0000000000157021.6046080000
1953.0000000000159897.1323045000
1954.0000000000162770.1634560000
1955.0000000000165626.1640625000
1956.0000000000168453.3357440000
1957.0000000000171242.4280205000
1958.0000000000173986.5505920000
1959.0000000000176680.9856185000
1960.0000000000179323.0000000000
1961.0000000000181911.6576565000
1962.0000000000184447.6318080000
1963.0000000000186933.0172545000
1964.0000000000189371.1426560000
1965.0000000000191766.3828125000
1966.0000000000194123.9709440000
1967.0000000000196449.8109705000
1968.0000000000198750.2897920000
1969.0000000000201032.0895685000
1970.0000000000203302.0000000000
1971.0000000000205566.7306065000
1972.0000000000207832.7230080000
1973.0000000000210105.9632045000
1974.0000000000212391.7938560000
1975.0000000000214694.7265625000
1976.0000000000217018.2541440000
1977.0000000000219364.6629205000
1978.0000000000221734.8449920000
1979.0000000000224128.1105185000
1980.0000000000226542.0000000000
1981.0000000000228972.0965565000
1982.0000000000231411.8382080001
1983.0000000000233852.3301545001
1984.0000000000236282.1570560000
1985.0000000000238687.1953125000
1986.0000000000241050.4253440000
1987.0000000000243351.7438705000
1988.0000000000245567.7761920000
1989.0000000000247671.6884685000
1990.0000000000249633.0000000000
> Terminated with exit code 1.
```

Lagrange Interpolation Polynomial



Joseph-Loise Lagrange

Lagrange interpolation polynomial is the reformulated version of the Newton's interpolation polynomial. Lagrange Formula do not require calculation of divided differences.

$f_n(x) = \sum_{i=0}^{n} L_i(x) f(x_i)$ (6.4-3) Where

$L_i(x) = \prod_{\substack{j=0 \\ j \neq i}}^{n} \frac{x - x_i}{x_i - x_j}$. (6.4-4) Where $\prod$ indicates multiplication process.

Open form of the equation;

for the first degree (linear) polinomial (n=1)

$$f_1(x) = \frac{x - x_1}{x_0 - x_1} f(x_0) + \frac{x - x_0}{x_1 - x_0} f(x_1) \quad (6.4\text{-}5)$$

for the first degree (quadratic) polinomial (n=2)

$$f_2(x) = \frac{(x - x_1)(x - x_2)}{(x_0 - x_1)(x_0 - x_2)} f(x_0) + \frac{(x - x_0)(x - x_2)}{(x_1 - x_0)(x_1 - x_2)} f(x_1) + \frac{(x - x_0)(x - x_1)}{(x_2 - x_0)(x_2 - x_2)} f(x_2) \quad (6.4\text{-}6)$$

Program 6.4-2 Lagrange Interpolation formula

```
 //Lagrange interpolation polynomial

import java.io.*;
import javax.swing.*;
import java.lang.*;
import java.util.Locale;
class NA48
{
 public static double[][]  Lagrange(double xi[],double yi[])
 {
            int n=xi.length;
   double L[][]=new double[2][n];
   for(int i=0;i<n;i++)
   {L[0][i]=yi[i];
   for(int j=0;j<n;j++)
   {if(i!=j) L[0][i]/=(xi[i]-xi[j]);}
   }
   for(int i=0;i<n;i++)
   {L[1][i]=xi[i];}
   return L;
 }


 public static double  funcLagrange(double L[][],double x)
 {
            int n=L[0].length;
   double carpim=1.0;
   double toplam=0;
   for(int i=0;i<n;i++)
   {carpim=L[0][i];
   for(int j=0;j<n;j++)
   {if(i!=j) carpim*=(x-L[1][j]);}
   toplam+=carpim;
   }
   return toplam;
 }

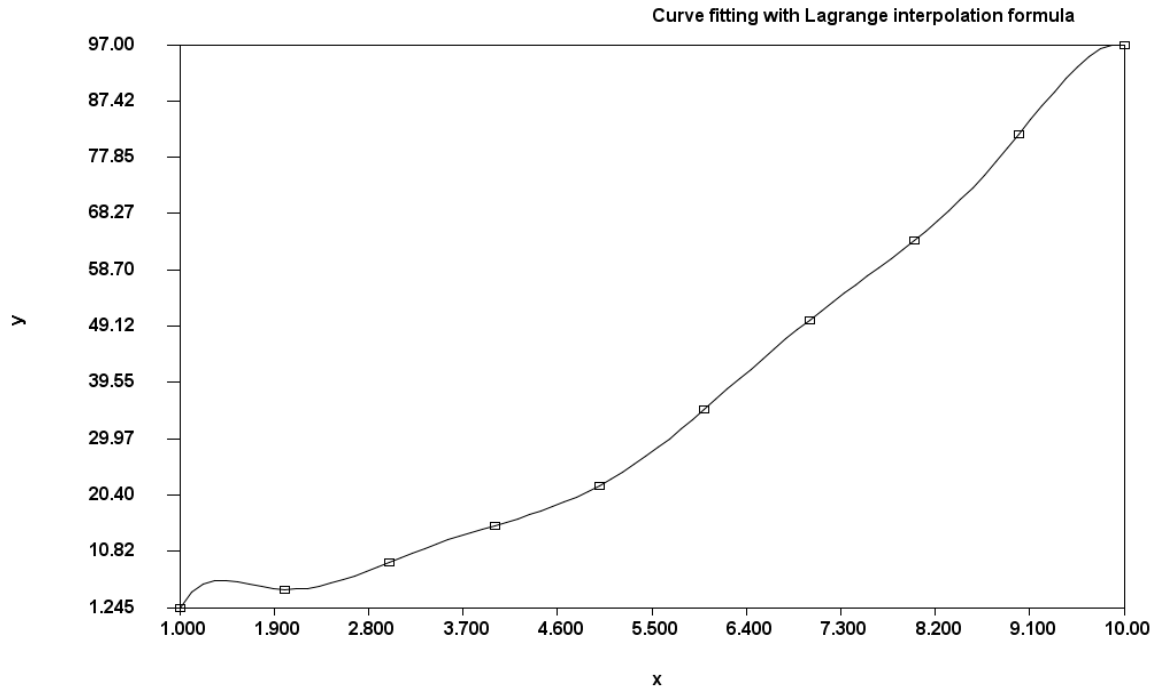

public static double[][]  funcLagrange(double xi[],double yi[],int numberofmidpoints)
{
```

```
            //numberofmidpoints :  in x--o--o--x--o--o--x chain if x's are esxperimental points
            // numberofmidpoints is 2
            int n=xi.length;
            int nn=(n-1)*(numberofmidpoints+1)+1;
            double z[][]=new double[2][nn];
   double Q[][]=Lagrange(xi,yi);
   double dx=0;
   int k=0;
   int i;
   for(i=0;i<(n-1);i++)
     {  z[0][k]=xi[i];z[1][k]=funcLagrange(Q,z[0][k]);k++;
       for(int j=0;j<numberofmidpoints;j++)
         {dx=(xi[i+1]-xi[i])/((double)numberofmidpoints+1.0);
       z[0][k]=z[0][k-1]+dx;z[1][k]=funcLagrange(Q,z[0][k]);k++;}
     }
     z[0][k]=xi[i];z[1][k]=funcLagrange(Q,z[0][k]);
   return z;
 }
  public static String toString(double z[][])
  { String s="";
   for(int i=0;i<z.length;i++)
   {  for(int j=0;j<z[0].length;j++)
              {s+=String.format("%f ",z[i][j]);}
              s+="\n";
   }
   return s;
  }
  public static String toStringT(double z[][])
  { String s="";
   for(int i=0;i<z[0].length;i++)
   {  for(int j=0;j<z.length;j++)
              {s+=String.format(Locale.US,"%f ",z[j][i]);}
              s+="\n";
   }
   return s;
  }
  public static void main(String args[]) throws IOException
  {
double x[];
  double y[];
  String s1=JOptionPane.showInputDialog("Data file name (input1.txt)");
  //JFileChooser fc=new JFileChooser();
  // if (fc.showOpenDialog(null) == JFileChooser.APPROVE_OPTION) {File file = fc.getSelectedFile();s1=file.getName();
}
  double a[][]=Text.readDoubleT(s1);
  x=a[0];
  y=a[1];
  double z1[][]=funcLagrange(x,y,8);
  System.out.println("Lagrange interpolation\n"+toStringT(z1));
  Text.print(Text.T(z1),"Curve fitting with Lagrange interpolation formula");
  Plot pp=new Plot(x,y);
  pp.setPlabel("Curve fitting with Lagrange interpolation formula");
  pp.setPlotType(0,20);
  pp.addData(z1[0],z1[1]);
  pp.plot();
  }
}
```

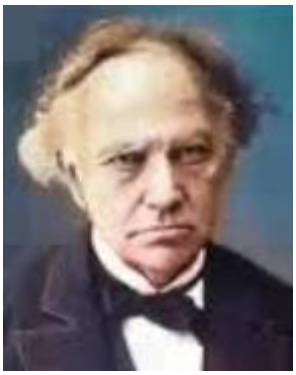
```
 ---------- Capture Output ----------
> "E:\co\java\bin\java.exe" NA48
Lagrange interpolation
1.000000 1.245200
1.111111 3.844709
1.222222 5.220168
1.333333 5.766842
1.444444 5.792022
1.555556 5.529201
1.666667 5.150603
1.777778 4.778211
1.888889 4.493406
2.000000 4.345340
2.111111 4.358154
2.222222 4.537148
2.333333 4.873996
2.444444 5.351107
2.555556 5.945224
2.666667 6.630327
2.777778 7.379936
2.888889 8.168880
3.000000 8.974600
3.111111 9.778041
3.222222 10.564209
3.333333 11.322430
3.444444 12.046372
3.555556 12.733864
3.666667 13.386567
3.777778 14.009523
3.888889 14.610618
4.000000 15.200000
4.111111 15.789457
4.222222 16.391798
4.333333 17.020253
4.444444 17.687896
4.555556 18.407124
4.666667 19.189190
4.777778 20.043809
4.888889 20.978831
5.000000 22.000000
5.111111 23.110791
5.222222 24.312326
5.333333 25.603376
5.444444 26.980433
5.555556 28.437861
5.666667 29.968105
5.777778 31.561966
5.888889 33.208924
6.000000 34.897500
6.111111 36.615652
6.222222 38.351190
6.333333 40.092204
6.444444 41.827482
6.555556 43.546928
6.666667 45.241938
6.777778 46.905758
```

```
6.888889 48.533778
7.000000 50.123780
7.111111 51.676109
7.222222 53.193771
7.333333 54.682444
7.444444 56.150395
7.555556 57.608292
7.666667 59.068921
7.777778 60.546780
7.888889 62.057579
8.000000 63.617610
8.111111 65.243027
8.222222 66.949006
8.333333 68.748812
8.444444 70.652774
8.555556 72.667180
8.666667 74.793108
8.777778 77.025198
8.888889 79.350409
9.000000 81.746750
9.111111 84.182044
9.222222 86.612727
9.333333 88.982735
9.444444 91.222502
9.555556 93.248116
9.666667 94.960673
9.777778 96.245886
9.888889 96.973988
10.000000 97.000000

> Terminated with exit code 0.
```



Hermit Interpolation Polynomial



Charles Hermite(1822-1901)

Hermit interpolation requires and uses derivative data in addition to the function data.

General formula

$H(x) = Q_{0,0}+ Q_{1,1}(x-x_0)+ Q_{2,2}(x-x_0)^2+ Q_{3,3}(x-x_0)^2(x-x_1)+ Q_{4,4}(x-x_0)^2(x-x_1)^2+\ldots+ Q_{2n+1,2n+1}(x-x_0)^2(x-x_1)^2\ldots(x-x_{n-1})^2(x-x_n)$

Calculation of the coefficients is explain at Algorithm 6.4-1 and Table 6.4-1.

Algorithm 6.4-1 Hermit interpolation

INPUT $x_0,x_1,\ldots,x_n$ değerleri; $f(x_0)$, $f(x_1)$, …,$f(x_n)$ değerleri; $f'(x_0)$, $f'(x_1)$, …,$f'(x_n)$ değerleri

OUTPUT : $Q_{0,0},Q_{1,1},\ldots..$, $Q_{2n+1,2n+1}$ değerleri

$H(x)= Q_{0,0}+ Q_{1,1}(x-x_0)+ Q_{2,2}(x-x_0)^2+ Q_{3,3}(x-x_0)^2(x-x_1)+ Q_{4,4}(x-x_0)^2(x-x_1)^2+\ldots+ Q_{2n+1,2n+1}(x-x_0)^2(x-x_1)^2\ldots(x-x_{n-1})^2(x-x_n)$

Step 1: for i=0,1,…,n

{

Step 2: $z_{2i}=x_i$;

$z_{2i+1}=x_i$;

$Q_{2i,0}=f(x_i)$;

$Q_{2i+1,0}=f(x_i)$;

$Q_{2i+1,0}=f'(x_i)$;

Step 3: if i≠0 then $Q_{2i,j} = \frac{Q_{2i,0}-Q_{2i-1,0}}{z_{2i}-z_{2i-1}}$

} End of Step 1 in sonu

Step 4: for i=2,3,…,2n+1

{ for j=2,3,…,i $Q_{i,j} = \frac{Q_{i,j-1}-Q_{i-1,j-1}}{z_i-z_{2i-j}}$ }

Table 6.4-1 Hermit interpolation difference formulation

Z	f(z)	First difference equation	Second difference equation
$\mathbf{Z_0 = x_0}$	$f[z_0]=f(x_0)$	$f[z_0,z_1] = f(x_0)$	$f[z_0,z_1,z_2] = \frac{f[z_1,z_2]-f[z_0,z_1]}{z_2-z_0}$
$\mathbf{Z_1 = x_0}$	$f[z_1]=f(x_0)$	$f[z_1,z_2] = \frac{f[z_2]-f[z_1]}{z_2-z_1}$	$f[z_1,z_2,z_3] = \frac{f[z_2,z_3]-f[z_1,z_2]}{z_2-z_1}$
$\mathbf{Z_2 = x_1}$	$f[z_2]=f(x_1)$	$f[z2,z3] = f(x1)$	$f[z_2,z_3,z_4] = \frac{f[z_3,z_4]-f[z_2,z_3]}{z_4-z_2}$
$\mathbf{Z_3 = x_1}$	$f[z_3]=f(x_1)$	$f[z_3,z_4] = \frac{f[z_4]-f[z_3]}{z_4-z_3}$	$f[z_3,z_4,z_5] = \frac{f[z_4,z_5]-f[z_3,z_4]}{z_5-z_4}$
$\mathbf{Z_4 = x_2}$	$f[z_4]=f(x_2)$	$f[z4,z5] = f(x2)$	
$\mathbf{Z_5 = x_2}$	$f[z_5]=f(x_2)$		

An example problem :

k	x_k	$f(x_k)$	$f'(x_k)$
0	1.3	0.6200860	-0.5220232

1	1.6	0.4554022	-05698959
2	1.9	0.2818186	-0.5811571

Difference table:

1.3	0.620086					
		-0.5220232				
1.3	0.620086		-0.08974267			
		-0.548946		0.06636556		
1.6	0.4554022		-0.069833		0.00266667	
		-0.5698959		0.06796556		-0.00277469
1.6	0.4554022		-0.02905367		0.00100185	
		-0.578612		0.06856667		
1.9	0.2818186		-0.00848367			
		-0.5811571				
1.9	0.2818186					

Interpolation value at x=1.5:

$H_5(1.5) = 0.620086+(1.5-1.3)*(-0.5220232)+ (1.5-1.3)^2*(1.5-1.6)*(0.06636556+ (1.5-1.3)^2* (1.5-1.6)^2*(0.00266667)+ 1.5-1.3)^2* (1.5-1.6)^2*(1.5-1.9) (-0.00277469)=0.5118277$

A program carrying out Hermit interpolation is given

Program 6.4-3 Hermite polynomial interpolation formula

```
//Hermit Interpolation
import java.io.*;

class NA123
{
 public static double [] Hermite(double xi[],double fi[],double dfi[])
 {
 //m degree of polynomial
 int i,j;
 int m=xi.length;
 int n=m-1;
 double Q[][]=new double[2*n+2][2*n+2];
 double QQ[]=new double[2*n+2];
 double z[]=new double[2*n+2];
 for(i=0;i<=n;i++)
  {
          z[2*i]=xi[i];
          z[2*i+1]=xi[i];
          Q[2*i][0]=fi[i];
          Q[2*i+1][0]=fi[i];
          Q[2*i+1][1]=dfi[i];
          if(i!=0) { Q[2*i][1]=(Q[2*i][0]-Q[2*i-1][0])/(z[2*i]-z[2*i-1]);}
          }
          for(i=2;i<=(2*n+1);i++)
            {for(j=2;j<=i;j++){Q[i][j]=(Q[i][j-1]-Q[i-1][j-1])/(z[i]-z[i-j]);}
            }
          for(i=0;i<=(2*n+1);i++)
           {QQ[i]=Q[i][i];}
          //System.out.println(QQ.length+"\n"+Matrix.toStringT(QQ));
          return QQ;
 }

 public static double funcHermite(double QQ[],double xi[],double x)
```

```
 {
            //Hermit polynomial
 int i,j;
 int m=QQ.length-1;
 double xx=1;
 double toplam=QQ[0];
 for(i=1;i<m;i+=2)
  {   j=(i-1)/2;
              xx*=(x-xi[j]);
              toplam+=QQ[i]*xx;
              xx*=(x-xi[j]);
              toplam+=QQ[i+1]*xx;
  }
 return toplam;
 }

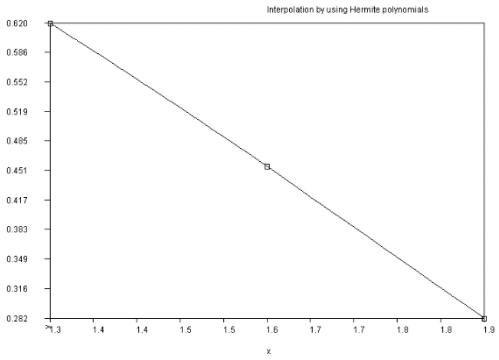
public static double[][]  funcHermite(double xi[],double yi[],double dyi[],int aradegersayisi)
{
           //number of midpoints:  x--o--o--x--o--o--x  x data points midpoints are 2
           int n=xi.length;
           int nn=(n-1)*(aradegersayisi+1)+1;
           double z[][]=new double[2][nn];
  double Q[]=Hermite(xi,yi,dyi);
  double dx=0;
  int k=0;
  int i;
  for(i=0;i<(n-1);i++)
    {  z[0][k]=xi[i];z[1][k]=funcHermite(Q,xi,z[0][k]);k++;
      for(int j=0;j<aradegersayisi;j++)
        {dx=(xi[i+1]-xi[i])/((double)aradegersayisi+1.0);
         z[0][k]=z[0][k-1]+dx;z[1][k]=funcHermite(Q,xi,z[0][k]);k++;
        }
    }
    z[0][k]=xi[i];z[1][k]=funcHermite(Q,xi,z[0][k]);
  return z;
}

 public static void main(String args[]) throws IOException
 {
 double x[]={1.3,1.6,1.9};
 double y[]={0.620086,0.4554022,0.281818186};
 double dy[]={-0.5220232,-0.5698959,-0.5811571};
 double z1[][]=funcHermite(x,y,dy,8);
 Text.print(Text.T(z1),"Interpolation by using Hermite polynomials");
 System.out.println("Hermite polynomial interpolation\n"+Matrix.toStringT(z1));
 Plot pp=new Plot(x,y);
 pp.setPlabel("Interpolation by using Hermite polynomials");
 pp.setPlotType(0,20);
 pp.addData(z1[0],z1[1]);
 pp.plot();

 }
}
```

```
---------- Capture Output ----------
> "C:\turhan\java\bin\javaw.exe" NA123
Hermite polynomial interpolation
```

1.3000000000	0.6200860000
1.3333333333	0.6025660584
1.3666666667	0.5848174157
1.4000000000	0.5668545838
1.4333333333	0.5486921534
1.4666666667	0.5303447937
1.5000000000	0.5118272527
1.5333333333	0.4931543568
1.5666666667	0.4743410113
1.6000000000	0.4554022000
1.6333333333	0.4363529853
1.6666666667	0.4172085082
1.7000000000	0.3979839884
1.7333333333	0.3786947244
1.7666666667	0.3593560931
1.8000000000	0.3399835500
1.8333333333	0.3205926294
1.8666666667	0.3011989443
1.9000000000	0.2818181860



6.5 VANDERMONDE INTERPOLATION



6.5.1 Nicolaas van der Monde, Dutch

If the interpolating polynomial is written as

$$y = c_0 + c_1 x + c_2 x^2 + \cdots + c_N x^N \quad \textbf{(6.5-1)}$$

then the c_i's are required to satisfy the linear equation

$$\begin{bmatrix} 1 & x_0 & x_0^2 & \cdots & x_0^N \\ 1 & x_1 & x_1^2 & \cdots & x_1^N \\ \cdots & \cdots & \cdots & \cdots & \cdots \\ 1 & x_{N-1} & x_{N-1}^2 & \cdots & x_{N-1}^2 \\ 1 & x_N & x_N^2 & \cdots & x_N^2 \end{bmatrix} \begin{Bmatrix} c_0 \\ c_1 \\ \cdots \\ c_{N-1} \\ c_N \end{Bmatrix} = \begin{Bmatrix} y_0 \\ y_1 \\ \cdots \\ y_{N-1} \\ y_N \end{Bmatrix} \quad (6.5\text{-}2)$$

This is a *Vandermonde matrix*. The matrix can be solved by using some of the Matrix solving method. In The next program, it is solved by using pivoted Gauss eliminaton method. Remember that Vandermonde systems can be quite ill-conditioned. In such a case, *no* numerical method is going to give a very accurate answer. So the method should be used for relatively small number of data points.

Program 6.5-1 Vandermonde matrix interpolation with Gauss elimination solution

```
import java.io.*;
import java.util.*;
import javax.swing.*;

class Vandermonde
{
//Vandermonde interpolation
  public static double[] gausswithpartialpivot(double a[][],double b[])
  { //Gauss elimination with partial pivoting
  int n=b.length;
  double x[]=new double[n];
  double carpan=0;
  double toplam=0;
  double buyuk;
  double dummy=0;
  //gauss elimination
  int i,j,k,p,ii,jj;
  for(k=0;k<(n-1);k++)
  { //partial pivoting
              p=k;
              buyuk=Math.abs(a[k][k]);
              for(ii=k+1;ii<n;ii++)
              {  dummy=Math.abs(a[ii][k]);
                if(dummy > buyuk) {buyuk=dummy;p=ii;}
              }
              if(p!=k)
              { for(jj=k;jj<n;jj++)
                { dummy=a[p][jj];
                  a[p][jj]=a[k][jj];
                  a[k][jj]=dummy;
                }
                dummy=b[p];
                b[p]=b[k];
                b[k]=dummy;
              }
              //
    for(i=k+1;i<n;i++)
    {  carpan=a[i][k]/a[k][k];
       a[i][k]=0;
       for(j=k+1;j<n;j++)
       {   a[i][j]-=carpan*a[k][j]; }
          b[i]   =b[i]   -carpan*b[k];
              }
  }
  //backward substitution
  x[n-1]=b[n-1]/a[n-1][n-1];
  for(i=n-2;i>=0;i--)
  {
    toplam=0;
    for(j=i+1;j<n;j++)
    {  toplam+=a[i][j]*x[j];}
```

```
    x[i]=(b[i]-toplam)/a[i][i];
  }
  return x;
}


public static double[] Vandermonde_coef(double xi[],double yi[])
{
int n=xi.length;
fa f=new fa();
int l=xi.length;
int i,j,k;
double A[][];
A=new double[n][n];
double B[];
B=new double[n];
double X[];
X=new double[n];
double cc=1.0;
for(i=0;i<n;i++)
  {  B[i]=yi[i];
     cc=1.0;
              for(j=0;j<n;j++)
     {
              A[i][j]=cc;
              cc*=xi[i];
     }
  }
X=gausswithpartialpivot(A,B);
return X;
}

public static double func(double c[],double x)
{
// this function calculates the value of
// least square curve fitting function
int n=c.length;
double ff=0;
double xx=1.0;
if(n!=0.0)
  { xx=1.0;
  for(int i=0;i<n;i++)
   {ff+=c[i]*xx;
    xx*=x;
   }
  }
return ff;
}

public static double[][]  Vandermonde(double xi[],double yi[],int aradegersayisi)
 {
              //aradegersayisi:  x--o--o--x--o--o--x zincirinde x deneysel noktalar ise
              // ara değer sayısı 2 dir
              int n=xi.length;
              int nn=(n-1)*(aradegersayisi+1)+1;
              double z[][]=new double[2][nn];
    double E[]=Vandermonde_coef(xi,yi);
    System.out.println("coefficients :\n"+Matrix.toStringT(E));
    double dx=0;
    int k=0;
    int i;
    for(i=0;i<(n-1);i++)
       {z[0][k]=xi[i];z[1][k]=func(E,z[0][k]);k++;
       for(int j=0;j<aradegersayisi;j++)
       {dx=(xi[i+1]-xi[i])/((double)aradegersayisi+1.0);
       z[0][k]=z[0][k-1]+dx;z[1][k]=func(E,z[0][k]);k++;}
```

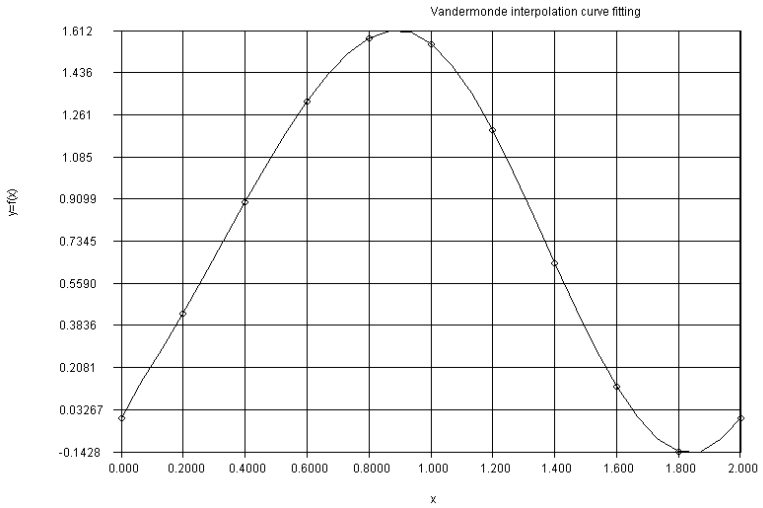
```
      }
      z[0][k]=xi[i];z[1][k]=func(E,z[0][k]);
    return z;
  }

  public static void main(String args[]) throws IOException
  {
  double x[];
  double y[];
  String s1=JOptionPane.showInputDialog("Data file name (input1.txt)");
  // JFileChooser fc=new JFileChooser();
  // if (fc.showOpenDialog(null) == JFileChooser.APPROVE_OPTION) {File file = fc.getSelectedFile();s1=file.getName();
}
  double a[][]=Text.readDoubleT(s1);
  x=a[0];
  y=a[1];
  double z[][]=Vandermonde(x,y,10);
  System.out.println("Vandermonde interpolation curve fitting\n"+Matrix.toStringT(z));
  Plot pp=new Plot(a[0],a[1]);
  pp.setPlabel("Vandermonde interpolation curve fitting");
  pp.setXlabel("x");
  pp.setYlabel("y=f(x)");
  pp.setPlotType(0,22);
  pp.addData(z[0],z[1]);
  pp.setPlotType(1,0);
  pp.setGrid(1,1);
  pp.plot();
  }
}
```

```
---------- Capture Output ----------
> "D:\java\bin\javaw.exe" Vandermonde
coefficients  :
        0.000000000000000000000000
        2.522238688245352000000000
       -6.499687449381024000000000
       43.219537011556255000000000
     -139.17792182712440000000000
      260.28892984853250000000000
     -302.86673648874705000000000
      218.09728893170060000000000
      -94.06798950652913000000000
       22.26860946298418400000000
       -2.22686094623513360000000

Vandermonde interpolation curve fitting
   0.0000000000   0.0000000000
   0.0666666667   0.1496357599
   0.1333333333   0.2886288414
   0.2000000000   0.4340028520
   0.2666666667   0.5869147066
   0.3333333333   0.7432097976
   0.4000000000   0.8981438220
   0.4666666667   1.0477593840
   0.5333333333   1.1886496189
   0.6000000000   1.3172323490
   0.6666666667   1.4291826777
   0.7333333333   1.5193154343
   0.8000000000   1.5819574910
   0.8666666667   1.6116896704
   0.9333333333   1.6042547278
   1.0000000000   1.5574077250
```

1.0666666667	1.4715139871
1.1333333333	1.3497637445
1.2000000000	1.1979574910
1.2666666667	1.0239080269
1.3333333333	0.8365900852
1.4000000000	0.6452323490
1.4666666667	0.4585755448
1.5333333333	0.2845001247
1.6000000000	0.1301438220
1.6666666667	0.0024690569
1.7333333333	-0.0910112194
1.8000000000	-0.1419971480
1.8666666667	-0.1427785660
1.9333333333	-0.0909568327
2.0000000000	-0.0000000000



The Vandermonde matrix can easily be inverted in terms of Lagrange basis polynomials: each *column* is the coefficients of the Lagrange basis polynomial, with terms in increasing order going down. The resulting solution to the interpolation problem is called the Lagrange polynomial. Remembering lagrange polynomial has the form

$$L_i(x) = \prod_{\substack{j=0 \\ j \neq i}}^{n} \frac{x - x_i}{x_i - x_j} = \sum_{k=0}^{N} A_{ik} x^k \quad (6.5-3)$$

Here the meaning of the last equality is to define the components of the matrix A_{ij} as the coefficients that arise when the product is multiplied out and like terms collected. The polynomial $P_j(x)$ is a function of x generally. But you will notice that it is specifically designed so that it takes on a value of zero at all x_i with $i \neq j$, and has a value of unity at $x = x_j$. In other words

$$L_i(x) = \delta_{ji} = \sum_{k=0}^{N} A_{ik} x^k \quad (6.5-4)$$

with

$$\delta_{ji} = \begin{Bmatrix} 0 \ if \ j \neq i \\ 1 \ if \ j = i \end{Bmatrix} \quad (6.5-6)$$

A_{ik} is exactly the inverse of the matrix of components x_i^k , which appears in Vandermonde Matrix, so

$$c_i = \sum_{k=0}^{N} A_{ki} x^k \quad (6.5-7)$$

A second version of the Vandermonde interpolation prosess is given as the next code, utilising Lagrange interpolation solution of the Matrix, instead of Gauss elimination

Program 6.5-1 Vandermonde matrix interpolation with Lagrange interpolation formula solution

```
import java.io.*;
import java.util.*;
import javax.swing.*;

class Vandermonde1
{
//Vandermonde interpolation
public static double[] Vandermonde_coef(double x[], double y[])
{ // Based on the Lagrange interpolation method to find coefficient of
  // Vandermonde interpolation MAtrix
int n=x.length-1;
double cof[]=new double[n+1];
int k,j,i;
double phi,ff,b;
double s[]=new double[n+1];
s[n] = -x[0];
for (i=1;i<=n;i++) {
for (j=n-i;j<=(n-1);j++)
s[j] -= x[i]*s[j+1];
s[n] -= x[i];
}
for (j=0;j<=n;j++) {
phi=n+1;
for (k=n;k>=1;k--)
phi=k*s[k]+x[j]*phi;
ff=y[j]/phi;
b=1.0;
for (k=n;k>=0;k--)
{
cof[k] += b*ff;
b=s[k]+x[j]*b;
}
}
return cof;
}

public static double func(double c[],double x)
{
// this function calculates the value of
// least square curve fitting function
int n=c.length;
double ff=0;
double xx=1.0;
if(n!=0.0)
  { xx=1.0;
  for(int i=0;i<n;i++)
   {ff+=c[i]*xx;
    xx*=x;
   }
  }
return ff;
}

public static double[][]  Vandermonde(double xi[],double yi[],int aradegersayisi)
 {
            //aradegersayisi:  x--o--o--x--o--o--x zincirinde x deneysel noktalar ise
            // ara değer sayısı 2 dir
            int n=xi.length;
            int nn=(n-1)*(aradegersayisi+1)+1;
            double z[][]=new double[2][nn];
```

```
    double E[]=Vandermonde_coef(xi,yi);
    System.out.println("katsayılar :\n"+Matrix.toStringT(E));
    double dx=0;
    int k=0;
    int i;
    for(i=0;i<(n-1);i++)
       {z[0][k]=xi[i];z[1][k]=func(E,z[0][k]);k++;
       for(int j=0;j<aradegersayisi;j++)
       {dx=(xi[i+1]-xi[i])/((double)aradegersayisi+1.0);
       z[0][k]=z[0][k-1]+dx;z[1][k]=func(E,z[0][k]);k++;}
       }
       z[0][k]=xi[i];z[1][k]=func(E,z[0][k]);
    return z;
  }

  public static void main(String args[]) throws IOException
  {
  double x[];
  double y[];
  String s1=JOptionPane.showInputDialog("Data file name (input1.txt)");
  // JFileChooser fc=new JFileChooser();
  // if (fc.showOpenDialog(null) == JFileChooser.APPROVE_OPTION) {File file = fc.getSelectedFile();s1=file.getName();
}
  double a[][]=Text.readDoubleT(s1);
  x=a[0];
  y=a[1];
  double z[][]=Vandermonde(x,y,2);
  System.out.println("Vandermonde interpolation curve fitting\n"+Matrix.toStringT(z));
  Plot pp=new Plot(a[0],a[1]);
  pp.setPlabel("Vandermonde interpolation curve fitting");
  pp.setXlabel("x");
  pp.setYlabel("y=f(x)");
  pp.setPlotType(0,22);
  pp.addData(z[0],z[1]);
  pp.setPlotType(1,0);
  pp.setGrid(1,1);
  pp.plot();
  }
}
```

```
---------- Capture Output ----------
> "D:\java\bin\javaw.exe" Vandermonde1
coefficients :
      -0.00000000008741448393914 12
       2.52223900766072440000000 00
      -6.49969183314451000000000 00
       43.21956104045553000000000 00
     -139.17799227629200000000000 00
      260.28905328308370000000000 00
     -302.86687190777195000000000 00
      218.09738271154805000000000 00
      -94.06802929631061000000000 00
       22.26861889956762800000000 00
       -2.22686190366278640000000 00

Vandermonde interpolation curve fitting
  0.0000000000  -0.0000000001
  0.0666666667   0.1496357675
  0.1333333333   0.2886288452
  0.2000000000   0.4340028519
  0.2666666667   0.5869147051
  0.3333333333   0.7432097966
  0.4000000000   0.8981438219
  0.4666666667   1.0477593843
  0.5333333333   1.1886496191
  0.6000000000   1.3172323488
  0.6666666667   1.4291826772
  0.7333333333   1.5193154337
```

```
0.8000000000  1.5819574906
0.8666666667  1.6116896704
0.9333333333  1.6042547280
1.0000000000  1.5574077250
1.0666666667  1.4715139868
1.1333333333  1.3497637436
1.2000000000  1.1979574897
1.2666666667  1.0239080255
1.3333333333  0.8365900841
1.4000000000  0.6452323486
1.4666666667  0.4585755447
1.5333333333  0.2845001244
1.6000000000  0.1301438204
1.6666666667  0.0024690536
1.7333333333 -0.0910112234
1.8000000000 -0.1419971501
1.8666666667 -0.1427785627
1.9333333333 -0.0909568244
2.0000000000 -0.0000000025
```

The same data is entered as the previous version. Note the small differences in the coefficient vector and interpolation values.

6.6 CUBIC SPLINE INTERPOLATION POLYNOMIALS

Another way for the interpolation process is to fit a different polynomial in between each two point. If a third degree polinomial is considered:

$$r_k(x) = a_k(x - x_k)^3 + b_k(x - x_k)^2 + c_k(x - x_k)^3 + y_k \quad 1 \le k \le n \quad (6.6-1)$$

In the interpolation proses polinoms should be passing through all data points

$$r_k(x_{k+1}) = y_{k+1} \quad 1 \le k \le n \quad (6.6-2)$$

In the same time the first derivative of the polynomial should also be continious while passing from one polynomial to the next one at the data point

$$r'_{k-1}(x_k) = r'_k(x_k) \quad 1 \le k \le n \quad (6.6-3)$$

For the third degree polinomial second derivative of the polynomial should also be continious while passing from one polynomial to the next one at the data point

$$r''_{k-1}(x_k) = r''_k(x_k) \quad 1 \le k \le n \quad (6.6-4)$$

All these conditions are not enough to solve the coefficients of the polinomials. Two more conditions are required. This two additional conditions (A and B of the following equation) can be given by user

$$r''_1(x_1) = A \qquad r''_{n-1}(x_n) = B \quad (6.6\text{-}5)$$

They are the second derivatives at the both hand of the series of polinomials. If A and B values are taken equals to 0, it is called a **natural cubic spline**. Other end conditions such as the ones depends one the first derivatives can also be set to solve the system of equations.

Defining $h_k = x_{k+1} - x_k \quad 1 \le k \le n \quad (6.6-5)$

System of equations become:

$$a_k h_k^3 + b_k h_k^2 + b_k h_k = y_{k+1} - y_k \quad 1 \le k \le n \quad (6.6-6)$$
$$3a_{k-1}h_{k-1}^2 + 2b_{k-1}h_{k-1} + c_{k-1} - c_k = 0$$
$$6a_{k-1}h_{k-1} + 2b_{k-1} + 2b_k = 0$$
$$3b_0 = 0$$
$$6a_{n-1}h_{n-1} + 2b_{n-1} = 0$$

This set contains 3n-3 equations. This could a considerable load to the system of equation solving programs. To make calculation load simpler a special third degree polinomial can be considered. If our cubic polinomial is in the form of:

$$h_k = x_{k+1} - x_k \quad 1 \le k \le n$$
$$s_k(x) = a_k(x - x_k) + b_k(x_{k+1} - x) + [(x - x_k)^3 c_{k+1} + (x_{k+1} - x)^3 c_k] / (6h_k) \quad 1 \le k \le n \quad (6.6-7)$$

then derivative equations becomes

$$s'_k(x) = a_k - b_k + [(x - x_k)^2 c_{k+1} - (x_{k+1} - x)^2 c_k]/h_k \quad 1 \le k \le n \qquad (6.6\text{-}8)$$
$$s''_k(x) = [(x - x_k)c_{k+1} - (x_{k+1} - x)c_k]/h_k \quad 1 \le k \le n$$

a_k ve b_k coefficients can be expressed as a function of c_k

$$b_k = \frac{[6y_k - h_k c_k]}{6h_k} \qquad 1 \le k \le n \qquad (6.6-9)$$

$$a_k = \frac{[6y_{k+1} - h_k^2 c_{k+1}]}{6h_k} \qquad 1 \le k \le n \qquad (6.6-10)$$

In this case only c_k terms left in the system of equations to be solved.

$$h_{k-1}c_{k-1} + 2(h_{k-1} - h_k)c_{k-1} + h_k c_{k+1}c_{k+1} = 6\left[\frac{y_{k+1} - y_k}{h_k} - \frac{y_k - y_{k-1}}{h_{k-1}}\right] \qquad 1 \le k \le n \qquad (6.6-10)$$

This system of equation has only n-2 terms to be solved. By making definition

$w_k = \frac{y_{k+1} - y_k}{h_k} = \frac{y_{k+1} - y_k}{x_{k+1} - x_k}$, $\quad 1 \le k \le n$ (6.6-11)

System of equation becomes

$$\begin{bmatrix} 1 & & & & & & \\ h_1 & 2(h_1+h_2) & h_2 & & & & \\ & h_2 & 2(h_2+h_3) & \cdots & & & \\ & \cdots & \cdots & \cdots & \cdots & \cdots & \\ & & & \cdots & 2(h_{n-3}+h_{n-2}) & h_{n-2} & \\ & & & & h_{n-2} & 2(h_{n-2}+h_{n-1}) & h_{n-1} \\ & & & & & 1 & \end{bmatrix} \begin{Bmatrix} c_0 \\ c_1 \\ c_2 \\ \cdots \\ c_{n-2} \\ c_{n-1} \\ c_n \end{Bmatrix} = \begin{Bmatrix} A \\ 6(w_2 - w_1) \\ 6(w_3 - w_2) \\ \cdots \\ 6(w_{n-2} - w_{n-3}) \\ 6(w_{n-1} - w_{n-2}) \\ B \end{Bmatrix} \quad (6.6-12)$$

Where A and B are the second derivative end conditions. A and B should be defined by user. Another important property of the above matrix is that it is a band matrix, therefore less amount of calculation is required to solve it (by using band matrix algorithms such as Thomas algorithm) compare to gauss elimination type of solution methods.

Program 6.6-1 Cubic spline curve fitting

```
import java.io.*;

class NA49
{

 public static double [] thomas(double f[],double e[],double g[],double r[])
 {
 // 3 band matrix system of equation solving algorithm
 int n=f.length;
 double x[]=new double[n];
 for(int k=1;k<n;k++)
   {e[k]=e[k]/f[k-1];
    f[k]=f[k]-e[k]*g[k-1];
   }
 for(int k=1;k<n;k++)
   {r[k]=r[k]-e[k]*r[k-1];
   }
 x[n-1]=r[n-1]/f[n-1];
 for(int k=(n-2);k>=0;k--)
   {x[k]=(r[k]-g[k]*x[k+1])/f[k];}
 return x;
 }

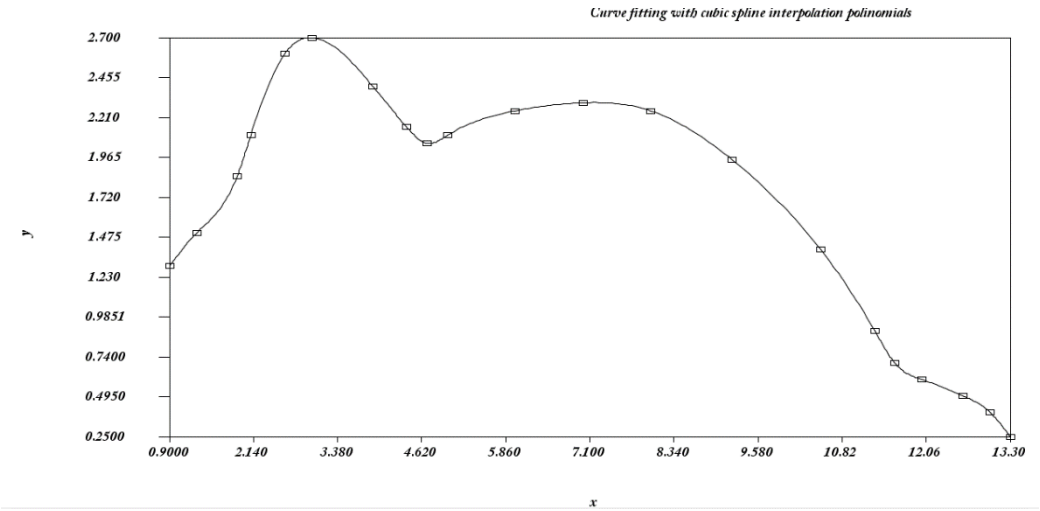

 public static double [][] cubic_spline(double xi[],double yi[],double c0,double cn)
 {
 int n=xi.length;
```

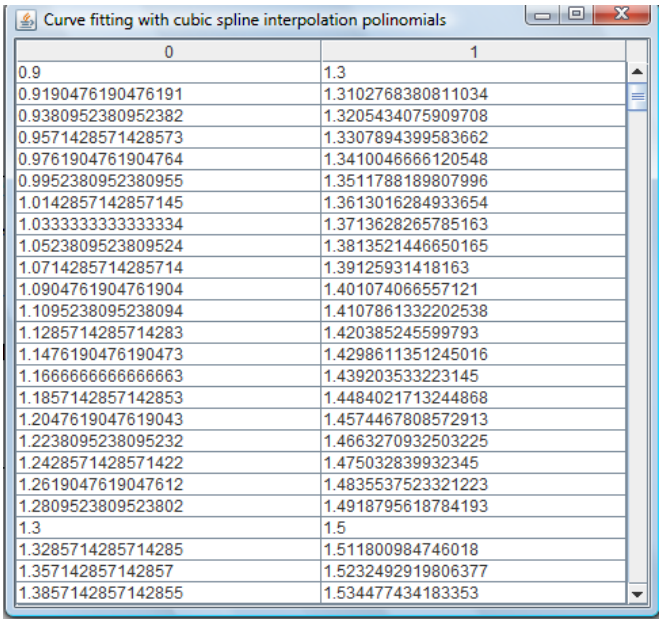
```
double h[]=new double[n];
double w[]=new double[n];
double f[]=new double[n];
double e[]=new double[n];
double g[]=new double[n];
double d[]=new double[n];
double x[]=new double[n];
double S[][]=new double[4][n];
int k;
for(k=0;k<(n-1);k++)
  {h[k]=xi[k+1]-xi[k];
   w[k]=(yi[k+1]-yi[k])/h[k];
        }
d[0]=c0;
d[n-1]=cn;
for(k=1;k<(n-1);k++)
  {d[k]=6.0*(w[k]-w[k-1]);}
f[0]=1.0;
f[n-1]=1.0;
g[0]=0.0;
g[n-1]=0.0;
e[0]=0.0;
e[n-1]=0.0;
for(k=1;k<(n-1);k++)
{f[k]=2.0*(h[k]+h[k-1]);e[k]=h[k-1];g[k]=h[k];}
S[2]=thomas(f,e,g,d);
S[3]=xi;
for(k=0;k<(n-1);k++)
 {S[0][k]=(6.*yi[k+1]-h[k]*h[k]*S[2][k+1])/(6.0*h[k]);
  S[1][k]=(6.*yi[k]-h[k]*h[k]*S[2][k])/(6.0*h[k]);
 }
return S;
}

public static double funcSpline(double S[][],double x)
{
int n=S[0].length;
double xx1=0;
double xx2=0;
double y=0;
double hk=0;
for(int k=0;k<(n-1);k++)
  {if(S[3][k]<=x && x<=S[3][k+1])
   {hk=(S[3][k+1]-S[3][k]);
    xx1=(x-S[3][k]);
    xx2=(S[3][k+1]-x);
    y=S[0][k]*xx1+S[1][k]*xx2+(xx1*xx1*xx1*S[2][k+1]+xx2*xx2*xx2*S[2][k])/(6.0*hk);
    break;
    }
  }
    if(y==0 && S[3][n-2]<=x )
    {
            int k=n-2;
            hk=(S[3][k+1]-S[3][k]);
    xx1=(x-S[3][k]);
    xx2=(S[3][k+1]-x);
    y=S[0][k]*xx1+S[1][k]*xx2+(xx1*xx1*xx1*S[2][k+1]+xx2*xx2*xx2*S[2][k])/(6.0*hk);
```

```
    }
 return y;
 }

public static double[][]  funcSpline(double xi[],double yi[],int numberofmidpoints)
{
            //numberofmidpoints :  in x--o--o--x--o--o--x chain if x's are esxperimental points
            // numberofmidpoints is 2
            int n=xi.length;
            int nn=(n-1)*(numberofmidpoints+1)+1;
            double z[][]=new double[2][nn];
  double S[][]=cubic_spline(xi,yi,0,0);
  double dx=0;
  int k=0;
  int i;
  for(i=0;i<(n-1);i++)
    {  z[0][k]=xi[i];z[1][k]=funcSpline(S,z[0][k]);k++;
      for(int j=0;j<numberofmidpoints;j++)
        {dx=(xi[i+1]-xi[i])/((double)numberofmidpoints+1.0);
      z[0][k]=z[0][k-1]+dx;z[1][k]=funcSpline(S,z[0][k]);k++;}
    }
    z[0][k]=xi[i];z[1][k]=funcSpline(S,z[0][k]);
  return z;
}
 public static void main(String args[]) throws IOException
 {double x[]={0.9,1.3,1.9,2.1,2.6,3.0,3.9,4.4,4.7,5.0,6.0,7.0,8.0,9.2,10.5,11.3,11.6,12.0,12.6,13.0,13.3};
 double y[]={1.3,1.5,1.85,2.1,2.6,2.7,2.4,2.15,2.05,2.1,2.25,2.3,2.25,1.95,1.4,0.9,0.7,0.6,0.5,0.4,0.25};
 double z3[][]=funcSpline(x,y,20);
 System.out.println("Cubic spline \n"+Matrix.toStringT(z3));
 Text.print(Text.T(z3),"Curve fitting with cubic spline interpolation polinomials");
 Plot pp=new Plot(x,y);
 pp.setPlabel("Curve fitting with cubic spline interpolation polinomials");
 pp.setPlotType(0,20);
 pp.addData(z3[0],z3[1]);
 pp.plot();
 }}
```





0	1
0.9	1.3
0.9190476190476191	1.3102768380811034
0.9380952380952382	1.3205434075909708
0.9571428571428573	1.3307894399583662
0.9761904761904764	1.3410046666120548
0.9952380952380955	1.3511788189807996
1.0142857142857145	1.3613016284933654
1.0333333333333334	1.3713628265785163
1.0523809523809524	1.3813521446650165
1.0714285714285714	1.39125931418163
1.0904761904761904	1.401074066557121
1.1095238095238094	1.4107861332202538
1.1285714285714283	1.420385245599793
1.1476190476190473	1.4298611351245016
1.1666666666666663	1.439203533223145
1.1857142857142853	1.4484021713244868
1.2047619047619043	1.4574467808572913
1.2238095238095232	1.4663270932503225
1.2428571428571422	1.475032839932345
1.2619047619047612	1.4835537523321223
1.2809523809523802	1.4918795618784193
1.3	1.5
1.3285714285714285	1.511800984746018
1.357142857142857	1.5232492919806377
1.3857142857142855	1.534477434183353

Program 6.6-2 Cubic spline curve fitting (python version)

```
# -*- coding: utf-8 -*-
"""
Created on Sun Jun  3 21:38:38 2018

@author: Mustafa Turhan Çoban
"""
from math import *
import numpy as np;
import matplotlib.pyplot as plt;

def thomas(f,e,g,r):
    n=len(f);
    x=[0.0 for i in range(n)];
    for k in range(0,n): e[k]=e[k]/f[k-1];f[k]=f[k]-e[k]*g[k-1];
    for k in range(1,n): r[k]=r[k]-e[k]*r[k-1];
    x[n-1]=r[n-1]/f[n-1];
    for k in range(n-2,-1,-1):  x[k]=(r[k]-g[k]*x[k+1])/f[k];
    return x;

def cubic_spline(xi,yi,c0,cn):
    n=len(xi);
    h=[0.0 for i in range(n)];
    w=[0.0 for i in range(n)];
    f=[0.0 for i in range(n)];
    e=[0.0 for i in range(n)];
    g=[0.0 for i in range(n)];
    d=[0.0 for i in range(n)];
    x=[0.0 for i in range(n)];
    S= [[0 for x in range(n)] for y in range(4)];
    for k in range(0,n-1):
        h[k]=xi[k+1]-xi[k];
        w[k]=(yi[k+1]-yi[k])/h[k];
    d[0]=c0;
    d[n-1]=cn;
    for k in range(0,n-1):
       d[k]=6.0*(w[k]-w[k-1]);
    f[0]=1.0;
    f[n-1]=1.0;
    g[0]=0.0;
    g[n-1]=0.0;
```

```
    e[0]=0.0;
    e[n-1]=0.0;
    for k in range(0,n-1):
        f[k]=2.0*(h[k]+h[k-1]);e[k]=h[k-1];g[k]=h[k];
    S[2]=thomas(f,e,g,d);
    S[3]=xi;
    for k in range(0,n-1):
       S[0][k]=(6.*yi[k+1]-h[k]*h[k]*S[2][k+1])/(6.0*h[k]);
       S[1][k]=(6.*yi[k]-h[k]*h[k]*S[2][k])/(6.0*h[k]);
    return S;

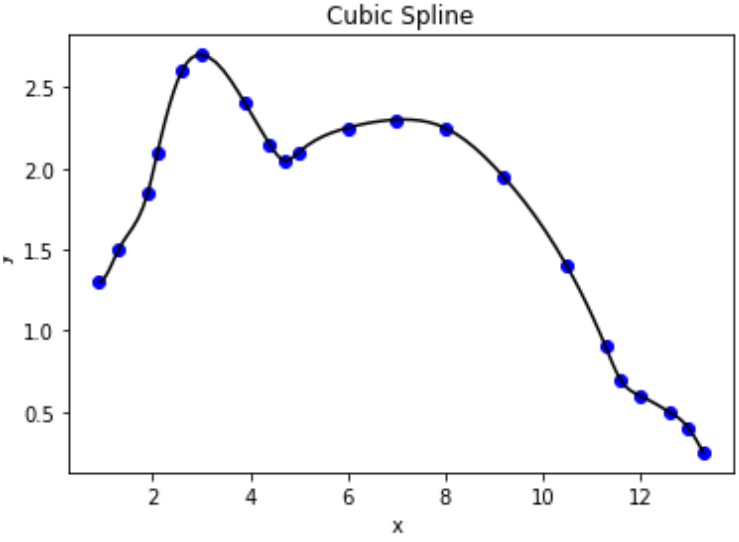
def funcSpline(S,x):
    n=len(S[0]);
    xx1=0;
    xx2=0;
    y=0;
    hk=0;
    for k in range(0,n-1):
       if S[3][k]<=x and x<=S[3][k+1]:
           hk=(S[3][k+1]-S[3][k]);
           xx1=(x-S[3][k]);
           xx2=(S[3][k+1]-x);
           y=S[0][k]*xx1+S[1][k]*xx2+(xx1*xx1*xx1*S[2][k+1]+xx2*xx2*xx2*S[2][k])/(6.0*hk);
           break;
    if y==0 and S[3][n-2]<=x:
         k=n-2;
         hk=(S[3][k+1]-S[3][k]);
         xx1=(x-S[3][k]);
         xx2=(S[3][k+1]-x);
         y=S[0][k]*xx1+S[1][k]*xx2+(xx1*xx1*xx1*S[2][k+1]+xx2*xx2*xx2*S[2][k])/(6.0*hk);
    return y;

def listSpline(xi,yi,numberofmidpoints,c0,cn):
    #numberofmidpoints :  in x--o--o--x--o--o--x chain if x's are esxperimental points
                #numberofmidpoints is 2
    n=len(xi);
    nn=(n-1)*(numberofmidpoints+1)+1;
    z=[[0 for x in range(nn)] for y in range(2)];
    S= [[0 for x in range(nn)] for y in range(4)];
    S=cubic_spline(xi,yi,c0,cn);
    dx=0;
    k=0;
    for i in range(0,n-1):
       z[0][k]=xi[i];z[1][k]=funcSpline(S,z[0][k]);k=k+1;
       for j in range(numberofmidpoints):
          dx=(xi[i+1]-xi[i])/(numberofmidpoints+1.0);
          z[0][k]=z[0][k-1]+dx;z[1][k]=funcSpline(S,z[0][k]);k=k+1;
    z[0][k]=xi[i+1];z[1][k]=funcSpline(S,z[0][k]);
    return z;

x=[0.9,1.3,1.9,2.1,2.6,3.0,3.9,4.4,4.7,5.0,6.0,7.0,8.0,9.2,10.5,11.3,11.6,12.0,12.6,13.0,13.3];
y=[1.3,1.5,1.85,2.1,2.6,2.7,2.4,2.15,2.05,2.1,2.25,2.3,2.25,1.95,1.4,0.9,0.7,0.6,0.5,0.4,0.25];
Z=listSpline(x,y,5,0,0);
print("Z=",Z);
plt.title('Cubic Spline')
plt.xlabel('x ')
plt.ylabel('y ');
plt.plot(x,y,'bo',Z[0],Z[1],'k');
```

```
 runfile('D:/okul/SCO1/cubic_spline.py', wdir='D:/okul/SCO1')
Reloaded modules: gauss
Z= [[0.9, 0.9666666666666667, 1.0333333333333334, 1.1, 1.1666666666666667, 1.2333333333333334, 1.3, 1.4, 1.5, 1.6,
1.7000000000000002, 1.8000000000000003, 1.9, 1.9333333333333333, 1.9666666666666668, 2.0, 2.033333333333333,
2.0666666666666664, 2.1, 2.1833333333333336, 2.266666666666667, 2.3500000000000005, 2.433333333333334, 2.5166666666666675,
```

2.6, 2.666666666666667, 2.7333333333333334, 2.8, 2.8666666666666663, 2.9333333333333327, 3.0, 3.15, 3.3, 3.4499999999999997, 3.5999999999999996, 3.7499999999999996, 3.9, 3.9833333333333334, 4.066666666666666, 4.1499999999999995, 4.2333333333333325, 4.3166666666666655, 4.4, 4.45, 4.5, 4.55, 4.6, 4.6499999999999995, 4.7, 4.75, 4.8, 4.85, 4.8999999999999995, 4.949999999999999, 5.0, 5.166666666666667, 5.333333333333334, 5.500000000000001, 5.666666666666668, 5.833333333333335, 6.0, 6.166666666666667, 6.333333333333334, 6.500000000000001, 6.666666666666668, 6.833333333333335, 7.0, 7.166666666666667, 7.333333333333334, 7.500000000000001, 7.666666666666668, 7.833333333333335, 8.0, 8.2, 8.399999999999999, 8.599999999999998, 8.799999999999997, 8.999999999999996, 9.2, 9.416666666666666, 9.633333333333333, 9.85, 10.066666666666666, 10.283333333333333, 10.5, 10.633333333333333, 10.766666666666666, 10.899999999999999, 11.033333333333331, 11.166666666666664, 11.3, 11.350000000000001, 11.400000000000002, 11.450000000000003, 11.500000000000004, 11.550000000000004, 11.6, 11.666666666666666, 11.733333333333333, 11.799999999999999, 11.866666666666665, 11.933333333333332, 12.0, 12.1, 12.2, 12.299999999999999, 12.399999999999999, 12.499999999999998, 12.6, 12.666666666666666, 12.733333333333333, 12.799999999999999, 12.866666666666665, 12.933333333333332, 13.0, 13.05, 13.100000000000001, 13.150000000000002, 13.200000000000003, 13.250000000000004, 13.3], [1.2999999999999998, 1.3093751894619083, 1.3344450507225516, 1.3706260230943061, 1.413334545889548, 1.4579870584206536, 1.4999999999999996, 1.551964750663964, 1.5957343748206128, 1.6393657921512435, 1.690915922337154, 1.7584416850596403, 1.8500000000000003, 1.887079217060214, 1.926982280840211, 1.9689675411820904, 2.0122933479279497, 2.0562180509198864, 2.1, 2.205651779159039, 2.3043189530703647, 2.394544441922051, 2.4748711659021705, 2.543842045198796, 2.6, 2.6349754723676666, 2.661593547897045, 2.680629632506552, 2.6928591321146036, 2.699057452639615, 2.7000000000000006, 2.6860390570520196, 2.652938927647158, 2.6042422845124706, 2.5434918003750124, 2.4742301479618374, 2.4, 2.357861812873503, 2.315552989956062, 2.2733429642288336, 2.231501168672976, 2.190297036269646, 2.1499999999999995, 2.1265238713656305, 2.1044571079150636, 2.0847855243507576, 2.0684949353751727, 2.056571155690767, 2.05, 2.0494053210004255, 2.0539631233699773, 2.062487449781682, 2.073792342908569, 2.086691845423666, 2.1, 2.1396051286242534, 2.1717712493844465, 2.1976960795913483, 2.2185773365557284, 2.235612737588356, 2.25, 2.262715615023503, 2.2738511695799972, 2.2832770245126888, 2.2908635406647826, 2.2964810788794843, 2.3, 2.301199077948401, 2.29949073896223, 2.2941958223578958, 2.284635167451807, 2.270129613560372, 2.25, 2.217799810228662, 2.1773615745280437, 2.129523884276508, 2.0751253308524147, 2.0150045056341255, 1.95, 1.8747328100503609, 1.7939415032693384, 1.7068493242653617, 1.61267951764686, 1.510655328022263, 1.4, 1.3272597895357943, 1.2506677564782636, 1.169934830911663, 1.0847719429202491, 0.9948900225882772, 0.9000000000000001, 0.8632203068251879, 0.8266057525736611, 0.7910663910053992, 0.7575122758803815, 0.7268534609585878, 0.6999999999999998, 0.6710795558939062, 0.6490500148136773, 0.632419332721743, 0.6196954655805323, 0.6093863693524749, 0.6, 0.585262369387163, 0.5694620676711136, 0.5528173873631121, 0.5355466209744191, 0.517868061016295, 0.5, 0.48794696771124646, 0.47511033878706393, 0.46069372039065026, 0.4439007196852034, 0.42393494383392094, 0.4000000000000001, 0.3790954038292061, 0.3559569510242995, 0.33103141291210286, 0.30476556081943873, 0.27760616607312966, 0.25]]



We can now investigate two dimensional cubic spline. Basically problem can be considered as spline of spline. If data is placed in a rectangular manner, data can be applied to create all the x axis data as one dimesional cubic spline curve fitting and then all created intermediate values can be used in a second cubic spline solution to find final interpolation value.
If x_i,y_i,z_i data set is given to find $z = f(x,y)$ first step is: to create $w_j = f(x)$ and then by using w_i,z_i find $z = f(x,y)$
Program for this is given below as two basic classes CS1 and CS2

Program 6.6-3 Cubic spline surface fitting first step CS1.java

```
//Cubic spline curve fitting
import java.io.*;
import javax.swing.*;

class CS1
{  public double S[][]; // cubic spline coefficients
```

```
 public double c[][]; // data matrix
 public int nx,ny;    // x and y data column numbers
 public int kk;       // equation number that given x located
 public CS1(String name,int nxi,int nyi,double c0,double cn)
 {nx=nxi;ny=nyi;
  c=Text.readDoubleT(name);
  double x[]=c[nx];double y[]=c[ny];
  S=cubic_spline(x,y,c0,cn);
 }
 public CS1(double a[][],int nxi,int nyi,double c0,double cn)
 {c=a;nx=nxi;ny=nyi;
  double x[]=c[nx];double y[]=c[ny];
  S=cubic_spline(x,y,c0,cn);
 }
 public CS1(double xi[],double yi[],double c0,double cn)
 {double c1[][]={xi,yi};c=c1;S=cubic_spline(xi,yi,c0,cn);}
public static double [] thomas(double f[],double e[],double g[],double r[])
{
// Thomas algorithm
// 3 band matrix system of equation solving algorithm
int n=f.length;
double x[]=new double[n];
for(int k=1;k<n;k++)
   {e[k]=e[k]/f[k-1];
    f[k]=f[k]-e[k]*g[k-1];
   }
for(int k=1;k<n;k++)
   {r[k]=r[k]-e[k]*r[k-1];
   }
x[n-1]=r[n-1]/f[n-1];
for(int k=(n-2);k>=0;k--)
   {x[k]=(r[k]-g[k]*x[k+1])/f[k];}
return x;
}

public static double [][] cubic_spline(double xi[],double yi[],double c0,double cn)
{
int n=xi.length;
double h[]=new double[n];
double w[]=new double[n];
double f[]=new double[n];
double e[]=new double[n];
double g[]=new double[n];
double d[]=new double[n];
double x[]=new double[n];
double S[][]=new double[4][n];
int k;
for(k=0;k<(n-1);k++)
   {h[k]=xi[k+1]-xi[k];
    w[k]=(yi[k+1]-yi[k])/h[k];
              }
d[0]=c0;
d[n-1]=cn;
for(k=1;k<(n-1);k++)
   {d[k]=6.0*(w[k]-w[k-1]);}
f[0]=1.0;
f[n-1]=1.0;
g[0]=0.0;
g[n-1]=0.0;
e[0]=0.0;
e[n-1]=0.0;
for(k=1;k<(n-1);k++)
{f[k]=2.0*(h[k]+h[k-1]);e[k]=h[k-1];g[k]=h[k];}
S[2]=thomas(f,e,g,d);
```

```
  S[3]=xi;
  for(k=0;k<(n-1);k++)
    {S[0][k]=(6.*yi[k+1]-h[k]*h[k]*S[2][k+1])/(6.0*h[k]);
     S[1][k]=(6.*yi[k]-h[k]*h[k]*S[2][k])/(6.0*h[k]);
    }
  return S;
  }

  public double func(double x)
  {
  int n=S[0].length;
  double xx1=0;
  double xx2=0;
  double y=0;
  double hk=0;
  int k=0;
  for(k=0;k<(n-1);k++)
     {if(S[3][k]<=x && x<=S[3][k+1])
      {hk=(S[3][k+1]-S[3][k]);
       xx1=(x-S[3][k]);
       xx2=(S[3][k+1]-x);
       y=S[0][k]*xx1+S[1][k]*xx2+(xx1*xx1*xx1*S[2][k+1]+xx2*xx2*xx2*S[2][k])/(6.0*hk);
       break;
       }
     }

       if(y==0 && S[3][n-2]<=x )
       {
                  k=n-2;
                  hk=(S[3][k+1]-S[3][k]);
       xx1=(x-S[3][k]);
       xx2=(S[3][k+1]-x);
       y=S[0][k]*xx1+S[1][k]*xx2+(xx1*xx1*xx1*S[2][k+1]+xx2*xx2*xx2*S[2][k])/(6.0*hk);
       }
     kk=k+1;
  return y;
  }

  // Türev formülleri
// =================


  public double dfunc(double x)
  {
  int n=S[0].length;
  double xx1=0;
  double xx2=0;
  double y=0;
  double hk=0;
  for(int k=0;k<(n-1);k++)
     {if(S[3][k]<=x && x<=S[3][k+1])
      {hk=(S[3][k+1]-S[3][k]);
       xx1=(x-S[3][k]);
       xx2=(S[3][k+1]-x);
       y=S[0][k]-S[1][k]+(xx1*xx1*S[2][k+1]-xx2*xx2*S[2][k])/(2.0*hk);
       break;
       }
     }
       if(y==0 && S[3][n-2]<=x )
       {
                  int k=n-2;
                  hk=(S[3][k+1]-S[3][k]);
       xx1=(x-S[3][k]);
       xx2=(S[3][k+1]-x);
```

```
     y=S[0][k]*xx1+S[1][k]*xx2+(xx1*xx1*xx1*S[2][k+1]+xx2*xx2*xx2*S[2][k])/(6.0*hk);
     }
return y;
}

public double integral(double a,double b)
{
  //kübik şerit integral[S(x)]dx  formülü
int n=S[0].length;
double xx1=0;
double xx2=0;
double y1,y2;
double hk=0;
double toplam=0;
y1=0;y2=0;
for(int k=0;k<(n-1);k++)
   { hk=(S[3][k+1]-S[3][k]);
    if(a>S[3][k+1])
    {toplam=0;
    }
              else if(S[3][k]<=a && a<=S[3][k+1] && b>S[3][k+1])//şart 2
     {
     xx1=(a-S[3][k]);
     xx2=(S[3][k+1]-a);
     y1=S[0][k]*xx1*xx1/2.0-S[1][k]*xx2*xx2/2.0+(xx1*xx1*xx1*xx1*S[2][k+1]-xx2*xx2*xx2*xx2*S[2][k])/(24.0*hk);
     xx1=hk;
     xx2=0;
     y2=S[0][k]*xx1*xx1/2.0-S[1][k]*xx2*xx2/2.0+(xx1*xx1*xx1*xx1*S[2][k+1]-xx2*xx2*xx2*xx2*S[2][k])/(24.0*hk);
     toplam+=(y2-y1);
     }
     else if(S[3][k]<=a && a<=S[3][k+1] && S[3][k]<=b && b<=S[3][k+1])//şart 3
     {
     xx1=(a-S[3][k]);
     xx2=(S[3][k+1]-a);
     y1=S[0][k]*xx1*xx1/2.0-S[1][k]*xx2*xx2/2.0+(xx1*xx1*xx1*xx1*S[2][k+1]-xx2*xx2*xx2*xx2*S[2][k])/(24.0*hk);
     xx1=(b-S[3][k]);
     xx2=(S[3][k+1]-b);
     y2=S[0][k]*xx1*xx1/2.0-S[1][k]*xx2*xx2/2.0+(xx1*xx1*xx1*xx1*S[2][k+1]-xx2*xx2*xx2*xx2*S[2][k])/(24.0*hk);
     toplam+=(y2-y1);
     }
     else if(a<S[3][k] && b>=S[3][k+1]) //şart 4
     {
     xx1=0;
     xx2=hk;
     y1=S[0][k]*xx1*xx1/2.0-S[1][k]*xx2*xx2/2.0+(xx1*xx1*xx1*xx1*S[2][k+1]-xx2*xx2*xx2*xx2*S[2][k])/(24.0*hk);
     xx1=hk;
     xx2=0;
     y2=S[0][k]*xx1*xx1/2.0-S[1][k]*xx2*xx2/2.0+(xx1*xx1*xx1*xx1*S[2][k+1]-xx2*xx2*xx2*xx2*S[2][k])/(24.0*hk);
     toplam+=(y2-y1);
     }
     else if(S[3][k]<=b && b<=S[3][k+1])//şart 5
     {xx1=0;
      xx2=hk;
      y1=S[0][k]*xx1*xx1/2.0-S[1][k]*xx2*xx2/2.0+(xx1*xx1*xx1*xx1*S[2][k+1]-xx2*xx2*xx2*xx2*S[2][k])/(24.0*hk);
      xx1=(b-S[3][k]);
      xx2=(S[3][k+1]-b);
      y2=S[0][k]*xx1*xx1/2.0-S[1][k]*xx2*xx2/2.0+(xx1*xx1*xx1*xx1*S[2][k+1]-xx2*xx2*xx2*xx2*S[2][k])/(24.0*hk);
      toplam+=(y2-y1);
      }
      else break;
   }
return toplam;
}
```

```
public double golden(double a,double b)
{
//  find the minimum of the function
//  note  maximum f(x) = minimum (-f(x))
    double epsilon=1.0e-10;
    double delta=1.0e-8;
    int print=0;
    double r1 = (Math.sqrt(5.0)-1.0)/2.0; // golden ratio
    double r2 = r1*r1;
    double h = b - a;
    double ya = -func(a);
    double yb = -func(b);
    double c = a + r2*h;
    double d = a + r1*h;
    double yc = -func(c);
    double yd = -func(d);
    int k = 1;
    double dp,dy,p,yp;
    while ((Math.abs(yb-ya)>epsilon) || (h>delta))
    {
      k++;
      if (yc<yd)
        {
        b = d;
        yb = yd;
        d = c;
        yd = yc;
        h = b - a;
        c = a + r2 * h;
        yc = -func(c);
        }
      else
       {
         a = c;
         ya = yc;
         c = d;
         yc = yd;
         h = b - a;
         d = a + r1 * h;
         yd = -func(d);
       }//end of if
    }//end of while
    dp = Math.abs(b-a);
    dy = Math.abs(yb-ya);
    p = a;
    yp = ya;
    if (yb<ya)
    {
      p = b;
      yp = yb;
    }
    if(print==1)
    {System.out.println("x min = "+p+"ymin = "+yp+"error of x ="+dp+"error of y"+dy); }
    return p;
}
public static void  printData(double c[][])
  {       String s3[]={"x[i]","y[i]"};
       Text.print(Text.T(c),s3,"Input data");
  }

  public static void main(String args[]) throws IOException
  { CS1 cs=new CS1("a100.txt",0,2,0.0,0.0);
    double T=300.0;
```

```
    double ro=cs.func(T);
    System.out.println("T="+T+"ro="+ro);
  }
}
```

Program 6.6-4 Cubic spline surface fitting second step CS2.java

```
 //3D Cubic spline surface fitting
import java.io.*;
import javax.swing.*;

class CS2
{  public double S[][];
   String name[];
   public CS1 cs[];
   public int n;
   double xx[],yy[];
   public double c0,cn;
   public int nyy;
   public int nx,ny;
   public double d[][][];
   //use set of files each with a constant property

   // namei: name of datafile to read data
   // use one file with a repeated column nselect
   // c: data matrix with a repeated one column
   // nyi y selection column
   // nxi  x selection column
   // nzi  z selection column
   // c0x and cnx x axis cubic spline end conditions (second derivatives)
   // c0y and cny y axis cubic spline end conditions (second derivatives)
   public CS2(String namei,int nxi,int nyi,int nzi,double c0x,double cnx,double c0y,double cny)
   { double c[][]=Text.readDouble(namei);
     set(c,nxi,nyi,nzi,c0x,cnx,c0y,cny);
   }
   public CS2(double c[][],int nxi,int nyi,int nzi,double c0x,double cnx,double c0y,double cny)
   {  set(c,nxi,nyi,nzi,c0x,cnx,c0y,cny);}

   // use one file with a repeated column nselect
   // nyi y selection column
   // nxi  x selection column
   // nzi  z selection column
   // c0x and cnx x axis cubic spline end conditions (second derivatives)
   // c0y and cny y axis cubic spline end conditions (second derivatives)

   public void set(double c[][],int nxi,int nyi,int nzi,double c0x,double cnx,double c0y,double cny)
   { nyy=nyi;
     nx=nxi;
     ny=nzi;
     c0=c0y;
     cn=cny;
     d=readDouble(c,nyy);
     n=d.length;
     cs=new CS1[n];
     xx=new double[n];
     yy=new double[n];
     for(int i=0;i<n;i++)
     {   cs[i]=new CS1(Text.T(d[i]),nx,ny,c0x,cnx);}
   }

  public static double[][][] readDouble(String name,int nc)
  {double c[][]=Text.readDouble(name);
   int n=c.length;
   int m=c[0].length;
   double dd[][][]=new double[1000][n][m];
```

```
 int k=0;
 int i=0;
 int j=0;
 int nj[]=new int[100];
 for(i=0;i<n-1;i++)
 {if(c[i][nc]==c[i+1][nc]) {dd[k][j]=c[i];j++;}
  else {dd[k][j]=c[i];nj[k]=j;j=0;
        k++;
       }
 }
 double ee[][][]=new double[k][][];
 for(i=0;i<k;i++)
 {   ee[i]=new double[nj[i]+1][m];
     for(j=0;j<nj[i]+1;j++) ee[i][j]=dd[i][j];
 }
 return ee;
}
public static double[][][] readDouble(double c[][],int nc)
{int n=c.length;
 int m=c[0].length;
 double dd[][][]=new double[1000][n][m];
 int k=0;
 int i=0;
 int j=0;
 int nj[]=new int[1000];
 for(i=0;i<n-1;i++)
 {if(c[i][nc]==c[i+1][nc]) {dd[k][j]=c[i];j++;}
  else {dd[k][j]=c[i];nj[k]=j;j=0;
        k++;
       }
 }
 double ee[][][]=new double[k][][];
 for(i=0;i<k;i++)
 {
                ee[i]=new double[nj[i]+1][m];
     for(j=0;j<nj[i]+1;j++) ee[i][j]=dd[i][j];
 }
 return ee;
}
public static double [] thomas(double f[],double e[],double g[],double r[])
{
// 3 band matrix system of equation solving algorithm
int n=f.length;
double x[]=new double[n];
for(int k=1;k<n;k++)
   {e[k]=e[k]/f[k-1];
    f[k]=f[k]-e[k]*g[k-1];
   }
for(int k=1;k<n;k++)
   {r[k]=r[k]-e[k]*r[k-1];
   }
x[n-1]=r[n-1]/f[n-1];
for(int k=(n-2);k>=0;k--)
   {x[k]=(r[k]-g[k]*x[k+1])/f[k];}
return x;
}

public static double [][] cubic_spline(double xi[],double yi[],double c0,double cn)
{
int n=xi.length;
double h[]=new double[n];
double w[]=new double[n];
double f[]=new double[n];
double e[]=new double[n];
```

```
double g[]=new double[n];
double d[]=new double[n];
double x[]=new double[n];
double S[][]=new double[4][n];
int k;
for(k=0;k<(n-1);k++)
   {h[k]=xi[k+1]-xi[k];
    w[k]=(yi[k+1]-yi[k])/h[k];
              }
d[0]=c0;
d[n-1]=cn;
for(k=1;k<(n-1);k++)
   {d[k]=6.0*(w[k]-w[k-1]);}
f[0]=1.0;
f[n-1]=1.0;
g[0]=0.0;
g[n-1]=0.0;
e[0]=0.0;
e[n-1]=0.0;
for(k=1;k<(n-1);k++)
{f[k]=2.0*(h[k]+h[k-1]);e[k]=h[k-1];g[k]=h[k];}
S[2]=thomas(f,e,g,d);
S[3]=xi;
for(k=0;k<(n-1);k++)
  {S[0][k]=(6.*yi[k+1]-h[k]*h[k]*S[2][k+1])/(6.0*h[k]);
   S[1][k]=(6.*yi[k]-h[k]*h[k]*S[2][k])/(6.0*h[k]);
  }
return S;
}
public double func(double x1,double y1)
{ double xx1[]={x1,y1};
  return func(xx1);
}
public double func(double x[])
{ //create data set

            for(int i=0;i<n;i++)
            {  yy[i]=cs[i].func(x[0]);
               xx[i]=cs[i].c[nyy][0];
               //xx[i]=cs[i].c[nyy][cs[i].kk];
            }
            S=cubic_spline(xx,yy,c0,cn);
 int n=S[0].length;
double xx1=0;
double xx2=0;
double y=0;
double hk=0;
int k=0;
for(k=0;k<n-1;k++)
   {if(S[3][k]<=x[1] && x[1]<=S[3][k+1])
    {hk=(S[3][k+1]-S[3][k]);
     xx1=(x[1]-S[3][k]);
     xx2=(S[3][k+1]-x[1]);
     y=S[0][k]*xx1+S[1][k]*xx2+(xx1*xx1*xx1*S[2][k+1]+xx2*xx2*xx2*S[2][k])/(6.0*hk);
     break;
     }
   }
     if(y==0 && S[3][n-2]<=x[1] )
     {
               k=n-2;
               hk=(S[3][k+1]-S[3][k]);
     xx1=(x[1]-S[3][k]);
     xx2=(S[3][k+1]-x[1]);
     y=S[0][k]*xx1+S[1][k]*xx2+(xx1*xx1*xx1*S[2][k+1]+xx2*xx2*xx2*S[2][k])/(6.0*hk);
```

```
      }
  return y;
  }

  // Türev formülleri
// =================

  public double dfunc(double x1,double y1)
  {double xx1[]={x1,y1};
   return dfunc(xx1) ;
  }
  public double dfunc(double x[])
  {
  double xx1=0;
  double xx2=0;
  double y=0;
  double hk=0;
  double yy[]=new double[n];
              for(int i=0;i<=n;i++)
              {yy[i]=cs[i].func(x[0]);}
               S=cubic_spline(xx,yy,c0,cn);
  for(int k=0;k<(n-1);k++)
    {if(S[3][k]<=x[1] && x[1]<=S[3][k+1])
     {hk=(S[3][k+1]-S[3][k]);
      xx1=(x[1]-S[3][k]);
      xx2=(S[3][k+1]-x[1]);
      y=S[0][k]-S[1][k]+(xx1*xx1*S[2][k+1]-xx2*xx2*S[2][k])/(2.0*hk);
      break;
      }
    }
      if(y==0 && S[3][n-2]<=x[1] )
      {
                int k=n-2;
                hk=(S[3][k+1]-S[3][k]);
      xx1=(x[1]-S[3][k]);
      xx2=(S[3][k+1]-x[1]);
      y=S[0][k]*xx1+S[1][k]*xx2+(xx1*xx1*xx1*S[2][k+1]+xx2*xx2*xx2*S[2][k])/(6.0*hk);
      }
  return y;
  }

  public static void main(String args[]) throws IOException
  { //String
name[]={"a10.txt","a50.txt","a90.txt","a95.txt","a100.txt","a101.txt","a110.txt","a150.txt","a200.txt","a500.txt","a1000.txt","a2000.txt","a30
00.txt"};
              //CS2 c2=new CS2(name,2,0,1,0.0,0.0,0.0,0.0);
              //column of P = 1
              //column of T 0 column of ro 2
              double c[][]=Text.readDouble("axy.txt");
              Text.print(c);
              CS2 c2=new CS2(c,1,0,2,0.0,0.0,0.0,0.0);
    double x=-1.0;
    double y=-1.0;
    double z=c2.func(x,y);
    System.out.println("x="+x+"y="+y+"z="+z);
  }
}
```

Sample data:

x	y	z	x	y	z	x	y	z	x	y	z	x	y	z	x	y	z
-2	-2	8	-1.6	-2	6.56	-1.2	-2	5.44	-0.8	-2	4.64	-0.4	-2	4.16	0	-2	4.00
-2	-1.8	7.24	-1.6	-1.8	5.8	-1.2	-1.8	4.68	-0.8	-1.8	3.88	-0.4	-1.8	3.40	0	-1.8	3.24
-2	-1.6	6.56	-1.6	-1.6	5.12	-1.2	-1.6	4	-0.8	-1.6	3.2	-0.4	-1.6	2.72	0	-1.6	2.56
-2	-1.4	5.96	-1.6	-1.4	4.52	-1.2	-1.4	3.4	-0.8	-1.4	2.6	-0.4	-1.4	2.12	0	-1.4	1.96
-2	-1.2	5.44	-1.6	-1.2	4	-1.2	-1.2	2.88	-0.8	-1.2	2.08	-0.4	-1.2	1.60	0	-1.2	1.44

-2	-1	5	-1.6	-1	3.56	-1.2	-1	2.44	-0.8	-1	1.64	-0.4	-1	1.16	0	-1	1.00
-2	-0.8	4.64	-1.6	-0.8	3.2	-1.2	-0.8	2.08	-0.8	-0.8	1.28	-0.4	-0.8	0.80	0	-0.8	0.64
-2	-0.6	4.36	-1.6	-0.6	2.92	-1.2	-0.6	1.8	-0.8	-0.6	1	-0.4	-0.6	0.52	0	-0.6	0.36
-2	-0.4	4.16	-1.6	-0.4	2.72	-1.2	-0.4	1.6	-0.8	-0.4	0.8	-0.4	-0.4	0.32	0	-0.4	0.16
-2	-0.2	4.04	-1.6	-0.2	2.6	-1.2	-0.2	1.48	-0.8	-0.2	0.68	-0.4	-0.2	0.20	0	-0.2	0.04
-2	0	4	-1.6	0	2.56	-1.2	0	1.44	-0.8	0	0.64	-0.4	0	0.16	0	0	0.00
-2	0.2	4.04	-1.6	0.2	2.6	-1.2	0.2	1.48	-0.8	0.2	0.68	-0.4	0.2	0.20	0	0.2	0.04
-2	0.4	4.16	-1.6	0.4	2.72	-1.2	0.4	1.6	-0.8	0.4	0.8	-0.4	0.4	0.32	0	0.4	0.16
-2	0.6	4.36	-1.6	0.6	2.92	-1.2	0.6	1.8	-0.8	0.6	1	-0.4	0.6	0.52	0	0.6	0.36
-2	0.8	4.64	-1.6	0.8	3.2	-1.2	0.8	2.08	-0.8	0.8	1.28	-0.4	0.8	0.80	0	0.8	0.64
-2	1	5	-1.6	1	3.56	-1.2	1	2.44	-0.8	1	1.64	-0.4	1	1.16	0	1	1.00
-2	1.2	5.44	-1.6	1.2	4	-1.2	1.2	2.88	-0.8	1.2	2.08	-0.4	1.2	1.60	0	1.2	1.44
-2	1.4	5.96	-1.6	1.4	4.52	-1.2	1.4	3.4	-0.8	1.4	2.6	-0.4	1.4	2.12	0	1.4	1.96
-2	1.6	6.56	-1.6	1.6	5.12	-1.2	1.6	4	-0.8	1.6	3.2	-0.4	1.6	2.72	0	1.6	2.56
-2	1.8	7.24	-1.6	1.8	5.8	-1.2	1.8	4.68	-0.8	1.8	3.88	-0.4	1.8	3.40	0	1.8	3.24
-2	2	8	-1.6	2	6.56	-1.2	2	5.44	-0.8	2	4.64	-0.4	2	4.16	0	2	4.00
-1.8	-2	7.24	-1.4	-2	5.96	-1	-2	5	-0.6	-2	4.36	-0.2	-2	4.04	0.2	-2	4.04
-1.8	-1.8	6.48	-1.4	-1.8	5.2	-1	-1.8	4.24	-0.6	-1.8	3.6	-0.2	-1.8	3.28	0.2	-1.8	3.28
-1.8	-1.6	5.8	-1.4	-1.6	4.52	-1	-1.6	3.56	-0.6	-1.6	2.92	-0.2	-1.6	2.60	0.2	-1.6	2.60
-1.8	-1.4	5.2	-1.4	-1.4	3.92	-1	-1.4	2.96	-0.6	-1.4	2.32	-0.2	-1.4	2.00	0.2	-1.4	2.00
-1.8	-1.2	4.68	-1.4	-1.2	3.4	-1	-1.2	2.44	-0.6	-1.2	1.8	-0.2	-1.2	1.48	0.2	-1.2	1.48
-1.8	-1	4.24	-1.4	-1	2.96	-1	-1	2	-0.6	-1	1.36	-0.2	-1	1.04	0.2	-1	1.04
-1.8	-0.8	3.88	-1.4	-0.8	2.6	-1	-0.8	1.64	-0.6	-0.8	1	-0.2	-0.8	0.68	0.2	-0.8	0.68
-1.8	-0.6	3.6	-1.4	-0.6	2.32	-1	-0.6	1.36	-0.6	-0.6	0.72	-0.2	-0.6	0.40	0.2	-0.6	0.40
-1.8	-0.4	3.4	-1.4	-0.4	2.12	-1	-0.4	1.16	-0.6	-0.4	0.52	-0.2	-0.4	0.20	0.2	-0.4	0.20
-1.8	-0.2	3.28	-1.4	-0.2	2	-1	-0.2	1.04	-0.6	-0.2	0.4	-0.2	-0.2	0.08	0.2	-0.2	0.08
-1.8	0	3.24	-1.4	0	1.96	-1	0	1	-0.6	0	0.36	-0.2	0	0.04	0.2	0	0.04
-1.8	0.2	3.28	-1.4	0.2	2	-1	0.2	1.04	-0.6	0.2	0.4	-0.2	0.2	0.08	0.2	0.2	0.08
-1.8	0.4	3.4	-1.4	0.4	2.12	-1	0.4	1.16	-0.6	0.4	0.52	-0.2	0.4	0.20	0.2	0.4	0.20
-1.8	0.6	3.6	-1.4	0.6	2.32	-1	0.6	1.36	-0.6	0.6	0.72	-0.2	0.6	0.40	0.2	0.6	0.40
-1.8	0.8	3.88	-1.4	0.8	2.6	-1	0.8	1.64	-0.6	0.8	1	-0.2	0.8	0.68	0.2	0.8	0.68
-1.8	1	4.24	-1.4	1	2.96	-1	1	2	-0.6	1	1.36	-0.2	1	1.04	0.2	1	1.04
-1.8	1.2	4.68	-1.4	1.2	3.4	-1	1.2	2.44	-0.6	1.2	1.8	-0.2	1.2	1.48	0.2	1.2	1.48
-1.8	1.4	5.2	-1.4	1.4	3.92	-1	1.4	2.96	-0.6	1.4	2.32	-0.2	1.4	2.00	0.2	1.4	2.00
-1.8	1.6	5.8	-1.4	1.6	4.52	-1	1.6	3.56	-0.6	1.6	2.92	-0.2	1.6	2.60	0.2	1.6	2.60
-1.8	1.8	6.48	-1.4	1.8	5.2	-1	1.8	4.24	-0.6	1.8	3.6	-0.2	1.8	3.28	0.2	1.8	3.28
-1.8	2	7.24	-1.4	2	5.96	-1	2	5	-0.6	2	4.36	-0.2	2	4.04	0.2	2	4.04

Result:

```
---------- Capture Output ----------
> "D:\co\java\bin\java.exe" CS2
x=-1.0y=-1.0z=2.0
```

It should be note that repeated column of the data should be selected as y column so fort he given data column selection is done as 1,0,2, i.e first column is taken as y axis (axis calculated by second cubic spline in CS2). And data must be given this rectangualr form to utilise thissurface cubic spline interpolation process.

6.7 CUBIC B-SPLINE INTERPOLATION

B-spline interpolation will be investigated in this section. In general B-spline definition spline functions are defined in the region from minus infinity to plus infinity. We will specify cubic B-spline here for given $k_0,\ldots,k_N$ knots general curve fitting functions will be defined as

$$S(x) = \sum_{i=-3}^{N} a_i B_i(x) \quad (6.7-1)$$

$B_i(x)$ in the equation is called B-spline function. B-spline function for k_i, i=-3,-2,0,…,N,N+1,N+2 knots for the x value in the [a,b] interval , and with the limit values k_0=a, k_N=b can have the following property

$\sum_{i=-3}^{N} B_i(x) = 1 \quad (6.7-2)$

Interpolation equation S(x) has N+3 coefficient. There are N+1 data in the [a,b] data range so that additional equations will be needed to solve the data set. In order to define B-spline function B(x) a recursive calculation process will be defined. We first define B-splines that may be piecewise constant(n=1), linear(n=2), quadratic (n=3)and cubic (n=4). The condition that the sum of all B-spline function in all regions are still holds.

$\sum_{i=-3}^{N+3} B_i(x) = 1$ (n=1,2,3,4) (6.7-3)

The B-splines of order n are related to those of order n-1 by the recurrence relation

$$B_{i,n}(x) = \frac{(x-k_i)}{k_{i+n-1}-k_i} B_{i,n-1}(x) + \frac{(k_{i+n}-x)}{k_{i+n}-k_{i+1}} B_{i+1,n-1}(x) \quad (6.7\text{-}4)$$

Where i=-3,…N-1 range and n=(1,2,3,4) .In computing with this relation, B-spline of order one will be taken as a starting point. Therefore first order B spline should be independently defined.

$$B_{i,1}(x) = \begin{cases} 0 & x < k_i \\ 1 & k_i \le x < k_{i+1} \\ 0 & k_{i+1} \le x \end{cases} \quad \text{i=-3,-2,…,N+3} \quad (6.7\text{-}5)$$



Figure 6.7-1 first order(constant) B-spline function

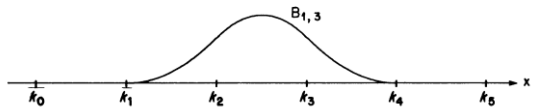


Figure 6.7-2 3th order(quadratic) B-spline function

By using first order equation and recurrence relation, linear, quadratic and cubic B spline functions can be calculated. First order(constant) and third order(quadratic) B-spline functions are shown in Figure 6.7-1 and 6.7-2 .

Now that B-spline functions are defined, we can return to solve coefficients of the iterations equation. By suppliying given k_j and f_j data points into the quation N+1 equations will be formed to solve

$\sum_{i=-3}^{N-1} a_i B_i(k_j) = f_j$ j=0,1,…,N (6.7-6)

In the jth equation only terms with B_{j-3}, B_{j-2}, B_{j-1} are different than zero the rest will be zero. In order to solve the data set two more equation will be needed. In order to established this, second derivative is at both end of the B-spline equation will be defined. If Natural cubic spline conditions are applied, this boundary values will be zero. Of course the problem can be solve by using any other type of boundary conditions.

$S"(k_0) = 0$

$S"(k_N) = 0$ (6.7-7)

When this condition applied to the curve fitting equations, they become

$a_{-3}B"_{-3}(k_0) + a_{-2}B"_{-2}(k_0) + a_{-1}B"_{-1}(k_0) = 0$

$a_{N-3}B"_{N-3}(k_N) + a_{N-2}B"_{N-2}(k_N) + a_{N-1}B"_{N-1}(k_N) = 0$ (6.7-8)

Second derivatives in the equation defined as:

$B"_{-3}(k_0) = \frac{6}{(k_1-k_{-1})(k_1-k_{-2})}$ (6.7-9)

$$B"_{-2}(k_0) = \frac{-6}{(k_1-k_{-1})}\left\{\frac{1}{(k_2-k_{-1})} + \frac{1}{(k_1-k_{-2})}\right\} \quad (6.7-10)$$

$$B"_{-1}(k_0) = \frac{6}{(k_1-k_{-1})(k_2-k_{-1})} \quad (6.7-11)$$

$$B"_{N-3}(k_0) = \frac{6}{(k_{N+1}-k_{N-1})(k_{N+1}-k_{N-2})} \quad (6.7-12)$$

$$B"_{N-2}(k_N) = \frac{-6}{(k_{N+1}-k_{N-1})}\left\{\frac{1}{(k_{N+2}-k_{N-1})} + \frac{1}{(k_{N+1}-k_{N-2})}\right\} \quad (6.7-13)$$

$$B"_{N-1}(k_0) = \frac{6}{(k_{N+1}-k_{N-1})(k_{N+2}-k_{N-1})} \quad (6.7-14)$$

It should be noted that k_{-3}, k_{-2}, k_{-1}, k_{N+1}, k_{N+2}, k_{N+3} values are not defined in the data set. This values can be set arbitrarily, but should satisfy $k_{-3}<k_{-2}<k_{-1}<k_0$ and $k_N<k_{N+1}<k_{N+2}<k_{N+3}$ conditions.

Program 6.7-1 B-spline curve fitting

```
import javax.swing.*;
```

```
public class B_spline
{ // cubic B-spline interpolation
  public double x[];
  public double f[];
  public double k[];
  int N;
  int n1;
  int m;
  double B[];
  double B2[];
  double a[];

  public B_spline(double xi[],double yi[])
  {   N=yi.length-1;
      n1=N+7;
              x=new double[N+1];
              f=new double[N+1];
              k=new double[n1];
      for(int i=0;i<=N;i++)
      {x[i]=xi[i];f[i]=yi[i];k[i+3]=x[i];
      }
      double dx=(x[N]-x[0]);
      k[0]=x[0]-6.0*dx;
      k[1]=x[0]-4.0*dx;
      k[2]=x[0]-2.0*dx;
      k[N+4]=x[N]+2.0*dx;
      k[N+5]=x[N]+4.0*dx;
      k[N+6]=x[N]+6.0*dx;
      m=5;
      B=new double[N+7];
      B2=new double[N+7];
      A();
  }

public  double[] B(double t)
{
double b[][]=new double[m][N+7];
for(int i=0;i<k.length-1;i++)
{if(t>=k[i] && t<k[i+1]) b[1][i]=1;
 else b[1][i]=0.0;
}
for(int n=2;n<m;n++)
{
for(int i=0;i<k.length-4;i++)
{
b[n][i]=(t-k[i])/(k[i+n-1]-k[i])*b[n-1][i]+(k[i+n]-t)/(k[i+n]-k[i+1])*b[n-1][i+1];
B[i]=b[n][i];
}
}
return B;
}

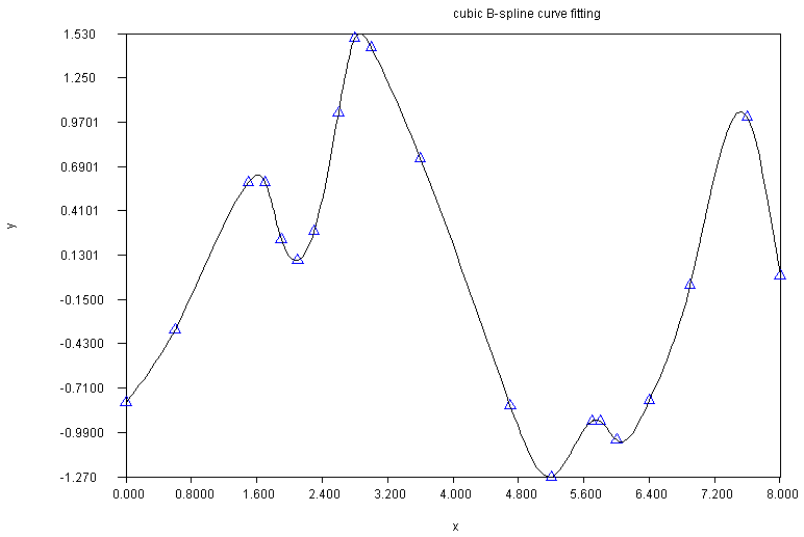
public void A()
{
B2[0]=6.0/(k[4]-k[2])/(k[4]-k[1]);
B2[1]=-6.0/(k[4]-k[2])*(1.0/((k[5]-k[2])+1.0/(k[4]-k[1]) ));
B2[2]=6.0/(k[4]-k[2])/(k[5]-k[2]);
```

```
B2[N]=6.0/(k[N+4]-k[N+2])/(k[N+4]-k[N+1]);
B2[N+1]=6.0/(k[N+4]-k[N+2])*(1.0/((k[N+5]-k[N+2])+1.0/(k[N+4]-k[N+1]) ));
B2[N+2]=6.0/(k[N+4]-k[N+2])/(k[N+5]-k[N+2]);
double AA[][]=new double[N+3][N+3];
double BB[]=new double[N+3];
AA[0][0]=B2[0];
AA[0][1]=B2[1];
AA[0][2]=B2[2];
BB[0]=0;
AA[N+2][N]=B2[N];
AA[N+2][N+1]=B2[N+1];
AA[N+2][N+2]=B2[N+2];
BB[N+2]=0;
for(int j=0;j<=N;j++)
{ B=B(k[j+3]);
  for(int i=0;i<=N+2;i++)
  { AA[j+1][i]=B[i];}
  BB[j+1]=f[j];
}
//Matrix is a 3-band matrix
a=thomas(AA,BB);
}
public static double [] thomas(double a[][],double r[])
  {
  int n=a.length;
  double f[]=new double[n];
  double e[]=new double[n];
  double g[]=new double[n];
  double x[]=new double[n];
  for(int i=0;i<n;i++)     {f[i]=a[i][i];}
  for(int i=0;i<(n-1);i++) {g[i]=a[i][i+1];}
  for(int i=0;i<(n-1);i++) {e[i+1]=a[i+1][i];}
  for(int k=1;k<n;k++)
     {e[k]=e[k]/f[k-1];
      f[k]=f[k]-e[k]*g[k-1];
     }
  for(int k=1;k<n;k++)
     {r[k]=r[k]-e[k]*r[k-1];
     }
  x[n-1]=r[n-1]/f[n-1];
  for(int k=(n-2);k>=0;k--)
     {x[k]=(r[k]-g[k]*x[k+1])/f[k];}
  return x;
  }
public double S(double x)
{ B=B(x);
  double ss=0;
  for(int i=0;i<=N+2;i++)
  {ss+=a[i]*B[i];}
  return ss;
}
public double func(double x)
{ return S(x);}

 public static  double[][]  funcB_Spline(double xi[],double yi[],int numberofmidpoints)
 {
    B_spline b1=new B_spline(xi,yi);
```

```
    double aa[][]=b1.funcB_Spline(numberofmidpoints);
    return aa;
}

public  double[][]  funcB_Spline(int numberofmidpoints)
{
            // numberofmidpoints:  x--o--o--x--o--o--x in the curve fitting chain x
            // is experimental data and o is mid point (in this case equal to 2)
            // This function generates curve fitted data
            int n=x.length;
            int nn=(n-1)*(numberofmidpoints+1)+1;
            double z[][]=new double[2][nn];
   double dx=0;
   int k=0;
   int i;
   for(i=0;i<(n-1);i++)
     {  z[0][k]=x[i];z[1][k]=S(z[0][k]);k++;
       for(int j=0;j<numberofmidpoints;j++)
         {dx=(x[i+1]-x[i])/((double)numberofmidpoints+1.0);
       z[0][k]=z[0][k-1]+dx;z[1][k]=S(z[0][k]);k++;}
     }
     z[0][k]=x[i];z[1][k]=S(z[0][k]);
   return z;
}
 public static void main(String[] arg)
 {
    double a[][]={{0.0,  0.6 ,1.5 ,1.7 ,1.9 ,2.1,2.3 ,2.6 ,2.8,3.0 ,3.6 ,4.7  ,5.2  ,5.7  ,5.8  , 6.0 , 6.4 ,6.9,7.6,8.0},
             {-0.8,-0.34,0.59,0.59,0.23,0.1,0.28,1.03,1.5,1.44,0.74,-0.82,-1.27,-0.92,-0.92,-1.04,-0.79,-0.06,1.0,0.0}};
    //String dosya=JOptionPane.showInputDialog("enter input file");
    //double a[][]=Matrix.T(Text.readDouble(dosya));
    int numberofmidpoints=10;
    double aa[][]=funcB_Spline(a[0],a[1],10);
    Plot pp=new Plot(a[0],a[1]);
    pp.setPlotType(0,24);
    pp.setColor(0,0,0,255);
    pp.addData(aa[0],aa[1]);
    pp.plot();
  }
}
```



Program 6.7-2 B-spline curve fitting python version

```
 # -*- coding: utf-8 -*-
"""
Created on Sun Jun  3 21:38:38 2018

@author: Mustafa Turhan Çoban
"""
from math import *
import matplotlib.pyplot as plt;

def thomas(f,e,g,r):
    n=len(f);
    x=[0.0 for i in range(n)];
    for k in range(0,n): e[k]=e[k]/f[k-1];f[k]=f[k]-e[k]*g[k-1];
    for k in range(1,n): r[k]=r[k]-e[k]*r[k-1];
    x[n-1]=r[n-1]/f[n-1];
    for k in range(n-2,-1,-1):  x[k]=(r[k]-g[k]*x[k+1])/f[k];
    return x;

def cubic_spline(xi,yi,c0,cn):
    n=len(xi);
    h=[0.0 for i in range(n)];
    w=[0.0 for i in range(n)];
    f=[0.0 for i in range(n)];
    e=[0.0 for i in range(n)];
    g=[0.0 for i in range(n)];
    d=[0.0 for i in range(n)];
    S= [[0 for x in range(n)] for y in range(4)];
    for k in range(0,n-1):
        h[k]=xi[k+1]-xi[k];
        w[k]=(yi[k+1]-yi[k])/h[k];
    d[0]=c0;
    d[n-1]=cn;
    for k in range(0,n-1):
       d[k]=6.0*(w[k]-w[k-1]);
    f[0]=1.0;
    f[n-1]=1.0;
    g[0]=0.0;
    g[n-1]=0.0;
    e[0]=0.0;
    e[n-1]=0.0;
    for k in range(0,n-1):
        f[k]=2.0*(h[k]+h[k-1]);e[k]=h[k-1];g[k]=h[k];
    S[2]=thomas(f,e,g,d);
    S[3]=xi;
    for k in range(0,n-1):
       S[0][k]=(6.*yi[k+1]-h[k]*h[k]*S[2][k+1])/(6.0*h[k]);
       S[1][k]=(6.*yi[k]-h[k]*h[k]*S[2][k])/(6.0*h[k]);
    return S;


def funcSpline(S,x):
    n=len(S[0]);
    xx1=0;
    xx2=0;
    y=0;
    hk=0;
    for k in range(0,n-1):
       if S[3][k]<=x and x<=S[3][k+1]:
           hk=(S[3][k+1]-S[3][k]);
           xx1=(x-S[3][k]);
           xx2=(S[3][k+1]-x);
           y=S[0][k]*xx1+S[1][k]*xx2+(xx1*xx1*xx1*S[2][k+1]+xx2*xx2*xx2*S[2][k])/(6.0*hk);
           break;
```

```
    if y==0 and S[3][n-2]<=x:
        k=n-2;
        hk=(S[3][k+1]-S[3][k]);
        xx1=(x-S[3][k]);
        xx2=(S[3][k+1]-x);
        y=S[0][k]*xx1+S[1][k]*xx2+(xx1*xx1*xx1*S[2][k+1]+xx2*xx2*xx2*S[2][k])/(6.0*hk);
    return y;

def listSpline(xi,yi,numberofmidpoints,c0,cn):
    # numberofmidpoints :  in x--o--o--x--o--o--x chain if x's are esxperimental points
            # numberofmidpoints is 2
    n=len(xi);
    nn=(n-1)*(numberofmidpoints+1)+1;
    z=[[0 for x in range(nn)] for y in range(2)];
    S= [[0 for x in range(nn)] for y in range(4)];
    S=cubic_spline(xi,yi,c0,cn);
    dx=0;
    k=0;
    for i in range(0,n-1):
       z[0][k]=xi[i];z[1][k]=funcSpline(S,z[0][k]);k=k+1;
       for j in range(numberofmidpoints):
          dx=(xi[i+1]-xi[i])/(numberofmidpoints+1.0);
          z[0][k]=z[0][k-1]+dx;z[1][k]=funcSpline(S,z[0][k]);k=k+1;
    z[0][k]=xi[i+1];z[1][k]=funcSpline(S,z[0][k]);
    return z;

def read_array2D(d):
    array2D = []
    f=open(d, 'r');
    lines=f.readlines();
    for line in lines:
       line.rstrip('\n');
       numbers_str=line.split();
       numbers_float=[float(x) for x in numbers_str]
       array2D.append(numbers_float)
    return T(array2D);

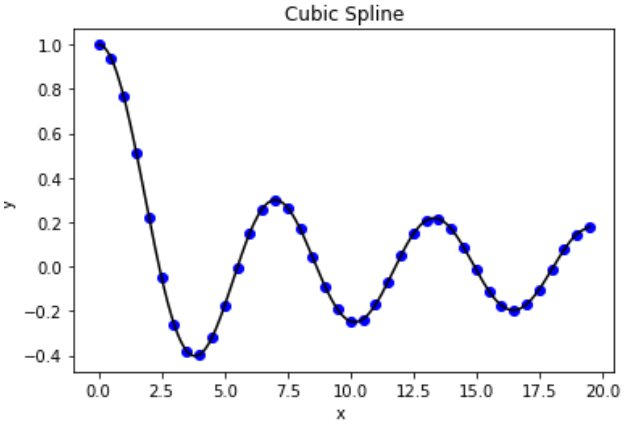
def T(a):
    n1=len(a);
    n2=len(a[0]);
    b = [[a[j][i] for j in range(n1)] for i in range(n2)];
    return b;

d=input("input file name (a.txt) : ");
a=read_array2D(d);
print(a);
x=a[0];
y=a[1];
Z=listSpline(x,y,5,0,0);
plt.title('Cubic Spline')
plt.xlabel('x ')
plt.ylabel('y ');
plt.plot(x,y,'bo',Z[0],Z[1],'k');
```

```
runfile('C:/Users/TURHAN/Desktop/NA_W8/cubic_spline.py', wdir='C:/Users/TURHAN/Desktop/NA_W8')

input file name (a.txt) : a.txt
[[0.0, 0.5, 1.0, 1.5, 2.0, 2.5, 3.0, 3.5, 4.0, 4.5, 5.0, 5.5, 6.0, 6.5, 7.0, 7.5, 8.0, 8.5, 9.0, 9.5, 10.0, 10.5, 11.0, 11.5, 12.0, 12.5, 13.0, 13.5, 14.0,
14.5, 15.0, 15.5, 16.0, 16.5, 17.0, 17.5, 18.0, 18.5, 19.0, 19.5], [1.0, 0.938469807240813, 0.7651976865579666, 0.5118276717359181,
0.22389077914123562, -0.048383776468198025, -0.2600519549019334, -0.38012773998726385, -0.39714980986384796, -
0.32054250898512326, -0.1775967713143408, -0.006843869417827212, 0.15064525725098935, 0.2600946055815953,
0.3000792705195197, 0.2663396578803943, 0.1716508071375183, 0.041939251842874115, -0.09033361118311638, -
0.19392874768828677, -0.24593576445249363, -0.23664819446450427, -0.17119030040887326, -0.06765394812043596,
0.04768931078146386, 0.14688405468945884, 0.20692610234959996, 0.21498916577372518, 0.17107347599270106,
```

```
0.08754486788129101, -0.014224472951772184, -0.10923065060064199, -0.17489907408784924, -0.19638069227528462, -
0.16985425053436704, -0.10311039483282625, -0.013355803592061321, 0.07716482976695192, 0.14662945902756847,
0.17885385052619363]]
```



6.8 LEAST SQUARE CURVE FITTING OF NONLINEAR EQUATIONS

If curve fitting to a function $y = f(a; x)$ with non-linear coefficients is required. Process will be a little bit different compare to equations with linear coefficients.

$y = f(a; x)$ is given and data x_i,f_i, i=0...n will be fit into the equation. In this equation coefficients are placed in a non-linear form. For example : $y = a_0(1 - e^{-a_1 x})$ nonlinear

$y = a_0(1 - e^{-x}) + a_1(1 - e^{-2x}) = a_0\phi_0(x) + a_1\phi_1(x)$ linear

Least square error formula was defined as

$$H(a_j^{(m)}; x_0, x_{1,}.., x_n) = \sum_{i=1}^{n} w(x_0, x_{1,}.., x_n)\left[y_i - f(a_j^{(m)}; x_0, x_{1,}.., x_n)\right]^2 \quad (6.8-1)$$

This equation should be solved by using one of the multidimensional optimization methods such as Nelder-Mead, Newton-Raphson, Steepest descent, Generic algorithms

sample : (input3.txt)

```
5.0 16.0
10.0 25.0
15.0 32.0
20.0 33.0
25.0 38.0
30.0 36.0
```

A nonlinear function to fit the data:

double pi=ai[0]*(1.0-Math.exp(-ai[1]*x));

In this function ai are coefficient of the equations and x is independent variable.

Program 6.8-1 Nelder-Mead one independent variable least square curve fitting java version

```
import java.io.*;
import javax.swing.*;

abstract class f_xj
{
  // single function  multi independent variable
  // a single value is returned  (
  // example f=x[0]+sin(x[1])
  // func(x) returns the value of f
  abstract public double func(double x[]);
}
```

```
class yy extends f_xj
{
        double xi[]; // independent variable data set
double yi[]; //dependent variable data set
double a[];  //fit function coefficient set
int nn;
        public yy(String filename,double ia[])
{
  //read the data to curvefit
  //get the data file and initial fit coefficient when class is defined
          xi=new double[500];
  yi=new double[500];
  int n=ia.length;
  a=new double[n];
  seta(ia);
  int i=-1;
  try{
          BufferedReader fin=new BufferedReader(new FileReader(filename));
  try {
  while(fin != null)
  {
  i++;
  xi[i]=Text.readDouble(fin);
  yi[i]=Text.readDouble(fin);
  }
  } catch(EOFException e_eof) {System.out.println("end of file"); }
  }catch(IOException e_io) {System.out.println("dosya bulunamadi"); }
nn=i;
a=ia;
}
public void seta(double ia[])
{
            //assign new fit coefficient set
            for(int ii=0;ii<nn;ii++)
      a[ii]=ia[ii];
        }
        public double[] geta()
{
          // return fit coefficient set
          return a;
        }
        double Ps(double x,double ai[])
{
//sample specific function
        // saturation pressure as a function of saturation temperature
a=ai;
double pi=ai[0]*(1.0-Math.exp(-ai[1]*x));
return pi; //kPa
}

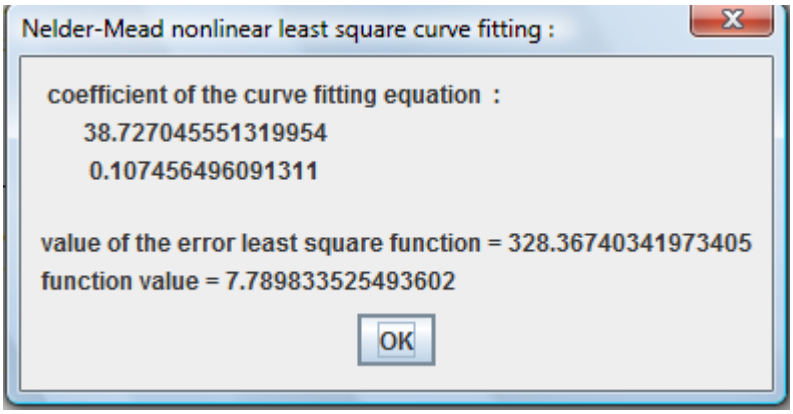
public double func(double ai[])
{
double ff=0;
double w;
double yy;
 for(int i=0;i<nn;i++)
          {w=Ps(xi[i],ai);
```

```
                yy=yi[i];
       //System.out.println("x="+xi[i]+"w="+w+"yy="+yy+"ff="+ff);
                w-=yy;
       ff+=w*w;
       //System.out.println("x="+xi[i]+"w="+w+"yy="+yy+"ff="+ff);
                }
     return ff;
             }
}

class zz extends yy
{     public zz(String filename,double ia[])
     {super(filename,ia);}
     public double func(double ai[])
     {
     return -super.func(ai);
             }
}
public class NA50
{  public static void main(String args[]) throws IOException
   {
   //String in_name=JOptionPane.showInputDialog(" enter name of the input file : ");
   String in_name="input3.txt";
   double a[];
   a=new double[2];
   a[0] = 30.0;
   a[1] = 1.0;
   double da[];
   da=new double[2];
   da[0] =1.0;
   da[1] =0.2;

   yy f=new yy(in_name,a);
   int i=0;
   for(double t=-100;t<101;t+=1)
   {System.out.println("t="+t+"ti="+f.xi[i]+"Pi="+f.yi[i]+"P="+f.Ps(t,a));i++;};
   System.out.println("f input = "+f.func(a));
   double p[]=NA41.nelder(f,a,da,1000,1e-20,1);
   String s1=" coefficient of the curve fitting equation  : \n"+Matrix.toStringT(p)+"\n";
   s1+="value of the error least square function = "+f.func(a)+"\n";
   s1+="function value = "+f.func(p);
   String s2="Nelder-Mead nonlinear least square curve fitting : ";
   JOptionPane.showMessageDialog(null,s1,s2,JOptionPane.PLAIN_MESSAGE);
   }

}
```



The problem can be solved by using different optimization programs. For example id Davidon-Fletcher-Powell algorithms is used:

Program 6.8-1a Nelder-Mead one independent variable least square curve fitting python version

```
from math import *
from f_xj import *
from matplotlib import pyplot as plt
import numpy as np


class y(f_xj):
    def __init__(self,filename,ia):
        self.name = 'yy',
        self.filename=filename
        path = filename
        fin=open(path,'r')
        n=len(ia)
        self.a=ia
        path = self.filename
        file1 = open(path,'r')
        file1.seek(0,2)  #jumps to the end
        endLocation=file1.tell()
        file1.seek(0) #jumps to the start
        nn=0
        while True:
            line=file1.readline()
            nn+=1
            if  file1.tell()==endLocation:
                break
        xi=[0.0 for i in range(nn)]
        yi=[0.0 for i in range(nn)]
        file1.seek(0) #jumps to the start
        for i in range(nn):
            line=file1.readline()
            ee=line.split()
            xi[i]=float(ee[0])
            yi[i]=float(ee[1])
            #print(xi[i]," ",yi[i])
        file1.close()
        self.xi=xi
        self.yi=yi
        self.nn=nn

    def Ps(self,x,ai):
        pi=ai[0]*(1-exp(-ai[1]*x))
        return pi
    def func(self,ai):
        ff=0
        for i in range(self.nn):
```

```
            w=self.Ps(self.xi[i],ai)
            yy=self.yi[i]
            w=w-yy
            ff=ff+w*w
        return ff

def  nelder(fnelder,a,da,maxiteration,tolerance,printlist):
    #print("a",a)
    #print("da",da)
    n=len(a)
    m=n+1
    x=[[0.0 for j in range(n)] for i in range(n+1)]
    p=[[0.0 for j in range(m)] for i in range(m)]
    s=""
    NDIMS = len(x)-1
    NPTS = len(x)
    FUNC = NDIMS
    ncalls = 0;
    for i in range(m):
        for j in range(n):
            if i==j:
                x[i][j]=a[j]+da[j]
                p[i][j]=x[i][j]
            else:
                x[i][j]=a[j]
                p[i][j]=x[i][j]
        p[i][FUNC] = fnelder.func(p[i])
        #print("x[",i,"]=",x[i])
        #print("p[",i,"]=",p[i])
    #print("p",p)
    #Inlet variable definitions
    #fnelder : abstract multivariable function f(x)
    #x : independent variable set of n+1 simplex elements
    #maxiteration : maximum iteration number
    #tolerance :

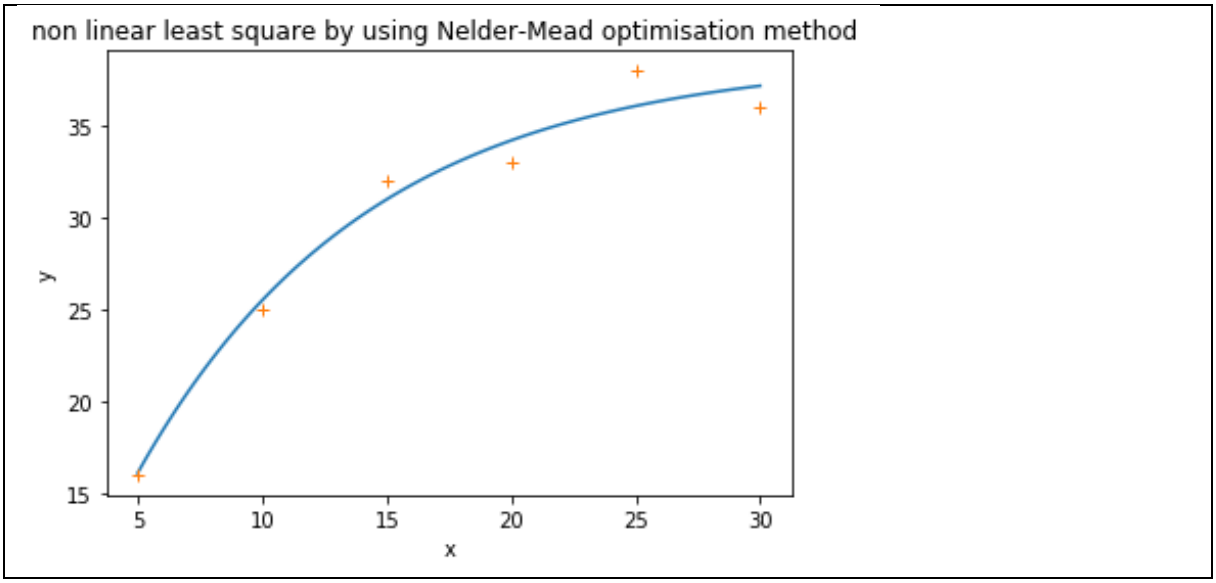
    ##### construct the starting simplex ///////////////////
    z=[0.0 for i in range(NDIMS)]
    best = 1.0E99;
    ################ calculate the first function values for the simplex ////////////////
    iter=0;
    for iter in range(1,maxiteration):
        ########  define lo, nhi, hi (low high next_to_high //////////////
        ilo=0
        ihi=0
        inhi = -1
        # -1 means missing
        flo = p[0][FUNC];
        fhi = flo
        #print("flo=",flo,"fhi=",fhi)
        #double pavg,sterr;
        for i in range(1,NPTS):
            if p[i][FUNC] < flo:
                flo=p[i][FUNC]
                ilo=i
            if  p[i][FUNC] > fhi:
                fhi=p[i][FUNC]
                ihi=i
            #print("i=",i,"flo=",flo,"fhi=",fhi,"ilo=",ilo,"ihi=",ihi,"pifunc=",p[i][FUNC])
        fnhi = flo
        inhi = ilo
        #print("fnhi=",fnhi,"inhi=",inhi)
        for i in range(NPTS):
            if (i != ihi) and (p[i][FUNC] > fnhi):
```

```
            fnhi=p[i][FUNC]
            inhi=i
    #//////// exit criteria //////////////
    if (iter % 4*NDIMS) == 0:
       #calculate the avarage (including maximum value)
       pavg=0
       for i in range(NPTS):
           pavg=pavg+p[i][FUNC]
       pavg=pavg/NPTS
       tot=0.0
       if printlist!=0:
          print(iter)
          for j in range(NDIMS):
             print(p[ilo][j]," ")
       for i in range(NPTS):
          tot=(p[i][FUNC]-pavg)*(p[i][FUNC]-pavg)
       sterr=sqrt(tot/NPTS)
       for j in range(NDIMS):
          z[j]=p[ilo][j]
       best = p[ilo][FUNC];
    # end of if iter%4*NDMS)==0
    #/// calculate avarage without maximum value //////
    ave=[0.0 for i in range(NDIMS)]
    for j in range(NDIMS):
       ave[j] = 0
    for i in range(NPTS):
       if i != ihi:
          for j in range(NDIMS):
             ave[j] = ave[j]+p[i][j];
    for j in range(NDIMS):
       ave[j]=ave[j]/(NPTS-1)
    #print("ave[",j,"]=",ave[j])
    #//////// reflect ////////////////
    #print("reflect")
    r=[0.0 for i in range(NDIMS)]
    for j in range(NDIMS):
       r[j] = 2.0*ave[j] - p[ihi][j]
    fr = fnelder.func(r)
    if (flo <= fr) and (fr < fnhi): #in zone: accept
       for j in range(NDIMS):
          p[ihi][j] = r[j]
       p[ihi][FUNC] = fr
       continue
    #///(////////// expand
    if fr < flo:
       #print("expand")
       e=[0.0 for i in range(NDIMS)]
       for j in range(NDIMS):
          e[j] = 3.0*ave[j] - 2.0*p[ihi][j];
       fe = fnelder.func(e);
       if fe < fr:
          for j in range(NDIMS):
             p[ihi][j] = e[j]
          p[ihi][FUNC] = fe
          continue
       else:
          for j in range(NDIMS):
             p[ihi][j] = r[j]
          p[ihi][FUNC] = fr
          continue
    #////////// shrink:
    if fr < fhi:
       #print("srink")
       c=[0.0 for i in range(NDIMS)]
```

```
            for j in range(NDIMS):
                c[j] = 1.5*ave[j] - 0.5*p[ihi][j]
            fc = fnelder.func(c);
            if fc <= fr:
                for j in range(NDIMS):
                    p[ihi][j] = c[j];
                p[ihi][FUNC] = fc
                continue
            else:   #////// shrink
                for i in range(NPTS):
                    if i != ilo:
                        for j in range(NDIMS):
                            p[i][j] = 0.5*p[ilo][j] + 0.5*p[i][j]
                        p[i][FUNC] = fnelder.func(p[i])
                continue
        #/////////////// contract
        if fr >= fhi:
            #print("contract")
            cc=[0.0 for i in range(NDIMS)]
            for j in range(NDIMS):
                cc[j] = 0.5*ave[j] + 0.5*p[ihi][j];
            fcc = fnelder.func(cc)
            if fcc < fhi:
                for j in range(NDIMS):
                    p[ihi][j] = cc[j];
                p[ihi][FUNC] = fcc
                continue
            else:
                for i in range(NPTS):
                    if i != ilo:
                        for j in range(NDIMS):
                            p[i][j] = 0.5*p[ilo][j] + 0.5*p[i][j]
                        p[i][FUNC] = fnelder.func(p[i]);
    return z

a=[30.0,1.0]
da=[1.0,0.2]
name="input3.txt"
f=y(name,a)
ai=nelder(f,a,da,100,1.0e-15,0)
print("ai = ",ai)
n1=100
xx1 =[0 for w in range(n1)]
yy1 =[0 for w in range(n1)]
dx=(30.0-5.0)/(n1-1)
for i in range(n1):
    xx1[i]=5.0+dx*i
    yy1[i]=f.Ps(xx1[i],ai)
plt.plot(xx1,yy1,f.xi,f.yi,"+")
plt.xlabel("x")
plt.ylabel("y")
plt.title("non linear least square by using Nelder-Mead optimisation method ")
plt.show()
```

```
runfile('E:/okul/SCO1/nelder_opt.py', wdir='E:/okul/SCO1')
Reloaded modules: f_xj
ai =  [38.727045551319954, 0.1074564960913112]
```



Program 6.8-2 Davison-Fletcher-Powell one independent variable nonlinear least square curve fitting

```
// Davidon-Fletcher-Powell optimization
// Method Referance : Numerical Analysis, 8th edition Richard L. Burden, J. Douglas Faires 3
// page 617
import java.util.*;
import java.awt.*;
import java.applet.Applet;
import java.awt.event.*;
import javax.swing.*;
import java.io.*;
     class fa extends f_xj
     { double xi[]; // independent variable data set
       double yi[]; //dependent variable data set
       double a[];  //fit function coefficient set
       int nn;
      public fa(String filename,double ia[])
     { //read the data to curvefit
       //get the data file and initial fit coefficient when class is defined
      xi=new double[500];
      yi=new double[500];
      int n=ia.length;
      a=new double[n];
      seta(ia);
      int i=-1;
      try { BufferedReader fin=new BufferedReader(new FileReader(filename));
      try {
      while(fin != null)
      { i++;
      xi[i]=Text.readDouble(fin);
      yi[i]=Text.readDouble(fin);
      }
      } catch(EOFException e_eof) {System.out.println("end of file"); }
      }catch(IOException e_io) {System.out.println("dosya bulunamadi"); }
     nn=i;
     a=ia;
```

```
    }

    public void seta(double ia[])
    {  //assign new fit coefficient set
       for(int ii=0;ii<nn;ii++)   a[ii]=ia[ii];}

    public double[] geta()
    {// return fit coefficient set
      return a;}
    double Ps(double x,double ai[])
    {
    //sample specific function
    // saturation pressure as a function of saturation temperature
    a=ai;
    double pi=ai[0]*(1.0-Math.exp(-ai[1]*x));
    //pi=Math.pow(10.0,pi);
    return pi; //kPa
    }

    public double func(double ai[])
    {
    double ff=0;
    double w;
    double yy;
     for(int i=0;i<nn;i++)
                {w=Ps(xi[i],ai);
                yy=yi[i];
      //System.out.println("x="+xi[i]+"w="+w+"yy="+yy+"ff="+ff);
                w-=yy;
      ff+=w*w;
      //System.out.println("x="+xi[i]+"w="+w+"yy="+yy+"ff="+ff);
                }
    return ff;
              }
}

public class NA125
{
public static double VT_V(double [] left)
{
//multiplys a vector transpose with a vector
int n=left.length;
double tot=0;
for(int i=0;i<n;i++)
  {
  tot+=left[i]*left[i];
  }
return tot;
}

public static double[][] V_VT(double [] left)
{
//multiplys a vector transpose with a vector
int n=left.length;
double aa[][]=new double[n][n];
for(int i=0;i<n;i++)
  {
```

```
  for(int j=0;j<n;j++)
    {
              aa[i][j]=left[i]*left[j];
              }
  }
return aa;
}

public static double  VT_X(double [] left,double [] right)
{
//multiplies a vector transpose with a vector
int n=left.length;
double tot=0;
for(int i=0;i<n;i++)
  {
  tot+=left[i]*right[i];
  }
return tot;
}

public static double norm(double v[])
{
// vector norm
double total=0;;
for(int i=0;i<v.length;i++)
  {total+=v[i]*v[i];}
return Math.sqrt(total);
}

public static double turev(f_xj f,double x[],int denklem_ref,int x_ref)
{
//
// verilen fonksiyonun ikinci türevlerinin matrisi
// fonksiyon f in denklem_ref sayili fonksiyonunun(sayilar 0 dan başlar)
// x[x_ref] değişkenine göre türevi (sayilar 0 dan başlar)
// df_denklem_ref(x)/d_x_ref
// bu metod newtond metodu içinde kullanilmak içindir.
double h0=0.256808;
int i,m;
int n=7;
double f1[];
f1=new double[x.length];
double f2[];
f2=new double[x.length];
double x1[];
x1=new double[x.length];
double x2[];
x2=new double[x.length];
for(i=0;i<x.length;i++)
{
x1[i]=x[i];
x2[i]=x[i];
}
//turev of a simple function
double T[][];
T=new double[n][n];
double h[];
```

```
h=new double[n];
//vector<double> h(n,0);
for(i=0;i<n;i++)
{
  h[i]=0;
  for(int j=0;j<n;j++)
    T[i][j]=0;
}
h[0]=h0;
double r=0.5;
for( i=1;i<n;i++)
{
h[i]=h0*Math.pow(r,i);
}

for(i=0;i<n;i++)
{
x1[x_ref]+=h[i];
x2[x_ref]-=h[i]*1.0094847;
f1=df(f,x1);
f2=df(f,x2);
T[i][0]=( f1[denklem_ref] - f2[denklem_ref])/(2.0*h[i]);
x1[x_ref]=x[x_ref];
x2[x_ref]=x[x_ref];
}
for(m=1;m<n;m++)
{
  for(i=0;i<n-m;i++)
  {
  T[i][m]=(h[i]*h[i]*T[i+1][m-1] - h[i+m]*h[i+m]*T[i][m-1])/(h[i]*h[i]
  - h[i+m]*h[i+m]);
  }
}
double xx=T[0][n-1];
return xx;
}
public static double[] df(f_xj f,double xxi[])
{
//Gredyen vektörü
// f(x0,x1,x2,...xn) founksiyonunun türev vektörü
// [df/dx0, df/dx1,df/dx2,.....,df/dxn]
// df/dxj  j=0...x.length
//
// bu fonksiyon bilgisayar tarafindan hesaplanmaktadir
// kullanici tarafindan tanimlanan f(x) fonksiyonunun
// sayisal olarak alinan türevidir.

//
double a[]=new double[xxi.length];
for(int i=0;i<xxi.length;i++)
   {
   a[i]=turev(f,xxi,i);
   }
return a;
}
public static double turev(f_xj f,double xxi[],int x_ref)
{
```

```
// df/dxj
double h0=0.256808;
int i,m;
int n=7;
double x1[];
x1=new double[xxi.length];
double x2[];
x2=new double[xxi.length];
for(i=0;i<xxi.length;i++)
{
x1[i]=xxi[i];
x2[i]=xxi[i];
}
//turev of a simple function
double T[][];
T=new double[n][n];
double h[];
h=new double[n];
for(i=0;i<n;i++)
{
  h[i]=0;
  for(int j=0;j<n;j++)
    T[i][j]=0;
}
h[0]=h0;
double r=0.5;
for( i=1;i<n;i++)
{
//h[i]=h0*Math.pow(r,i);
h0/=2.0;
h[i]=h0;
}
//first turev (difference formula)
for(i=0;i<n;i++)
{
x1[x_ref]+=h[i];
x2[x_ref]-=h[i];
T[i][0]=(f.func(x1)-f.func(x2))/(2.0*h[i]);
x1[x_ref]=xxi[x_ref];
x2[x_ref]=xxi[x_ref];
}

for(m=1;m<n;m++)
{
  for(i=0;i<n-m;i++)
  {
  T[i][m]=(h[i]*h[i]*T[i+1][m-1] -
  h[i+m]*h[i+m]*T[i][m-1])/(h[i]*h[i] - h[i+m]*h[i+m]);
  }
}
double xx=T[0][n-1];
return xx;
}
public static double[] turev(f_xj f_deriv,double x[])
{
// df/dxj  j=0...x.length
// This method calculates turev of a function with more than one independent variable.
```

```
// Accuracy of method can be adjusted by changing variables h0 and n
// function input should be in the form given in abstract class
// f_xj,j=0...x.length = df/dx(x_ref)
double a[]=new double[x.length];
for(int x_ref=0;x_ref<x.length;x_ref++)
  {
  a[x_ref]=turev(f_deriv,x,x_ref);
  }
return a;
}

public static double SIGN(double a,double b) {return (b > 0.0 ? Math.abs(a) : -Math.abs(a));}
public static double MAX(double a,double b) {return (a > b ? a : b);}


public static double linmin(f_xj f,double p[],double xi[])
{
        double tol=2.0e-4;
   int n=p.length;
   double pcom[],xicom[];
        int j;
        double xx,xmin,bx,ax;
        f1dim f1=new f1dim(f,p,xi);
        ax=0.0;
        xx=1.0;
        bx=2.0;
        double aa[]=mnbrak(f1,ax,xx,bx);
        ax=aa[0];
        xx=aa[1];
        bx=aa[2];
        xmin=brent(f1,ax,xx,bx,tol);
        return xmin;
}

public static double[][] linminiter(f_xj f,double p[],double xi[])
{   // yeni iterasyon noktasini verir
   int n=p.length;
   double xmin=linmin(f,p,xi);
   double aa[][]=new double[2][n];
        for (int j=0;j<n;j++)
        {   xi[j] *= xmin;
                p[j] += xi[j];
                aa[0][j]=p[j];
                aa[1][j]=xi[j];
        }
   return aa;
}
public  static double[] brentf(f_x f,double ax,double bx,double cx,double tol)
{
// ax,bx,cx üç adet tahmin noktasi (bxax ile cx arasinda yer almalidir)
// f : fonksiyon (f_x abstract sinifinda tanimlanmiş)
// tol tolerans
int ITMAX=100;
double CGOLD=(3.0-Math.sqrt(5))/2.0; //altin oran
double ZEPS=1.0e-10;
double xmin;
double aa[]=new double[2];
```

```
// SHFT(double a,double b,double c,d) (a)=(b);(b)=(c);(c)=(d);
        int iter;
        double a,b,d,etemp,fu,fv,fw,fx,p,q,r,tol1,tol2,u,v,w,x,xm;
        double e=0.0;
        d=0.0;
        a=((ax < cx) ? ax : cx);
        b=((ax > cx) ? ax : cx);
        x=w=v=bx;
        fw=fv=fx=f.func(x);
        for (iter=1;iter<=ITMAX;iter++) {
                xm=0.5*(a+b);
                tol2=2.0*(tol1=tol*Math.abs(x)+ZEPS);
                if (Math.abs(x-xm) <= (tol2-0.5*(b-a))) {
                        xmin=x;
            aa[0]=xmin;
            aa[1]=fx;
            return aa;
                }
                if (Math.abs(e) > tol1) {
                        r=(x-w)*(fx-fv);
                        q=(x-v)*(fx-fw);
                        p=(x-v)*q-(x-w)*r;
                        q=2.0*(q-r);
                        if (q > 0.0) p = -p;
                        q=Math.abs(q);
                        etemp=e;
                        e=d;
                        if (Math.abs(p) >= Math.abs(0.5*q*etemp) || p <= q*(a-x) || p >= q*(b-x))
                                d=CGOLD*(e=(x >= xm ? a-x : b-x));
                        else {
                                d=p/q;
                                u=x+d;
                                if (u-a < tol2 || b-u < tol2)
                                        d=SIGN(tol1,xm-x);
                        }
                } else {
                        d=CGOLD*(e=(x >= xm ? a-x : b-x));
                }
                u=(Math.abs(d) >= tol1 ? x+d : x+SIGN(tol1,d));
                fu=f.func(u);
                if (fu <= fx) {
                        if (u >= x) a=x; else b=x;
                        {v=w;w=x;x=u;}
                        {fv=fw;fw=fx;fx=fu;}
                } else {
                        if (u < x) a=u; else b=u;
                        if (fu <= fw || w == x) {
                                v=w;
                                w=u;
                                fv=fw;
                                fw=fu;
                        } else if (fu <= fv || v == x || v == w) {
                                v=u;
                                fv=fu;
                        }
                }
        }
```

```
            System.out.println("BRENT metodunda maksimum iterasyon sayisi aşildi");
            xmin=x;
            //minumum değer a[0] ile fonksiyon değeri a[1] ile geridöndürülmektedir
            aa[0]=xmin;
            aa[1]=fx;
            return aa;
}
public  static double[] mnbrak(f_x f,double ax,double dx)
{
// dx ikinci derece polinom için gerekli iki ek noktayi oluşturmak için
// ana noktadan seçilen uzaklik
return mnbrak(f,ax,(ax-dx),(ax+dx));
}

public  static double[] mnbrak(f_x f,double ax)
{
// dx ikinci derece polinom için gerekli iki ek noktayi oluşturmak için
// ana noktadan seçiln uzaklik
return mnbrak(f,ax,(ax-0.1),(ax+0.1));
}

public  static double[] mnbrak(f_x f,double ax,double bx,double cx)
{
// ax,bx,cx üç adet tahmin noktasi verildiğinde içinde minimum bulunan ax,bx,cx üç adetnoktayi bize arama
// sonucu bulur
// f : fonksiyon (f_x abstract sinifinda tanimlanmiş)
// tol tolerans
int ITMAX=100;
double xmin;
double GOLD=(Math.sqrt(5.0)+1.0)/2.0;;
double GLIMIT=100.0;
double TINY=1.0e-20;
double fa,fb,fc;
fa=0;
fb=0;
fc=0;
            double ulim,u,r,q,fu,dum;
            double aa[]=new double[3];
            fa=f.func(ax);
            fb=f.func(bx);
            if (fb > fa) {
                        //SHFT(dum,*ax,*bx,dum)
                        {dum=ax;ax=bx;bx=dum;}
                        //SHFT(dum,*fb,*fa,dum)
                        {dum=fb;fb=fa;fa=dum;}
            }
            cx=(bx)+GOLD*(bx-ax);
            fc=f.func(cx);
                        aa[0]=ax;
                        aa[1]=bx;
                        aa[2]=cx;
            while (fb > fc) {
                        r=(bx-ax)*(fb-fc);
                        q=(bx-cx)*(fb-fa);
                        u=(bx)-((bx-cx)*q-(bx-ax)*r)/
                                    (2.0*SIGN(MAX(Math.abs(q-r),TINY),q-r));
                        ulim=(bx)+GLIMIT*(cx-bx);
```

```
                        if ((bx-u)*(u-cx) > 0.0) {
                                fu=f.func(u);
                                if (fu < fc) {
                                        ax=bx;
                                        bx=u;
                                        fa=fb;
                                        fb=fu;
                                        return aa;
                                } else if (fu > fb) {
                                        cx=u;
                                        fc=fu;
                                        return aa;
                                }
                                u=(cx)+GOLD*(cx-bx);
                                fu=f.func(u);
                        } else if ((cx-u)*(u-ulim) > 0.0) {
                                fu=f.func(u);
                                if (fu < fc) {
                                        //SHFT(bx,cx,u,cx+GOLD*(cx-bx))
                                        {bx=cx;cx=u;u=cx+GOLD*(cx-bx);}
                                        //SHFT(fb,fc,fu,f.func(u))
                                        {fb=fc;fc=fu;fu=f.func(u);}
                                }
                        } else if ((u-ulim)*(ulim-cx) >= 0.0) {
                                u=ulim;
                                fu=f.func(u);
                        } else {
                                u=cx+GOLD*(cx-bx);
                                fu=f.func(u);
                        }
                        //SHFT(ax,bx,cx,u)
                        {ax=bx;bx=cx;cx=u;}
                        //SHFT(fa,fb,fc,fu)
                        {fa=fb;fb=fc;fc=fu;}
                        aa[0]=ax;
                        aa[1]=bx;
                        aa[2]=cx;
                }
                        return aa;
}
public  static double brent(f_x f,double ax,double bx,double cx,double tol)
{
 double aa[]=brentf(f,ax,bx,cx,tol);
 return aa[0];
}

public static double[][] I(int n)
{
//unit matrix
double b[][];
b=new double[n][n];
for(int i=0;i<n;i++)
   b[i][i]=1.0;
return b;
}
// Davidon - Fletcher-Powell metodu
//_____________________________________________________________
```

```
public static double[] davidon_fletcher_powell( f_xj f,double x[])
{
int i,j,k;
int nmax=500;
double gamma=1.0;
double tolerance=1.0e-12 ;
int n=x.length;
double alpha;
double dx[]=new double[n];
double g[] =new double[n];
double ge[]=new double[n];
double dg[]=new double[n];
double d[] =new double[n];
double r[] =new double[n];
double rdg,dxdg;
double beta;
double alpha_d;
double Q[][]=I(n);
double Q1[][]=new double[n][n];
double Q2[][]=new double[n][n];
int nn=15;
i=0;
j=0;k=0;
g=turev(f,x);
while( j++ < nmax && gamma > tolerance )
{
   for(int ii=0;ii<n;ii++)
   {
    d[ii]=0.0;
    for(int kk=0;kk<n;kk++)
                {d[ii]=d[ii]-Q[ii][kk]*g[kk];}
   }
   System.out.println("j="+j+"d="+Matrix.toString(d));
   alpha=linmin(f,x,d);
   System.out.println("alfa="+alpha);
   if(alpha==0) break;
   for(i=0;i<n;i++){dx[i]=alpha*d[i];x[i]+=dx[i];ge[i]=g[i];}
   System.out.println("j="+j+"dx=\n"+Matrix.toString(dx));
   System.out.println("j="+j+"x=\n"+Matrix.toString(x));
   g=turev(f,x);
   System.out.println("j="+j+"g=\n"+Matrix.toString(g));
   for(i=0;i<n;i++){dg[i]=g[i]-ge[i];}
   gamma=norm(g);
   for(int ii=0;ii<n;ii++)
   {r[ii]=0.0;
                for(int kk=0;kk<n;kk++)
                {r[ii]+=Q[ii][kk]*dg[kk];}
   }
   rdg=VT_X(r,dg);
   dxdg=VT_X(dx,dg);
   Q1=V_VT(r);
   Q2=V_VT(dx);
   for(int ii=0;ii<n;ii++)
     for(int jj=0;jj<n;jj++)
       Q[ii][jj]=Q[ii][jj]-Q1[ii][jj]/rdg+Q2[ii][jj]/dxdg;
   gamma=norm(dx);
   System.out.println("j="+j+"Q=\n"+Matrix.toString(Q));
```

```
j++;k++;
if(k>=nn) {k=0;Q=I(n);}
}
if(j >= nmax) JOptionPane.showMessageDialog(null,"Uyari maximum iterasyon sayisi aşildi \n"+
  " çözüm geçerli olmiyabilir","MAKSİMUM ITERASYON SAYISI UYARISI",JOptionPane.WARNING_MESSAGE);
return x;
}
public static double[] davidon_fletcher_powell_V1( f_xj f,double x[])
{
int i,j,k;
int nmax=500;
double gamma=1.0;
double tolerance=1.0e-12 ;
int n=x.length;
double alpha;
double dx[]=new double[n];
double g[] =new double[n];
double ge[]=new double[n];
double dg[]=new double[n];
double d[] =new double[n];
double r[] =new double[n];
double rdg,dxdg;
double beta;
double alpha_d;
double Q[][]=I(n);
double Q1[][]=new double[n][n];
double Q2[][]=new double[n][n];
int nn=15;
i=0;
j=0;k=0;
g=turev(f,x);
while( j++ < nmax && gamma > tolerance )
{

   for(int ii=0;ii<n;ii++)
   {
    d[ii]=0.0;
    for(int kk=0;kk<n;kk++)
                {d[ii]=d[ii]-Q[ii][kk]*g[kk];}
   }
   alpha=linmin(f,x,d);
   if(alpha==0) break;
   for(i=0;i<n;i++){dx[i]=alpha*d[i];x[i]+=dx[i];ge[i]=g[i];}
   g=turev(f,x);
   for(i=0;i<n;i++){dg[i]=g[i]-ge[i];}
   gamma=norm(g);
   for(int ii=0;ii<n;ii++)
   {  r[ii]=0.0;
                for(int kk=0;kk<n;kk++)
                {r[ii]+=Q[ii][kk]*dg[kk];}
   }
   rdg=VT_X(r,dg);
   dxdg=VT_X(dx,dg);
   beta=dxdg/rdg;
   Q1=V_VT(r);
   Q2=V_VT(dx);
   for(int ii=0;ii<n;ii++)
```

```
     for(int jj=0;jj<n;jj++)
       Q[ii][jj]=beta*(Q[ii][jj]-Q1[ii][jj]/rdg)+Q2[ii][jj]/dxdg;
   gamma=norm(dx);
j++;k++;
if(k>=nn) {k=0;Q=I(n);}
}
if(j >= nmax) JOptionPane.showMessageDialog(null,"Uyari maximum iterasyon sayisi aşildi \n"+
  " çözüm geçerli olmiyabilir","MAKSİMUM ITERASYON SAYISI UYARISI",JOptionPane.WARNING_MESSAGE);
return x;
}
            public static double[] verigir(String s)
    {
    String s1=JOptionPane.showInputDialog(s);
    StringTokenizer token=new StringTokenizer(s1);
    int n=token.countTokens()-1;
    int m=n+1;
    double a[]=new double[m];
    int j=0;
    while(token.hasMoreTokens())
    {
    Double ax=new Double(token.nextToken());
    a[j++]=ax.doubleValue();
    }
    return a;
    }
            public static void main (String args[])
    {
            String s="write root estimates with one space in between: ";
    double [] x0=verigir(s);
    fa f_x=new fa("a.txt",x0);
    double [] r1= davidon_fletcher_powell(f_x,x0);
    double [] r2= davidon_fletcher_powell_V1(f_x,x0);
    s=" solution set DFP : \n"+Matrix.toStringT(r1);
    s+="solution set corrected DFP : \n"+Matrix.toStringT(r2);
    String s2="Davidon - Fletcher - Powell multivariable optimization:";
    JOptionPane.showMessageDialog(null,s,s2,JOptionPane.PLAIN_MESSAGE);
    System.exit(0);
    }
}
```

A real engineering problem will be defined here to show how actual need would arise and solution

As a research project, drying of sliced apple by a heat pump is investigated. The data includes t time(hour), slice thickness h(mm) and P is the retaining moisture in the apple slice(kg moisture/kg fresh apple moisture). When the previous researchs are investigated, it is found that the following equations are used.

$a=a_0+a_1h$

$k=a_2+a_3h$

$P=ae^{(-kt)}$

$$\boldsymbol{P = (a_0 + a_1 h)e^{(a_2+a_3 h)}}$$

In order to compare we would like to fit our data into the same equations.

The data (T,h,P)

0	6	1.000	3	8	0.332	4	10	0.328
0.5	6	0.736	3.5	8	0.276	4.5	10	0.287
1	6	0.603	4	8	0.229	5	10	0.251

1.5	6	0.482	4.5	8	0.192	5.5	10	0.218
2	6	0.390	5	8	0.160	6	10	0.185
2.5	6	0.307	5.5	8	0.128	6.5	10	0.161
3	6	0.245	6	8	0.102	7	10	0.138
3.5	6	0.192	6.5	8	0.083	7.5	10	0.117
4	6	0.143	7	8	0.067	8	10	0.101
4.5	6	0.108	7.5	8	0.055	8.5	10	0.086
5	6	0.081	8	8	0.046	9	10	0.071
5.5	6	0.059	0	10	1.000	9.5	10	0.063
6	6	0.044	0.5	10	0.824	10	10	0.056
0	8	1.000	1	10	0.718	10.5	10	0.051
0.5	8	0.765	1.5	10	0.618	11	10	0.045
1	8	0.649	2	10	0.542			
1.5	8	0.541	2.5	10	0.474			
2	8	0.456	3	10	0.424			
2.5	8	0.382	3.5	10	0.372			

Let us fit data into the above equation:

Program 6.8-3 Nelder-Mead one independent variable nonlinear least square curve fitting, apple slice drying problem

```
import java.io.*;
import javax.swing.*;
    class fxa extends f_xj
    {
            double xi[][]; // m independent variable data set
    double yi[]; //dependent variable data set
    double a[];  //fit function coefficient set
    int nn;//number of data
    int m; //number of idependent variables xi
    double xmax;
            public fxa(String filename,double ia[])
    {
     //read the data to curvefit
     //last column is treated as independent data set
     //get the data file and initial fit coefficient when class is defined
     int n=ia.length;
     a=new double[n];
     seta(ia);
     xmax=-1.0e99;
     double b[][]=Text.readDoubleT(filename);
     m=b.length-1;
     yi=b[m];
     nn=yi.length;
     //System.out.println("m="+m);
     xi=new double[m][nn];
     for(int i=0;i<nn;i++) {
            xi[0][i]=b[0][i];if(xi[0][i]>xmax) {xmax=xi[0][i];}
     xi[1][i]=b[1][i];
            yi[i]=b[m][i];
     }
     //System.out.println("b=\n"+Matrix.toString(Matrix.T(b)));
     System.out.println("xi=\n"+Matrix.toString(xi));
     System.out.println("yi=\n"+Matrix.toString(yi));
    nn=yi.length;
```

```
    a=ia;
    }

    public void seta(double ia[])
    {
                //assign new fit coefficient set
                for(int ii=0;ii<nn;ii++)
          a[ii]=ia[ii];
             }

             public double[] geta()
    {
               // return fit coefficient set
               return a;
             }
     double Ps(double x[],double ai[])
    {
    // sample specific function m dimensional independent variable
    // independent variables : x[0],x[1]...x[m-1]
    // x[0] t
    // x[1] kalinlik
    // ai[0] a
    //
    a=ai;
    double t=x[0];
    double h=x[1];
    double a=ai[0]+ai[1]*h;
    double k=ai[2]+ai[3]*h;
    double pi=a*Math.exp(-k*t);
    return pi; //kPa
    }

    public double func(double ai[])
    {
    double ff=0;
    double w;
    double yy;
    double xx[]=new double[xi.length];
     for(int i=0;i<nn;i++)
               {   for(int j=0;j<m;j++) {xx[j]=xi[j][i];}
                                    w=Ps(xx,ai);
               yy=yi[i];
      //System.out.println("x="+xi[i]+"w="+w+"yy="+yy+"ff="+ff);
               w-=yy;
      ff+=w*w;
      //System.out.println("x="+xi[i]+"w="+w+"yy="+yy+"ff="+ff);
               }
    return ff;
               }
}

public class NA126
{//2. Nelder ve Mead method
//_____________________________________________________________
public static double[] nelder(f_xj fnelder,double a[],double da[],int maxiteration,double tolerance,int printlist)
{
double x[][]=new double[a.length+1][a.length];
```

```
for(int i=0;i<x.length;i++)
  {for(int j=0;j<x[0].length;j++)
   {if(i==j){x[i][j]=a[j]+da[j];}
    else    {x[i][j]=a[j]; }
   }
  }
  //  Nelder mead multidimensional simplex minimization method
  //  Reference : Nelder & Mead 1965 Computer J, v.7, 308-313.
       int NDIMS = x.length-1;
       int NPTS = x.length;
       int FUNC = NDIMS;
       int ncalls = 0;
         ////// starting simplex ///////////////////
         double p[][]=new double[NPTS][NPTS]; // [row][col] = [whichvx][coord,FUNC]
         double z[]=new double[NDIMS];
         double best = 1E99;
         ///////////////// calculate function values /////////////////
         for (int i=0; i<NPTS; i++)
         {
          for (int j=0; j<NDIMS; j++)
             {p[i][j] = x[i][j];}
          p[i][NDIMS] = fnelder.func(p[i]);
         }
         int iter=0;
         for (iter=1; iter<maxiteration; iter++)
         {
            ///////////  define lo, nhi, hi points ///////////////
            int  ilo=0, ihi=0, inhi = -1; // -1 means missing
            double flo = p[0][FUNC];
            double fhi = flo;
            double pavg,sterr;
            for (int i=1; i<NPTS; i++)
            {
              if (p[i][FUNC] < flo)
                {flo=p[i][FUNC]; ilo=i;}
              if (p[i][FUNC] > fhi)
                {fhi=p[i][FUNC]; ihi=i;}
            }
            double fnhi = flo;
            inhi = ilo;
            for (int i=0; i<NPTS; i++)
              if ((i != ihi) && (p[i][FUNC] > fnhi))
                {fnhi=p[i][FUNC]; inhi=i;}
            //////////exit criterieas///////////////
            if ((iter % 4*NDIMS) == 0)
            {
               pavg=0;
               for(int i=0;i<NPTS;i++)
                 pavg+=p[i][FUNC];
               pavg/=NPTS;
               double tot=0;
               if(printlist!=0)
               {  System.out.print(iter);
                  for (int j=0; j<=NDIMS; j++)
                     { System.out.print(p[ilo][j]+" ");}
                  System.out.println("");
               }
```

```
    for(int i=0;i<NPTS;i++)
    { tot=(p[i][FUNC]-pavg)*(p[i][FUNC]-pavg);}
    sterr=Math.sqrt(tot/NPTS);
    //if(sterr < tolerance)
      { for (int j=0; j<NDIMS; j++)
          { z[j]=p[ilo][j];}
        //break;
      }
    best = p[ilo][FUNC];
}

///// calculate ave[] vector except the largest value //////

double ave[] = new double[NDIMS];
for (int j=0; j<NDIMS; j++)
  ave[j] = 0;
for (int i=0; i<NPTS; i++)
  if (i != ihi)
    for (int j=0; j<NDIMS; j++)
      ave[j] += p[i][j];
for (int j=0; j<NDIMS; j++)
   ave[j] /= (NPTS-1);
//////// reflect ////////////////
double r[] = new double[NDIMS];
for (int j=0; j<NDIMS; j++)
  r[j] = 2*ave[j] - p[ihi][j];
double fr = fnelder.func(r);

if ((flo <= fr) && (fr < fnhi))  // in zone: accept
{
    for (int j=0; j<NDIMS; j++)
      p[ihi][j] = r[j];
    p[ihi][FUNC] = fr;
    continue;
}

if (fr < flo)  //// expand, else accept
{
    double e[] = new double[NDIMS];
    for (int j=0; j<NDIMS; j++)
      e[j] = 3*ave[j] - 2*p[ihi][j];
    double fe = fnelder.func(e);
    if (fe < fr)
    {
      for (int j=0; j<NDIMS; j++)
        p[ihi][j] = e[j];
      p[ihi][FUNC] = fe;
      continue;
    }
    else
    {
      for (int j=0; j<NDIMS; j++)
        p[ihi][j] = r[j];
      p[ihi][FUNC] = fr;
      continue;
    }
}
```

```
            ////////////// squeese:
            if (fr < fhi)
            {
               double c[] = new double[NDIMS];
               for (int j=0; j<NDIMS; j++)
                 c[j] = 1.5*ave[j] - 0.5*p[ihi][j];
               double fc = fnelder.func(c);
               if (fc <= fr)
               {
                  for (int j=0; j<NDIMS; j++)
                    p[ihi][j] = c[j];
                  p[ihi][FUNC] = fc;
                  continue;
               }
               else   /////// squeese
               {
                  for (int i=0; i<NPTS; i++)
                    if (i != ilo)
                    {
                       for (int j=0; j<NDIMS; j++)
                         p[i][j] = 0.5*p[ilo][j] + 0.5*p[i][j];
                       p[i][FUNC] = fnelder.func(p[i]);
                    }
                  continue;
               }
            }

            if (fr >= fhi)   ///
            {
               double cc[] = new double[NDIMS];
               for (int j=0; j<NDIMS; j++)
                 cc[j] = 0.5*ave[j] + 0.5*p[ihi][j];
               double fcc = fnelder.func(cc);
               if (fcc < fhi)
               {
                  for (int j=0; j<NDIMS; j++)
                    p[ihi][j] = cc[j];
                  p[ihi][FUNC] = fcc;
                  continue;
               }
               else    /////////
               {
                  for (int i=0; i<NPTS; i++)
                    if (i != ilo)
                    {
                       for (int j=0; j<NDIMS; j++)
                         p[i][j] = 0.5*p[ilo][j] + 0.5*p[i][j];
                       p[i][FUNC] = fnelder.func(p[i]);
                    }
               }
            }
         }
      return z;
}
     public static double[] nelder(f_xj fnelder,double a[],double da[],double tolerance)
     {return nelder(fnelder,a,da,300,tolerance,0);}
```

```
  public static double[] nelder(f_xj fnelder,double a[],double da[])
  {return nelder(fnelder,a,da,500,1.0e-20,0);}

  public static double[] nelder(f_xj fnelder,double x0[])
  {  double [] dx0=new double[x0.length];
     for(int i=0;i<x0.length;i++) dx0[i]=0.2*x0[i];
     return nelder(fnelder,x0,dx0,500,1.0e-15,0);
  }

  public static double[] nelder(f_xj fnelder,double x0[],double tolerance)
  {
    double [] dx0=new double[x0.length];
    for(int i=0;i<x0.length;i++) dx0[i]=0.2*x0[i];
              return nelder(fnelder,x0,dx0,tolerance);
  }

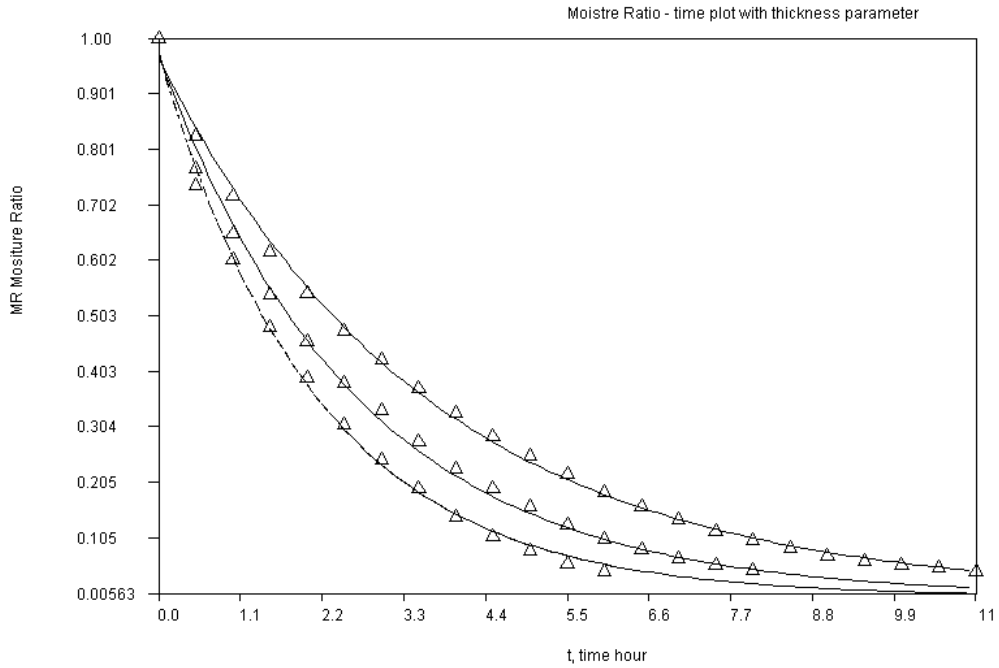
  public static double[] nelder(f_xj fnelder,double x0[],int maxiteration,double tolerance)
  {
    double [] dx0=new double[x0.length];
    for(int i=0;i<x0.length;i++) dx0[i]=0.2*x0[i];
    return nelder(fnelder,x0,dx0,maxiteration,tolerance,0);
  }

public static void main(String args[]) throws IOException
{
//String in_name=JOptionPane.showInputDialog(" veri dosyasinin isminigiriniz : ");
String in_name="kalinlik1.txt";
double a[];
double x[][],y[]; //uydurulan veri
double xu1[][],yu1[]; //uydurulan veri
double xu2[][],yu2[];
double xu3[][],yu3[];
a=new double[4];
a[0] = 1.0;
a[1] = 1.0;
a[2] = 1.0;
a[3] = 1.0;
fxa f=new fxa(in_name,a);
int i=0,j=0;
double p[]=nelder(f,a);
f=new fxa(in_name,p);
double tmax=f.xmax;
double dt=0.1; //ZAMAN ARALIĞI
int n=(int)(tmax/dt);
System.out.println("length="+f.nn+"tmax="+tmax+"n="+n);
xu1=new double[f.m][n];
yu1=new double[n];
xu2=new double[f.m][n];
yu2=new double[n];
xu3=new double[f.m][n];
yu3=new double[n];
double z[]=new double[2];
for(i=0;i<n;i++)
{
xu1[0][i]=dt*i;
xu1[1][i]=6;
z[0]=xu1[0][i];
```

```
    z[1]=xu1[1][i];
    yu1[i]=f.Ps(z,p);
    System.out.println("x0="+xu1[0][i]+"x1="+xu1[1][i]+"y="+yu1[i]);
    }
    for(i=0;i<n;i++)
    {
    xu2[0][i]=dt*i;
    xu2[1][i]=8;
    z[0]=xu2[0][i];
    z[1]=xu2[1][i];
    yu2[i]=f.Ps(z,p);
    System.out.println("x0="+xu2[0][i]+"x1="+xu2[1][i]+"y="+yu2[i]);
    }
    for(i=0;i<n;i++)
    {
    xu3[0][i]=dt*i;
    xu3[1][i]=10;
    z[0]=xu3[0][i];
    z[1]=xu3[1][i];
    yu3[i]=f.Ps(z,p);
    System.out.println("x0="+xu3[0][i]+"x1="+xu3[1][i]+"y="+yu3[i]);
    }
    Plot pp=new Plot(f.xi[0],f.yi);
    pp.setPlabel("Moistre Ratio - time plot with thickness parameter");
    pp.setXlabel("t, time hour");
    pp.setYlabel("MR Mositure Ratio ");
    pp.setPlotType(0,24);
    pp.addData(xu1[0],yu1);
    pp.setPlotType(1,0);
    pp.addData(xu2[0],yu2);
    pp.setPlotType(1,1);
    pp.addData(xu3[0],yu3);
    pp.setPlotType(1,2);

    //pp.addData(x,y);
    //pp.setPlotType(1,24);// üçgen olarak çiz
    pp.plot();

    String s1=" Optimization value  : \n"+Matrix.toStringT(p)+"\n";
    s1+="Value of the original function = "+f.func(a)+"\n";
    s1+="Function vlue = "+f.func(p);
    s1+="standard deviation= "+Math.sqrt(f.func(p)/(f.nn-1));
    String s2="Nelder-Mead non linear least square curve fitting : ";
    System.out.println(s1);
    }
}
```



Resulting curve :

a_0= *0.990155880325089*

a_1= *-0.002517036051923*

a_2= *0.765464843543500*

a_3= **-0.048774936274541**

In the next example harris' hawks optimisation is used to find non-linear curve fitting function.

Data:

```
5.0 16.0
10.0 25.0
15.0 32.0
20.0 33.0
25.0 38.0
30.0 36.0
```

Function to curvefit:

$f(x) = a_0[1 - \exp(-a_1 x)]$

Program 6.8-4 Harris' hawks one independent variable nonlinear least square curve fitting

```
 import java.util.*;
import java.io.*;
    class yy implements if_xj
    {
            double xi[]; // independent variable data set
    double yi[]; //dependent variable data set
    double a[];  //fit function coefficient set
    int nn;
            public yy(String filename,int n)
    {
     //read the data to curvefit
     //get the data file and initial fit coefficient when class is defined
              xi=new double[5000];
     yi=new double[5000];
     a=new double[n];
     //seta(ia);
     int i=-1;
     try{
             BufferedReader fin=new BufferedReader(new FileReader(filename));
     try {
```

```
       while(fin != null)
       {
       i++;
       xi[i]=Text.readDouble(fin);
       yi[i]=Text.readDouble(fin);
       }
       } catch(EOFException e_eof) {System.out.println("end of file"); }
       }catch(IOException e_io) {System.out.println("dosya bulunamadi"); }
     nn=i;
     }
     public void seta(double ia[])
     {
                    //assign new fit coefficient set
                    for(int ii=0;ii<nn;ii++)
            a[ii]=ia[ii];
                }
                public double[] geta()
     {
                  // return fit coefficient set
                  return a;
                }
                double Ps(double x,double ai[])
     {
     //sample specific function
                // saturation pressure as a function of saturation temperature
     a=ai;
     double pi=ai[0]*(1.0-Math.exp(-ai[1]*x));
     return pi; //kPa
     }

     public double func(double ai[])
     {
     double ff=0;
     double w;
     double yy;
      for(int i=0;i<nn;i++)
                  {w=Ps(xi[i],ai);
                  yy=yi[i];
      //System.out.println("x="+xi[i]+"w="+w+"yy="+yy+"ff="+ff);
                  w-=yy;
      ff+=w*w;
      //System.out.println("x="+xi[i]+"w="+w+"yy="+yy+"ff="+ff);
                  }
     return ff;
                }
}

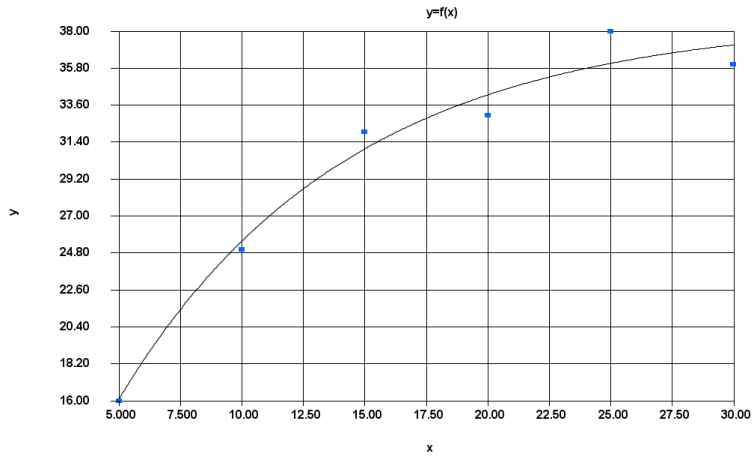
public class harris_hawks_optimization_test1
{

public static void main(String arg[])
   {   String in_name="input3.txt";
   int iN=100;
   yy f=new yy(in_name,2);
   double[] iLower={0.0,0.0};
   double[] iUpper={50,50};
   int iMaxiter=200;
   //harris_hawks_optimization(f_xj iff,int iN,double[] iLower,double[] iUpper, int iMaxiter)
              harris_hawks_optimizationi hho=new harris_hawks_optimizationi(f,iN,iLower,iUpper,iMaxiter);
   double[][] x0=hho.solution();
   String s="";
   s+="harris-hawks optimisation : \n"+Matrix.toStringT(x0[0]);
              //double [] r1= NA125.davidon_fletcher_powell_V1(f,x0[0]);
              //s+="solution set corrected DFP : \n"+Matrix.toStringT(r1);
```

```
            double [] r2=NA41.nelder(f,x0[0]);
            s+="Nelder-Mead solution : \n"+Matrix.toStringT(r2);
            System.out.println(s);
            if_x ff=x->r2[0]*(1.0-Math.exp(-r2[1]*x));
            Plot pp=new Plot(ff,5.0,30.0);
            double a[][]=Text.readDoubleT("input3.txt");
            pp.addData(a[0],a[1]);
            pp.setPlotType(1,21);
            pp.plot();
}
}
```

```
---------- Capture Output ----------
> "E:\co\java\bin\javaw.exe" harris_hawks_optimization_test1
end of file
harris-hawks optimisation :
        38.657896877219976
         0.108016570513192
Nelder-Mead solution :
        38.727045456466600
         0.107456496673013

> Terminated with exit code 0.
```



We will carry out the same optimisation by using Genetic algorithms and Thermal Exchange optimisation as well. It should be note that Genetic algorithms find maximum value, Programs integrates also with Nelder Mead optimisation methods which finds minimum value.

Program 6.8-5 Genetic algorithm one independent variable nonlinear least square curve fitting

```
 import java.util.*;
import java.io.*;
    class yy1 implements if_xj
    { yy f;
            double xi[]; // independent variable data set
    double yi[]; //dependent variable data set
    double a[];  //fit function coefficient set
    int nn;
            public yy1(String filename,int n)
    {
     //read the data to curvefit
     //get the data file and initial fit coefficient when class is defined
              xi=new double[5000];
     yi=new double[5000];
     a=new double[n];
     //seta(ia);
     int i=-1;
```

```
      try{
                BufferedReader fin=new BufferedReader(new FileReader(filename));
      try {
      while(fin != null)
      {
      i++;
      xi[i]=Text.readDouble(fin);
      yi[i]=Text.readDouble(fin);
      }
      } catch(EOFException e_eof) { }
      }catch(IOException e_io) {System.out.println("dosya bulunamadi"); }
    nn=i;
    }
    public void seta(double ia[])
    {
                  //assign new fit coefficient set
                  for(int ii=0;ii<nn;ii++)
          a[ii]=ia[ii];
               }
               public double[] geta()
    {
                 // return fit coefficient set
                 return a;
               }
               double Ps(double x,double ai[])
    {
    //sample specific function
               // saturation pressure as a function of saturation temperature
    a=ai;
    double pi=ai[0]*(1.0-Math.exp(-ai[1]*x));
    return pi; //kPa
    }
    public double func(double ai[])
    {
    double ff=0;
    double w;
    double yy;
     for(int i=0;i<nn;i++)
                 {w=Ps(xi[i],ai);
                 yy=yi[i];
                 w-=yy;
      ff+=w*w;              }
    return -ff;
               }}

public class Genetic3_test1
{

public static void main(String arg[])
   {   String in_name="input3.txt";
   int N=63;
             int POPSIZE=500;
   int MAXGENS=500;
   int NVARS=2;
   double PXOVER=0.3;
   double PMUTATION=0.02;
   Genetic3i xx=new Genetic3i(POPSIZE,MAXGENS,NVARS,N,PXOVER,PMUTATION);
   yy1 fx=new yy1(in_name,2);
   yy fz=new yy(in_name,2);
   double[] iLower={0.0,0.0};
   double[] iUpper={50,50};
   int iMaxiter=200;
   xx.setPopulation(iLower,iUpper,fx);
   double[] x0=xx.solution(fx);
```

```
   String s="";
   s+="Genetic3 optimisation : \n"+Matrix.toStringT(x0);
              double [] r2=NA41_i.nelder(fz,x0);
              s+="Nelder-Mead solution : \n"+Matrix.toStringT(r2);
              System.out.println(s);
              if_x ff=x->r2[0]*(1.0-Math.exp(-r2[1]*x));
              Plot pp=new Plot(ff,5.0,30.0);
              double a[][]=Text.readDoubleT("input3.txt");
              pp.addData(a[0],a[1]);
              pp.setPlotType(1,21);
              pp.plot();
}
}
```

```
---------- Capture Output ----------
> "E:\co\java\bin\javaw.exe" Genetic3_test1
Genetic3 optimisation :
          38.800000088888890
           0.106943935273514
Nelder-Mead solution :
          38.727045433140990
           0.107456496707803

> Terminated with exit code 0.
```

Program 6.8-6 Thermal exhange optimisation algorithm one independent variable nonlinear least square curve fitting

```
 import java.util.*;
import java.io.*;
public class Thermal_Exchange_Optimization_test3
{

public static void main(String arg[])
   {   String in_name="input3.txt";
   int iN=100;
   yy f=new yy(in_name,2);
   double[] iLower={0.0,0.0};
   double[] iUpper={50,50};
   int iMaxiter=200;
   Thermal_Exchange_Optimizationi wo=new   Thermal_Exchange_Optimizationi(f,iN,iLower,iUpper,iMaxiter);
   double[][] x0=wo.solution();
   String s="";
   s+="Thermal exchange optimisation  : \n"+Matrix.toStringT(x0[0]);
              double [] r2=NA41_i.nelder(f,x0[0]);
              s+="Nelder-Mead solution : \n"+Matrix.toStringT(r2);
              System.out.println(s);
              if_x ff=x->r2[0]*(1.0-Math.exp(-r2[1]*x));
              Plot pp=new Plot(ff,5.0,30.0);
              double a[][]=Text.readDoubleT("input3.txt");
              pp.addData(a[0],a[1]);
              pp.setPlotType(1,21);
              pp.plot();
}
}
```

```
---------- Capture Output ----------
> "E:\co\java\bin\javaw.exe" Thermal_Exchange_Optimization_test3
end of file
Thermal exchange optimisation  :
          38.558306736224620
           0.108852415410211
Nelder-Mead solution :
          38.727045546680750
```

```
0.107456496004458
```

Now let us try drying of sliced apple problem we previously solved by using Hawkins' hawks algorith merged with Nelder Mead algorithm

Program 6.8-7 Harris' hawks algorithm with two independent variable nonlinear least square curve fitting

```
 import java.io.*;
import javax.swing.*;

    class ifxa implements if_xj
    {
            double xi[][]; // m independent variable data set
    double yi[]; //dependent variable data set
    double a[];  //fit function coefficient set
    int nn;//number of data
    int m; //number of idependent variables xi
    double xmax;
            public ifxa(String filename,double ia[])
    {
      //read the data to curvefit
      //last column is treated as independent data set
      //get the data file and initial fit coefficient when class is defined
      int n=ia.length;
      a=new double[n];
      seta(ia);
      xmax=-1.0e99;
      double b[][]=Text.readDoubleT(filename);
      m=b.length-1;
      yi=b[m];
      nn=yi.length;
      xi=new double[m][nn];
      for(int i=0;i<nn;i++) {
              xi[0][i]=b[0][i];if(xi[0][i]>xmax) {xmax=xi[0][i];}
      xi[1][i]=b[1][i];
              yi[i]=b[m][i];
      }
    nn=yi.length;
    a=ia;
    }
    public void seta(double ia[])
    {
              //assign new fit coefficient set
              for(int ii=0;ii<nn;ii++)
          a[ii]=ia[ii];
            }

            public double[] geta()
    {
              // return fit coefficient set
              return a;
            }
     double Ps(double x[],double ai[])
    {
    // sample specific function m dimensional independent variable
    // independent variables : x[0],x[1]...x[m-1]
    // x[0] t
    // x[1] kalınlık
    // ai[0] a
    //
    a=ai;
    double t=x[0];
    double h=x[1];
```

```
    double a=ai[0]+ai[1]*h;
    double k=ai[2]+ai[3]*h;
    double pi=a*Math.exp(-k*t);
    return pi; //kPa
    }

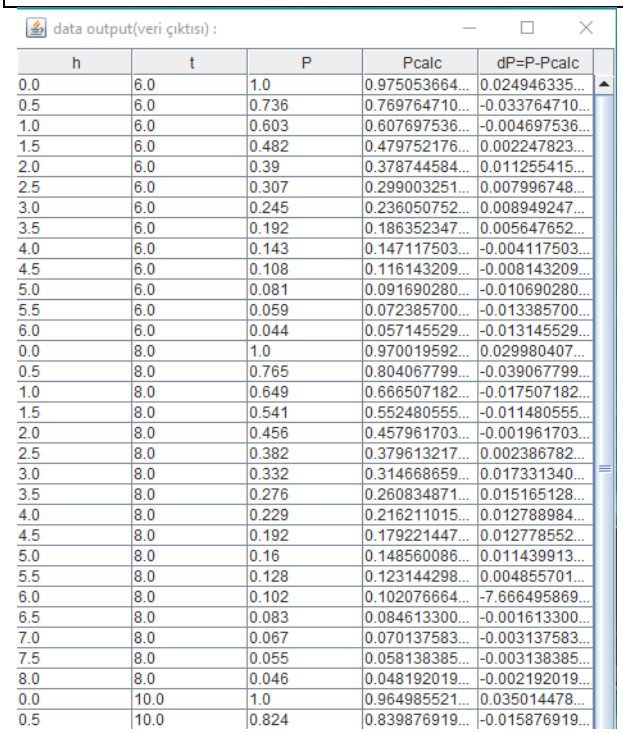
    public double func(double ai[])
    {
    double ff=0;
    double w;
    double yy;
    double xx[]=new double[xi.length];
     for(int i=0;i<nn;i++)
                {   for(int j=0;j<m;j++) {xx[j]=xi[j][i];}
                                          w=Ps(xx,ai);
                yy=yi[i];
                w-=yy;
      ff+=w*w;
                }
    return ff;
               }
}
class fx implements if_xj
{   double ai[];
              public fx(double aj[]) {ai=aj;}
   public double func(double x[])
   {// sample specific function m dimensional independent variable
    // independent variables : x[0],x[1]...x[m-1]
    // x[0] t
    // x[1] kalınlık
    // ai[0] a
    // ;
    double t=x[0];
    double h=x[1];
    double a=ai[0]+ai[1]*h;
    double k=ai[2]+ai[3]*h;
    double pi=a*Math.exp(-k*t);
    return pi; //kPa
   }
}

public class harris_hawks_optimization_test5
{

   public static void main(String args[]) throws IOException
   {
   //String in_name=JOptionPane.showInputDialog(" veri dosyasının isminigiriniz : ");
   String in_name="kalinlik1.txt";
   double a[];
   a=new double[4];
   a[0] = 1.0;
   a[1] = 1.0;
   a[2] = 1.0;
   a[3] = 1.0;
   double x[][],y[]; //curve fitting value
   double xu1[][],yu1[]; ////curve fitting value
   double xu2[][],yu2[];
   double xu3[][],yu3[];
   int iN=500;
   ifxa f=new ifxa(in_name,a);
   double[] iLower={-1.0,-1.0,-1.0,-1.0};
   double[] iUpper={2,2,2,2};
   int iMaxiter=500;
   harris_hawks_optimizationi hho=new harris_hawks_optimizationi(f,iN,iLower,iUpper,iMaxiter);
```

```
    double[][] x0=hho.solution();
    int i=0,j=0;
    double [] r2=NA41.nelder(f,x0[0]);
    String s="";
    s+="harris-hawks optimisation : \n"+Matrix.toStringT(x0[0]);
    s+="Nelder-Mead solution : \n"+Matrix.toStringT(r2);
    System.out.println(s);
    double E[][]=Text.readDoubleT(in_name);
    fx f1=new fx(r2);
    double xx[]=E[0];
    double yy[]=E[1];
    double zz[]=E[2];
    double ww[]=new double[xx.length];
    double ee[]=new double[xx.length];
    for(int ii=0;ii<xx.length;ii++)
    {   double xx1[]={xx[ii],yy[ii]};
                ww[ii]=f1.func(xx1);
                ee[ii]=zz[ii]-ww[ii];
              }
              double qq[][]={xx,yy,zz,ww,ee};
              String s1[]={"h","t","P","Pcalc","dP=P-Pcalc"};
    Text.printT(qq,s1);
    }
}
```

```
---------- Capture Output ----------
> "E:\co\java\bin\javaw.exe" harris_hawks_optimization_test5
harris-hawks optimisation :
          0.963470015426018
          0.000756980450594
          0.758892779991807
         -0.047965637605179
Nelder-Mead solution :
          0.990155888634559
         -0.002517037118643
          0.765464844854249
         -0.048774936394621
```



data output(veri çıktısı) :

h	t	P	Pcalc	dP=P-Pcalc
0.0	6.0	1.0	0.975053664...	0.024946335...
0.5	6.0	0.736	0.769764710...	-0.033764710...
1.0	6.0	0.603	0.607697536...	-0.004697536...
1.5	6.0	0.482	0.479752176...	0.002247823...
2.0	6.0	0.39	0.378744584...	0.011255415...
2.5	6.0	0.307	0.299003251...	0.007996748...
3.0	6.0	0.245	0.236050752...	0.008949247...
3.5	6.0	0.192	0.186352347...	0.005647652...
4.0	6.0	0.143	0.147117503...	-0.004117503...
4.5	6.0	0.108	0.116143209...	-0.008143209...
5.0	6.0	0.081	0.091690280...	-0.010690280...
5.5	6.0	0.059	0.072385700...	-0.013385700...
6.0	6.0	0.044	0.057145529...	-0.013145529...
0.0	8.0	1.0	0.970019592...	0.029980407...
0.5	8.0	0.765	0.804067799...	-0.039067799...
1.0	8.0	0.649	0.666507182...	-0.017507182...
1.5	8.0	0.541	0.552480555...	-0.011480555...
2.0	8.0	0.456	0.457961703...	-0.001961703...
2.5	8.0	0.382	0.379613217...	0.002386782...
3.0	8.0	0.332	0.314668659...	0.017331340...
3.5	8.0	0.276	0.260834871...	0.015165128...
4.0	8.0	0.229	0.216211015...	0.012788984...
4.5	8.0	0.192	0.179221447...	0.012778552...
5.0	8.0	0.16	0.148560086...	0.011439913...
5.5	8.0	0.128	0.123144298...	0.004855701...
6.0	8.0	0.102	0.102076664...	-7.666495869...
6.5	8.0	0.083	0.084613300...	-0.001613300...
7.0	8.0	0.067	0.070137583...	-0.003137583...
7.5	8.0	0.055	0.058138385...	-0.003138385...
8.0	8.0	0.046	0.048192019...	-0.002192019...
0.0	10.0	1.0	0.964985521...	0.035014478...
0.5	10.0	0.824	0.839876919...	-0.015876919...

Program 6.8-9 Thermal Exchange algorithm with two independent variable nonlinear least square curve fitting

```
import java.io.*;
import javax.swing.*;

public class  Thermal_Exchange_Optimization_test5
{

   public static void main(String args[]) throws IOException
   {
   //String in_name=JOptionPane.showInputDialog(" veri dosyasının isminigiriniz : ");
   String in_name="kalinlik1.txt";
   double a[];
   a=new double[4];
   a[0] = 1.0;
   a[1] = 1.0;
   a[2] = 1.0;
   a[3] = 1.0;
   double x[][],y[]; //curve fitting value
   double xu1[][],yu1[]; ////curve fitting value
   double xu2[][],yu2[];
   double xu3[][],yu3[];
   int iN=500;
   ifxa f=new ifxa(in_name,a);
   double[] iLower={-1.0,-1.0,-1.0,-1.0};
   double[] iUpper={2,2,2,2};
   int iMaxiter=500;
    Thermal_Exchange_Optimizationi wo=new   Thermal_Exchange_Optimizationi(f,iN,iLower,iUpper,iMaxiter);
   double[][] x0=wo.solution();
   int i=0,j=0;
   double [] r2=NA41.nelder(f,x0[0]);
   String s="";
   s+="harris-hawks optimisation : \n"+Matrix.toStringT(x0[0]);
   s+="Nelder-Mead solution : \n"+Matrix.toStringT(r2);
   System.out.println(s);
   double E[][]=Text.readDoubleT(in_name);
   fx f1=new fx(r2);
   double xx[]=E[0];
   double yy[]=E[1];
   double zz[]=E[2];
   double ww[]=new double[xx.length];
   double ee[]=new double[xx.length];
   for(int ii=0;ii<xx.length;ii++)
   {   double xx1[]={xx[ii],yy[ii]};
                ww[ii]=f1.func(xx1);
                ee[ii]=zz[ii]-ww[ii];
              }
              double qq[][]={xx,yy,zz,ww,ee};
              String s1[]={"h","t","P","Pcalc","dP=P-Pcalc"};
   Text.printT(qq,s1);
   }
}
```

¶

```
 ---------- Capture Output ----------
> "E:\co\java\bin\javaw.exe" Thermal_Exchange_Optimization_test5
Thermal exchange optimisation :
           1.051951878301539
          -0.010213933271042
           0.764537767561333
          -0.048899008341797
Nelder-Mead solution :
           0.990155877760489
          -0.002517035624054
```

```
0.765464840908594
-0.048774935944010
```

6.9 **USAGE OF PARAMETRIC VARIABLES IN CURVE FITTING**

x and y values are given in Table 6.8-1. If it is tried to curve fit an equation to this data, it will fail no matter which method will be used. For the same x values two different y values are defined. In cases like this in order to curve fit the data an artificial parameter is added and x and y curvefit seperately with this artificial parametric variable.

Table 6.9-1 input data a.txt

i	0	1	2	3	4
T	0	1	2	3	4
X	-1	0	1	0	1
Y	0	1	0.5	0	-1

Program 6.9-1 Cubic spline interpolation with indirect spline interpolation

```
import java.io.*;
import javax.swing.*;

class NA127
{
 public static double [] thomas(double a[][],double r[])
 {
 //
 int n=a.length;
 double f[]=new double[n];
 double e[]=new double[n];
 double g[]=new double[n];
 double x[]=new double[n];
 for(int i=0;i<n;i++)     {f[i]=a[i][i];}
 for(int i=0;i<(n-1);i++) {g[i]=a[i][i+1];}
 for(int i=0;i<(n-1);i++) {e[i+1]=a[i+1][i];}
 for(int k=1;k<n;k++)
   {e[k]=e[k]/f[k-1];
    f[k]=f[k]-e[k]*g[k-1];
   }
 for(int k=1;k<n;k++)
   {r[k]=r[k]-e[k]*r[k-1];
   }
 x[n-1]=r[n-1]/f[n-1];
 for(int k=(n-2);k>=0;k--)
   {x[k]=(r[k]-g[k]*x[k+1])/f[k];}
 return x;
 }

 public static double [] thomas(double f[],double e[],double g[],double r[])
 {
 int n=f.length;
 double x[]=new double[n];
 for(int k=1;k<n;k++)
   {e[k]=e[k]/f[k-1];
    f[k]=f[k]-e[k]*g[k-1];
   }
 for(int k=1;k<n;k++)
```

```
   {r[k]=r[k]-e[k]*r[k-1];
   }
x[n-1]=r[n-1]/f[n-1];
for(int k=(n-2);k>=0;k--)
   {x[k]=(r[k]-g[k]*x[k+1])/f[k];}
return x;
}

public static double [][] cubic_spline(double xi[],double yi[],double c0,double cn)
{
int n=xi.length;
double h[]=new double[n];
double w[]=new double[n];
double f[]=new double[n];
double e[]=new double[n];
double g[]=new double[n];
double d[]=new double[n];
double x[]=new double[n];
double S[][]=new double[4][n];
int k;
for(k=0;k<(n-1);k++)
   {h[k]=xi[k+1]-xi[k];
    w[k]=(yi[k+1]-yi[k])/h[k];
            }
d[0]=c0;
d[n-1]=cn;
for(k=1;k<(n-1);k++)
   {d[k]=6.0*(w[k]-w[k-1]);}
f[0]=1.0;
f[n-1]=1.0;
g[0]=0.0;
g[n-1]=0.0;
e[0]=0.0;
e[n-1]=0.0;
for(k=1;k<(n-1);k++)
{f[k]=2.0*(h[k]+h[k-1]);e[k]=h[k-1];g[k]=h[k];}
S[2]=thomas(f,e,g,d);
S[3]=xi;
for(k=0;k<(n-1);k++)
  {S[0][k]=(6.*yi[k+1]-h[k]*h[k]*S[2][k+1])/(6.0*h[k]);
   S[1][k]=(6.*yi[k]-h[k]*h[k]*S[2][k])/(6.0*h[k]);
  }
return S;
}

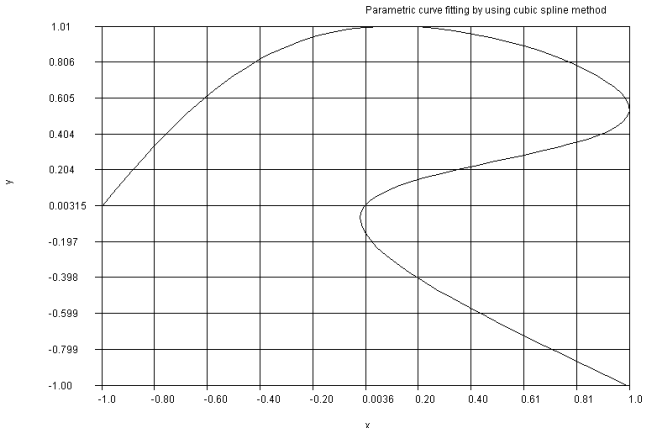
public static double funcSpline(double S[][],double x)
{
int n=S[0].length;
double xx1=0;
double xx2=0;
double y=0;
double hk=0;
for(int k=0;k<(n-1);k++)
   {if(S[3][k]<=x && x<=S[3][k+1])
    {hk=(S[3][k+1]-S[3][k]);
     xx1=(x-S[3][k]);
     xx2=(S[3][k+1]-x);
```

```
      y=S[0][k]*xx1+S[1][k]*xx2+(xx1*xx1*xx1*S[2][k+1]+xx2*xx2*xx2*S[2][k])/(6.0*hk);
      break;
      }
    }
      if(y==0 && S[3][n-2]<=x )
      {
               int k=n-2;
               hk=(S[3][k+1]-S[3][k]);
      xx1=(x-S[3][k]);
      xx2=(S[3][k+1]-x);
      y=S[0][k]*xx1+S[1][k]*xx2+(xx1*xx1*xx1*S[2][k+1]+xx2*xx2*xx2*S[2][k])/(6.0*hk);
      }
 return y;
 }

public static double[][]  funcSpline(double xi[],double yi[],int aradegersayisi)
{
            //aradegersayisi:  x--o--o--x--o--o--x zincirinde x deneysel noktalar ise
            // ara değer sayisi 2 dir
            int n=xi.length;
            int nn=(n-1)*(aradegersayisi+1)+1;
            double z[][]=new double[2][nn];
   double S[][]=cubic_spline(xi,yi,0,0);
   double dx=0;
   int k=0;
   int i;
   for(i=0;i<(n-1);i++)
      {  z[0][k]=xi[i];z[1][k]=funcSpline(S,z[0][k]);k++;
        for(int j=0;j<aradegersayisi;j++)
          {dx=(xi[i+1]-xi[i])/((double)aradegersayisi+1.0);
        z[0][k]=z[0][k-1]+dx;z[1][k]=funcSpline(S,z[0][k]);k++;}
      }
     z[0][k]=xi[i];z[1][k]=funcSpline(S,z[0][k]);
   return z;
}

 public static void main(String args[]) throws IOException
 {
 double x[];
 double y[];
 String s1="a.txt";
 JFileChooser fc=new JFileChooser();
 if (fc.showOpenDialog(null) == JFileChooser.APPROVE_OPTION) {File file = fc.getSelectedFile();s1=file.getName(); }
 double a[][]=Text.readDoubleT(s1);
 x=a[0];
 y=a[1];
 int n=x.length;
 double t[]=new double [n];
 for(int i=0;i<n;i++) t[i]=i;
 double z1[][]=funcSpline(t,x,20);
 double z2[][]=funcSpline(t,y,20);
 System.out.println(" z1\n"+Matrix.toString(z1));
 System.out.println(" z2\n"+Matrix.toString(z1));
 Plot pp=new Plot(z1[1],z2[1]);
 pp.setPlabel("Parametric curve fitting by using cubic spline method");
 pp.setXlabel("x");
 pp.setYlabel("y");
```

```
pp.setPlotType(0,0);
pp.addData(z1[1],z2[1]);
pp.plot();
}
}
```



6.10 MOVİNG LEAST SQUARES WITH SINGLE AND MULTI DIMENSIONAL VARIABLES (INTERPOLATION)

Polynomial least square equation was defined as

$$\frac{\partial H\left(a_0^{(m)},..,a_0^{(m)}\right)}{\partial a_k^{(m)}} = 2\sum_{i=1}^{n} w(x_i)\left[f(x_i) - \sum_{j=0}^{m} a_j^{(m)} x^j\right] x^k = 0 \quad k = 0,\dots,m \qquad (6.1-6)$$

and this equation was solved for weight function $w(x_i)$=1. If it is desired that the accuracy of the equation to be improved, it will be seen that increasing coeefficients will not always give beter accuracy. To see this effect assume the following data and make 3rd 4th and 5th degree curve fitting into the data.

x	1.47	1.83	3.02	3.56	5.86	8.75	9.45
f	2.09	1.92	2.19	2.64	3.19	3.13	3.61

Program 6.10-1 3,4 ve 5th degree polynomial least square curve fitting

```
import java.io.*;
import javax.swing.*;

class NA128
{
//polynomial leat square
  public static double[] pivotlugauss(double a[][],double b[])
  { //gauss elimination with partial pivoting
  int n=b.length;
  double x[]=new double[n];
  double carpan=0;
  double toplam=0;
  double buyuk;
  double dummy=0;
  int i,j,k,p,ii,jj;
  for(k=0;k<(n-1);k++)
  { //pivotlama
            p=k;
            buyuk=Math.abs(a[k][k]);
```

```
            for(ii=k+1;ii<n;ii++)
            {  dummy=Math.abs(a[ii][k]);
              if(dummy > buyuk) {buyuk=dummy;p=ii;}
            }
            if(p!=k)
            { for(jj=k;jj<n;jj++)
              { dummy=a[p][jj];
                a[p][jj]=a[k][jj];
                a[k][jj]=dummy;
              }
              dummy=b[p];
              b[p]=b[k];
              b[k]=dummy;
            }
   for(i=k+1;i<n;i++)
   {  carpan=a[i][k]/a[k][k];
      a[i][k]=0;
      for(j=k+1;j<n;j++)
      {   a[i][j]-=carpan*a[k][j]; }
         b[i]   =b[i]   -carpan*b[k];
            }
  }
  //back substitution
  x[n-1]=b[n-1]/a[n-1][n-1];
  for(i=n-2;i>=0;i--)
  {
    toplam=0;
    for(j=i+1;j<n;j++)
    {  toplam+=a[i][j]*x[j];}
    x[i]=(b[i]-toplam)/a[i][i];
  }
  return x;
}

public static double[] LSTSQR(double xi[],double yi[],int n)
{
int l=xi.length;
int i,j,k;
int np1=n+1;
double A[][];
A=new double[np1][np1];
double B[];
B=new double[np1];
double X[];
X=new double[np1];
for(i=0;i<n+1;i++)
  {  for(j=0;j<n+1;j++)
     {if(i==0 && j==0) A[i][j]=l;
     else  for(k=0;k<l;k++) A[i][j] += Math.pow(xi[k],(i+j));
     }
     for(k=0;k<l;k++) { if(i==0)  B[i]+= yi[k];
                  else      B[i] += Math.pow(xi[k],i)*yi[k];}
  }
System.out.println(Matrix.toString(A));
System.out.println(Matrix.toStringT(B));
X=pivotlugauss(A,B);
//X=B/A;
```

```
double max=0;
for(i=0;i<n+1;i++)
 if(Math.abs(X[i]) > max) max = Math.abs(X[i]);
for(i=0;i<n+1;i++)
 if((Math.abs(X[i]/max) > 0) && (Math.abs(X[i]/max) < 1.0e-100)) X[i]=0;
return X;
}

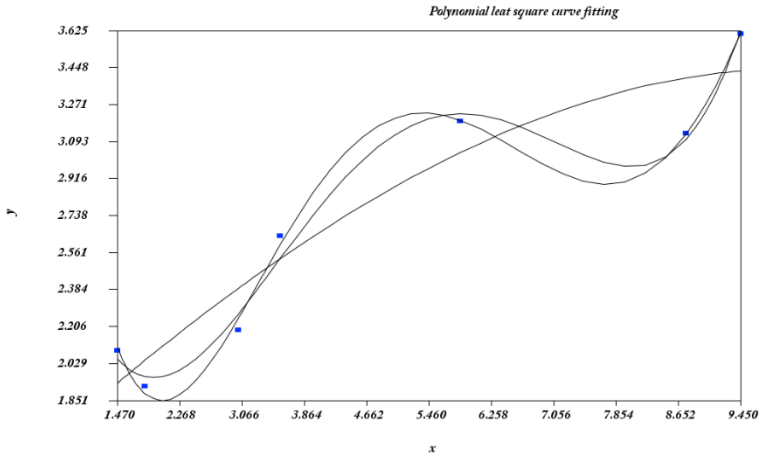
public static double funcLSTSQR(double e[],double x)
{
// this function calculates the value of
// least square curve fitting function
int n=e.length;
double ff;
if(n!=0.0)
  { ff=e[n-1];
  for(int i=n-2;i>=0;i--)
   { ff=ff*x+e[i]; }
  }
else
  ff=0;
return ff;
}

public static double error(double x[],double y[],double e[])
{
//calculates absolute square root error of a least square approach
double n=x.length;
 int k;
 double total=0;
 for(k=0;k<n;k++)
 {
 total+=(y[k]-funcLSTSQR(e,x[k]))*(y[k]-funcLSTSQR(e,x[k]));
 }
total=Math.sqrt(total);
return total;
}

public static double[][]  funcLSTSQR(double xi[],double yi[],int polinomkatsayisi,int aradegersayisi)
 {
            //aradegersayisi:  x--o--o--x--o--o--x zincirinde x deneysel noktalar ise
            // ara değer sayisi 2 dir
             int n=xi.length;
            int nn=(n-1)*(aradegersayisi+1)+1;
            double z[][]=new double[2][nn];
   double E[]=LSTSQR(xi,yi,polinomkatsayisi);
   System.out.println("coefficients :"+Matrix.toStringT(E));
   double dx=0;
   int k=0;
   int i;
   for(i=0;i<(n-1);i++)
     {z[0][k]=xi[i];z[1][k]=funcLSTSQR(E,z[0][k]);k++;
     for(int j=0;j<aradegersayisi;j++)
     {dx=(xi[i+1]-xi[i])/((double)aradegersayisi+1.0);
     z[0][k]=z[0][k-1]+dx;z[1][k]=funcLSTSQR(E,z[0][k]);k++;}
     }
     z[0][k]=xi[i];z[1][k]=funcLSTSQR(E,z[0][k]);
```

```
    return z;
  }


  public static void main(String args[]) throws IOException
  {
  double x[];
  double y[];
  String s1="a.txt";
  //JFileChooser fc=new JFileChooser();
  //if (fc.showOpenDialog(null) == JFileChooser.APPROVE_OPTION) {File file = fc.getSelectedFile();s1=file.getName();
}
  double a[][]=Text.readDoubleT(s1);
  x=a[0];
  y=a[1];
  int n=x.length;
  double t[]=new double [n];
  for(int i=0;i<n;i++) t[i]=i;
  double z1[][]=funcLSTSQR(x,y,3,10);
  System.out.println("Least Square. z1\n"+Matrix.toString(z1));
   System.out.println("Least Square. z2\n"+Matrix.toString(z1));
  Plot pp=new Plot(z1[0],z1[1]);
  pp.addData(x,y);
  z1=funcLSTSQR(x,y,4,10);
  pp.addData(z1[0],z1[1]);
  z1=funcLSTSQR(x,y,5,10);
  pp.addData(z1[0],z1[1]);
  pp.setPlabel("Polynomial least square curve fitting");
  pp.setXlabel("x");
  pp.setYlabel("y");
  pp.setPlotType(0,0);
  pp.setPlotType(1,18);
  pp.plot();
  }
}
```



In here all the equations given different kind of results. Which one is correct or closer to the data then? It is not always easy to answer. As a different approach to this curve fitting, lets apply a continiously changing weight factor

$$w_i(x) = \frac{1}{|x - x_i|^k}$$

In this function when x value become equal to x_i the value of weight factor will go to infinity. $x \to x_i$ $w_i(x) \to \infty$. If it is assumed that for $x=x_i$, $f=f_i$ and solve the least square for the remaining points solution will be existed. But in this type of conditions due to the fact that w_i was changing in each region, least square curve fitting coefficients should be calculated for each point(region) separetely. This type of least square method is called **moving least squares.** An example program to moving least squares are presented below

Program 6.10-2 Moving least square curve fitting

```
import java.io.*;
import javax.swing.*;

class NA129
{
//polynomial moving least square
  public static double[] pivotlugauss(double a[][],double b[])
  { //gauss elimination with pivoting
  int n=b.length;
  double x[]=new double[n];
  double carpan=0;
  double toplam=0;
  double buyuk;
  double dummy=0;
  //gauss eleme
  int i,j,k,p,ii,jj;
  for(k=0;k<(n-1);k++)
  { //pivotlama
              p=k;
              buyuk=Math.abs(a[k][k]);
              for(ii=k+1;ii<n;ii++)
              {  dummy=Math.abs(a[ii][k]);
                 if(dummy > buyuk) {buyuk=dummy;p=ii;}
              }
              if(p!=k)
              { for(jj=k;jj<n;jj++)
                { dummy=a[p][jj];
                  a[p][jj]=a[k][jj];
                  a[k][jj]=dummy;
                }
                dummy=b[p];
                b[p]=b[k];
                b[k]=dummy;
              }
              //gauss elemeyi çözme
    for(i=k+1;i<n;i++)
    {  carpan=a[i][k]/a[k][k];
       a[i][k]=0;
       for(j=k+1;j<n;j++)
       {   a[i][j]-=carpan*a[k][j]; }
           b[i]   =b[i]   -carpan*b[k];
              }
  }
  //geriye doğru yerine koyma
  x[n-1]=b[n-1]/a[n-1][n-1];
  for(i=n-2;i>=0;i--)
  {
    toplam=0;
    for(j=i+1;j<n;j++)
    {  toplam+=a[i][j]*x[j];}
    x[i]=(b[i]-toplam)/a[i][i];
  }
  return x;
}

public static double movingLSTSQR(double x,double xi[],double yi[],int n,int kk)
{
```

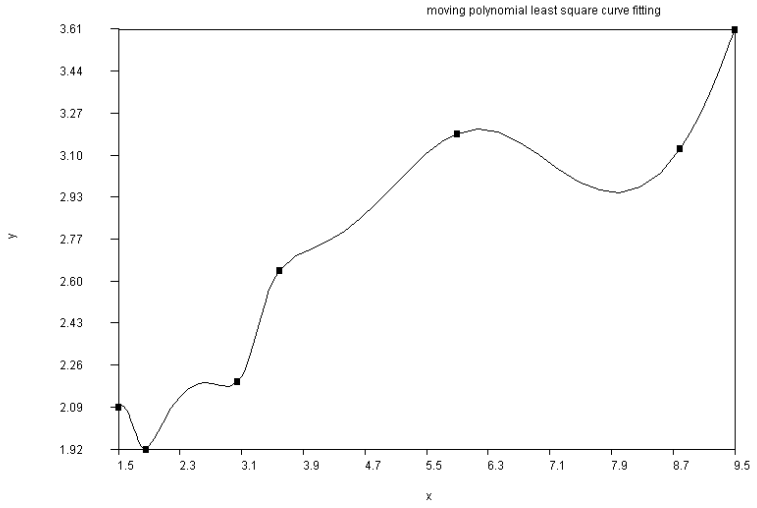
```
int l=xi.length;
int i,j,k;
int np1=n+1;
double A[][];
A=new double[np1][np1];
double B[];
B=new double[np1];
double X[];
X=new double[np1];
double w[]=new double[l];
for(i=0;i<l;i++) {if(x==xi[i]) return yi[i];}
for(i=0;i<l;i++) {w[i]=1.0/Math.pow(Math.abs(x-xi[i]),kk);}
for(i=0;i<np1;i++)
  {  for(j=0;j<np1;j++)
     {if(i==0 && j==0) A[i][j]=l;
     else  for(k=0;k<l;k++) A[i][j] += w[k]*Math.pow(xi[k],(i+j));
     }
     for(k=0;k<l;k++) { if(i==0)  B[i]+= w[k]*yi[k];
                    else      B[i] += w[k]*Math.pow(xi[k],i)*yi[k];}
  }
X=pivotlugauss(A,B);
double max=0;
for(i=0;i<n+1;i++)
 if(Math.abs(X[i]) > max) max = Math.abs(X[i]);
for(i=0;i<n+1;i++)
 if((Math.abs(X[i]/max) > 0) && (Math.abs(X[i]/max) < 1.0e-100)) X[i]=0;
return funcEKK(X,x);
}

public static double funcEKK(double e[],double x)
{
// this function calculates the value of
// least square curve fitting function
int n=e.length;
double ff;
if(n!=0.0)
  { ff=e[n-1];
  for(int i=n-2;i>=0;i--)
   { ff=ff*x+e[i]; }
  }
else
  ff=0;
return ff;
}

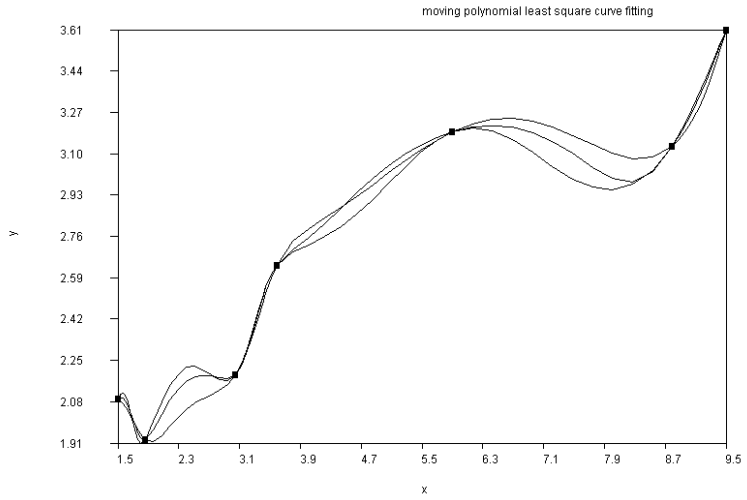
public static double[][]  funcEKK(double xi[],double yi[],int polinomkatsayisi,int aradegersayisi,int kk)
 {
                 int n=xi.length;
                int nn=(n-1)*(aradegersayisi+1)+1;
                double z[][]=new double[2][nn];
    double dx=0;
    int k=0;
    int i;
    for(i=0;i<(n-1);i++)
      {z[0][k]=xi[i];z[1][k]=movingLSTSQR(z[0][k],xi,yi,polinomkatsayisi,kk);k++;
      for(int j=0;j<aradegersayisi;j++)
      {dx=(xi[i+1]-xi[i])/((double)aradegersayisi+1.0);
      z[0][k]=z[0][k-1]+dx;z[1][k]=movingLSTSQR(z[0][k],xi,yi,polinomkatsayisi,kk);k++;}
      }
      z[0][k]=xi[i];z[1][k]=movingLSTSQR(z[0][k],xi,yi,polinomkatsayisi,kk);
    return z;
 }
  public static void main(String args[]) throws IOException
  {
```

```
  double x[];
  double y[];
  String s1="a.txt";
  //JFileChooser fc=new JFileChooser();
  //if (fc.showOpenDialog(null) == JFileChooser.APPROVE_OPTION) {File file = fc.getSelectedFile();s1=file.getName(); }
  double a[][]=Text.readDoubleT(s1);
  x=a[0];
  y=a[1];
  int n=x.length;
  double t[]=new double [n];
  for(int i=0;i<n;i++) t[i]=i;
  double z1[][]=funcEKK(x,y,4,10,2);
  //System.out.println("E.K.K. z1\n"+Matrix.toString(z1));
  Plot pp=new Plot(z1[0],z1[1]);
  pp.setPlabel("moving polynomial least square curve fitting");
  pp.setXlabel("x");
  pp.setYlabel("y");
  pp.setPlotType(0,0);
  pp.addData(x,y);
   pp.setPlotType(1,21);
  pp.plot();
  }
}
```

The same data in the previous example will be now evaluated by moving least square of 4th order polynomial With the weight factor $w_i(x) = \frac{1}{|x-x_i|^2}$



If this data is covered with different degree of polynomials(3rd, 4th and 5th degrees) :



The biggest disadvantage of this method is the requirement of matrix solution for each curve fitting point seperately. The equations are passing through each point. Therefore, this method is closer to interpolation methods compare to least square interpolations. Method can be applied also by using orthogonal polynomials, instead of normal polynomials, and this way disadvantage of applying many matrix solutions will be removed from the equation set.

Program 6.10-3 Moving least square curve fitting by using orthogonal polynomials

```
import java.io.*;
import javax.swing.*;

class NA121A
{
public static double movingOPEKK(double x,double xi[],double fi[],int m,int kk)
//ortogonal polinomial moving least square method
//Referance : A First Course in Numerical Analysis
// Anthony Ralston,Philip Rabinowitz, Mc Graw Hill ISBN 0-07-051158-6
// m degree of polynomial xi fi input data
// kk weight factor power wi=1/|(x-xi)|^kk
{
int i,j,k;
int n=xi.length;
int mp2=n+2;
int mp1=n+1;
double p[][]=new double[mp2][n];
double gamma[]=new double[mp1];
double beta[]=new double[mp1];
double omega[]=new double[mp1];
double alpha[]=new double[mp1];
double b[]=new double[mp1];
double wi[]=new double[n];
double X[][]=new double[3][mp1];
for(i=0;i<n;i++) {if(x==xi[i]) return fi[i];}
for(i=0;i<n;i++)
  {
   p[1][i]=1.0;
   p[0][i]=0.0;
   wi[i]=1.0/Math.pow(Math.abs(x-xi[i]),kk);
  }
gamma[0]=0;
for(i=0;i<n;i++)
  {
  gamma[0]+=wi[i];
  }
beta[0]=0.0;
```

```
for(j=0;j<m+1;j++)
  {
  omega[j]=0;
  for(i=0;i<n;i++)
    {omega[j]+=wi[i]*fi[i]*p[j+1][i];}
  b[j]=omega[j]/gamma[j];
if( j != m)
    {
    alpha[j+1]=0;
    for(i=0;i<n;i++)
      { alpha[j+1]+=wi[i]*xi[i]*p[j+1][i]*p[j+1][i]/gamma[j];}
    for(i=0;i<n;i++)
      {p[j+2][i]=(xi[i]-alpha[j+1])*p[j+1][i]-beta[j]*p[j][i];}
      gamma[j+1]=0;
    for(i=0;i<n;i++)
      {gamma[j+1]+=wi[i]*p[j+2][i]*p[j+2][i];}
    beta[j+1]=gamma[j+1]/gamma[j];
    }
}//end of j
for(j=0;j<m+1;j++)
  {
  X[0][j]=b[j];
  X[1][j]=alpha[j];
  X[2][j]=beta[j];
  }
return funcOPEKK(X,x);
}

public static double funcOPEKK(double a[][],double x)
{
// polinom değerleri hesaplama fonksiyonu
double yy=0;
int k;
int m=a[0].length-1;
int mp2=m+2;
double q[];
q=new double[mp2];
//vector<double> q(m+2,0.0);
for(k=m-1;k>=0;k--)
  {
  q[k]=a[0][k]+(x-a[1][k+1])*q[k+1]-a[2][k+1]*q[k+2];
  yy=q[k];
  }
return yy;
}

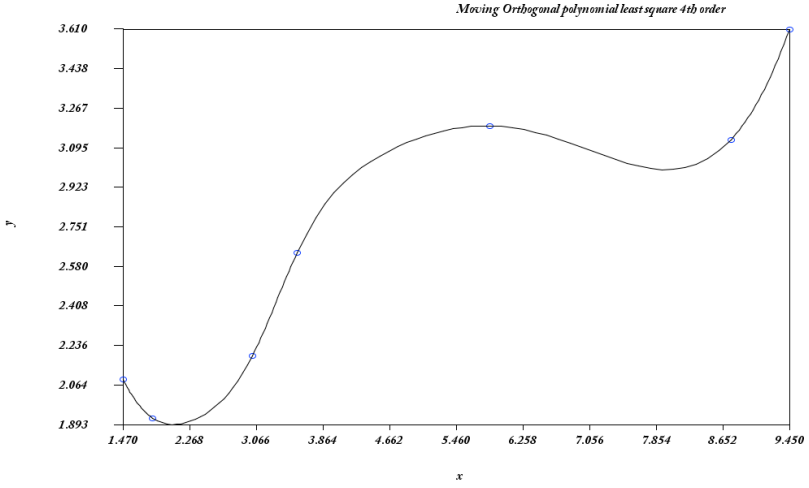
public static double[][]  funcOPEKK(double xi[],double yi[],int polynomialcoefficient,int numberofmidpoints,int kk)
 {
              //number of midpoints:  x--o--o--x--o--o--x  x data points midpoints are 2
              int n=xi.length;
              int nn=(n-1)*(numberofmidpoints+1)+1;
              double z[][]=new double[2][nn];
    //double E[][]=OPEKK(xi,yi,polynomialcoefficient);
    double dx=0;
    int k=0;
    int i;
    for(i=0;i<(n-1);i++)
      {z[0][k]=xi[i];z[1][k]=movingOPEKK(z[0][k],xi,yi,polynomialcoefficient,kk);
       k++;
      for(int j=0;j<numberofmidpoints;j++)
      {dx=(xi[i+1]-xi[i])/((double)numberofmidpoints+1.0);
      z[0][k]=z[0][k-1]+dx;z[1][k]=movingOPEKK(z[0][k],xi,yi,polynomialcoefficient,kk);k++;}
      }
      z[0][k]=xi[i];z[1][k]=movingOPEKK(z[0][k],xi,yi,polynomialcoefficient,kk);
```

```
    return z;
  }

  public static void main(String args[]) throws IOException
  {
  //double x[]={0.0,0.1,0.2,0.3,0.4,0.5,0.6,0.7,0.8,0.9,1.0};
  //double y[]={0.0,0.099833,0.1986693,0.2955202,0.38941834,0.479425,0.5646424,0.6442176,0.717356,0.783326,0.84147};
  String s1=JOptionPane.showInputDialog("file name : ");
  int n=Integer.parseInt(JOptionPane.showInputDialog("polinom derecesi (n)"));
  double c[][]=Text.readDoubleT(s1);
  double z3[][]=funcOPEKK(c[0],c[1],n,20,2);
  Text.print(Text.T(z3),"Moving Orthogonal polynomial least square ");
  Plot pp=new Plot(c[0],c[1]);
  pp.setPlabel("Moving Orthogonal polynomial least square ");
  pp.setPlotType(0,22);
  pp.addData(z3[0],z3[1]);
  pp.plot();
  }
}
```

Moving Orthogonal polynomial least square

0	1
1.47	2.09
1.487142857142857	2.07779704441685
1.5042857142857142	2.066029414208825
1.5214285714285714	2.054693812893271
1.5385714285714285	2.043786845502805
1.5557142857142856	2.0333050096474197
1.5728571428571427	2.023244686415547
1.5899999999999999	2.0136021311686294
1.607142857142857	2.004373464290527
1.624285714285714	1.9955546619634126
1.6414285714285712	1.987141547050553
1.6585714285714284	1.9791297801764047
1.6757142857142855	1.971514851104714
1.6928571428571426	1.9642920705252138
1.7099999999999997	1.9574565623699305
1.7271428571428569	1.9510032567899005
1.744285714285714	1.9449268839326925
1.7614285714285711	1.9392219686698753
1.7785714285714282	1.9338828264315624
1.7957142857142854	1.9289035603115945
1.8128571428571425	1.9242780596122262
1.83	1.92
1.8666666666666667	1.9082473193485172
1.9433333333333334	1.8999661657125702



Moving least square method can similarly be applied into multi-independent variable curve fitting problems. In principles number of independent variables in the equation does not effect the curve fitting process, except the definition of the functions. If a degree m polynomial

$$f(x_0, x_{1,}.., x_n) = \sum_{j=0}^{m} a_j^{(m)} \phi_j((x_0, x_{1,}.., x_n) \quad (6.3-1)$$

j th degree function is given. It is desired to fit x_0, $x_{1,}$ x_2,…, x_n ,$f_{i,}$ i=0...n data set into this function. The best fitted $a_j^{(m)}$ values to be found. For this purpose Minimum value of

$$H(x_0, x_{1,}.., x_n) = \sum_{i=1}^{n} w(x_0, x_{1,}.., x_n) \left[f_i - \sum_{j=0}^{m} a_j^{(m)} \phi_j(x_0, x_{1,}.., x_n) \right]^2 \quad (6.3-2)$$

function should be found. $w(x_0, x_{1,}.., x_n)$ is called weight function and it should satisfy the condition $w(x_0, x_{1,}.., x_n) \geq 0 \quad i = 1, \dots, n.$ The minimum of the function is the root of the derivative of the function.

$$\frac{\partial H(x_0, x_{1,}.., x_n)}{\partial a_k^{(m)}} = 2 \sum_{i=1}^{n} w(x_0, x_{1,}.., x_n) \left[f_i - \sum_{j=0}^{m} a_j^{(m)} \phi_j(x_0, x_{1,}.., x_n) \right] \phi_k(x_0, x_{1,}.., x_n) = 0 \quad k = 0,.., m \quad (6.3-2)$$

If unit value is taken as weight function, equation becomes:

$$2 \sum_{i=1}^{n} \left[f_i - \sum_{j=0}^{m} a_j^{(m)} \phi_j(x_0, x_{1,}.., x_n) \right] \phi_k(x_0, x_{1,}.., x_n) = 0 \qquad k = 0,.., m \quad (6.3-2a)$$

This equation was solved in chapter 6.3 by using $w(x_0, x_{1,}.., x_n)$=1 as classical least square approach. In order to define weight factor function Euclidian distance between the points will be used. If two vectoral point is defined in Euclidian space as:

$x = (x_0, x_{1,}.., x_n)$ (6.10-1)

$x^i = (x_0^i, x_1^i, .., x_n^i)$ (6.10-2)

Euclidian distance between x and x^i will be given by

$$d(x, x^i) = \sqrt{\sum_{j=0}^{n} (x_j - x_j^i)^2} \quad (6.10-3)$$

$w_i(x) = w(d(x, x^i))$ (6.10-4)

Now the weight function can be defined as a function of Euclidian distance as:

.

As an example definition of an actual weight function the following equation can be given

$$w(d) = \frac{e^{-d^2/16}}{d^2 + 0.001}$$

In this equations x is the point that curve fitting value is desired and x^i is the neighbouring point. When this weight factor is applied, program in the chapter 6.3 will be converted to the moving least square form.

Program 6.10-3 moving least square curve fitting of multidimensional independent variable function

```
 import java.io.*;
import java.util.*;
import javax.swing.*;
import java.awt.event.*;

//  Surface fitting by using moving least square model
//  f(x,y)=a0+a1*x+a2*y+a3*x^2+a4*y^2+a5*xy+a6*x^3+a7*y^3+a8*x^2y
//  +a9*xy^2+a10*x^4+a11*y^4+a12*x^2y^2+a13x^3y+a14*xy^3

abstract class f_xir
{ abstract double func(double x[],int equation_ref);}

//You can define your own fitting function
class fa extends f_xir
{
        double func(double x[],int i)
```

```
            {//A general two dimensional surface polynomial equation
             // of order (up to) 14
   double xx=0.0;
   //density function
   if(i==0)      {xx=1.0;          }
   else if (i==1) {xx=x[0];         }
   else if (i==2) {xx=x[1];         }
   else if (i==3) {xx=x[0]*x[0];    }
   else if (i==4) {xx=x[1]*x[1];    }
   else if (i==5) {xx=x[0]*x[1];    }
   else if (i==6) {xx=x[0]*x[0]*x[0];}
   else if (i==7) {xx=x[1]*x[1]*x[1];}
   else if (i==8) {xx=x[0]*x[0]*x[1];}
   else if (i==9) {xx=x[0]*x[1]*x[1];}
   else if (i==10){xx=x[0]*x[0]*x[0]*x[0];}
   else if (i==11){xx=x[1]*x[1]*x[1]*x[1];}
   else if (i==12){xx=x[0]*x[0]*x[1]*x[1];}
   else if (i==13){xx=x[0]*x[0]*x[0]*x[1];}
   else if (i==14){xx=x[0]*x[1]*x[1]*x[1];}
   return xx;
            }
}

class NA129H
{
            public static double d(double x[],double xi[])
            {// Euler distance
            double total=0;
            for(int i=0;i<x.length;i++)
            {total+=(x[i]-xi[i])*(x[i]-xi[i]);}
            return Math.sqrt(total);
   }

   public static double w(double x[],double xi[])
   {
            // Moving least square weight function
            double d=d(x,xi);
            double eps=0.001;
            double alpha=1.0/16.0;
            double ww=Math.exp(-alpha*d*d)/(d*d+eps);
            return ww;
            }

            public static double[][] Transpose(double [][] left)
   { //transpose matrix (if A=a(i,j) Transpose(A)=a(j,i)
   int i,j;
   int n=left.length;
   int m=left[0].length;
   double b[][];
   b=new double[m][n];
   for(i=0;i<n;i++)
     {for(j=0;j<m;j++)  {b[j][i]=left[i][j];} }
   return b;
   }

  public static double[] pivotlugauss(double a[][],double b[])
  { //gauss elimination with partial pivoting
  int n=b.length;
  double x[]=new double[n];
  double carpan=0;
  double toplam=0;
  double buyuk;
  double dummy=0;
  //gauss elimination
```

```
  int i,j,k,p,ii,jj;
  for(k=0;k<(n-1);k++)
  { //pivoting
                p=k;
                buyuk=Math.abs(a[k][k]);
                for(ii=k+1;ii<n;ii++)
                {  dummy=Math.abs(a[ii][k]);
                  if(dummy > buyuk) {buyuk=dummy;p=ii;}
                }
                if(p!=k)
                { for(jj=k;jj<n;jj++)
                  { dummy=a[p][jj];
                    a[p][jj]=a[k][jj];
                    a[k][jj]=dummy;
                  }
                  dummy=b[p];
                  b[p]=b[k];
                  b[k]=dummy;
                }
                //gauss elimination process
    for(i=k+1;i<n;i++)
    {  carpan=a[i][k]/a[k][k];
       a[i][k]=0;
       for(j=k+1;j<n;j++)
       {   a[i][j]-=carpan*a[k][j]; }
          b[i]   =b[i]   -carpan*b[k];
                }
  }
  //back substitution
  x[n-1]=b[n-1]/a[n-1][n-1];
  for(i=n-2;i>=0;i--)
  {
    toplam=0;
    for(j=i+1;j<n;j++)
    {  toplam+=a[i][j]*x[j];}
    x[i]=(b[i]-toplam)/a[i][i];
  }
  return x;
}

public static double HEKK(double x[],double xi[][],double yi[],int n)
{ // n dimension surface curve fitting for the vector point x[]
fa f=new fa();
int l=xi.length; //number of independent variables
int i,j,k;
int np1=n+1;
double A[][];
A=new double[np1][np1];
double B[];
B=new double[np1];
double E[];
E=new double[np1];
for(i=0;i<n+1;i++)
  {  B[i]=0;
                for(j=0;j<np1;j++)
    {
                A[i][j]=0.0;
    for(k=0;k<l;k++) A[i][j]+= w(x,xi[k])*f.func(xi[k],i)*f.func(xi[k],j);
    }
    for(k=0;k<l;k++) B[i]+= w(x,xi[k])*f.func(xi[k],i)*yi[k];
  }
E=pivotlugauss(A,B);
//X=B/A;
double max=0;
```

```
for(i=0;i<n+1;i++)
 if(Math.abs(E[i]) > max) max = Math.abs(E[i]);
for(i=0;i<n+1;i++)
 if((Math.abs(E[i]/max) > 0) && (Math.abs(E[i]/max) < 1.0e-100)) E[i]=0;
double z=funcEKKgenel(E,x);
return z;
}


public static double funcEKKgenel(double e[],double x[])
{
// Moving least square function evaluation
fa f=new fa();
int n=e.length;
double ff=0;
if(n!=0.0)
  {
  for(int i=n-1;i>=0;i--)
   {ff+=e[i]*f.func(x,i);}
  }
return ff;
}

public static double[] HEKK(double x[][],double xi[][],double yi[],int n)
{
                // n dimension surface curve fitting for the array of vector points x[]
int m=x.length;
double z[]=new double[m];
for(int i=0;i<m;i++)
{
z[i]= HEKK(x[i],xi,yi,n);
}
return z;
}

public static double[][] output(double x[][],double xi[][],double yi[],int n)
{  //gives curve fitting values together with the independent value set
int m=x.length;
int nn=x[0].length;
double c[][]=new double[m][nn+1];
double z[]=new double[m];
for(int i=0;i<m;i++)
{for(int j=0;j<nn;j++)
                {c[i][j]=x[i][j];}
}
for(int i=0;i<m;i++)
{
c[i][nn]= HEKK(x[i],xi,yi,n);
}
return c;
}
  public static double[][] z(double xmin, double xmax, double dx,double ymin,double ymax, double dy)
  {int n=(int)((xmax-xmin)/dx+0.0001)+1;int m=(int)((ymax-ymin)/dy+0.0001)+1;
   double z[][]=new double[(n+1)*(m+1)][2];
   double x=0,y=0;
   for(int i=0;i<n;i++)
   {x=xmin+i*dx;
    for(int j=0;j<m;j++)
    {y=ymin+j*dy;
     z[i*m+j][0]=x;
     z[i*m+j][1]=y;
    }
  }
  return z;
```

```
  }
  public static void main(String[] args)
  {
  String s1=JOptionPane.showInputDialog("curve fitting input data : ");
  double c[][]=Text.readDouble(s1);
  int n=c[0].length;
  int m=n-1;
  int l=c.length;
  double x[][]=new double[l][m];
  double y[]=new double[l];
  for(int i=0;i<l;i++)
  {for(int j=0;j<m;j++)
   {x[i][j]=c[i][j];}
   y[i]=c[i][m];
  }
  //String s2=JOptionPane.showInputDialog("required curve fitting independent variable list data : ");
  //double xy[][]=Text.readDouble(s2);
  double xy[][]=z(0.0,4.0,1,0.0,4.0,1);
  //double xy[][]=z(0.0,3.0,1.0,0.0,15.0,1.0);
  Text.print(xy,"xy");
  double z[][]=output(xy,x,y,12);
  Text.print(z);
  //surfacePlot p=new surfacePlot(c);
  //p.plot();
  surfacePlot p1=new surfacePlot(z);
  p1.plot();

  }
}
```

Example data for the program x_{ij},y_i sample data is generated by using the function $y=x_0^2+x_1^2$

0	0	0	3	0	9
0	1	1	3	1	10
0	2	4	3	2	13
0	3	9	3	3	18
0	4	16	3	4	25
1	0	1	4	0	16
1	1	2	4	1	17
1	2	5	4	2	20
1	3	10	4	3	25
1	4	17	4	4	32
2	0	4			
2	1	5			
2	2	8			
2	3	13			
2	4	20			

Independent variable data to curvefit

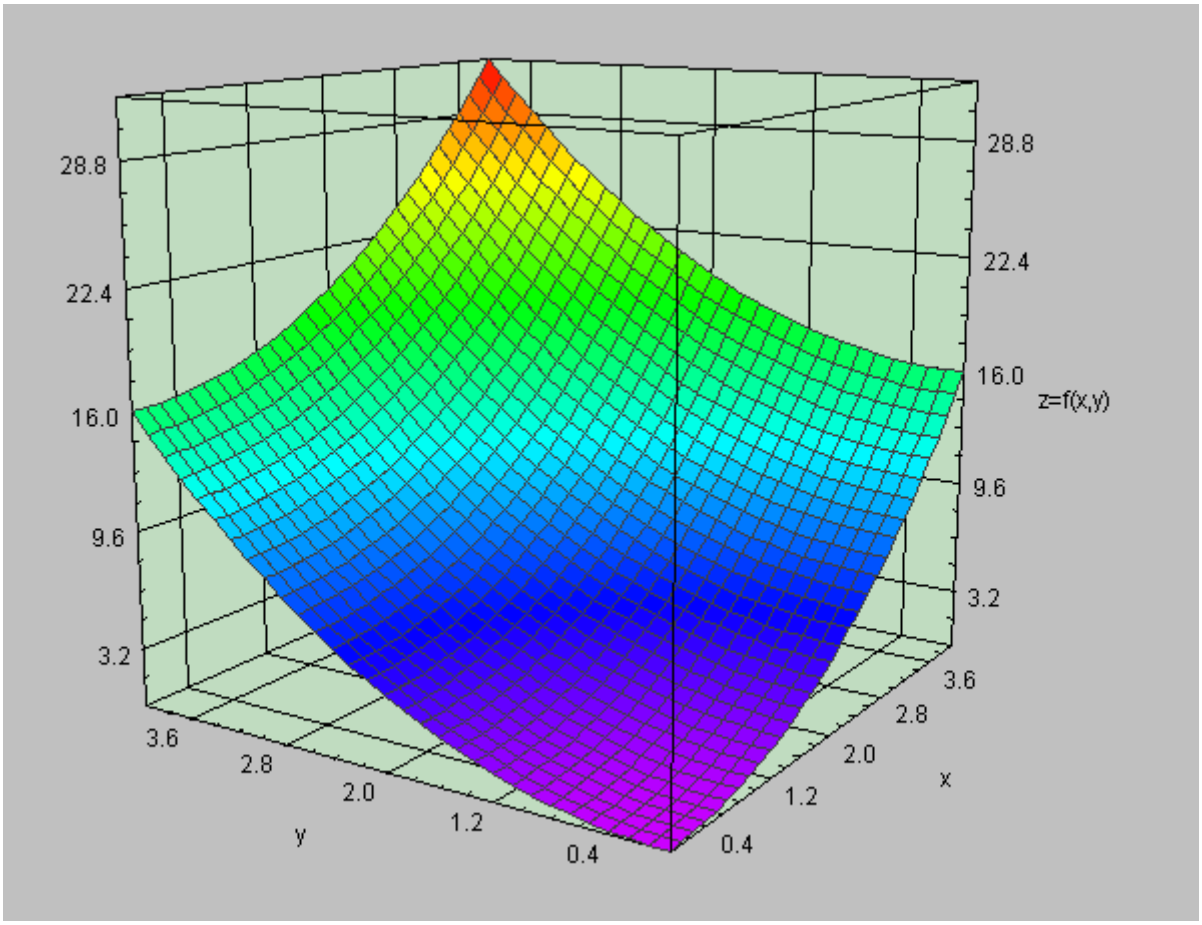
0	1	1.5	0
0	1.5	1.5	0.5
0	2	1.5	1
0.5	0	1.5	1.5
0.5	0.5	1.5	2
0.5	1	2	0
0.5	1.5	2	0.5
0.5	2	2	1
1	0	2	1.5
1	0.5	2	2
1	1		
1	1.5		

1 2	

Output values:

data output(veri çıktısı) :

0	1	2
0.0	1.0	1.0
0.0	1.5	2.2499999999999987
0.0	2.0	4.000000000000002
0.5	0.0	0.24999999999999853
0.5	0.5	0.5000000000000003
0.5	1.0	1.2500000000000004
0.5	1.5	2.4999999999999964
0.5	2.0	4.250000000000001
1.0	0.0	0.9999999999999998
1.0	0.5	1.2499999999999998
1.0	1.0	2.0000000000000004
1.0	1.5	3.249999999999996
1.0	2.0	4.999999999999998
1.5	0.0	2.249999999999997
1.5	0.5	2.5000000000000018
1.5	1.0	3.249999999999998
1.5	1.5	4.499999999999994
1.5	2.0	6.250000000000002
2.0	0.0	4.000000000000001
2.0	0.5	4.2499999999999964
2.0	1.0	4.999999999999993
2.0	1.5	6.249999999999994
2.0	2.0	7.999999999999991



6.11 BERNSTEIN POLINOMIALS AND BEZIER CURVES



Figure 6.11.1 Felix Bernstein, German Mathematician

Bernstein polynomials are a serial function approximation using the following form in a range of (a,b)

$$B_n(f;x) = \sum_{i=0}^{n} \binom{n}{i} (x-a)^i (b-x)^{n-i} f(x_i) \quad (6.11-1)$$

x_i in the equation can be written for constant point differences $x_i=a+ih=a+(i/n)(b-a)$ i=0..n . If the same function is written in (0,1) range:

$$B_n(f;x) = \sum_{i=0}^{n} \binom{n}{i} x^i (1-x)^{n-i} f\left(\frac{i}{n}\right) = \sum_{i=0}^{n} f\left(\frac{i}{n}\right) B_{n,i}(x) \quad (6.11-2)$$

$$B_{n,i}(x) = \sum_{i=0}^{n} \binom{n}{i} x^i (1-x)^{n-i} \quad (6.11-3)$$

When Bernstein polynomials are used for curve fitting approximations, it is seen that convertion rates are very low. Very high number of iterations are required for relatively good approximations. They will pass from starting and end points exactly but for the rest of the points it approximate slowly. But if mid-points are only taken as a reference. A practical method for drawing complicated shapes is obtained. This type of Bernstein polynomial applications have a special name: Bezier curves. Bezier curves uses parametric variables. For any (x_i , y_i) set with the parametric variable t

$$x(t) = \sum_{i=0}^{n} x_i \, B_{n,i}(t) \quad (6.11-4)$$

$$y(t) = \sum_{i=0}^{n} y_i \, B_{n,i}(t) \quad (6.11-5)$$

Let us use the following Example data to create th Bezier curve:

-1.0000000000 0.0000000000
-0.6666666667 0.6666666667
-0.3333333333 0.6666666667
0.3333333333 -0.6666666667
0.6666666667 -0.6666666667
1.0000000000 0.0000000000
0.6666666667 0.6666666667

0.3333333333 0.6666666667
-0.3333333333 -0.6666666667
-0.6666666667 -0.6666666667
-1.0000000000 0.0000000000

Program 6.11-1 Bezier curve generator

```
import java.io.*;
import javax.swing.*;

class NA130
{
//Bernstein polinomials and Bezier curves
  public static double[] B(int n,double t)
  { //  0<=t<=1
              double b[]=new double[n+1];
    b[0]=1.0;
    for(int i=1;i<=n;i++)
    {b[i]=t*b[i-1];
     for(int j=i-1;j>=1;j--)
     {b[j]=(1.0-t)*b[j]+t*b[j-1];}
     b[0]=(1.0-t)*b[0];
    }
  return b;
  }

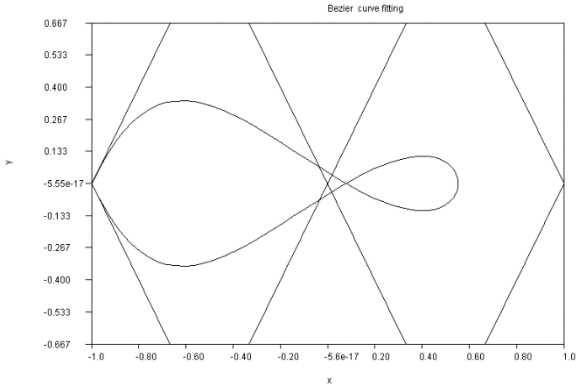
  public static double funcB(double a[],double t)
  { double xx=0;
    int n=a.length-1;
    double b[]=B(n,t);
    for(int i=0;i<=n;i++)
    {xx+=a[i]*b[i];}
  return xx;
  }

  public static double[][] Bezier(double xi[],double yi[],int noktasayisi)
  {
              int n=xi.length;
              double z[][]=new double[2][noktasayisi];
    double dt=1.0/(double)noktasayisi;
    double t=0;
    int k=0;
    int i;
    for(i=0;i<noktasayisi;i++)
       {z[0][i]=funcB(xi,t);z[1][i]=funcB(yi,t);
        t=t+dt;
       }
    return z;
  }



  public static void main(String args[]) throws IOException
  {
  double x[];
  double y[];
  String s1="a.txt";
```

```
  //JFileChooser fc=new JFileChooser();
  //if (fc.showOpenDialog(null) == JFileChooser.APPROVE_OPTION) {File file = fc.getSelectedFile();s1=file.getName();
}
  double a[][]=Text.readDoubleT(s1);
  x=a[0];
  y=a[1];
  int n=x.length;
  //for(double t=0;t<=1.0;t+=0.1) {System.out.println(funcB(4,x,t)+" "+funcB(4,y,t));}
  double z1[][]=Bezier(x,y,100);
  //System.out.println("E.K.K. z1\n"+Matrix.toString(Matrix.T(z1)));
  Plot pp=new Plot(z1[0],z1[1]);
  pp.setPlabel("Bezier  curve fitting");
  pp.setXlabel("x");
  pp.setYlabel("y");
  pp.setPlotType(0,0);
  pp.addData(x,y);
  pp.addData(x,y);
  pp.setPlotType(2,18);
  pp.plot();
  }
}
```



By changing reference points very complicated shapes can be drawn For example

0 0

3 5

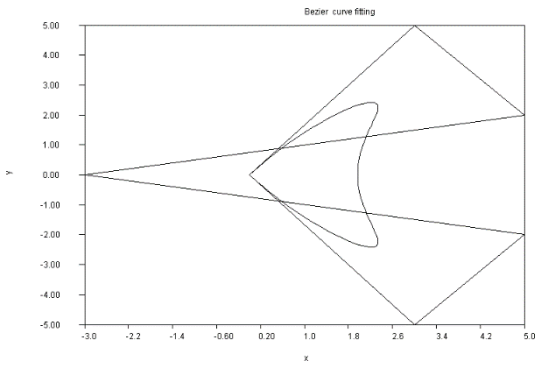
5 2

-3 0

5 -2

3 -5

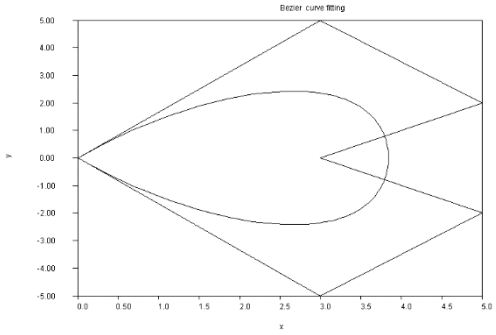
0 0



0 0

3 5
5 2
3 0
5 -2
3 -5

0 0



6.12 CURVE FITTING BY USING PARTIALLY CONTINOUS FUNCTIONS

One of the problems we faced in the real world least square curve fitting applications is that when the degree of equations are small, the curve fitting equation is unaccurate and when the degree of equation is increased round of error will considerably grow. If equation are defined for a smaller ranges, they tend to approach to the linear relations. So If the error level of a curve fitting equation can not be reduced to the desired level, one of the usefull way is defining partially continious functions for smaller regions of the curve. The method mathematically similar to the method of defining one curve, except that a family of the curves will be defined here instead of one curve. For each curve equation valid limits of the equations should also be given as additional coefficients. As an example, specific heat of perfect gases should be defined accurately for an accurate termodynamic model of the perfect gases. The specific heat of the gases will be defined with a partially continous function

$Cp=a_0 + a_1x10^{-3}T + a_2x10^5/T^2 + a_3x10^{-5}T^2$ where $T_L<T<T_H$. Nitrogen data will be taken as an example data. C_p values are given as (kJ/kg K) T unit is in degree Kelvin.

Table 6.12-1 Temperature – ideal gas Specific heat data for Nitrogen (N_2) gas

298.15 29.123	1900 35.794	3600 37.373
300 29.125	2000 35.969	3700 37.42
400 29.246	2100 36.126	3800 37.464
500 29.583	2200 36.267	3900 37.507
600 30.113	2300 36.395	4000 37.548
700 30.752	2400 36.511	4100 37.588
800 31.43	2500 36.616	4200 37.627
900 32.094	2600 36.713	4300 37.665
1000 32.696	2700 36.802	4400 37.701
1100 33.24	2800 36.884	4500 37.737
1200 33.723	2900 36.961	4600 37.773
1300 34.148	3000 37.031	4700 37.808
1400 34.519	3100 37.097	4800 37.842
1500 34.844	3200 37.159	4900 37.877
1600 35.128	3300 37.217	5000 37.911
1700 35.378	3400 37.272	
1800 35.598	3500 37.324	

Program 6.12-1 Partially continious curve-fit example

```
import java.io.*;
```

```
import java.util.*;
import javax.swing.*;


abstract class f_xr
{ abstract double func(double x,int equation_ref);}

class fh extends f_xr
{
double func(double t,int i)
{
double T=t+273.15;
            double xx=0.0;
if(i==0) xx=T;
else if(i==1) xx=1.0e-3/2.0*T*T;
else if(i==2) xx=-1.0e5/T;
else if(i==3) xx=1.0e-6/3.0*T*T*T;
return xx;
}
}

class fcp extends f_xr
{
double func(double t,int i)
{ // Cp(T) = xa[i]+xb[i]*1e-3*T+xc[i]*1.0e5/T^2+xd[i]*1e-6*T^2
double T=t;
            double xx=0.0;
if(i==0) xx=1.0;
else if(i==1) xx=1.0e-3*T;
else if(i==2) xx=1.0e5/(T*T);
else if(i==3) xx=1.0e-6*T*T;
return xx;
}
}

class Gas_GEN_Data
{
//this class reads and store and proses ideal gas data
public double M;       // kg/kmol
public double h0;      // KJ/kmol at temperature a[0][0]=Tref
public double s0;      // KJ/kmolK at temperature a[0][0]=Tref
public double hf;      // KJ/kmol at 298 K
public double R=8.3145; // KJ/kgmolK
public double a[][];
public String GasName;
public String openName;
public String ekbilgi;
int fitoption;
f_xr f1;
f_xr f2;

public double cpl[][];  // ideal gas heat capacitiy
public double visl[][]; // saturated liquid viscosity
public double kl[][]; // saturated liquid thermal conductivity

//General partial continious least square method
public Gas_GEN_Data(int fitoptioni,String GasNamei,String openNamei,double ai[][],int ngroup,double Mi, double h0i,double
s0i,double hfi)
{
//fitoption : 0   Cp
//other         h
fitoption=fitoptioni;
GasName=GasNamei;
a=ai;
```

```
ekbilgi=" ";
openName=openNamei;
int n=a.length;
int i,j;
if(fitoption==0) {f1=new fcp(); }
else            {f1=new fh(); }
f2=new fcp();
cpl=EKKgeneral(f1,a[0],a[2],ngroup);
visl=EKKgeneral(f2,a[0],a[3],ngroup);
kl=EKKgeneral(f2,a[0],a[4],ngroup);
M=Mi;
h0=h0i;
s0=s0i;
hf=hfi;
}

public static double[] pivotlugauss(double a[][],double b[])
{ //gauss eliminiation with partial pivoting
int n=b.length;
double x[]=new double[n];
double carpan=0;
double toplam=0;
double buyuk;
double dummy=0;
//gausselimination
int i,j,k,p,ii,jj;
for(k=0;k<(n-1);k++)
{ //pivoting
p=k;
buyuk=Math.abs(a[k][k]);
for(ii=k+1;ii<n;ii++)
{  dummy=Math.abs(a[ii][k]);
            if(dummy > buyuk) {buyuk=dummy;p=ii;}
}
if(p!=k)
{ for(jj=k;jj<n;jj++)
            { dummy=a[p][jj];
            a[p][jj]=a[k][jj];
            a[k][jj]=dummy;
            }
            dummy=b[p];
            b[p]=b[k];
            b[k]=dummy;
}
//solving of gauss elimination
for(i=k+1;i<n;i++)
{  carpan=a[i][k]/a[k][k];
a[i][k]=0;
for(j=k+1;j<n;j++)
{   a[i][j]-=carpan*a[k][j]; }
b[i]   =b[i]   -carpan*b[k];
}
}
//back substitution
x[n-1]=b[n-1]/a[n-1][n-1];
for(i=n-2;i>=0;i--)
{
toplam=0;
for(j=i+1;j<n;j++)
{  toplam+=a[i][j]*x[j];}
x[i]=(b[i]-toplam)/a[i][i];
}
return x;
}
```

```
public static double[][] EKKgeneral(f_xr f, double xi[],double yi[],int ngroup)
{
//ngroup : number of data in each group
//negri  : degree of curve fitting
//groupnumber : number of group
int negri=3;
int n=negri;
double y[]=new double[xi.length];
for(int i=0;i<xi.length;i++)
        {y[i]=yi[i];}
if(ngroup<negri) ngroup=negri;
int Nxi=xi.length; //total number of data
int groupnumber=Nxi/ngroup;
int ngroup1=ngroup+Nxi%ngroup;
//
int i,j,k;
int np1=4;
int np5=np1+2;
double A[][];
A=new double[np1][np1];
double B[];
B=new double[np1];
double X[][];
X=new double[groupnumber][np5];
System.out.println("group number = "+ groupnumber);
double Y[]=new double[np1];
int l=0;
double max=0;
//
for(l=0;l<groupnumber-1;l++)
{
for(i=0;i<np1;i++)
{  B[i]=0;
        for(j=0;j<np1;j++)
{
        A[i][j]=0.0;
for(k=l*ngroup;k<=(l+1)*ngroup;k++) {A[i][j]+=f.func(xi[k],i)*f.func(xi[k],j);}
}
for(k=l*ngroup;k<=(l+1)*ngroup;k++) B[i]+= f.func(xi[k],i)*y[k];
}
Y=pivotlugauss(A,B);//pivotlugauss(A,B);

int ii;
for(ii=0;ii<Y.length;ii++) {X[l][ii]=Y[ii];}
X[l][ii]=xi[l*ngroup];
X[l][ii+1]=xi[(l+1)*ngroup];
for(i=0;i<n+1;i++)
if(Math.abs(X[l][i]) > max) max = Math.abs(X[l][i]);
for(i=0;i<n+1;i++)
if((Math.abs(X[l][i]/max) > 0) && (Math.abs(X[l][i]/max) < 1.0e-100))X[l][i]=0;
}
l=groupnumber-1;
int k1=l*ngroup;
int k2=(l+1)*ngroup+Nxi%ngroup;
for(i=0;i<np1;i++)
{  B[i]=0;
        for(j=0;j<np1;j++)
{
        A[i][j]=0.0;
for(k=k1;k<k2;k++) {A[i][j]+=f.func(xi[k],i)*f.func(xi[k],j);}
}
for(k=k1;k<k2;k++) {B[i]+= f.func(xi[k],i)*y[k];}
}
```

```
Y=pivotlugauss(A,B);
int ii;
for(ii=0;ii<Y.length;ii++) {X[l][ii]=Y[ii];}
X[l][ii]=xi[k1];
X[l][ii+1]=xi[k2-1];

return X;
}

public static double funcEKKgeneral(f_xr f,double e[][],double x)
{
// this function calculates the value of
// least square curve fitting function
int n1=e.length;
int n2=e[0].length;
int n3=n2-2;
System.out.println("n1="+n1+"n2="+n2+"n3="+n3);
double ff=0;
double xlow,xhigh;
for(int j=0;j<n1;j++)
{
xlow=e[j][n3];
xhigh=e[j][n3+1];
System.out.println("xlow="+xlow+"xhigh="+xhigh);
if((x>=xlow) && (x<=xhigh))
{
ff=0;
for(int i=0;i<n3;i++)
{ff+=e[j][i]*f.func(x,i);System.out.println("ff="+ff+"i="+i);}
break;
}
}
return ff;
}

public static double[][] xyfit(f_xr f,double x[],double yi[],double e[][])
{
//calculates absolute square root error of a least square approach
double n=x.length;
int k;
double total=0;
double yy=0;
double z[][]=new double[2][x.length];
for(k=0;k<n;k++)
{   z[0][k]=x[k];
            z[1][k]=funcEKKgeneral(f,e,x[k]);
}
return z;
}


public static double[][] dxyfit(f_xr f,double x[],double yi[],double e[][])
{
//calculates absolute square root error of a least square approach
double n=x.length;
int k;
double total=0;
double yy=0;
double z[][]=new double[2][x.length];
for(k=0;k<n;k++)
{   z[0][k]=x[k];
            z[1][k]=yi[k]-funcEKKgeneral(f,e,x[k]);
}
return z;
```

```
}

public static double hata(f_xr f,double x[],double yi[],double e[][])
{
//calculates absolute square root error of a least square approach
double n=x.length;
int k;
double total=0;
double yy=0;
for(k=0;k<n;k++)
{
        yy=yi[k]-funcEKKgeneral(f,e,x[k]);
total+=yy*yy;
}
total=Math.sqrt(total/(n-1));
return total;
}
public static double[][]  hataEKKPlot(f_xr f,double xi[],double yi[],int ngroup)
{
//ngroup : Number of data in each group
int n=xi.length;
int l;
double z[][];
double E[][];//=EKKgeneral(xi,yi,polinomkatnumber,ngroup);
double dx=0;
int k=0;
int i;
E=EKKgeneral(f,xi,yi,ngroup);
z=dxyfit(f,xi,yi,E);
Plot pp=new Plot(z[0],z[1]);
pp.setPlabel("Cp=a0+a1*1e-3*T+a2*1e5/T^2+a3*1e-5*T^2  Tl<=T<=Th");
pp.setXlabel("T degree K");
pp.setYlabel("Error Cp KJ/kmolK");
pp.plot();
return z;
}

public static double[][]  funcEKKPlot(f_xr f,double xi[],double yi[],int ngroup)
{
int n=xi.length;
int l;
double z[][];
double E[][];
double dx=0;
int k=0;
int i;
E=EKKgeneral(f,xi,yi,ngroup);
z=xyfit(f,xi,yi,E);
Plot pp=new Plot(z[0],z[1]);
pp.setPlabel("Cp=a0+a1*1e-3*T+a2*1e5/T^2+a3*1e-5*T^2  Tl<=T<=Th");
pp.setXlabel("T temperature degree K");
pp.setYlabel("Cp specific heat KJ/kmolK");
pp.addData(xi,yi,1);
pp.plot();
return z;
}

public static double[][]  funcEKKgeneral(f_xr f,double xi[],double yi[],int ngroup,int subintervalnumber)
{
int n=xi.length;
int nn=(n-1)*(subintervalnumber+1)+1;
double z[][]=new double[2][nn];
double E[][]=EKKgeneral(f,xi,yi,ngroup);
Text.print(E,"EfuncEKK");
```

```
double dx=0;
int k=0;
int i;
for(i=0;i<(n-1);i++)
{z[0][k]=xi[i];z[1][k]=funcEKKgeneral(f,E,z[0][k]);
for(int j=0;j<subintervalnumber;j++)
{dx=(xi[i+1]-xi[i])/((double)subintervalnumber+1.0);
k++;
z[0][k]=z[0][k-1]+dx;z[1][k]=funcEKKgeneral(f,E,z[0][k]);k++;}
}
z[0][k]=xi[i];z[1][k]=funcEKKgeneral(f,E,z[0][k]);
return z;
}

public double Cp_l(double t)
{ return funcEKKgeneral(f2,cpl,t);}

public double viscosity_l(double t)
{return funcEKKgeneral(f2,visl,t);}

public double k_l(double t)
{return funcEKKgeneral(f2,kl,t);}

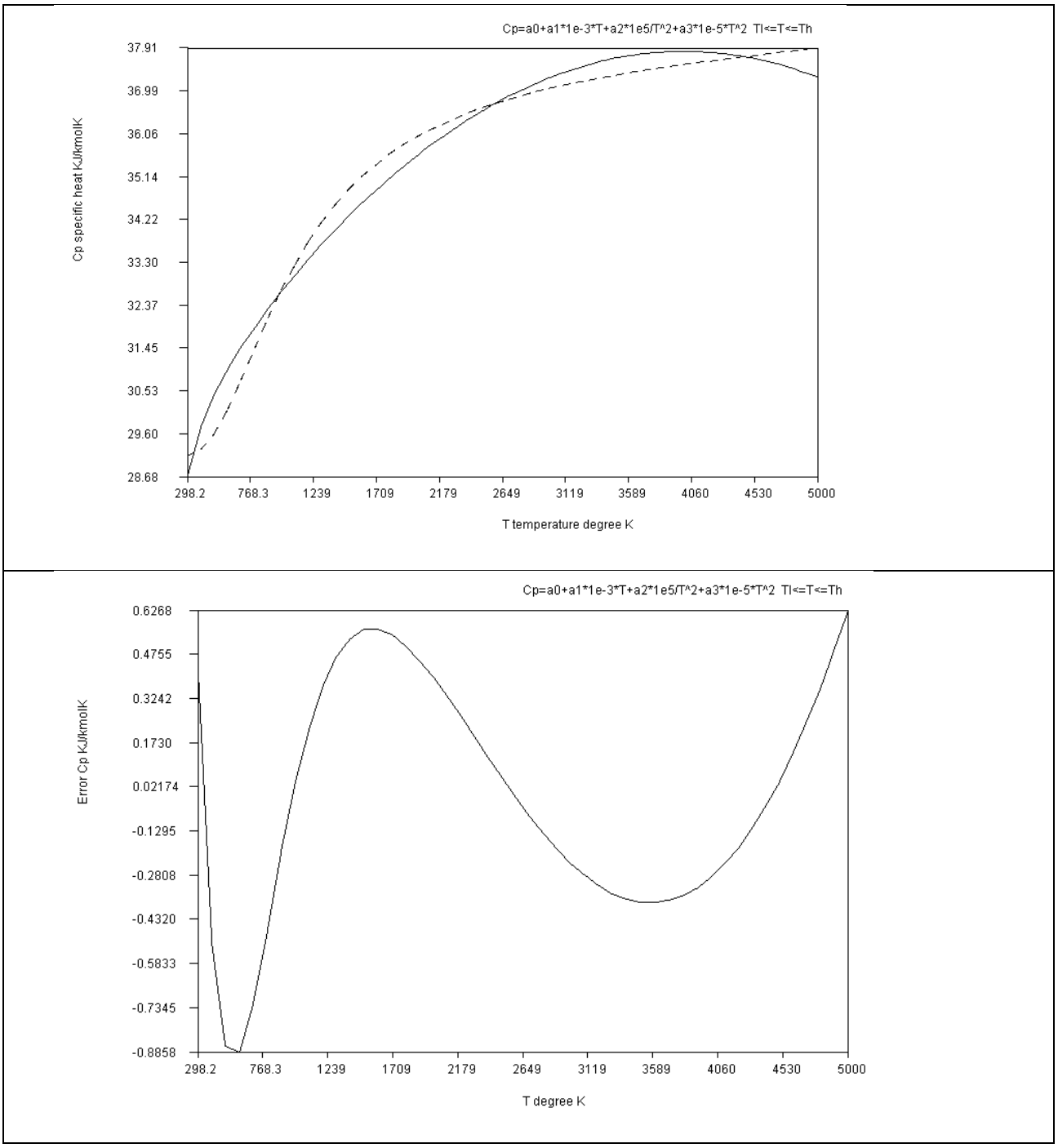

public static void main(String args[]) throws IOException
{
String s1=JOptionPane.showInputDialog("name of the input file : ");
double c[][]=Text.readDoubleT(s1);
Text.print(Text.T(c));
double d[][]=EKKgeneral(new fcp(), c[0],c[1],10);
funcEKKPlot(new fcp(), c[0],c[1],10);
Text.print(d);
hataEKKPlot(new fcp(), c[0],c[1],4);
System.out.println(Matrix.toString(d));
}

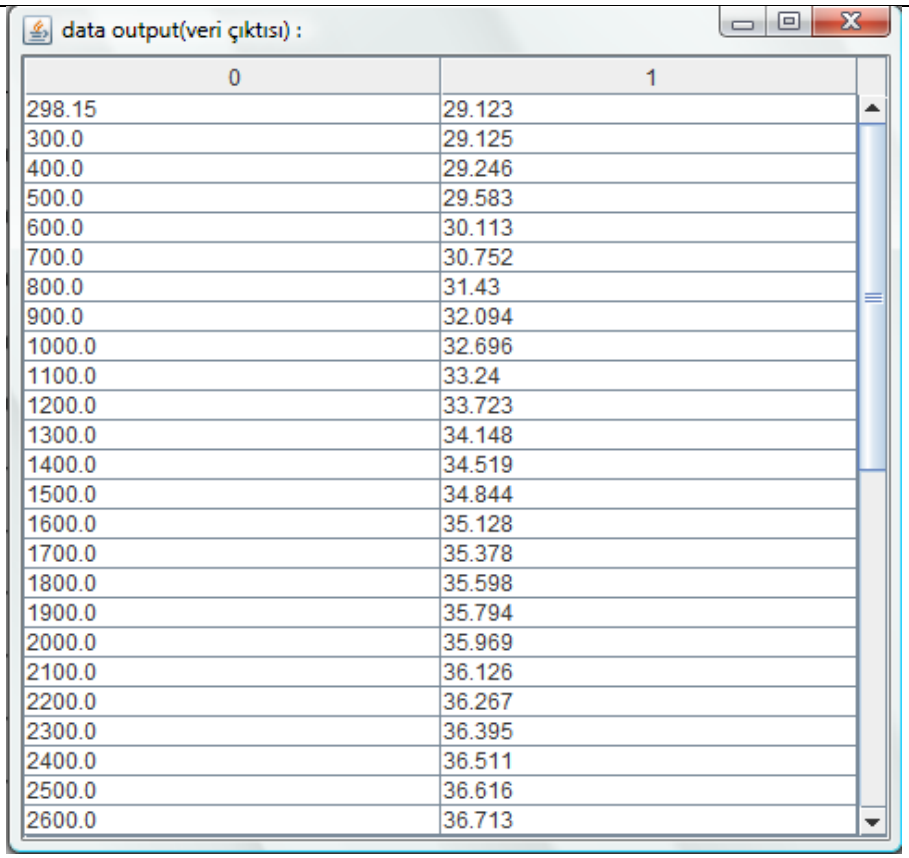
}
```

Output 6.11-1 Least square curve fitting, one equation for the full range (n=1)

data output(veri çıktısı) :

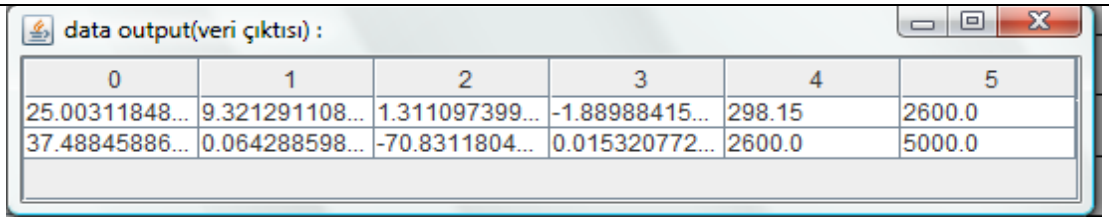
0	1	2	3	4	5
28.875689238984958	4.485071684423939	-1.3183008985398887	-0.5604610967288349	298.15	5000.0


Cp=a0+a1*1e-3*T+a2*1e5/T^2+a3*1e-5*T^2 Tl<=T<=Th
Cp specific heat KJ/kmolK
37.91
36.99
36.06
35.14
34.22
33.30
32.37
31.45
30.53
29.60
28.68
298.2
768.3
1239
1709
2179
2649
3119
3589
4060
4530
5000
T temperature degree K
Cp=a0+a1*1e-3*T+a2*1e5/T^2+a3*1e-5*T^2 Tl<=T<=Th
Error Cp KJ/kmolK
0.6268
0.4755
0.3242
0.1730
0.02174
-0.1295
-0.2808
-0.4320
-0.5833
-0.7345
-0.8858
298.2
768.3
1239
1709
2179
2649
3119
3589
4060
4530
5000
T degree K

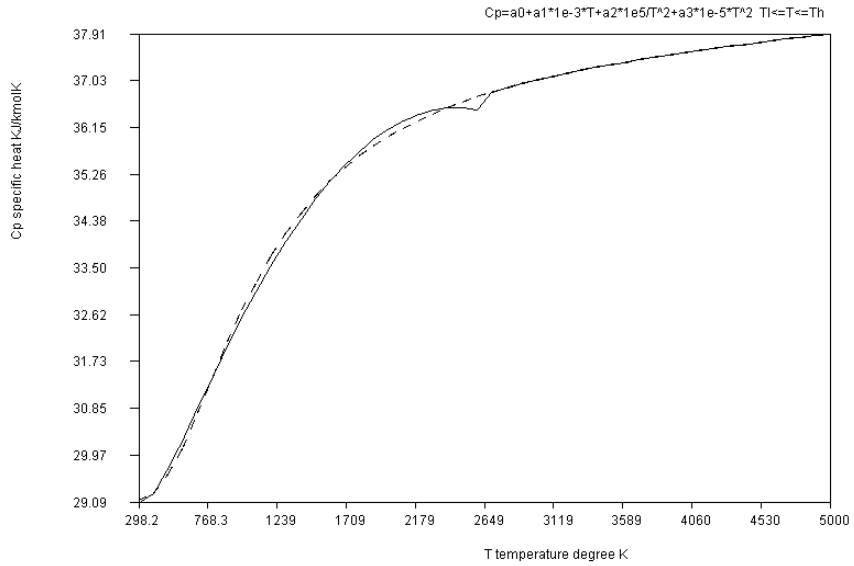


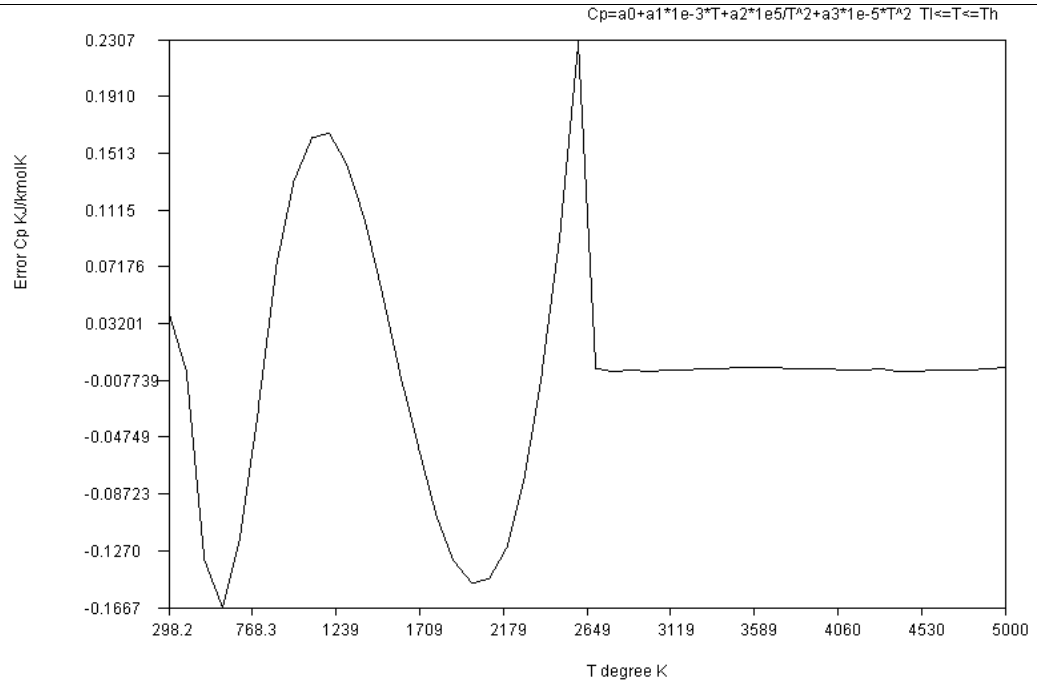
0	1
298.15	29.123
300.0	29.125
400.0	29.246
500.0	29.583
600.0	30.113
700.0	30.752
800.0	31.43
900.0	32.094
1000.0	32.696
1100.0	33.24
1200.0	33.723
1300.0	34.148
1400.0	34.519
1500.0	34.844
1600.0	35.128
1700.0	35.378
1800.0	35.598
1900.0	35.794
2000.0	35.969
2100.0	36.126
2200.0	36.267
2300.0	36.395
2400.0	36.511
2500.0	36.616
2600.0	36.713

Output 6.12-2 Partially continious least square curve fitting, two equations for the full range (n=2)



0	1	2	3	4	5
25.00311848...	9.321291108...	1.311097399...	-1.88988415...	298.15	2600.0
37.48845886...	0.064288598...	-70.8311804...	0.015320772...	2600.0	5000.0


Cp=a0+a1*1e-3*T+a2*1e5/T^2+a3*1e-5*T^2 Tl<=T<=Th
Cp specific heat KJ/kmolK
37.91
37.03
36.15
35.26
34.38
33.50
32.62
31.73
30.85
29.97
29.09
298.2
768.3
1239
1709
2179
2649
3119
3589
4060
4530
5000
T temperature degree K


Cp=a0+a1*1e-3*T+a2*1e5/T^2+a3*1e-5*T^2 Tl<=T<=Th
Error Cp KJ/kmolK
0.2307
0.1910
0.1513
0.1115
0.07176
0.03201
-0.007739
-0.04749
-0.08723
-0.1270
-0.1667
298.2
768.3
1239
1709
2179
2649
3119
3589
4060
4530
5000
T degree K

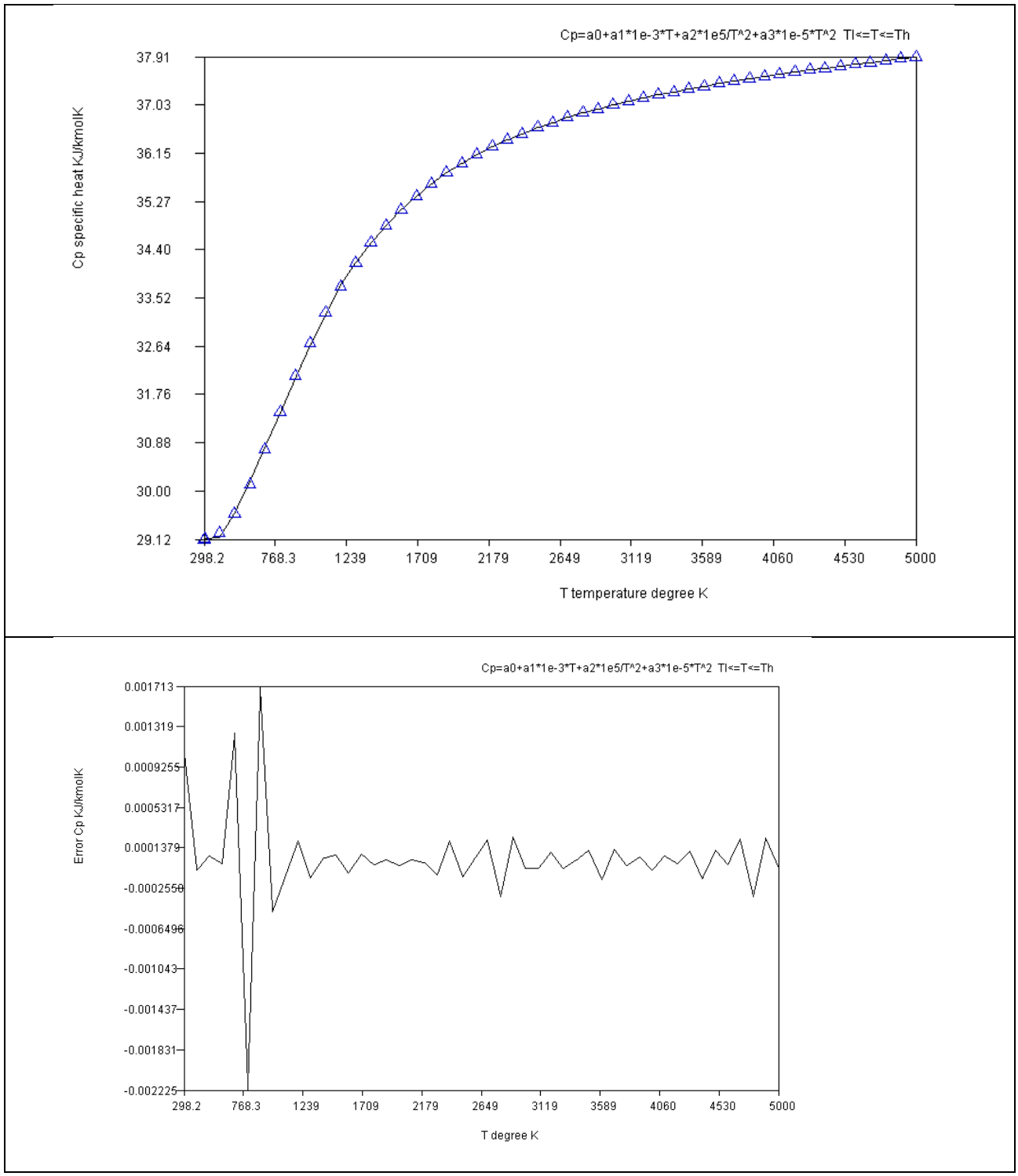
data output(veri çıktısı) :

0	1
298.15	29.123
300.0	29.125
400.0	29.246
500.0	29.583
600.0	30.113
700.0	30.752
800.0	31.43
900.0	32.094
1000.0	32.696
1100.0	33.24
1200.0	33.723
1300.0	34.148
1400.0	34.519
1500.0	34.844
1600.0	35.128
1700.0	35.378
1800.0	35.598
1900.0	35.794
2000.0	35.969
2100.0	36.126
2200.0	36.267
2300.0	36.395
2400.0	36.511
2500.0	36.616
2600.0	36.713

Output 6.12-3 Partially continious least square curve fitting, four equations for the full range (n=4)

data output(veri çıktısı) :

0	1	2	3	4	5
24.368546536...	10.007086647...	1.7415331317...	-1.8979238859...	298.15	1200.0
32.214041896...	3.2743928165...	-23.452661115...	-0.5513072397...	1200.0	2200.0
36.209932857...	0.6175206206...	-50.850743554...	-0.0518086414...	2200.0	3200.0
38.012874113...	-0.1102953868...	-84.323086856...	0.0314691118...	3200.0	5000.0


Cp=a0+a1*1e-3*T+a2*1e5/T^2+a3*1e-5*T^2 Tl<=T<=Th
Cp specific heat KJ/kmolK
37.91
37.03
36.15
35.27
34.40
33.52
32.64
31.76
30.88
30.00
29.12
298.2
768.3
1239
1709
2179
2649
3119
3589
4060
4530
5000
T temperature degree K
Cp=a0+a1*1e-3*T+a2*1e5/T^2+a3*1e-5*T^2 Tl<=T<=Th
Error Cp KJ/kmolK
0.001713
0.001319
0.0009255
0.0005317
0.0001379
-0.0002559
-0.0006496
-0.001043
-0.001437
-0.001831
-0.002225
298.2
768.3
1239
1709
2179
2649
3119
3589
4060
4530
5000
T degree K

data output(veri çıktısı) :

0	1
298.15	29.123
300.0	29.125
400.0	29.246
500.0	29.583
600.0	30.113
700.0	30.752
800.0	31.43
900.0	32.094
1000.0	32.696
1100.0	33.24
1200.0	33.723
1300.0	34.148
1400.0	34.519
1500.0	34.844
1600.0	35.128
1700.0	35.378
1800.0	35.598
1900.0	35.794
2000.0	35.969

6.13 RATIONAL FUNCTION APPROXIMATION : PADE APPROXIMATION METHOD



Figure 6.13.1 Henri Pade

The class of algebraic polynomials has some distinct advantages for use in approximations. The disadvantage of using polynomials for approximation is their tendency to oscilate. This often causes error bounds in polynomial approximation to significantly exceed the avarage approximation error. Another family of functions existed that spread the approximation error more evenly over the approximation interval. This family of functions are called **rational functions.** A rational function r of degree N has the form :

$r(x) = \frac{p(x)}{q(x)}$ (6.13-1)

Where p(x) and q(x) are polynomials whose degrees sum to N.

$$r(x) = \frac{p(x)}{q(x)} = \frac{p_0 + p_1 x + p_2 x^2 + \cdots + p_n x^n}{q_0 + q_1 x + q_2 x^2 + \cdots + q_n x^n} \quad (6.13-2)$$

Pade′ rational approximation is based on Taylor expansion of the functions.

$$f(x) = f(x_0) + f'^{(x_0)}(x - x_0) + \frac{f"(x_0)}{2!}(x - x_0)^2 + \frac{f^{(3)}(x_0)}{3!}(x - x_0)^3 + .. + \frac{f^{(n)}(x_0)}{n!}(x - x_0)^n \quad (6.13-3)$$

The special form of the Taylor expansion when x_0=0 was called MacLaurin expansion.

$$f(x) = f(0) + f'(0)x + \frac{f"(0)}{2!}x^2 + \frac{f^{(3)}(0)}{3!}x^3 + .. + \frac{f^{(n)}(0)}{n!}x^n \quad (6.13-4)$$

$$f(x) = \sum_{i=0}^{\infty} a_i x^i \quad (6.13-5)$$

Now, if the function f(x) is opened to a MacLaurin series of degree N

$$r(x) = a_0 + a_1 x + a_2 x^2 + a_3 x^3 + \cdots + a_N x^N$$

Consider the error term difference of the function f(x) and r(x) due to finite series approximation of the function.

$$f(x) - r(x) = f(x) - \frac{p(x)}{q(x)} = \frac{f(x)q(x) - p(x)}{q(x)} = \frac{\sum_{i=0}^{\infty} a_i x^i \sum_{i=0}^{\infty} q_i x^i - \sum_{i=0}^{\infty} p_i x^i}{q(x)} \quad (6.13-6)$$

We can assume that q_0=1. The object is to chose the constants $q_1, q_2, \ldots, q_m$ and $p_1, p_2, \ldots, p_n$ so that the right hand term of the above equation

$$(a_0 + a_1 x + a_2 x^2 + \cdots)(1 + q_1 x + q_2 x^2 + \cdots + q_n x^m) - (p_0 + p_1 x + p_2 x^2 + \cdots + p_n x^n) \quad (6.13-7)$$

has no term of degree less than or equal to N. To simplify the notation, we define $p_{n+1}=p_{n+2}=\ldots p_N=0$ and define $q_{m+1}=q_{m+2}=\ldots q_m=0$. We can then express the coefficient of x^k in the expression as

$$\left(\sum_{i=0}^{k} a_i q_{k-i}\right) - p_k \quad (6.13-8)$$

So the rational function pade approximation results from the solution of n+1 linear equation

$$\left(\sum_{i=0}^{k} a_i q_{k-i}\right) = p_k \quad k = 0, \ldots, N$$

In the N+1 unknowns $q_1, q_2, \ldots, q_m, p_0, p_1, \ldots, p_m,$

As a first example take function $f(x) = e^x$

MacLaurin expansion of the function $f(x) = e^x$ for error $O(x^5)$,

$f(x) = e^x, \quad f(0) = 1$

$f'(x) = e^x, \quad f'(0) = 1$

$f"(x) = e^x, \quad f"(0) = 1$

$f'''(x) = e^x, \quad f'''(0) = 1$

$f""(x) = e^x, \quad f""(0) = 1$

Maclaurin equation then becomes;

$f(x) = e^x = 1 + x + \frac{x^2}{2} + \frac{x^3}{6} + \frac{x^4}{24} + O(x^5)$

$p(x) = p_0 + xp_1 + x^2 p_2$

$q(x) = 1 + xq_1 + x^2 q_2$

$f(x)q(x) - p(x) = 0$

$$(1-p_0)+(1-p_1+q_1)x+\left(\frac{1}{2}-p_2+q_1+q_2\right)x^2+\left(\frac{1}{6}+\frac{q_1}{2}+q_2\right)x^3+\left(\frac{1}{24}+\frac{q_1}{6}+\frac{q_2}{2}\right)x^4+O(x^4)=0$$

$(1-p_0)=0$

$(1-p_1+q_1)=0$

$\left(\frac{1}{2}-p_2+q_1+q_2\right)=0$

$\left(\frac{1}{6}+\frac{q_1}{2}+q_2\right)=0$

$\left(\frac{1}{24}+\frac{q_1}{6}+\frac{q_2}{2}\right)=0$

Solution of the equation set gives:

$p_0=1\ \ p_1=1/2\ \ \ p_1=1/12\ \ \ q_1=-1/2\ \ \ q_2=1/12$

And $r(x)=\frac{1+\frac{1}{2x}+\frac{1}{12}x^2}{1-\frac{1}{2x}+\frac{1}{12}x^2}$

by simplifying

$r(x)=\frac{12+6x+x^2}{12-6x+x^2}$

Maximum error for x=1 is found as

$$f(x)-r(x)=e^x-\frac{12+6x+x^2}{12-6x+x^2}=0.00399611$$

Let us solve the same equation, by developing the computer code. In the code MacLauring derivative terms are calculated up to term 8. But also another input possibility of directly inputting the polynomial coefficients are given. In case of higher degree of polynomials, coefficients can directly be inputted into the program.

Program 6.13-1 Pade rational function approximation program pade.java

```
import java.io.*;
import javax.swing.*;
class ff extends f_x
{
public double func(double x)
{return Math.exp(x);}
}

//Pade approximation
class pade
{
public double q[];
public double p[];
public double a[];

public static double[] Maclaurin(f_x f,double x,int n)
{
double a[]=new double[n+1];
double fact=1.0;
for(int i=0;i<=n;i++)
{a[i]=f.dfunc(x,i)/fact;fact*=(i+1);}
return a;
}

public pade(f_x f,int n,int m)
{
int N=m+n;
double a[]=Maclaurin(f,0.0,N);
initilize(a,n,m);
}

public pade(double a[],int n,int m)
{initilize(a,n,m);}
```

```
public void initilize(double ai[],int n,int m)
{
int N=m+n;
int NP1=N+1;
a=new double[NP1];
for(int i=0;i<NP1;i++)
{a[i]=ai[i];}
q=new double[m+1];
p=new double[n+1];
q[0]=1.0;
p[0]=a[0];
int i,j;
double b[][]=new double[N+1][N+1];
double bb[][]=new double[N][N];
double c[]=new double[N];
for(i=1;i<=N;i++)
{ for(j=1;j<(i-1);j++)
  {if(j<=n) b[i][j]=0;}
  if(i<=n) b[i][i]=1;
  for(j=i+1;j<=N;j++)
  {b[i][j]=0;}
  for(j=1;j<=i;j++)
  {if(j<=m)b[i][n+j]=-a[i-j];}
  for(j=n+i+1;j<=N;j++)
  {b[i][j]=0;}
  c[i-1]=a[i];
}
for(i=0;i<N;i++)
{for(j=0;j<N;j++)
 {bb[i][j]=b[i+1][j+1];}
}
double x[]=pivotlugauss(bb,c);
for(i=0;i<n;i++)
{p[i+1]=x[i];}
for(i=n;i<N;i++)
{q[i+1-n]=x[i];}
}

  public static double[] pivotlugauss(double a[][],double b[])
  { //gauss elimination with partial pivoting
  int n=b.length;
  double x[]=new double[n];
  double carpan=0;
  double toplam=0;
  double buyuk;
  double dummy=0;
  //gauss elimination
  int i,j,k,p,ii,jj;
  for(k=0;k<(n-1);k++)
  { //pivoting
                p=k;
                buyuk=Math.abs(a[k][k]);
                for(ii=k+1;ii<n;ii++)
                {  dummy=Math.abs(a[ii][k]);
                   if(dummy > buyuk) {buyuk=dummy;p=ii;}
                }
                if(p!=k)
                { for(jj=k;jj<n;jj++)
                  { dummy=a[p][jj];
                    a[p][jj]=a[k][jj];
                    a[k][jj]=dummy;
                  }
                  dummy=b[p];
                  b[p]=b[k];
```

```
                b[k]=dummy;
               }
   for(i=k+1;i<n;i++)
   {  carpan=a[i][k]/a[k][k];
     a[i][k]=0;
     for(j=k+1;j<n;j++)
     {   a[i][j]-=carpan*a[k][j]; }
        b[i]   =b[i]   -carpan*b[k];
               }
  }
  x[n-1]=b[n-1]/a[n-1][n-1];
  for(i=n-2;i>=0;i--)
  {
   toplam=0;
   for(j=i+1;j<n;j++)
   {  toplam+=a[i][j]*x[j];}
   x[i]=(b[i]-toplam)/a[i][i];
  }
  return x;
  }

public double func(double x)
{
return poly_func(p,x)/poly_func(q,x);
}

public static double poly_func(double e[],double x)
{
// this function calculates the value of
// least square curve fitting function
int n=e.length;
double ff;
if(n!=0.0)
  { ff=e[n-1];
  for(int i=n-2;i>=0;i--)
   { ff=ff*x+e[i]; }
  }
else
  ff=0;
return ff;
}

public String toString()
{
               String s="a : \n"+Matrix.toStringT(a)+"\n";
                    s+="p : \n"+Matrix.toStringT(p)+"\n";
                    s+="q : \n"+Matrix.toStringT(q)+"\n";
               return s;
}

public String toString(f_x f,double a,double b,int n)
{

return Matrix.toStringT(result(f,a,b,n));
}

public double[][] result(f_x f,double a,double b,int n)
{   double xx[][]=new double[4][n+1];
   for(int i=0;i<=n;i++)
               { xx[0][i]=a+i*(b-a)/n;
                 xx[1][i]=func(xx[0][i]);
                 xx[2][i]=f.func(xx[0][i]);
                 xx[3][i]=xx[2][i]-xx[1][i];
               }
```

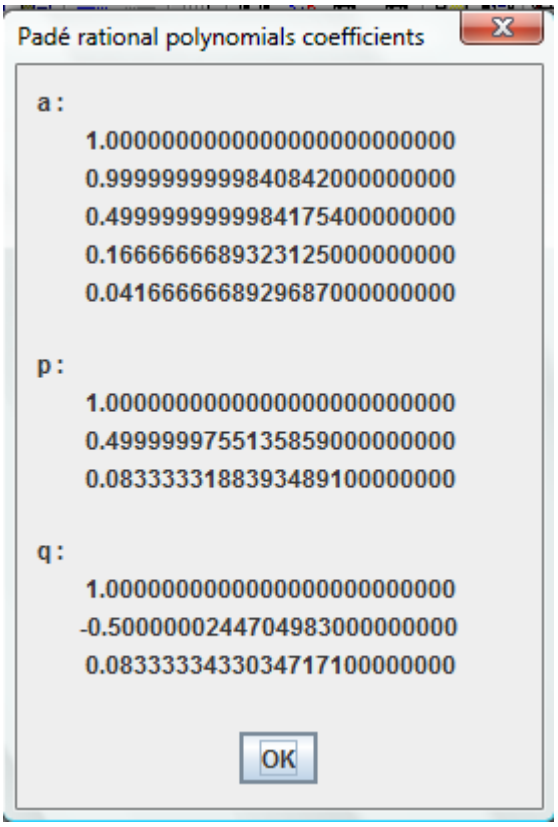
```
            return xx;
}
public void plot_func(f_x f,double a,double b)
{           double xx[]=new double[101];
   double yy[]=new double[101];
            for(int i=0;i<=100;i++)
            {xx[i]=a+i*(b-a)/100.0;yy[i]=func(xx[i]);}
            Plot pp=new Plot(xx,yy);
   pp.setPlabel("Pad"+'\u00E9'+" rational approximation function");
            pp.plot();}

            public void plot_error(f_x f,double a,double b)
   { double xx[]=new double[101];
     double yy[]=new double[101];
            for(int i=0;i<=100;i++)
            {xx[i]=a+i*(b-a)/100.0;yy[i]=f.func(xx[i])-func(xx[i]);}
            Plot pp=new Plot(xx,yy);
   pp.setPlabel("Pad"+'\u00E9'+" rational approximation error");
   pp.setYlabel("error");
            pp.plot();
}

public static void print(String s,String s1)
{JOptionPane.showMessageDialog(null,s,s1,JOptionPane.PLAIN_MESSAGE);}

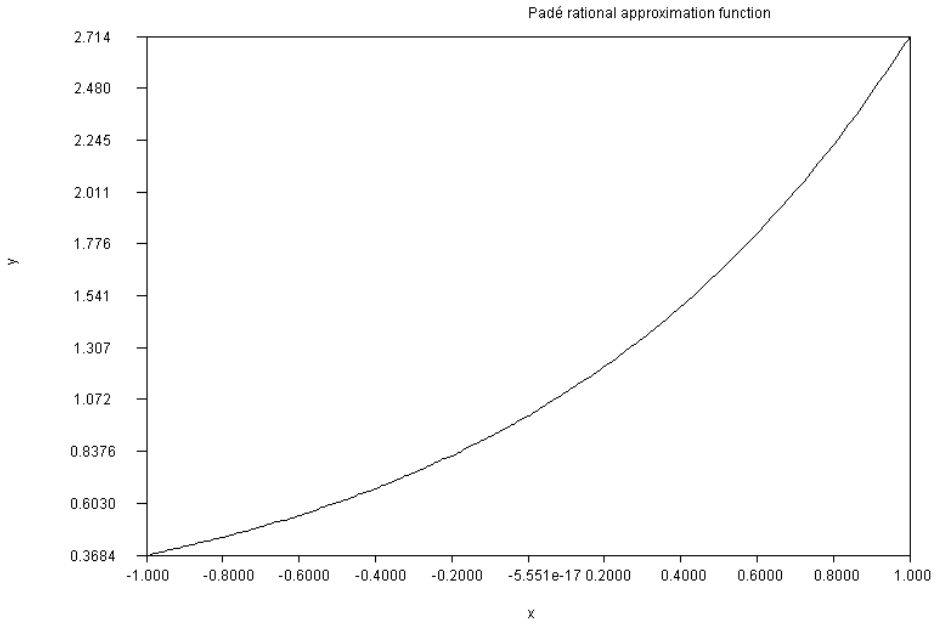
public static void main(String arg[])
{ ff f=new ff();
  //coefficients of function exp(x)
double a[]={1.00000000000000000000,
1.00000000000000000000,
0.50000000000000000000,
0.16666666666667000000,
0.04166666666666700000,
0.00833333333333330000,
0.00138888888888890000,
0.00019841269841269800,
0.00002480158730158730,
0.00000275573192239859,
0.00000027557319223985,
0.00000002505210838544,
0.00000000208767569878,
0.00000000016059043836,
0.00000000001147074559,
0.00000000000076471637,
0.00000000000004779477,
0.00000000000000281145,
0.00000000000000015619,
};
            int n=8;
            int m=8;
            pade Pa=new pade(f,n,m);
            //pade Pa1=new pade(a,n,m);
            print(Pa.toString(),"Pad"+'\u00E9'+" rational polynomials coefficients");
            String s1[]={"x","Pad"+'\u00E9'+" f(x)=p(x)/q(x)","f(x)","error"};
            Text.print(Text.T(Pa.result(f,-1.0,1.0,20)),s1,"Pad"+'\u00E9'+" rational polynomials n = "+n+" m = "+m);
            Pa.plot_func(f,-1.0,1.0);
            Pa.plot_error(f,-1.0,1.0);
}

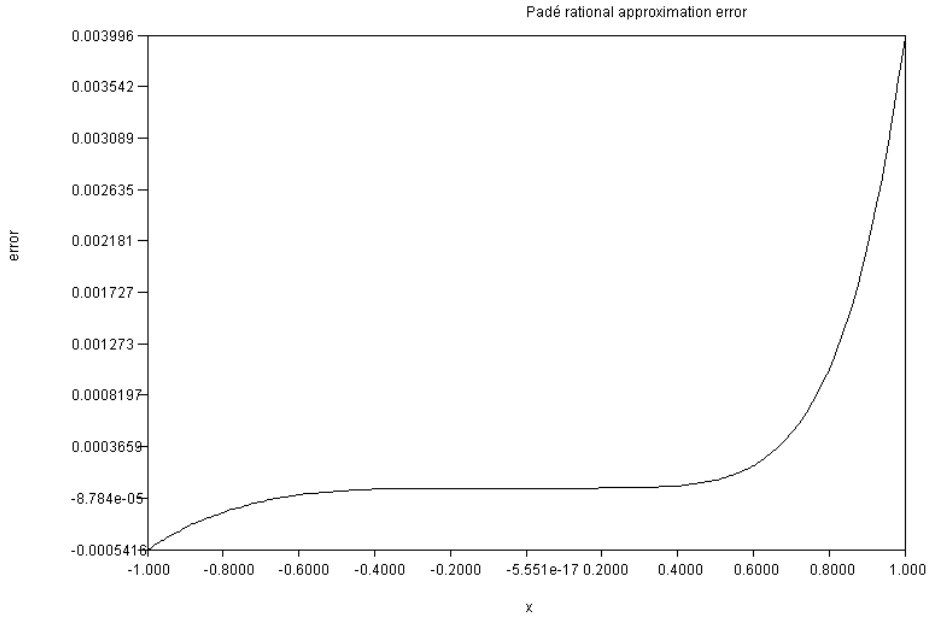
}
```



Padé rational polynomials n = 2 m = 2

x	Padé f(x)=p(x)/q(x)	f(x)	error
-1.0	0.3684210509286994	0.36787944117144233	-5.416097572570888E-4
-0.9	0.4069192738383368	0.4065696597405991	-3.496140977377049E-4
-0.8	0.4495412834707172	0.44932896411722156	-2.1231935349563802E-4
-0.7	0.4967046128965557	0.4965853037914095	-1.1930910514618143E-4
-0.6	0.5488721800340369	0.5488116360940265	-6.0543940010426844E-5
-0.5	0.6065573768024644	0.6065306597126334	-2.6717089830929908E-5
-0.4	0.6703296702018267	0.6703200460356393	-9.624166187327532E-6
-0.30000000000000004	0.7408207342878982	0.7408182206817178	-2.5136061804209575E-6
-0.19999999999999996	0.8187311178103238	0.8187307530779819	-3.647323418842774E-7
-0.09999999999999998	0.9048374306099658	0.9048374180359596	-1.2574006169607799E-8
0.0	1.0	1.0	0.0
0.10000000000000009	1.1051709027175882	1.1051709180756477	1.535805949082203E-8
0.19999999999999996	1.221402214037107	1.2214027581601699	5.441230628289162E-7
0.30000000000000004	1.3498542274610588	1.3498588075760032	4.5801149444191225E-6
0.3999999999999999	1.4918032788221807	1.4918246976412701	2.1418819089458196E-5
0.5	1.6486486489024128	1.6487212707001282	7.262179771538868E-5
0.6000000000000001	1.8219178086314263	1.822118800390509	2.0099175908283584E-4
0.7	2.013268999384152	2.0137527074704766	4.8370808632469675E-4
0.8	2.2244897966653	2.225540928492468	0.0010511318271677617
0.8999999999999999	2.457489879349433	2.4596031111569494	0.0021132318075163248
1.0	2.7142857149332364	2.7182818284590455	0.003996113525809086





6.14 RATIONAL FUNCTION APPROXIMATION : CHEBYSCHEV APPROXIMATION METHOD

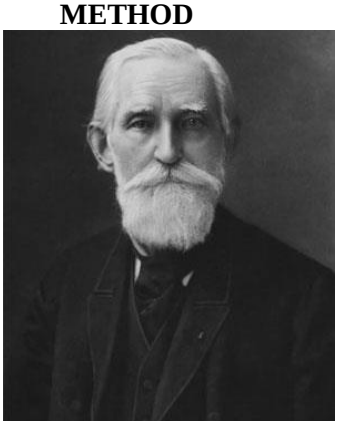


Figure 6.14.1 Pafnuty Chebyshev, Russian Mathematician

Pade approximation based on Taylor expansion of the functions. Taylor expansion has a wide variation of accuracy. To obtain more uniformly accurate rational-function approximation Chebyschev polynomials can be used. In chapter 6.2 Chebyschev polynomials were defined and used as orthogonal least square curve fitting functions.

$T_n(x)=\cos(n \arccos(x))$ (6.14-1)

If this equation is written in open form

$T_0(x)=1$

$T_1(x)=x$

$T_2(x)=2x^2-x$

…..

$T_{n+1}(x)=2x\ T_n(x)- T_{n-1}(x)$ (6.14-2)

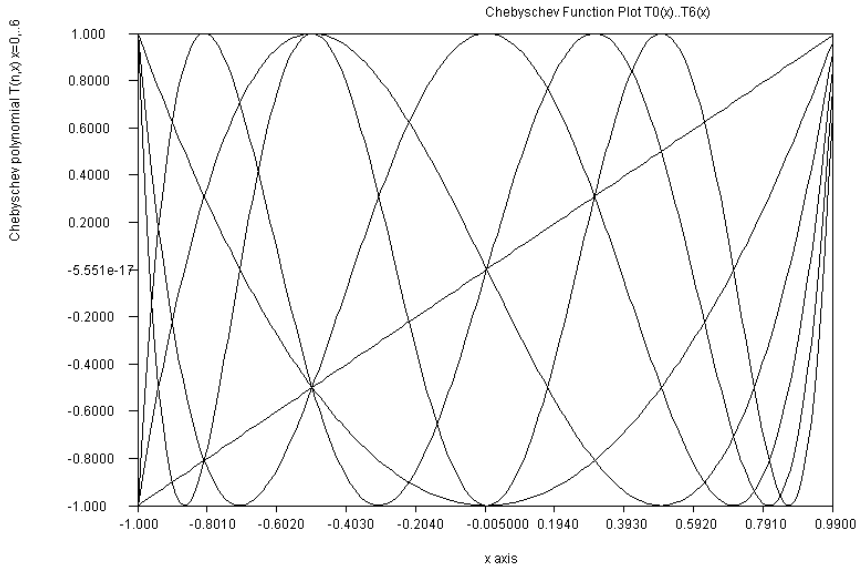


Figure 6.14-1 Chebyschev polynomials $T_0(x)$ through $T_6(x)$

Now if x^k terms of the pade approximations is replaced by Chebyschev polynomial $T_k(x)$ Chebyschev approximation is constructed. Suppose we want to approximate the function f by an N-th degree rational function r written in the form

$$r(x) = \frac{p(x)}{q(x)} = \frac{\sum_{k=0}^{n} p_k T_k(x)}{\sum_{k=0}^{n} q_k T_k(x)} \quad (6.14\text{-}3)$$

where N=n+m and $q_0=1$

Writing f(x) in a series involving Chebyschev polynomials as:

$f(x) = \sum_{k=0}^{n} a_k T_k(x)$ gives

$$f(x) - r(x) = f(x) - \frac{p(x)}{q(x)} = \frac{\sum_{k=0}^{n} a_k T_k(x) \sum_{k=0}^{n} q_k T_k(x) - \sum_{k=0}^{n} p_k T_k(x)}{\sum_{k=0}^{n} q_k T_k(x)} \quad (6.14-4)$$

The coefficients $q_1,q_2,\ldots,q_m$ and $p_1,p_2,\ldots,p_n$ are chosen so that the numerator on the right hand side of the equation has zero coefficients for $T_k(x)$ when k=0,1,..N. This implies that the series

$$(a_0T_0(x) + a_1T_1(x)+..)\big(T_0(x) + q_1T_1(x)+..+q_mT_m(x)\big) - \big(p_0T_0(x) + p_1T_1(x)+..+p_mT_m(x)\big) \quad (6.14-5)$$

Has no terms of degree less then or equal to N.

Solution of this equation involves few difficulties, the first one is the requirement of multiplication of Chebyschev terms. This problem is resolved by making use of the relationship

$$T_i(x)T_j(x) = 0.5\big[T_{i+1}(x) + T_{|i-j|}(x)\big] \quad (6.14-6)$$

Coefficients need to be calculated by using integration equations:

$$a_0 = \frac{1}{\pi}\int_{x=-1}^{1}\frac{f(x)}{\sqrt{1-x^2}}dx = \frac{1}{\pi}\int_{\theta=0}^{\pi} f(\cos(\theta))d\theta \quad (6.14-7)$$

$$a_k = \frac{1}{\pi}\int_{x=-1}^{1}\frac{f(x)T_k(x)}{\sqrt{1-x^2}}dx = \frac{1}{\pi}\int_{\theta=0}^{\pi} f(\cos(\theta))\cos(k\theta)\,d\theta \quad (6.14-8)$$

These are difficult integrals to solve analytically. Usually Numerical methods are involved to solve the integrals. Please refer Chapter 7 for the details of the numerical integrals (everything is related with each other in mathematics and we should give topics in some order). In our example code, Gander and Gautschi adaptive Gauss-Lobatto integral formula defined in chapter 7.5-5 is used as an accurate integration formula to solve coefficients.

Program 6.14-1 Chebyschev rational function approximation program Chebyshev.java

```
import java.io.*;
import javax.swing.*;
class ff extends f_x
{
public double func(double x)
{return Math.exp(x);}
}

//Chebyshev  rational approximation
class Chebyshev
{
public double q[];
public double p[];
public double a[];

            public static double Ti(double x,int k)
            {//Chebysev function
            double T[]=new double[k+1];
            T[0]=1.0;
            if(k>=1) T[1]=x;
            for(int i=2;i<=k;i++) {T[i]=2.0*x*T[i-1]-T[i-2];
   }
            return T[k];
            }

   public  double fk(f_x f,int k,double x)
   {   double a=0;
      double mult=2.0/Math.PI;
               a=mult*f.func(Math.cos(x))*Math.cos(k*x);
               return a;
            }

public double adaptive_Lobatto_integral(f_x f,double a,double b,int k)
{
            int i;
            double m=(a+b)/2.0; double h=(b-a)/2.0;
   double alpha=Math.sqrt(2.0/3.0);
   double beta=1.0/Math.sqrt(5.0);
   double x1,x2,x3;
   x1=0.94288241569547971905635175843185720232;
   x2=0.64185334234578130578123554132903188354;
```

```
    x3=0.23638319966214988028222377349205292599;
    double A,B,C,D,E,F,G;
    A=0.01582719197348018308716998673330551059l;
    B=0.09427384021885004553128250507710817l960;
    C=0.15507198733658539625363597980210298680;
    D=0.18882157396018245442000533937297167125;
    E=0.19977340522685852679206802206648840246;
    F=0.22492646533333952701601768799639508076;
    G=0.24261107190140773379964095790325635233;
            double x[]={a,(m-x1*h),(m-alpha*h),(m-x2*h),(m-beta*h),(m-
x3*h),m,(m+x3*h),(m+beta*h),(m+x2*h),(m+alpha*h),(m+x1*h),b};
            int n=x.length;
            double y[]=new double[n];
            for(i=0;i<n;i++) y[i]=fk(f,k,x[i]);
            double fa=y[0];
            double fb=y[12];
    double i2=(h/6.0)*(y[0]+y[12]+5.0*(y[4]+y[8]));
    double i1=(h/1470.0)*(77.0*(y[0]+y[12])+432.0*(y[2]+y[10])+
     625.0*(y[4]+y[8])+672.0*y[6]);
    double is=h*(A*(y[0]+y[12])+B*(y[1]+y[11])+C*(y[2]+y[10])+D*(y[3]+y[9])+E*(y[4]+y[8])+F*(y[5]+y[7])+G*y[6]);
    int s=(int)(is/Math.abs(is));
    if(s==0) s=1;
    double erri1=Math.abs(i1-is);
    double erri2=Math.abs(i2-is);
    double R;
    R=erri1/erri2;
    //if(R>0 && R<1) tol=tol/R;
    //is=s*Math.abs(is)*tol/eps;
    if(is==0) is=b-a;
  double Q=0;
  Q=Lobatto(f,a,b,fa,fb,is,k);
  return Q;
}

public  double Lobatto(f_x f,double a,double b,double fa,double fb,double is,int k)
{
  double h=(b-a)/2;
  double m=(a+b)/2;
  double alpha=Math.sqrt(2.0/3.0);
  double beta=1.0/Math.sqrt(5.0);
  double mll=m-alpha*h;
  double ml=m-beta*h;
  double mr=m+beta*h;
  double mrr=m+alpha*h;
  double x[]={mll,ml,m,mr,mrr};
  int n=x.length;
  double y[]=new double[n];
  for(int i=0;i<n;i++) y[i]=fk(f,k,x[i]);
  double fmll=y[0];
  double fml=y[1];
  double fm=y[2];
  double fmr=y[3];
  double fmrr=y[4];
  double Q;
  double i2=(h/6)*(fa+fb+5*(fml+fmr));
  double i1=(h/1470)*(77*(fa+fb)+432*(fmll+fmrr)+625*(fml+fmr)+672*fm);
  boolean c1,c2,c3;
```

```
  double w1=i1-i2;;
  double w2=is;
  double w3=w2+w1;
  c1=(w3==w2);
  c2=(mll<=a);
  c3=(b<=mrr);
  if( c1|| c2 ||c3 )
  {
   if (((m <= a) || (b<=m)))
   {
      System.out.println("istenilen toleransa ulaşılamadı");
   }
   Q=i1;
  }
  else
  {
    Q=Lobatto(f,a,mll,fa,fmll,is,k) + Lobatto(f,mll,ml,fmll,fml,is,k)+
      Lobatto(f,ml,m,fml,fm,is,k)   + Lobatto(f,m,mr,fm,fmr,is,k)+
      Lobatto(f,mr,mrr,fmr,fmrr,is,k)+Lobatto(f,mrr,b,fmrr,fb,is,k);
  }
  return Q;
}
//=============
public Chebyshev(f_x f,int n,int m)
{
int N=m+n;
int Npm=N+m;
double ai[]=new double[Npm+1];
int k;
for(k=0;k<=(N+m);k++)
{ai[k]=adaptive_Lobatto_integral(f,0.0,Math.PI,k);
}
initilize(ai,n,m);
}

public Chebyshev(double ai[],int n,int m)
{initilize(ai,n,m);}

public void initilize(double ai[],int n,int m)
{
int N=m+n;
int NP1=N+1;
int NPm=N+m;
int mp1=m+1;
int np1=n+1;
a=new double[NPm+1];
for(int i=0;i<=NPm;i++)
{a[i]=ai[i];}
q=new double[mp1];
p=new double[np1];
q[0]=1.0;
int i,j;
double b[][]=new double[NP1][NP1];
double c[]=new double[NP1];
for(i=0;i<=N;i++)
{  for(j=0;j<=i;j++)
   {if(j<=n) b[i][j]=0;}
```

```
if(i<=n) b[i][i]=1;
   for(j=i+1;j<=n;j++)
   {b[i][j]=0;}
   int k1,k2;
   for(j=n+1;j<=N;j++)
   {k1=i+j-n;
    k2=Math.abs(i-j+n);
   if(i!=0) b[i][j]=-0.5*(a[k1]+a[k2]);
   else     b[i][j]=-0.5*a[j-n];
  }
  if(i!=0) c[i]=a[i];
  else     c[i]=0.5*a[i];
}
double x[]=pivotlugauss(b,c);
for(i=0;i<=n;i++)
{p[i]=x[i];}
for(i=n+1;i<=N;i++)
{q[i-n]=x[i];}
}

  public static double[] pivotlugauss(double a[][],double b[])
  { //gauss elimination with partial pivoting
  int n=b.length;
  double x[]=new double[n];
  double carpan=0;
  double toplam=0;
  double buyuk;
  double dummy=0;
  //gauss elimination
  int i,j,k,p,ii,jj;
  for(k=0;k<(n-1);k++)
  { //pivoting
              p=k;
              buyuk=Math.abs(a[k][k]);
              for(ii=k+1;ii<n;ii++)
              {  dummy=Math.abs(a[ii][k]);
                 if(dummy > buyuk) {buyuk=dummy;p=ii;}
              }
              if(p!=k)
              { for(jj=k;jj<n;jj++)
                { dummy=a[p][jj];
                  a[p][jj]=a[k][jj];
                  a[k][jj]=dummy;
                }
                dummy=b[p];
                b[p]=b[k];
                b[k]=dummy;
              }
    for(i=k+1;i<n;i++)
    {  carpan=a[i][k]/a[k][k];
       a[i][k]=0;
       for(j=k+1;j<n;j++)
       {   a[i][j]-=carpan*a[k][j]; }
           b[i]   =b[i]   -carpan*b[k];
              }
  }
  x[n-1]=b[n-1]/a[n-1][n-1];
```

```
  for(i=n-2;i>=0;i--)
  {
    toplam=0;
    for(j=i+1;j<n;j++)
    {  toplam+=a[i][j]*x[j];}
    x[i]=(b[i]-toplam)/a[i][i];
  }
  return x;
  }

public double func(double x)
{
return poly_func(p,x)/poly_func(q,x);
}

public static double poly_func(double a[],double x)
{
// this function calculates the value of
// least square curve fitting function
int n=a.length;
double b=0;
for(int k=0;k<n;k++)
{b+=a[k]*Ti(x,k);}
return b;
}

public void plot_func(f_x f,double a,double b)
{           double xx[]=new double[101];
   double yy[]=new double[101];
            for(int i=0;i<=100;i++)
            {xx[i]=a+i*(b-a)/100.0;yy[i]=func(xx[i]);}
            Plot pp=new Plot(xx,yy);
   pp.setPlabel("Chebyshev rational approximation function");
            pp.plot();}

            public void plot_error(f_x f,double a,double b)
   { double xx[]=new double[101];
     double yy[]=new double[101];
            for(int i=0;i<=100;i++)
            {xx[i]=a+i*(b-a)/100.0;yy[i]=f.func(xx[i])-func(xx[i]);}
            Plot pp=new Plot(xx,yy);
   pp.setPlabel("Chebyshev rational approximation error");
   pp.setYlabel("error");
            pp.plot();
}

public String toString()
{
            String s="p : \n"+Matrix.toStringT(p)+"\n";
                s+="q : \n"+Matrix.toStringT(q)+"\n";
            return s;
}

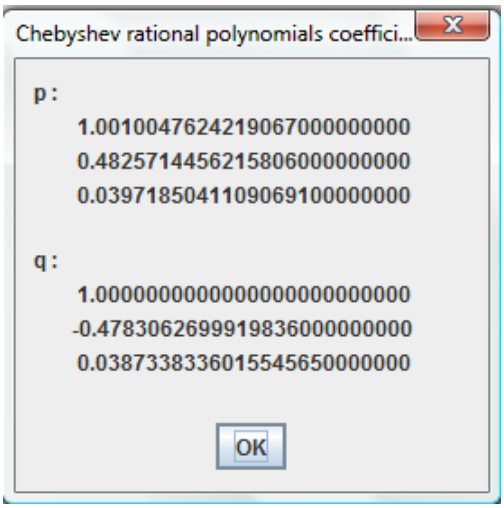
public String toString(f_x f,double a,double b,int n)
{

return Matrix.toStringT(result(f,a,b,n));
```

```
}

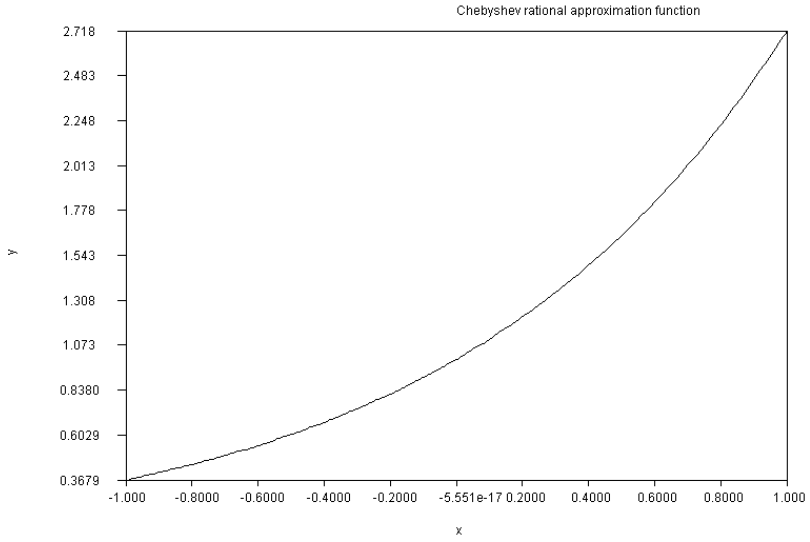
public double[][] result(f_x f,double a,double b,int n)
{   double xx[][]=new double[4][n+1];
   for(int i=0;i<=n;i++)
            { xx[0][i]=a+i*(b-a)/n;
              xx[1][i]=func(xx[0][i]);
              xx[2][i]=f.func(xx[0][i]);
              xx[3][i]=xx[2][i]-xx[1][i];
            }
            return xx;
}
public static void print(String s,String s1)
{JOptionPane.showMessageDialog(null,s,s1,JOptionPane.PLAIN_MESSAGE);}
public static void main(String arg[])
{
            ff f=new ff();
            int n=2;
            int m=2;
            Chebyshev Pa=new Chebyshev(f,n,m);
            print(Pa.toString(),"Chebyshev rational polynomials coefficients");
            String s1[]={"x","Chebyschev f(x)=p(x)/q(x)","f(x)","error"};
            Text.print(Text.T(Pa.result(f,-1.0,1.0,20)),s1,"Chebyshev rational polynomials n = "+n+" m = "+m);
            Pa.plot_func(f,-1.0,1.0);
            Pa.plot_error(f,-1.0,1.0);
}

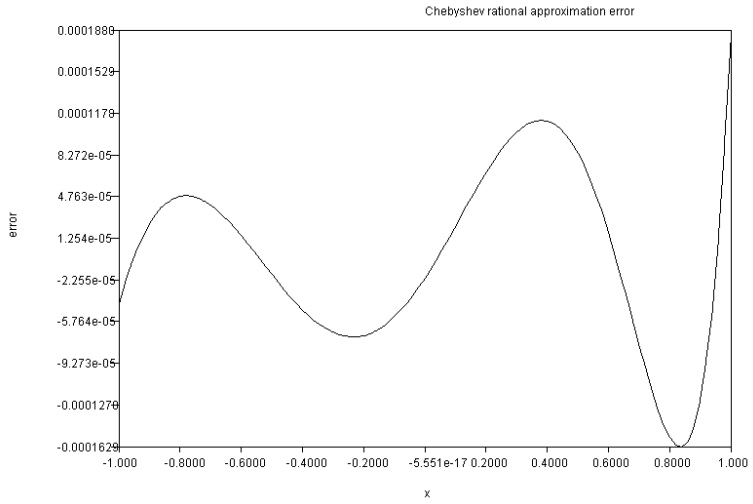
}
```



Chebyshev rational polynomials n = 2 m = 2

x	Chebyschev f(x)=p(x)/q(x)	f(x)	error
-1.0	0.3679215991647767	0.36787944117144233	-4.215799333434633E-5
-0.9	0.4065450308521668	0.4065696597405991	2.4628888432298712E-5
-0.8	0.44928099031688123	0.44932896411722156	4.79738003403285E-5
-0.7	0.49654472373346964	0.4965853037914095	4.058005793988739E-5
-0.6	0.5487968820945643	0.5488116360940265	1.4753999462158163E-5
-0.5	0.6065487349499269	0.6065306597126334	-1.8075237293468938E-5
-0.4	0.670367949939593	0.6703200460356393	-4.790390395370547E-5
-0.30000000000000004	0.7408849752494372	0.7408182206817178	-6.675456771942923E-5
-0.19999999999999996	0.8188000534499897	0.8187307530779819	-6.930037200780781E-5
-0.09999999999999998	0.9048908805870234	0.9048374180359596	-5.346255106375430 4E-5
0.0	1.0000209015080908	1.0	-2.090150809075375E-5
0.10000000000000009	1.105148198077676	1.1051709180756477	2.2719997971654138E-5
0.19999999999999996	1.2213348769950267	1.2214027581601699	6.788116514311326E-5
0.30000000000000004	1.3497567929670753	1.3498588075760032	1.0201460892789171E-4
0.3999999999999999	1.4917133440587107	1.4918246976412701	1.1135358255942585E-4
0.5	1.6486369404047452	1.6487212707001282	8.433029538301717E-5
0.6000000000000001	1.822101564395447	1.822118800390509	1.7235995062092613E-5
0.7	2.0138295972158415	2.0137527074704766	-7.688974536490534E-5
0.8	2.225695768869679	2.225540928492468	-1.5484037721114063E-4
0.8999999999999999	2.4597266816072687	2.4596031111569494	-1.2357045031929914E-4
1.0	2.71809384667528	2.7182818284590455	1.8798178376533414E-4





6.15 TRIGONOMETRIC FUNCTION (FOURIER) APPROXIMATION



Figure 6.15.1 Jean Baptiste Joseph Fourier 1768-1830, French Mathematician

Using of sine and cosine functions to represent data has a long history. For a function in the range of [-π,π] An approximation polynomial can be written as

$S_n(x) = \frac{a_0}{2} + a_n \cos(nx) + \sum_{k=1}^{n-1}[a_k \cos(kx) + b_k \sin(kx)]$ (6.15-1)

Since this set is orthogonal on [-π,π] range with respect to weight function w(x)=1, the selection of coefficients will be:

$$a_k = \frac{1}{\pi}\int_{-\pi}^{\pi} f(x)\cos(kx)\,dx \quad k = 0,1,2,..,n \qquad (6.15-2)$$

$$b = \frac{1}{\pi}\int_{-\pi}^{\pi} f(x)\sin(kx)\,dx \quad k = 0,1,2,..,n \qquad (6.15-3)$$

The limit of $S_n(x)$ when $n \to \infty$is called **Fourier series** of function f. A more general definition of the fourier series can be given in the range of [-l,l]

$$S(x) = \frac{a_0}{2} + \sum_{k=1}^{\infty}\left[a_k \cos\left(\frac{k\pi x}{l}\right) + b_k \sin\left(\left(\frac{k\pi x}{l}\right)\right)\right] \qquad (6.15-4)$$

$$a_k = \frac{1}{l}\int_{-l}^{l} f(x)\cos\left(\frac{k\pi x}{l}\right)dx = \frac{1}{l}\int_{\alpha}^{\alpha+2l} f(x)\cos\left(\frac{k\pi x}{l}\right)dx \quad k = 0,1,2,..,n \quad (6.15-5)$$

$$b_k = \frac{1}{l}\int_{-l}^{l} f(x)\sin\left(\frac{k\pi x}{l}\right)dx = \frac{1}{l}\int_{\alpha}^{\alpha+2l} f(x)\sin\left(\frac{k\pi x}{l}\right)dx \quad k = 0,1,2,..,n \quad (6.15-6)$$

A Computer program developed to calculate Trigonometric polynomials. Integrations are again taken numerically in this program similar to Chebyschev polynomial rational approximations. The first sample function to be investigated will be f(x)=|x|

Program 6.15-1 Trigonometric function approximation program Trigonometric.java

```
import java.io.*;
import javax.swing.*;
class ff extends f_x
{
public double func(double x)
{return Math.abs(x);}
}

// Trigonometric  rational approximation
class Trigonometric
{
public double a[];
public double b[];
```

```
public  double fak(f_x f,int k,double x)
   {   double a=0;
      double mult=1.0/Math.PI;
               a=mult*f.func(x)*Math.cos(k*x);
               return a;
            }

public  double fbk(f_x f,int k,double x)
   {   double a=0;
      double mult=1.0/Math.PI;
               a=mult*f.func(x)*Math.sin(k*x);
               return a;
            }

public double adaptive_Lobatto_integral(f_x f,double a,double b,int k,int eqn_type)
{
            int i;
            double m=(a+b)/2.0; double h=(b-a)/2.0;
   double alpha=Math.sqrt(2.0/3.0);
   double beta=1.0/Math.sqrt(5.0);
   double x1,x2,x3;
   x1=0.94288241569547971905635175843185720232;
   x2=0.64185334234578130578123554132903188354;
   x3=0.23638319966214988028222377349205292599;
   double A,B,C,D,E,F,G;
   A=0.015827191973480183087169986733305510591;
   B=0.094273840218850045531282505077108171960;
   C=0.155071987336585396253635979802102986 80;
   D=0.188821573960182454420005339372971671 25;
   E=0.199773405226858526792068022066488402 46;
   F=0.224926465333339527016017687996395080 76;
   G=0.242611071901407733799640957903256352 33;
            double x[]={a,(m-x1*h),(m-alpha*h),(m-x2*h),(m-beta*h),(m-
x3*h),m,(m+x3*h),(m+beta*h),(m+x2*h),(m+alpha*h),(m+x1*h),b};
            int n=x.length;
            double y[]=new double[n];
            for(i=0;i<n;i++)
            {
            if(eqn_type==0)
            {y[i]=fak(f,k,x[i]);}
            else
            {y[i]=fbk(f,k,x[i]);}
   }
            double fa=y[0];
            double fb=y[12];
   double i2=(h/6.0)*(y[0]+y[12]+5.0*(y[4]+y[8]));
   double i1=(h/1470.0)*(77.0*(y[0]+y[12])+432.0*(y[2]+y[10])+
    625.0*(y[4]+y[8])+672.0*y[6]);
   double is=h*(A*(y[0]+y[12])+B*(y[1]+y[11])+C*(y[2]+y[10])+D*(y[3]+y[9])+E*(y[4]+y[8])+F*(y[5]+y[7])+G*y[6]);
   int s=(int)(is/Math.abs(is));
   if(s==0) s=1;
   double erri1=Math.abs(i1-is);
   double erri2=Math.abs(i2-is);
   double R;
   R=erri1/erri2;
   //if(R>0 && R<1) tol=tol/R;
   //is=s*Math.abs(is)*tol/eps;
   if(is==0) is=b-a;
  double Q=0;
  Q=Lobatto(f,a,b,fa,fb,is,k,eqn_type);
  return Q;
}

public  double Lobatto(f_x f,double a,double b,double fa,double fb,double is,int k,int eqn_type)
{
  double h=(b-a)/2;
```

```
  double m=(a+b)/2;
  double alpha=Math.sqrt(2.0/3.0);
  double beta=1.0/Math.sqrt(5.0);
  double mll=m-alpha*h;
  double ml=m-beta*h;
  double mr=m+beta*h;
  double mrr=m+alpha*h;
  double x[]={mll,ml,m,mr,mrr};
  int n=x.length;
  double y[]=new double[n];
  for(int i=0;i<n;i++)
            {
   if(eqn_type==0)
            {y[i]=fak(f,k,x[i]);}
            else
            {y[i]=fbk(f,k,x[i]);}
    }
  double fmll=y[0];
  double fml=y[1];
  double fm=y[2];
  double fmr=y[3];
  double fmrr=y[4];
  double Q;
  double i2=(h/6)*(fa+fb+5*(fml+fmr));
  double i1=(h/1470)*(77*(fa+fb)+432*(fmll+fmrr)+625*(fml+fmr)+672*fm);
  boolean c1,c2,c3;
  double w1=i1-i2;;
  double w2=is;
  double w3=w2+w1;
  c1=(w3==w2);
  c2=(mll<=a);
  c3=(b<=mrr);
  if( c1|| c2 ||c3 )
  {
   if (((m <= a) || (b<=m)))
   {
      System.out.println("istenilen toleransa ulaşılamadı");
   }
   Q=i1;
  }
  else
  {
    Q=Lobatto(f,a,mll,fa,fmll,is,k,eqn_type) + Lobatto(f,mll,ml,fmll,fml,is,k,eqn_type)+
      Lobatto(f,ml,m,fml,fm,is,k,eqn_type)   + Lobatto(f,m,mr,fm,fmr,is,k,eqn_type)+
      Lobatto(f,mr,mrr,fmr,fmrr,is,k,eqn_type)+Lobatto(f,mrr,b,fmrr,fb,is,k,eqn_type);
  }
  return Q;
}

//==============


public Trigonometric(f_x f,int n)
{
double ai[]=new double[n+1];
double bi[]=new double[n+1];
for(int k=0;k<n;k++)
{ai[k]=adaptive_Lobatto_integral(f,-Math.PI,Math.PI,k,0);}
bi[0]=0.0;
for(int k=1;k<n;k++)
{bi[k]=adaptive_Lobatto_integral(f,-Math.PI,Math.PI,k,1);}
bi[n]=0;
initilize(ai,bi,n);
}

public Trigonometric(double ai[],double bi[],int n)
```

```
{initilize(ai,bi,n);}

public void initilize(double ai[],double bi[],int n)
{
a=new double[n+1];
b=new double[n+1];
for(int i=0;i<=n;i++)
{a[i]=ai[i];b[i]=bi[i];}
}

public  double func(double x)
{
int n=a.length-1;
double cc=a[0]/2.0+a[n]*Math.cos(n*x);
for(int k=1;k<n;k++)
{cc+=a[k]*Math.cos(k*x)+b[k]*Math.sin(k*x);}
return cc;
}

public void plot_func(f_x f,double x0,double xn)
{           double xx[]=new double[101];
   double yy[]=new double[101];
            for(int i=0;i<=100;i++)
            {xx[i]=x0+i*(xn-x0)/100.0;yy[i]=func(xx[i]);}
            Plot pp=new Plot(xx,yy);
            pp.addFunction(f,x0,xn,100);
   pp.setPlabel("Trigonometric rational approximation function n="+(a.length-1));
            pp.plot();
}

            public void plot_error(f_x f,double x0,double xn)
   { double xx[]=new double[101];
     double yy[]=new double[101];
            for(int i=0;i<=100;i++)
            {xx[i]=x0+i*(xn-x0)/100.0;yy[i]=f.func(xx[i])-func(xx[i]);}
            Plot pp=new Plot(xx,yy);
   pp.setPlabel("Trigonometric rational approximation error n = "+(a.length-1));
   pp.setYlabel("error");
            pp.plot();
}

public String toString()
{
            String s="a : \n"+Matrix.toStringT(a)+"\n";
                 s+="b : \n"+Matrix.toStringT(b)+"\n";
            return s;
}

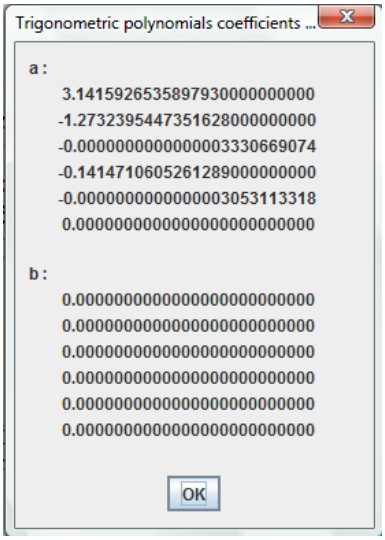
public String toString(f_x f,double a,double b,int n)
{

return Matrix.toStringT(result(f,a,b,n));
}

public double[][] result(f_x f,double a,double b,int n)
{   double xx[][]=new double[4][n+1];
   for(int i=0;i<=n;i++)
            { xx[0][i]=a+i*(b-a)/n;
              xx[1][i]=func(xx[0][i]);
              xx[2][i]=f.func(xx[0][i]);
              xx[3][i]=xx[2][i]-xx[1][i];
            }
            return xx;
}
```

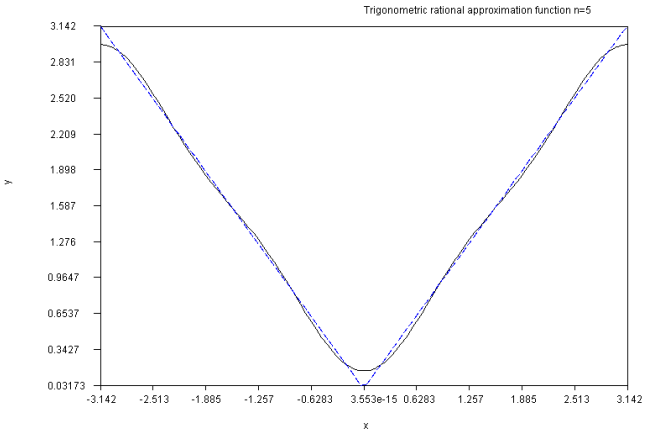
```
public static void print(String s,String s1)
{JOptionPane.showMessageDialog(null,s,s1,JOptionPane.PLAIN_MESSAGE);}
public static void main(String arg[])
{
        ff f=new ff();
        int n=5;
        Trigonometric Pa=new Trigonometric(f,n);
        print(Pa.toString(),"Trigonometric polynomials coefficients n="+n);
        String s1[]={"x","Trigonometric f(x)","f(x)","error"};
        Text.print(Text.T(Pa.result(f,-Math.PI,Math.PI,20)),s1,"Trigonometric polynomials n = "+n);
        Pa.plot_func(f,-Math.PI,Math.PI);
        Pa.plot_error(f,-Math.PI,Math.PI);
}

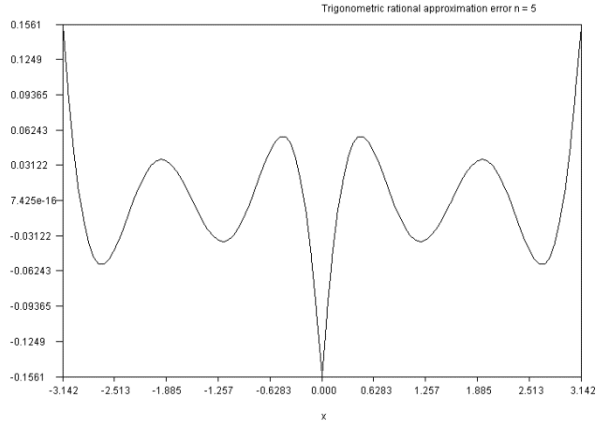
}
```



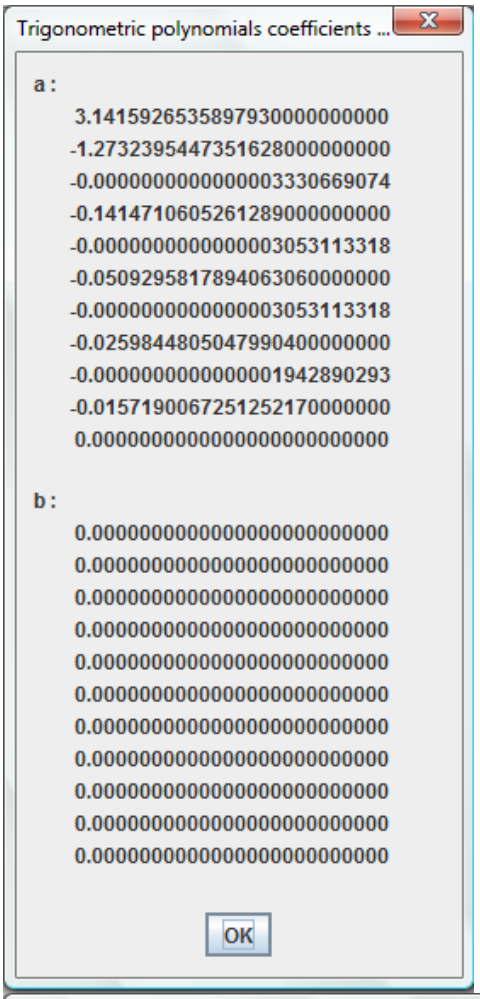
Trigonometric polynomials n = 5

x	Trigonometric f(x)	f(x)	error
-3.141592653589793	2.9855069320561873	3.141592653589793	0.15608572153360578
-2.827433388230814	2.864873695623382	2.827433388230814	-0.037440307392567984
-2.5132741228718345	2.5571517944810442	2.5132741228718345	-0.043877671609209745
-2.199114857512855	2.184640779845247	2.199114857512855	0.01447407766760822
-1.8849555921538759	1.849796491850398	1.8849555921538759	0.03515910030347791
-1.5707963267948966	1.5707963267948968	1.5707963267948966	-2.220446049250313E-16
-1.2566370614359172	1.2917961617393954	1.2566370614359172	-0.03515910030347813
-0.9424777960769379	0.9569518737445465	0.9424777960769379	-0.014474077667608554
-0.6283185307179586	0.5844408591087495	0.6283185307179586	0.04387767160920908
-0.3141592653589793	0.27671895796641055	0.3141592653589793	0.03744030739256876
0.0	0.15608572153360425	0.0	-0.15608572153360425
0.31415926535897887	0.27671895796641016	0.31415926535897887	0.037440307392568706
0.6283185307179586	0.5844408591087495	0.6283185307179586	0.04387767160920908
0.9424777960769379	0.9569518737445465	0.9424777960769379	-0.014474077667608554
1.2566370614359172	1.2917961617393954	1.2566370614359172	-0.03515910030347813
1.5707963267948966	1.5707963267948968	1.5707963267948966	-2.220446049250313E-16
1.8849555921538759	1.849796491850398	1.8849555921538759	0.03515910030347791
2.199114857512855	2.184640779845247	2.199114857512855	0.01447407766760822
2.5132741228718345	2.5571517944810442	2.5132741228718345	-0.043877671609209745
2.827433388230814	2.864873695623382	2.827433388230814	-0.037440307392567984
3.141592653589793	2.9855069320561873	3.141592653589793	0.15608572153360578



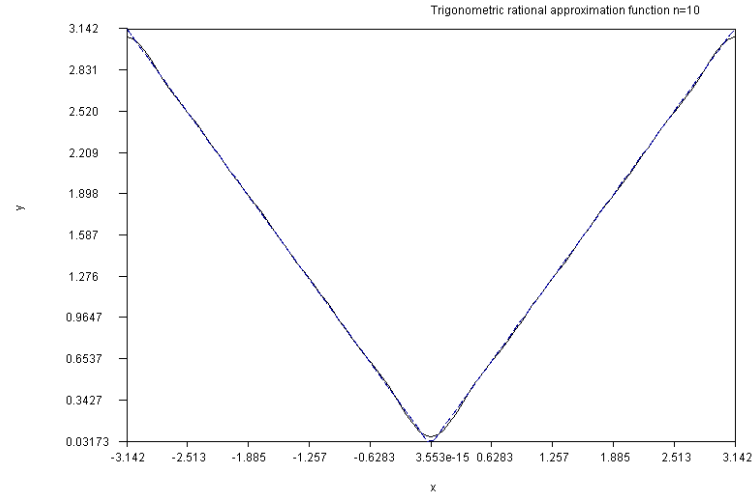


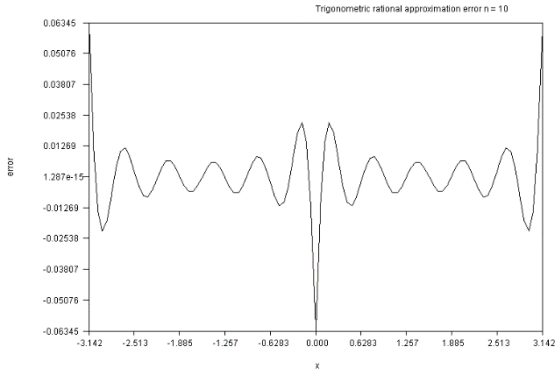
If number of the terms incresed, a better function fitting characteristics will be obtained.



Trigonometric polynomials n = 10

x	Trigonometric f(x)	f(x)	error
-3.141592653589793	3.0781400010755173	3.141592653589793	0.06345265251427579
-2.827433388230814	2.834650737418562	2.827433388230814	-0.007217349187748301
-2.5132741228718345	2.510909510200971	2.5132741228718345	0.0023646126708634085
-2.199114857512855	2.2001140890181654	2.199114857512855	-9.992315053102274E-4
-1.8849555921538759	1.8845616275341752	1.8849555921538759	3.9396461970064856E-4
-1.5707963267948966	1.5707963267948968	1.5707963267948966	-2.220446049250313E-16
-1.2566370614359172	1.2570310260556186	1.2566370614359172	-3.939646197013147E-4
-0.9424777960769379	0.9414785645716275	0.9424777960769379	9.992315053104495E-4
-0.6283185307179586	0.630683143388823	0.6283185307179586	-0.0023646126708644077
-0.3141592653589793	0.3069419161712307	0.3141592653589793	0.007217349187748634
0.0	0.06345265251427319	0.0	-0.06345265251427319
0.31415926535897887	0.3069419161712301	0.31415926535897887	0.007217349187748745
0.6283185307179586	0.630683143388823	0.6283185307179586	-0.0023646126708644077
0.9424777960769379	0.9414785645716275	0.9424777960769379	9.992315053104495E-4
1.2566370614359172	1.2570310260556186	1.2566370614359172	-3.939646197013147E-4
1.5707963267948966	1.5707963267948968	1.5707963267948966	-2.220446049250313E-16
1.8849555921538759	1.8845616275341752	1.8849555921538759	3.9396461970064856E-4
2.199114857512855	2.2001140890181654	2.199114857512855	-9.992315053102274E-4
2.5132741228718345	2.510909510200971	2.5132741228718345	0.0023646126708634085
2.827433388230814	2.834650737418562	2.827433388230814	-0.007217349187748301
3.141592653589793	3.0781400010755173	3.141592653589793	0.06345265251427579

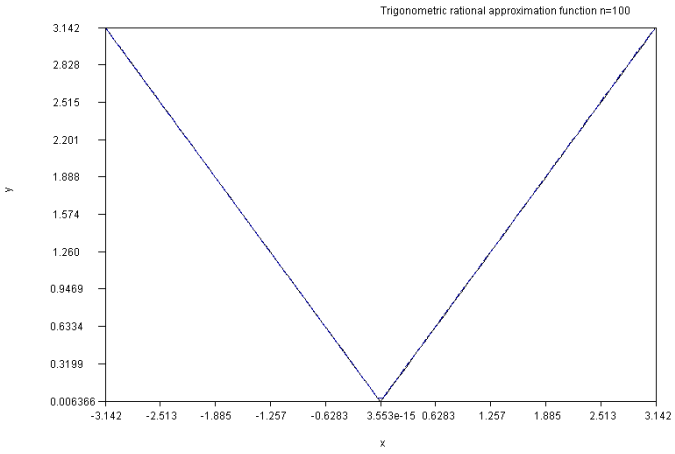


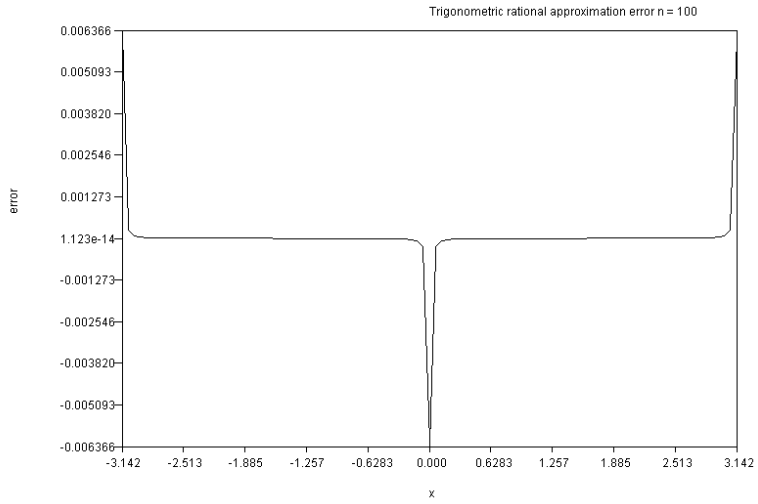


And finally error for 100 terms will be

Trigonometric polynomials n = 100

x	Trigonometric f(x)	f(x)	error
-3.141592653589793	3.135226668042987	3.141592653589793	0.006365985546806119
-2.827433388230814	2.8274208595836403	2.827433388230814	1.2528647173493113E-5
-2.5132741228718345	2.5132711510880275	2.5132741228718345	2.9717838070375535E-6
-2.199114857512855	2.1991137159337857	2.199114857512855	1.1415790694968564E-6
-1.8849555921538759	1.8849551576523567	1.8849555921538759	4.345015192086521E-7
-1.5707963267948966	1.5707963267948961	1.5707963267948966	4.440892098500626E-16
-1.2566370614359172	1.2566374959374371	1.2566370614359172	-4.345015198747859E-7
-0.9424777960769379	0.9424789376560113	0.9424777960769379	-1.141579073382637E-6
-0.6283185307179586	0.6283215025017691	0.6283185307179586	-2.971783810479245E-6
-0.3141592653589793	0.31417179400615136	0.3141592653589793	-1.2528647172049823E-5
0.0	0.006365985546783667	0.0	-0.006365985546783667
0.31415926535897887	0.3141717940061509	0.31415926535897887	-1.2528647172049823E-5
0.6283185307179586	0.6283215025017691	0.6283185307179586	-2.971783810479245E-6
0.9424777960769379	0.9424789376560113	0.9424777960769379	-1.141579073382637E-6
1.2566370614359172	1.2566374959374371	1.2566370614359172	-4.345015198747859E-7
1.5707963267948966	1.5707963267948961	1.5707963267948966	4.440892098500626E-16
1.8849555921538759	1.8849551576523567	1.8849555921538759	4.345015192086521E-7
2.199114857512855	2.1991137159337857	2.199114857512855	1.1415790694968564E-6
2.5132741228718345	2.5132711510880275	2.5132741228718345	2.9717838070375535E-6
2.827433388230814	2.8274208595836403	2.827433388230814	1.2528647173493113E-5
3.141592653589793	3.135226668042987	3.141592653589793	0.006365985546806119





There is a discreate analog that is useful for the discreate least square approximation and interpolation of the data. Assume that a collection of 2m paired data points (x_j,y_j) j=0..2m-1 is given. We assume that the interval is [-π,π]. If any other range is given, it should be converted by using linear conversion. Standart linear least square curve fitting method for w=1 can be applied here. For the trigonometric function

$$S_n(x) = \frac{a_0}{2} + a_n \cos(nx) + \sum_{k=1}^{n-1} [a_k \cos(kx) + b_k \sin(kx)] \quad (6.15-7)$$

Error term will be:

$$E(S_n) = \sum_{j=0}^{2m-1} \left\{ y_j - \left[\frac{a_0}{2} + a_n \cos(nx_j) + \sum_{k=1}^{n-1} [a_k \cos(kx_j) + b_k \sin(kx_j)] \right] \right\} \quad (6.15-8)$$

We need to choose a_k and b_k coefficients so that error will be minimum. It should be note that cosine and sine functions are orthogonal and for orthogonal functions relation

$$\sum_{k=1}^{2m-1} \emptyset_k(x_j)\emptyset_l(x_j) = 0 \quad for\ k \neq l \quad (6.15-9)$$

Will held, So solution set will not require a matrix solution and the coeeficients will be:

$$a_k = \frac{1}{m} \sum_{j=0}^{2m-1} y_j \cos(kx_j) \quad (6.15-10)$$

$$b_k = \frac{1}{m} \sum_{j=0}^{2m-1} y_j \sin(kx_j) \quad (6.15-11)$$

If the function is defined in some other interval than [-π,π] then a range correction should apply
For example for the range $[x_0, x_n]$ ranges can be converted by using transfer functions

$z = \frac{2\pi(x-x_0)}{(x_n-x_0)} - \pi$ (6.15-12)

$x = \frac{(x_n-x_0)(x+\pi)}{2\pi} + x_0$ (6.15-13)

So basic equations becomes:

$S_n(z) = \frac{a_0}{2} + a_n \cos(nz) + \sum_{k=1}^{n-1}[a_k \cos(kz) + b_k \sin(kz)]$ (6.15-14)

$a_k = \frac{1}{m}\sum_{j=0}^{2m-1} y_j \cos(kz_j)$ (6.15-16)

$b_k = \frac{1}{m}\sum_{j=0}^{2m-1} y_j \sin(kz_j)$ (6.15-17)

Note that if a discreate set is given problem can be solved either as a function fitting or data fitting problem. As an example problem Let us find discreate least square approximation $S_3(x)$, for m=5 (2m-1=9). The function is $f(x) = x^4 - 3x^3 + 2x^2\, tan(\, x(x-1))$. The problem first solved as an excel datasheet problem

FOURIER DISCRETE LEAST SQUARE CURVE

	xj			***k***	***0***	***1***	***2***	***3***				
j	**j/5**	**z_j**		**f(x)**	**a_0**	**a_1**	**a_2**	**a_3**	**b_0**	**b_1**	**b_2**	**b_3**
0	0	-3.14159	-	0	0	0	0	0	0	0	0	0
1	0.2	-2.51327	-	0.43400285	0.0868006	-0.0702231	0.0268228	0.0268229	0	-0.0510201	0.0825523	-0.082552
2	0.4	-1.88496	-	0.89814382	0.1796288	-0.0555083	-0.1453227	0.1453227	0	-0.1708371	0.1055831	0.1055832
3	0.6	-1.25664	-	1.31723235	0.2634465	0.0814094	-0.2131327	-	0	-0.2505525	-0.15485	0.1548499
4	0.8	-0.62832	-	1.58195749	0.3163915	0.2559661	0.0977704	-	0	-0.1859703	-0.3009062	-0.300906
5	1	0	0	1.55740772	0.3114815	0.3114815	0.3114815	0.3114815	0	0	0	0
6	1.2	0.62832	0.62832	1.19795749	0.2395915	0.1938336	0.0740378	-	0	0.1408283	0.2278651	0.2278651
7	1.4	1.25664	1.25664	0.64523235	0.1290465	0.0398776	-0.1044008	-	0	0.1227305	0.0758516	-0.075852
8	1.6	1.88496	1.88496	0.13014382	0.0260288	-0.0080433	-0.0210577	0.0210577	0	0.0247548	-0.0152993	-0.015299
9	1.8	2.51327	2.51327	-0.1419971	-	0.0229756	-0.0087759	-	0	-0.0166928	0.0270095	-0.027009
10	2	3.14159	3.14159	0	0	0	0	0	0	0	0	0
				7.62008075	1.5524156	0.7717691	0.0174228	0.0065673	0	-0.386759	0.047806	-0.013321

x	**z**	**f(x)**	**$S_3(x)$**	**$\lvert f(x)-S_3(x)\rvert$**
0	-3.14159	0	0.01529	0.01529429
0.125	-2.74889	0.2644	0.2548	0.00959471
0.375	-1.9635	0.84081	0.86573	0.02492368
0.625	-1.1781	1.3615	1.37668	0.015174
0.875	-0.3927	1.61282	1.61826	0.00544731
1.125	0.392699	1.36672	1.38986	0.02313645
1.375	1.178097	0.71697	0.72965	0.01267567
1.625	1.963495	0.07909	0.08349	0.00439694

A java program is developed to solve discreate least square problem (**Fourier series problem**)

Program 6.15-2 discreate trigonometric function approximation program Fourier.java

```
import java.io.*;
import javax.swing.*;
class ff extends f_x
{
public double func(double x)
{return x*x*x*x-3.0*x*x*x+2.0*x*x-Math.tan(x*(x-2.0));}
}

// Fourier approximation
class Fourier
{
public double a[];
public double b[];
public double xj[];
public double zj[];
public double yj[];
public double x0,xn;
public int m; //(2*m+1) data points
public int n; //degree n Fourier function
```

```
public double z_x(double x)
{double zz=2.0*Math.PI*(x-x0)/(xn-x0)-Math.PI;
 return zz;
}

public double x_z(double z)
{double xx=(z+Math.PI)/(2.0*Math.PI)*(xn-x0)+x0;
 return xx;
}

public Fourier(f_x f,double x0i,double xni,int mi,int ni)
{
m=mi;
n=ni;
a=new double[n+1];
b=new double[n+1];
xj=new double[2*m+1];
zj=new double[2*m+1];
yj=new double[2*m+1];
x0=x0i;
xn=xni;
for(int k=0;k<=2*m;k++)
{xj[k]=x0+(xn-x0)/(2*m)*k;
 yj[k]=f.func(xj[k]);
 zj[k]=z_x(xj[k]);
}
  for(int k=0;k<=n;k++)
  {
  a[k]=0;
  b[k]=0;
  for(int j=0;j<2*m;j++)
    { a[k]+=1.0/m*yj[j]*Math.cos(k*zj[j]);
      b[k]+=1.0/m*yj[j]*Math.sin(k*zj[j]);
    }
  }
}

public  double func(double x)
{
double cc=a[0]/2.0+a[n]*Math.cos(n*z_x(x));
for(int k=1;k<n;k++)
{cc+=a[k]*Math.cos(k*z_x(x))+b[k]*Math.sin(k*z_x(x));}
return cc;
}

public void plot_func(f_x f,double x0i,double xni,int ni)
{
            double xx[][]=new double[4][ni+1];
            double x1=0;
   for(int i=0;i<=ni;i++)
            { x1=x0i+(xni-x0i)*i/ni;
              xx[0][i]=x1;
              xx[1][i]=func(x1);
              xx[2][i]=f.func(x1);
              xx[3][i]=xx[2][i]-xx[1][i];
            }
            Plot pp=new Plot(xx[0],xx[1]);
            pp.addData(xx[0],xx[2]);
   pp.setPlabel("Discrete Trigonometric approximation function n="+(a.length-1));
            pp.plot();
}
            public void plot_error(f_x f,double x0i,double xni,int ni)
   { double xx[][]=new double[4][ni+1];
              double x1=0;
     for(int i=0;i<=ni;i++)
              { x1=x0i+(xni-x0i)*i/ni;
```

```
                xx[0][i]=x1;
                xx[1][i]=func(x1);
                xx[2][i]=f.func(x1);
                xx[3][i]=xx[2][i]-xx[1][i];
                }
            Plot pp=new Plot(xx[0],xx[3]);
    pp.setPlabel("Discrete Trigonometric approximation error n = "+(a.length-1));
    pp.setYlabel("error");
            pp.plot();
    }
public double[][] result(f_x f)
{   double xx[][]=new double[4][2*m];
    for(int i=0;i<2*m;i++)
            {
             xx[0][i]=xj[i];
             xx[1][i]=func(xj[i]);
             xx[2][i]=f.func(xj[i]);
             xx[3][i]=xx[2][i]-xx[1][i];
            }
            return xx;
}

public static void main(String arg[])
{
            ff f=new ff();
            int n=Integer.parseInt(JOptionPane.showInputDialog("number of terms in the Trigonometric function
approximation Sn(x): n="));
            int m=Integer.parseInt(JOptionPane.showInputDialog("number of data points (2m-1): m="));
            double x0=0;
            double xn=2.0;
            Fourier Pa=new Fourier(f,x0,xn,m,n);
    String s1[]={"x","Fourier f(x)","f(x)","error"};
            Text.print(Text.T(Pa.result(f)),s1,"Discrete Fourier curve fitting n = "+n+"m = "+m);
    Text.printT(Pa.a,"a");
    Text.printT(Pa.b,"b");
    Pa.plot_func(f,x0,xn,100);
    Pa.plot_error(f,x0,xn,100);
}

}
```

a

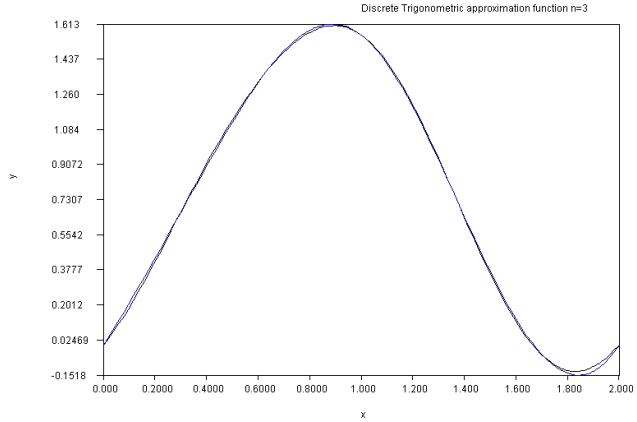
0
1.5240161502702365
0.771769039301839
0.017422790287928383
0.006567273148323184

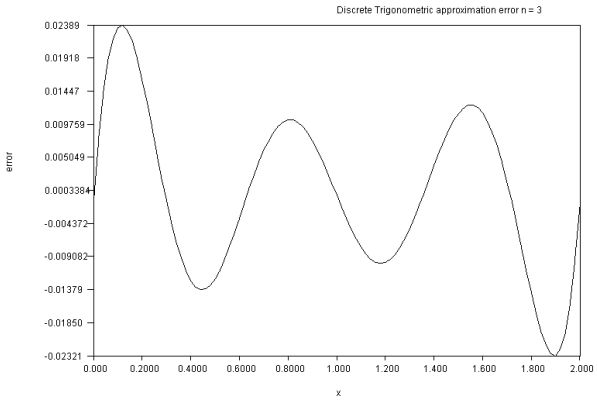
b

0
0.0
-0.3867590451331591
0.04780604706974956
-0.013320698468437413

Discrete Fourier curve fitting n = 3m = 5

x	Fourier f(x)	f(x)	error
0.0	0.0010945529728844348	0.0	-0.0010945529728844348
0.2	0.41784465940942667	0.434002851642027	0.016158192232600344
0.4	0.9106654279577352	0.8981438222209641	-0.01252160573677108
0.6000000000000001	1.3208194756677367	1.317232348561538	-0.0035871271061986842
0.8	1.571601893277591	1.5819574909236105	0.010355597646019454
1.0	1.557767177873209	1.5574077246549023	-3.594532183066601E-4
1.2000000000000002	1.2078718726056072	1.1979574909236106	-0.009914381681996653
1.4000000000000001	0.641359434323769	0.6452323485615388	0.003872914237769809
1.6	0.11880662886177322	0.130143822220963	0.01133719335918977
1.8	-0.12775037159854952	-0.1419971483579729	-0.014246776759423374

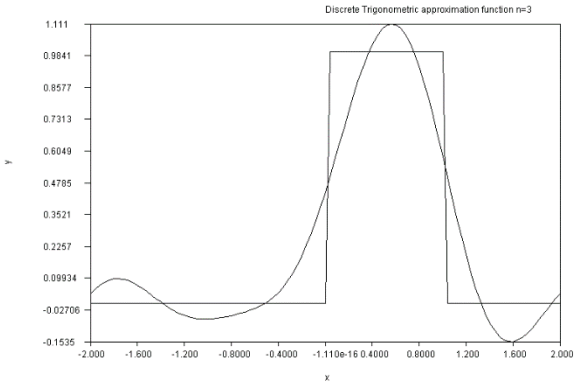


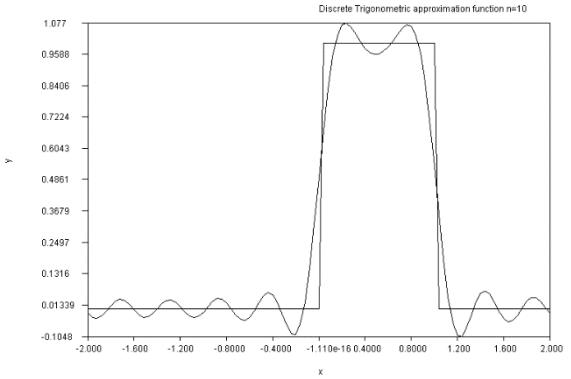


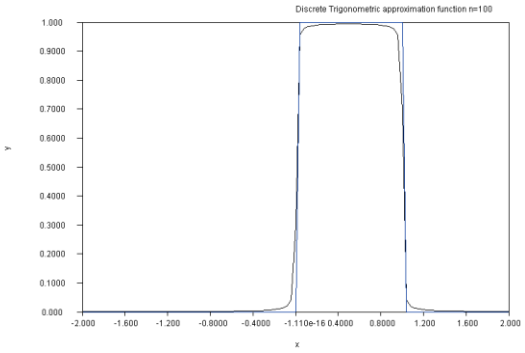
As another example, the following function is given

$$f(x) = \left\{ \begin{matrix} 0 \\ 1 \end{matrix} \right\} \begin{matrix} for x < 0 \\ for 0 <= x < 1 \\ for x > 1 \end{matrix}$$

Fourier series of different order fits to this function: $S_3(x)$, $S_{10}(x)$ and $S_{100}(x)$ function plots are given below:







If data is given and a curve fitting is required, the same equations can applied.

Program 6.16-3 discreate trigonometric data approximation(curve fitting) program Fourier1.java

```
import java.io.*;
import javax.swing.*;

// Fourier approximation
class Fourier1
{
public double a[];
public double b[];
public double xj[];
public double zj[];
public double yj[];
public double x0,xn;
public int m; //(2*m+1) data points
public int n; //degree n Fourier function

public void read_data(String s1)
{
double a[][]=Text.readDoubleT(s1);
int mm=a[0].length;
m=mm/2;
x0=a[0][0];
xn=a[0][mm-1];
xj=new double[mm];
zj=new double[mm];
yj=new double[mm];
for(int k=0;k<=2*m;k++)
{xj[k]=a[0][k];
 yj[k]=a[1][k];
 zj[k]=z_x(xj[k]);
}
}

public double z_x(double x)
{double zz=2.0*Math.PI*(x-x0)/(xn-x0)-Math.PI;
```

```
 return zz;
}

public double x_z(double z)
{double xx=(z+Math.PI)/(2.0*Math.PI)*(xn-x0)+x0;
 return xx;
}

public Fourier1(String name,int ni)
{
read_data(name);
n=ni;
a=new double[n+1];
b=new double[n+1];
  for(int k=0;k<=n;k++)
  {
  a[k]=0;
  b[k]=0;
  for(int j=0;j<2*m;j++)
    { a[k]+=1.0/m*yj[j]*Math.cos(k*zj[j]);
      b[k]+=1.0/m*yj[j]*Math.sin(k*zj[j]);
    }
  }
}

public  double func(double x)
{
double cc=a[0]/2.0+a[n]*Math.cos(n*z_x(x));
for(int k=1;k<n;k++)
{cc+=a[k]*Math.cos(k*z_x(x))+b[k]*Math.sin(k*z_x(x));}
return cc;
}

public void plot_func(double x0i,double xni,int ni)
{
            double xx[][]=new double[2][ni+1];
            double x1=0;
   for(int i=0;i<=ni;i++)
            { x1=x0i+(xni-x0i)*i/ni;
              xx[0][i]=x1;
              xx[1][i]=func(x1);
            }
            Plot pp=new Plot(xx[0],xx[1]);
            pp.addData(xj,yj);
            pp.setPlotType(1,32);
   pp.setPlabel("Discrete Trigonometric least square n="+(a.length-1));
            pp.plot();
}

public double[][] result()
{   double xx[][]=new double[4][2*m];
   for(int i=0;i<2*m;i++)
            {
              xx[0][i]=xj[i];
              xx[1][i]=yj[i];
              xx[2][i]=func(xj[i]);
              xx[3][i]=xx[2][i]-xx[1][i];
            }
            return xx;
}

public static void main(String arg[])
{
            int n=Integer.parseInt(JOptionPane.showInputDialog("number of terms in the Trigonometric function
approximation Sn(x): n="));
            String name=JOptionPane.showInputDialog("Data file name (a.txt)");
```

```
        Fourier1 Pa=new Fourier1(name,n);
   String s1[]={"x","Fourier f(x)","f(x)","error"};
        Text.print(Text.T(Pa.result()),s1,"Discrete Fourier curve fitting n = "+n+"m = "+Pa.m);
   Text.printT(Pa.a,"a");
   Text.printT(Pa.b,"b");
   Pa.plot_func(Pa.x0,Pa.xn,100);
}
}
```

Example data a.txt (generated from the function of the previous example)

```
0       0
0.2     0.434002852
0.4     0.898143822
0.6     1.317232349
0.8     1.581957491
1       1.557407725
1.2     1.197957491
1.4     0.645232349
1.6     0.130143822
1.8     -0.141997148
2       0
```

a

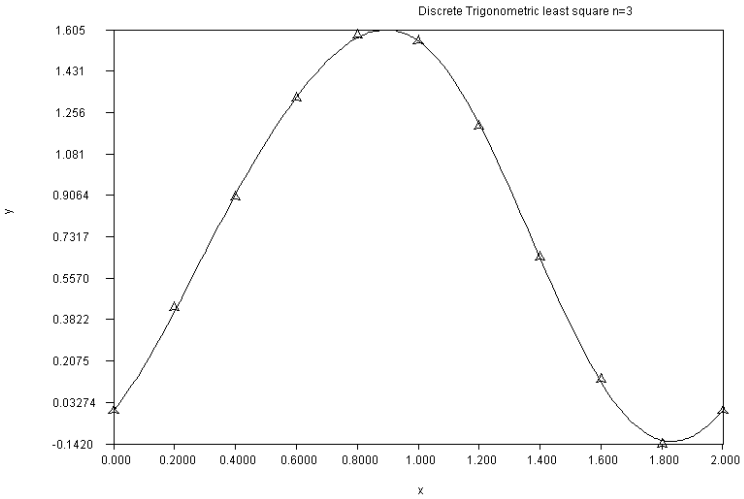
0
1.5240161506
0.7717690393612457
0.017422790340254395
0.006567273038754351

b

0
0.0
-0.38675904513315906
0.04780604706974935
-0.013320698468437184

Discrete Fourier curve fitting n = 3m = 5

x	Fourier f(x)	f(x)	error
0.0	0.0	0.001094553240254318	0.001094553240254318
0.2	0.434002852	0.4178446595085582	-0.016158192491441792
0.4	0.898143822	0.9106654279732834	0.012521605973283445
0.6	1.317232349	1.3208194758972864	0.0035871268972864634
0.8	1.581957491	1.5716018935405622	-0.010355597459437815
1.0	1.557407725	1.5577671780402547	3.5945304025464253E-4
1.2	1.197957491	1.2078718728685784	0.009914381868578515
1.4	0.645232349	0.6413594345533187	-0.0038729144466813636
1.6	0.130143822	0.11880662887732185	-0.01133719312267814
1.8	-0.141997148	-0.12775037149941762	0.01424677650058237



6.16 ORTHOGONAL FUNCTION APPROXIMATION: LEGENDRE POLYNOMIALS

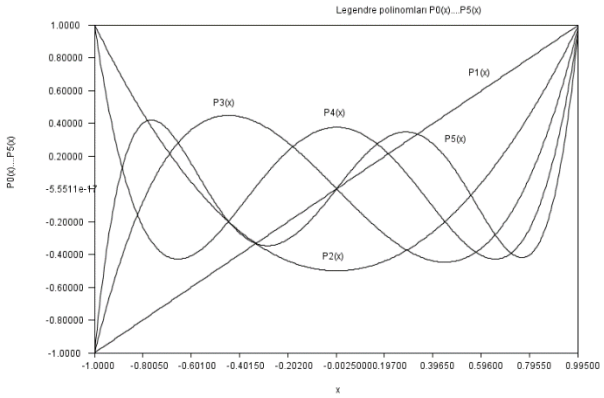


Figure 6.16.1 Adrien Marie Legendre 1752-1833 French Mathematician

Utilising sine and cosine functions as orthogonal function approximation is investigated in the previous sub-section. Similar tasks can be achieved by using other orthogonal polynomials. In this section we will be using Legendre polynomials as a means of function approximation. Definition of LEgendre polynomials are as follows:

$P_k(x) = \frac{(2k-1)}{k} x P_{k-1}(x) - \frac{(k-1)}{k} P_{k-2}(x)$ (6.16-1)

In this recursive equation first two terms are defined as: $P_0(x)=1$ and $P_1(x)=x$. The first 6 Legndre functions are shown in the plot below.



(bolum6_049.jpg,Legendre polinomları)

If the first 10 Legendre polynomials are listed in open form:

n	$P_n(x)$ -1<= x <=1
0	1
1	x
2	$(3x^2-1)/2$
3	$(5x^3-3x)/2$
4	$(35x^4-30x^2+3)/8$
5	$(63x^5-70x^3+15x)/8$
6	$(231x^6-315x^4+105x^2-5)/16$
7	$(429x^7-693x^5+315x^3-35x)/16$
8	$(6435x^8-12012x^6+6930x^4-1260x^2+35)/128$
9	$(12155x^9-25740x^7+18018x^5-4620x^3+315x)/128$
10	$(46189x^{10}-109395x^8+90090x^6-30030x^4+3465x^2-63)/256$

Equivalent form of a function f(x) in as Legendre function in the range [-1,1]

$f(x) = \sum_{k=0}^{m} b_k P_k(x) + error$

Basic definition of orthogonal form of Legendre function is based on [-1,1] region, if a function definition of a different range is concerned, coordinate system transfer is needed. For the region of [a,b] the following transfer function can be used.

$z = \left(\frac{b-a}{2}\right)x + \left(\frac{b+a}{2}\right) = \alpha x + \beta \qquad \alpha = \left(\frac{b-a}{2}\right) \quad \beta = \left(\frac{b+a}{2}\right)$ (6.16-2)

For the new coordinate system equation will be transferred as

$f(z) = \sum_{k=0}^{m} b_k q_k(z) + error$ (6.16-3) In this equation q_k converted Legendre function in the form of

$q_k(z) = q_k(\alpha x + \beta) = P_k(x)$ (6.16-4)

Due to orthogonality properties:

$\left[P_n(x)P_j(x)\right] = \int_{-1}^{1} P_n(x)P_j(x)dx$ where $\left[P_n(x)P_j(x)\right] = \begin{Bmatrix} 0 & if & n \neq j \\ \frac{2}{2n+1} & if & n = j \end{Bmatrix}$ (6.16-5)

so the coefficients of the function fitting will become $b_k = \frac{2k+1}{b-a}\int_a^b f(z)q_k(z)dz$ (6.16-6)

In order to achive numerical integration process, please referred to related section. As an example function let us assume **f(x)=|x|**

Program 6.16.1 Legendre function approximation for a given function or data (converted to a function by cubic spline interpolation formula)

```
public class legendre
{
//cubic spline function or external input function
public if_x f;
//Legendre function
public if_x f_Legendre;
public double alfa,beta,xl,xh;
public double x[];
public double z[];
public double y[];
public double b[]; //katsayılar matrisi;
public int m;   //eğri uydurma derecesi

public static double P(int k, double x)
{ // Legendre polynomial  -1<=x<=1
if(k==0) return 1.0;
else if(k==1) return x;
else
{return ((2.0*k-1.0)*P((k-1),x)*x-P((k-2),x)*(double)(k-1))/(double)k;}
}

public  double q(int k, double z)
{  // converted Legendre polynomial xl<= z <=xh
  double x=(z-beta)/alfa;
  return P(k,x);
}
public  double fak(int k,double z)
   {   //integral fonksiyonu
                double a=(2.0*k+1)/(xh-xl)*f.func(z)*q(k,z);
                return a;
             }

public legendre(if_x f1,double xli,double xhi,int mi)
{   f=f1;
             setdata(xli,xhi,mi);
}
// Constructor from data using cubic spline interpolation
public legendre(double xy[][],double xli,double xhi,int mi)
{   double xi[]=xy[0];
   double yi[]=xy[1];
```

```
   double c0=0.0;
   double cn=0.0;
            double S[][]=cubic_spline(xi,yi,c0,cn);
            f=(x)->funcSpline(S,x);
            setdata(xli,xhi,mi);
}
public legendre(double xi[],double yi[],double xli,double xhi,int mi)
{           double c0=0.0;
   double cn=0.0;
            double S[][]=cubic_spline(xi,yi,c0,cn);
            f=(x)->funcSpline(S,x);
            setdata(xli,xhi,mi);
}

public void setdata(double xli,double xhi,int mi)
{  //z  xl, xh
    m=mi;
    xl=xli;
    xh=xhi;
    alfa=(xh-xl)/2.0;
    beta=(xh+xl)/2.0;
    b=new double[m+1];
    for(int j=0;j<=m;j++)
            {b[j]=integral(xl,xh,30,j);}
            f_Legendre=(x)->funcLegendre(x);
            System.out.println("b=\n"+Matrix.toStringT(b));
}
public  double funcLegendre(double z)
{
   double x=(z-beta)/alfa;
   double ff=0;
   double pk=0;
   for(int j=0;j<=m;j++)
     {   pk=P(j,x);
              ff+=b[j]*pk;
         }
   return ff;
}

public static double[][] Gauss_Legendre_Coefficients(double x1,double x2,int n)
{
            // calculates Gauss-Legendre coefficients
            double EPS=3.0e-15;
            int m,j,i;
            double z1,z,xm,xl,pp,p3,p2,p1;
   double a[][]=new double[2][n];//a[0][i]=x[i]   a[1][i]=w[i]
            m=(n+1)/2;
            xm=0.5*(x2+x1);
            xl=0.5*(x2-x1);
            for (i=1;i<=m;i++)  {
                        z=Math.cos(Math.PI*((i-0.25)/(n+0.5)));
                        do {
                                    p1=1.0;
                                    p2=0.0;
                                    for (j=1;j<=n;j++) {
                                                p3=p2;
                                                p2=p1;
                                                p1=((2.0*j-1.0)*z*p2-(j-1.0)*p3)/j;
                                    }
                                    pp=n*(z*p1-p2)/(z*z-1.0);
                                    z1=z;
                                    z=z1-p1/pp;
                        } while (Math.abs(z-z1) > EPS);
                        a[0][i-1]=xm-xl*z;
```

```
                        a[0][n-i]=xm+xl*z;
                        a[1][i-1]=2.0*xl/((1.0-z*z)*pp*pp);
                        a[1][n-i]=a[1][i-1];
                                }
return a;
}

public  double integral(double xl,double xh,int n,int k)
{
// n : integral degree of polynomial
// This program calculates Gauss-Legendre coefficients in  [x1,x2] band
// and then calculates the integral in this band.
double a[][]=new double[2][n];
a=Gauss_Legendre_Coefficients(xl,xh,n);
double z=0;
for(int i=0;i<n;i++)
{ z+=a[1][i]*fak(k,a[0][i]);}
return z;
}
// Cubic spline interpolation section
  public static double [] thomas(double a[][],double r[])
  {
  //
  int n=a.length;
  double f[]=new double[n];
  double e[]=new double[n];
  double g[]=new double[n];
  double x[]=new double[n];
  for(int i=0;i<n;i++)     {f[i]=a[i][i];}
  for(int i=0;i<(n-1);i++) {g[i]=a[i][i+1];}
  for(int i=0;i<(n-1);i++) {e[i+1]=a[i+1][i];}
  for(int k=1;k<n;k++)
     {e[k]=e[k]/f[k-1];
      f[k]=f[k]-e[k]*g[k-1];
     }
  for(int k=1;k<n;k++)
     {r[k]=r[k]-e[k]*r[k-1];
     }
  x[n-1]=r[n-1]/f[n-1];
  for(int k=(n-2);k>=0;k--)
     {x[k]=(r[k]-g[k]*x[k+1])/f[k];}
  return x;
  }

  public static double [] thomas(double f[],double e[],double g[],double r[])
  {
  int n=f.length;
  double x[]=new double[n];
  for(int k=1;k<n;k++)
     {e[k]=e[k]/f[k-1];
      f[k]=f[k]-e[k]*g[k-1];
     }
  for(int k=1;k<n;k++)
     {r[k]=r[k]-e[k]*r[k-1];
     }
  x[n-1]=r[n-1]/f[n-1];
  for(int k=(n-2);k>=0;k--)
     {x[k]=(r[k]-g[k]*x[k+1])/f[k];}
  return x;
  }


  public static double [][] cubic_spline(double xi[],double yi[],double c0,double cn)
  {
```

```
// c0 second derivative of the first point in the first spline
// cn second derivative of the last point in last spline
int n=xi.length;
double h[]=new double[n];
double w[]=new double[n];
double f[]=new double[n];
double e[]=new double[n];
double g[]=new double[n];
double d[]=new double[n];
double x[]=new double[n];
double S[][]=new double[4][n];
int k;
for(k=0;k<(n-1);k++)
   {h[k]=xi[k+1]-xi[k];
    w[k]=(yi[k+1]-yi[k])/h[k];
               }
d[0]=c0;
d[n-1]=cn;
for(k=1;k<(n-1);k++)
   {d[k]=6.0*(w[k]-w[k-1]);}
f[0]=1.0;
f[n-1]=1.0;
g[0]=0.0;
g[n-1]=0.0;
e[0]=0.0;
e[n-1]=0.0;
for(k=1;k<(n-1);k++)
{f[k]=2.0*(h[k]+h[k-1]);e[k]=h[k-1];g[k]=h[k];}
S[2]=thomas(f,e,g,d);
S[3]=xi;
for(k=0;k<(n-1);k++)
  {S[0][k]=(6.*yi[k+1]-h[k]*h[k]*S[2][k+1])/(6.0*h[k]);
   S[1][k]=(6.*yi[k]-h[k]*h[k]*S[2][k])/(6.0*h[k]);
  }
return S;
}

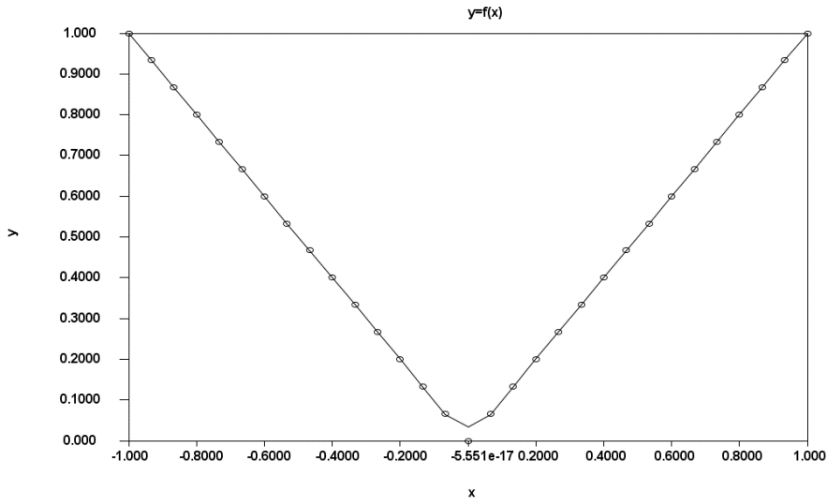
public static double funcSpline(double S[][],double x)
{
int n=S[0].length;
double xx1=0;
double xx2=0;
double y=0;
double hk=0;
for(int k=0;k<(n-1);k++)
   {if(S[3][k]<=x && x<=S[3][k+1])
    {hk=(S[3][k+1]-S[3][k]);
     xx1=(x-S[3][k]);
     xx2=(S[3][k+1]-x);
     y=S[0][k]*xx1+S[1][k]*xx2+(xx1*xx1*xx1*S[2][k+1]+xx2*xx2*xx2*S[2][k])/(6.0*hk);
     break;
     }
   }
     if(y==0 && S[3][n-2]<=x )
     {
               int k=n-2;
               hk=(S[3][k+1]-S[3][k]);
     xx1=(x-S[3][k]);
     xx2=(S[3][k+1]-x);
     y=S[0][k]*xx1+S[1][k]*xx2+(xx1*xx1*xx1*S[2][k+1]+xx2*xx2*xx2*S[2][k])/(6.0*hk);
     }
return y;
}
```

```
  public static double dfSpline(double xi[],double yi[],double c0,double cn,double x)
  { //cubic spline formulations
              double S[][]=cubic_spline(xi,yi,c0,cn);
    return dfSpline(S,x);
  }
  public static double dfSpline(double S[][],double x)
  {
    //cubic spline derivative formulation
  int n=S[0].length;
  double xx1=0;
  double xx2=0;
  double y=0;
  double hk=0;
  for(int k=0;k<(n-1);k++)
     {if(S[3][k]<=x && x<=S[3][k+1])
      {hk=(S[3][k+1]-S[3][k]);
       xx1=(x-S[3][k]);
       xx2=(S[3][k+1]-x);
       y=S[0][k]-S[1][k]+(xx1*xx1*S[2][k+1]-xx2*xx2*S[2][k])/(2.0*hk);
       break;
       }
     }
       if(y==0 && S[3][n-2]<=x )
       {
                int k=n-2;
                hk=(S[3][k+1]-S[3][k]);
       xx1=(x-S[3][k]);
       xx2=(S[3][k+1]-x);
       y=S[0][k]*xx1+S[1][k]*xx2+(xx1*xx1*xx1*S[2][k+1]+xx2*xx2*xx2*S[2][k])/(6.0*hk);
       }
  return y;
  }
 public static double  dfSpline(double xi[],double yi[],double x)
 {
    //natural cubic spline derivative formula
              double S[][]=cubic_spline(xi,yi,0,0);
    return dfSpline(S,x);
 }
//end of cubic spline
}
```

```
public class legendre_test1A
{
 public static void main(String args[])
  { int n=30;//number of evaluation points
    if_x f=(x)->Math.abs(x);
    legendre l=new legendre(f,-1.0,1.0,n);
    Plot p1=new Plot(l.f_Legendre,l.xl,l.xh,n);
    p1.addFunction(f,l.xl,l.xh,n);
    p1.setPlotType(1,22);
    p1.plot();
  }
}
```

```
---------- Capture Output ----------
> "E:\co\java\bin\java.exe" legendre_test1
b=
           0.500442496686119
           0.000000000000000
           0.623887571165741
           0.000000000000000
          -0.185978548243092
```

0.000000000000000
0.099693276449802
0.000000000000000
-0.064206907443573
-0.000000000000000
0.045318387689030
-0.000000000000000
-0.033735662852678
0.000000000000000
0.025936642703608
-0.000000000000000
-0.020306723986429
0.000000000000000
0.016004563758942
0.000000000000000
-0.012547099754516
-0.000000000000000
0.009633136817453
0.000000000000000
-0.007059268461882
-0.000000000000000
0.004675608843615
0.000000000000000
-0.002359881760763
0.000000000000000
0.000000000000002



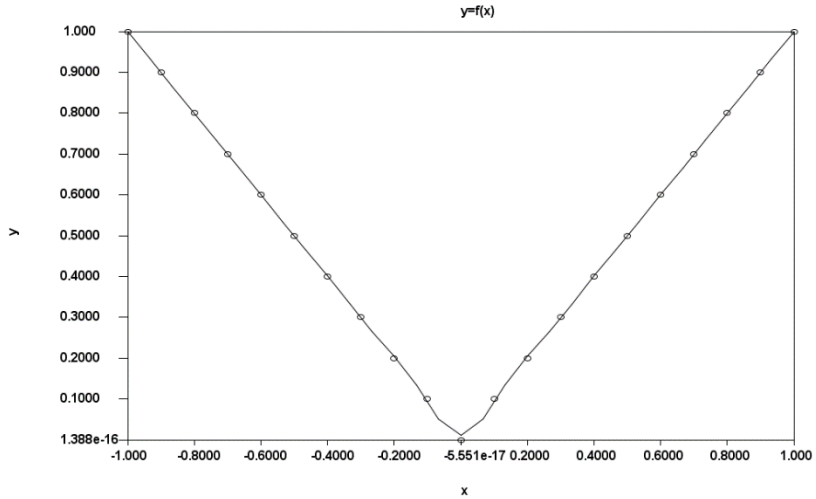
Now that we have a function fitting Legendre polynomial approach, we will apply this method to data fitting by creating a function from cubic spline interpolation and the apply Legendre function fitting method to find the curve fitting function

Data to curve-fit:

-1	1
-0.9	0.9
-0.8	0.8
-0.7	0.7
-0.6	0.6
-0.5	0.5
-0.4	0.4
-0.3	0.3
-0.2	0.2
-0.1	0.1
-1.38778E-16	1.38778E-16
0.1	0.1
0.2	0.2
0.3	0.3

```
0.4         0.4
0.5         0.5
0.6         0.6
0.7         0.7
0.8         0.8
0.9         0.9
1           1
```

```
public class legendre_test2A
{
 public static void main(String args[])
  { int n=30;//number of data points
    double a[][]=Text.readDoubleT("bb.txt");
    //x=a[0]  y=a[1]
    legendre l=new legendre(a,a[0][0],a[0][a[0].length-1],n);
    Plot p1=new Plot(l.f_Legendre,l.xl,l.xh,n);
    p1.addFunction(l.f,l.xl,l.xh,n);
    p1.setPlotType(1,22);
    p1.plot();
  }
}
```

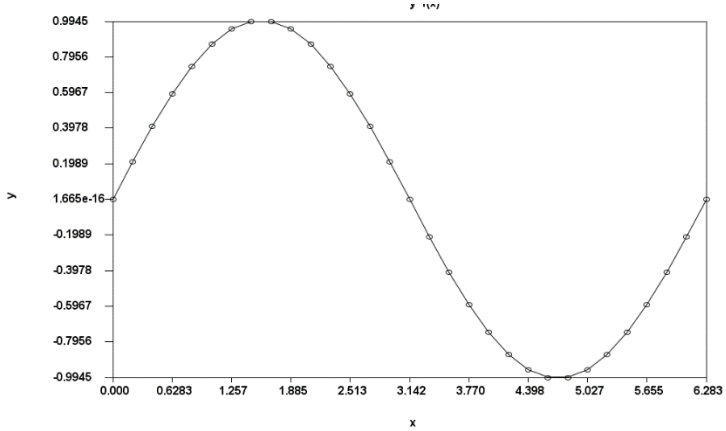


Now let us try a different function f=sin(x)

```
 public class legendre_test1
{public static void main(String args[])
  { int n=30;//number of evaluation points
    if_x f=(x)->Math.sin(x);
    legendre l=new legendre(f,0.0,2.0*Math.PI,n);
    Plot p1=new Plot(l.f_Legendre,l.xl,l.xh,n);
    p1.addFunction(f,l.xl,l.xh,n);
    p1.setPlotType(1,22);
    p1.plot();
  }}
```

```
---------- Capture Output ----------
> "E:\co\java\bin\java.exe" legendre_test1
b=       0.000000000000000
        -0.954929658551369
        -0.000000000000000
         1.158241912199413
         0.000000000000000
        -0.219289547216234
        -0.000000000000000
         0.016642266917960
        -0.000000000000000
        -0.000682206702639
         0.000000000000000
         0.000017538925948
        -0.000000000000000
        -0.000000309518242
        -0.000000000000000
```

0.000000003983935
0.000000000000000
-0.000000000039074
0.000000000000001
0.000000000000307
0.000000000000001
0.000000000000004
-0.000000000000001
0.000000000000003
0.000000000000001
-0.000000000000002
-0.000000000000001
-0.000000000000007
-0.000000000000002
-0.000000000000003
0.000000000000003



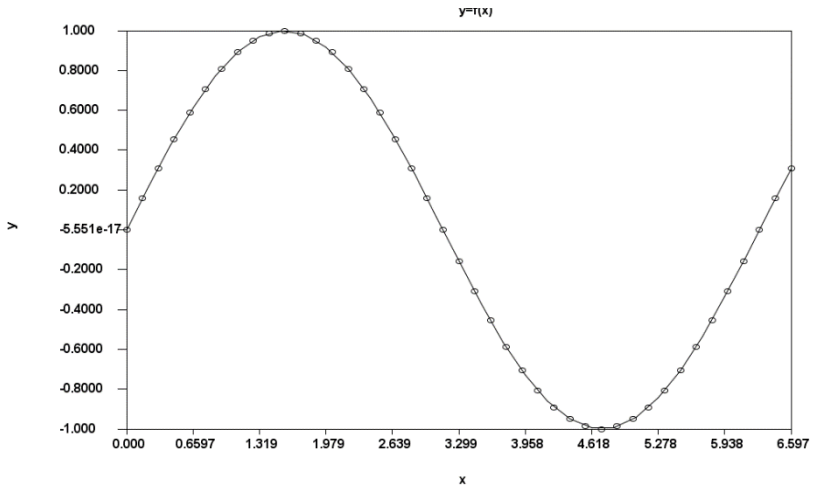
A data file of sinus function is created and Legendre polynomial function fitting is applied. Data file:

0.0000000000	0.000000000
0.1570796327	0.156434465
0.3141592654	0.309016994
0.4712388980	0.453990500
0.6283185307	0.587785252
0.7853981634	0.707106781
0.9424777961	0.809016994
1.0995574288	0.891006524
1.2566370614	0.951056516
1.4137166941	0.987688341
1.5707963268	1.000000000
1.7278759595	0.987688341
1.8849555922	0.951056516
2.0420352248	0.891006524
2.1991148575	0.809016994
2.3561944902	0.707106781
2.5132741229	0.587785252
2.6703537556	0.453990500
2.8274333882	0.309016994
2.9845130209	0.156434465
3.1415926536	0.000000000
3.2986722863	-0.156434465
3.4557519189	-0.309016994
3.6128315516	-0.453990500
3.7699111843	-0.587785252
3.9269908170	-0.707106781
4.0840704497	-0.809016994
4.2411500823	-0.891006524
4.3982297150	-0.951056516
4.5553093477	-0.987688341
4.7123889804	-1.000000000
4.8694686131	-0.987688341
5.0265482457	-0.951056516
5.1836278784	-0.891006524
5.3407075111	-0.809016994
5.4977871438	-0.707106781
5.6548667765	-0.587785252
5.8119464091	-0.453990500

```
5.9690260418    -0.309016994
6.1261056745    -0.156434465
6.2831853072    0.000000000
6.4402649399    0.156434465
6.5973445725    0.309016994
```

```
 public class legendre_test2
{
 public static void main(String args[])
  { int n=30;//number of data points
    double a[][]=Text.readDoubleT("aa.txt");
    //x=a[0]  y=a[1]
    legendre l=new legendre(a,a[0][0],a[0][a[0].length-1],n);
    Plot p1=new Plot(l.f_Legendre,l.xl,l.xh,n);
    p1.addFunction(l.f,l.xl,l.xh,n);
    p1.setPlotType(1,22);
    p1.plot();
  }}
```

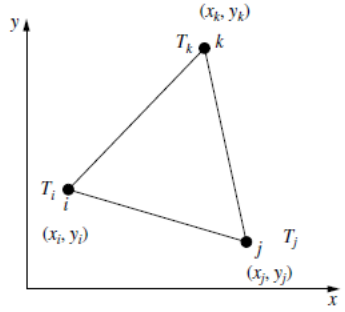
```
---------- Capture Output ----------
> "E:\co\java\bin\java.exe" legendre_test2
b=
          0.007414267330787
         -0.844614364918925
          0.239838145840113
          1.242916643035622
         -0.105404035077192
         -0.265311751791639
          0.013337733842588
          0.022383727555032
         -0.000851890424221
         -0.001058933715845
         -0.000003399532391
          0.000006612300055
         -0.000012688339823
          0.000000166584931
          0.000015221408603
          0.000029529923293
          0.000042576245059
          0.000054004185309
          0.000060956610402
          0.000063331749037
          0.000060975796011
          0.000052142373698
          0.000040400580168
          0.000025576784509
          0.000009029701868
         -0.000004223159282
         -0.000014649026394
         -0.000019613015178
         -0.000016868787182
         -0.000010864907456
         -0.000000000000003
```



6.17 FINITE ELEMENT (SIMPLEX) METHODS FOR SURFACE CURVE FITTING

The simplex elements were used in Nelder-Mead optimisation. By using a linear or non-linear simplex elements a surface curve fitting can be achieved. The simplest geometric shape that can be employed to approximate irregular surfaces is the triangle and it is one of the popular elements currently used in finite element calculations. The two-dimensional linear triangular element, also known as a *simplex element*, is represented by

$$T(x,y) = \alpha_1 + \alpha_2 x + \alpha_3 y$$



where the polynomial is linear in x and y and contains three coefficients. Since a linear triangle has three nodes, the values of α_1, α_2 and α_3 are determined from

$T_i=\alpha_1+\alpha_2 x_i+\alpha_3 y_i$

$T_j=\alpha_1+\alpha_2 x_j+\alpha_3 y_j$

$T_k=\alpha_1+\alpha_2 x_k+\alpha_3 y_k$

which results in the following:

$$\alpha_1 = \frac{1}{2A}\left[(x_j y_k - x_k y_j)T_i + (x_k y_i - x_i y_k)T_j + (x_i y_j - x_j y_i)T_k\right]$$

$$\alpha_2 = \frac{1}{2A}\left[(y_j - y_k)T_i + (y_k - y_i)T_j + (y_i - y_j)T_k\right]$$

$$\alpha_3 = \frac{1}{2A}\left[(x_k - x_j)T_i + (x_i - x_k)T_j + (x_j - x_i)T_k\right]$$

where 'A' is the area of the triangle given by

$$2A = \begin{bmatrix} 1 & x_i & y_i \\ 1 & x_j & y_j \\ 1 & x_k & y_k \end{bmatrix} = (x_i y_j - x_j y_i) + (x_k y_i - x_i y_k) + (x_j y_k - x_k y_j)$$

Substituting the values of α_1, α_2 and α_3 into the equation and collating the coefficients of T_i, T_j and T_k, we get

$$T = N_i T_i + N_j T_j + N_k T_k$$

$$N_i = \frac{1}{2A}[a_i + b_i x + c_i y]$$
$$N_j = \frac{1}{2A}[a_j + b_j x + c_j y]$$
$$N_k = \frac{1}{2A}[a_k + b_k x + c_k y]$$
$$a_i = (x_j y_k - x_k y_j) a_j = (x_k y_i - x_i y_k) a_k = (x_i y_j - x_j y_i)$$
$$b_i = (y_j - y_k) b_j = (y_k - y_i) b_k = (y_i - y_j)$$
$$c_i = (x_k - x_j) c_j = (x_i - x_k) c_k = (x_j - x_i)$$

If we evaluate N_i at node i, where the coordinates are (x_i,y_i), then we obtain $N_i(x_i,y_i)=1$ and similarly $N_i(x_j,y_j)=0$, $N_i(x_k,y_k)=0$

Thus, we see that the shape functions have a value of unity at the designated vertex and zero at all other vertices. It is possible to show that $N_i+N_j+N_k=1$ everywhere in the element, including the boundaries. Linear simplex is given us a relatively simple way of interpolating any point in the surface (a linear interpolation). An interpolation program is given below

Program 6.16.1 linear triangular finite simplex element

```
public class triangular_element1
{
public double a[],b[],c[];
public double dt;
public int P;
public double A;//Area
public double l[];
public double x[],y[];   //coordinates of data points m
public double xc,yc;     //center point m
public double N[],B[][]; // Shape factor and derivative
public double Ti[];      // Temperature nodes
public int n_point[];    // point numbers
public int n_element;    // element number

public triangular_element1(int n_elementi,int n_pointi[],double xi[],double yi[])
{dt=1.0;
 P=1;
 x=new double[3];
 y=new double[3];
 Ti=new double[3];
 l=new double[3];//0:lij,1:ljk,2:lik
 xc=0;yc=0;
 n_point=new int[3];
 n_element=n_elementi;
 for(int i=0;i<3;i++)
 { n_point[i]=n_pointi[i];
  x[i]=xi[i];y[i]=yi[i];
  xc+=x[i];yc+=y[i];
 }
 xc/=3.0;
 yc/=3.0;
 a=new double[3];
 b=new double[3];
 c=new double[3];
 N=new double[3];
 B=new double[2][3];
 a[0]=x[1]*y[2]-x[2]*y[1];
 a[1]=x[2]*y[0]-x[0]*y[2];
 a[2]=x[0]*y[1]-x[1]*y[0];
 b[0]=y[1]-y[2];
 b[1]=y[2]-y[0];
 b[2]=y[0]-y[1];
 c[0]=x[2]-x[1];
 c[1]=x[0]-x[2];
 c[2]=x[1]-x[0];
 A=0.5*((x[0]*y[1]-x[1]*y[0])+(x[2]*y[0]-x[0]*y[2])+(x[1]*y[2]-x[2]*y[1]));
 for(int i=0;i<3;i++)
 { B[0][i]=1.0/(2.0*A)*b[i];
   B[1][i]=1.0/(2.0*A)*c[i];
 }
```

```
 l[0]=Math.sqrt(b[2]*b[2]+c[2]*c[2]);
 l[1]=Math.sqrt(b[0]*b[0]+c[0]*c[0]);
 l[2]=Math.sqrt(b[1]*b[1]+c[1]*c[1]);
}
public triangular_element1(int n_elementi,int n_pointi[],double xi[],double yi[],double dti,int Pi)
{
 x=new double[3];
 y=new double[3];
 Ti=new double[3];
 l=new double[3];//0:lij,1:ljk,2:lik

 n_element=n_elementi;
 for(int i=0;i<3;i++)
 {n_point[i]=n_pointi[i];
  x[i]=xi[i];y[i]=yi[i];
  xc+=x[i];yc+=y[i];
 }
 xc/=3.0;
 yc/=3.0;
 a=new double[3];
 b=new double[3];
 c=new double[3];
 N=new double[3];
 B=new double[2][3];
 a[0]=x[1]*y[2]-x[2]*y[1];
 a[1]=x[2]*y[0]-x[0]*y[2];
 a[2]=x[0]*y[1]-x[1]*y[0];
 b[0]=y[1]-y[2];
 b[1]=y[2]-y[0];
 b[2]=y[0]-y[1];
 c[0]=x[2]-x[1];
 c[1]=x[0]-x[2];
 c[2]=x[1]-x[0];
 A=0.5*((x[0]*y[1]-x[1]*y[0])+(x[2]*y[0]-x[0]*y[2])+(x[1]*y[2]-x[2]*y[1]));
 for(int i=0;i<3;i++)
 { B[0][i]=1.0/(2.0*A)*b[i];
   B[1][i]=1.0/(2.0*A)*c[i];
 }
 l[0]=Math.sqrt(b[2]*b[2]+c[2]*c[2]);
 l[1]=Math.sqrt(b[0]*b[0]+c[0]*c[0]);
 l[2]=Math.sqrt(b[1]*b[1]+c[1]*c[1]);
}

public int setI(int cond,double L0,double L,double W0,double W)
{
   int i=100;
                if(cond==0) //x=0
                {if(x[0]==L0 && x[1]==L0) i=0;
                 else if(x[1]==L0 && x[2]==L0) i=1;
                 else if(x[0]==L0 && x[2]==L0) i=2;
                 else i=-1;
                }
                else if(cond==1) //x=L
                {if(x[0]==L && x[1]==L) i=0;
                 else if(x[1]==L && x[2]==L) i=1;
                 else if(x[0]==L && x[2]==L) i=2;
                 else i=-1;
                }
                else if(cond==2) //y=0
                {if(y[0]==W0 && y[1]==W0) i=0;
                 else if(y[1]==W0 && y[2]==W0) i=1;
                 else if(y[0]==W0 && y[2]==W0) i=2;
                 else i=-1;
                }
                else if(cond==3) //y=W
                {if(y[0]==W && y[1]==W) i=0;
                 else if(y[1]==W && y[2]==W) i=1;
                 else if(y[0]==W && y[2]==W) i=2;
                 else i=-1;
                }
                return i;
}
```

```
public void setN(double x,double y)
{for(int i=0;i<3;i++)
  {N[i]=1.0/(2.0*A)*(a[i]+b[i]*x+c[i]*y);}
}
public void setN(double z[])
{setN(z[0],z[1]);}

public void setT(double TT[])
{for(int i=0;i<3;i++)
  {Ti[i]=TT[i];}
}

public double[] getT()
{ return Ti;}

public double T(double x,double y)
{ //interpolation
            double TT=0;
 setN(x,y);
 for(int i=0;i<3;i++)
  {TT+=N[i]*Ti[i];}
   return TT;
}

public double T(double T[],double x,double y)
{ //set grid point temperatures and get interpolated value
            setT(T);
  setN(x,y);
  double TT=0;
 for(int i=0;i<3;i++)
  {TT+=N[i]*Ti[i];}
  return TT;
}
public double T(double T[],double z[])
{ return T(T,z[0],z[1]);}

}
```

Program 6.16.2 linear triangular finite simplex grid (curve fitting)

```
 import javax.swing.*;
import org.math.plot.*;
import static java.lang.Math.*;
import static org.math.array.DoubleArray.*;
import visad.*;
import visad.java3d.DisplayImplJ3D;
import visad.java2d.DisplayImplJ2D;
import java.rmi.RemoteException;
import java.awt.*;
import javax.swing.*;
import java.awt.event.*;
public class triangular_grid1
{
public triangular_element1 te[];
public setElement se[];
public int N,M,K,K1,K2,Nm1,Mm1;
public double x[][];
public double y[][];
public double T[][];
public double Ti[];
public double xi[];
public double yi[];
public boolean bi[];
public double dx,dy;
public int Bx0[],BxM[],By0[],ByN[];
public double L,W;
public double min(double xi[][])
{ double min=1e200;
  for(int i=0;i<xi.length;i++)
   {for(int j=0;j<xi[0].length;j++)
     {if(xi[i][j]<min) min=xi[i][j];}
 }
 return min;
}
```

```
public double max(double xi[][])
{double max=-1e200;
 for(int i=0;i<xi.length;i++)
 {for(int j=0;j<xi[0].length;j++)
   {if(xi[i][j]<max) max=xi[i][j];}
 }
 return max;
}
public triangular_grid1(double xi[][],double yi[][],double zi[][])
{x=xi;
 y=yi;
 T=zi;
 N=x.length-1;
 M=x[0].length-1;
 L=max(xi);
 W=max(yi);
 setElements();
}

public void generateTriangularGrid(double Li, double Wi,int Ni, int Mi,double zi[][])
{int i,j;
 N=Ni;M=Mi;
 L=Li;
 W=Wi;
 x=new double[N+1][M+1];
 y=new double[N+1][M+1];
 T=zi;
 setElements();
}

public void setElements()
{ int i,j;
 dx=L/M;
 dy=W/N;
 K=M*N;
 K1=K*2;
 K2=K1+1;
 //grid points
 xi=new double[K2];
 yi=new double[K2];
 Ti=new double[K2];
 bi=new boolean[K2];
 se=new setElement[K2];
 for(i=0;i<K2;i++) {se[i]=new setElement();}
 te=new triangular_element1[K1];
 int k1=-1,k=0,n=0,m=0;
for(n=0;n<=N;n++)
{ for(m=0;m<=M;m++)
  {k=(N+1)*n+m;
   x[n][m]=m*dx;y[n][m]=n*dy;
   xi[k]=x[n][m];
   yi[k]=y[n][m];
   bi[k]=false;
   Ti[k]=T[n][m];
  }
}

for(n=0;n<N;n++)
{for(m=0;m<M;m++)
  { k=(N+1)*n+m;
              int kp1=(N+1)*(n+1)+m;
              int kp1p1= kp1+1;
              double xi1[]={x[n][m],x[n][m+1],x[n+1][m]};
              double yi1[]={y[n][m],y[n][m+1],y[n+1][m]};
    double zi1[]={T[n][m],T[n][m+1],T[n+1][m]};
              k1++;
              int n_point1[]={k,(k+1),kp1};
              se[k].add(k1,0);
              se[k+1].add(k1,1);
              se[kp1].add(k1,2);
              te[k1]=new triangular_element1(k1,n_point1,xi1,yi1);
              te[k1].setT(zi1);
              double xi2[]={x[n][m+1],x[n+1][m+1],x[n+1][m]};
              double yi2[]={y[n][m+1],y[n+1][m+1],y[n+1][m]};
```

```
                double zi2[]={T[n][m+1],T[n+1][m+1],T[n+1][m]};
                k1++;
                int n_point2[]={k+1,kp1p1,kp1};
    se[k+1].add(k1,0);
                se[kp1p1].add(k1,1);
                se[kp1].add(k1,2);
                te[k1]=new triangular_element1(k1,n_point2,xi2,yi2);
                te[k1].setT(zi2);
  }
}
}
public triangular_grid1(double Li, double Wi,int Ni, int Mi,double zz[][])
{generateTriangularGrid(Li,Wi,Ni,Mi,zz);}
public int find_element(double z[])
{ return find_element(z[0],z[1]);}
public int find_element(double x,double y)
{double R1=Math.sqrt(x*x+y*y);
 int kmin=0;
 double min=1.0e50;
 for(int k=0;k<K1;k++)
 {   te[k].setN(x,y);
     double r0=Math.sqrt((te[k].x[0]-x)*(te[k].x[0]-x)+(te[k].y[0]-y)*(te[k].y[0]-y));
     double r1=Math.sqrt((te[k].x[1]-x)*(te[k].x[1]-x)+(te[k].y[1]-y)*(te[k].y[1]-y));
     double r2=Math.sqrt((te[k].x[2]-x)*(te[k].x[2]-x)+(te[k].y[2]-y)*(te[k].y[2]-y));
     double dr=Math.sqrt(r0*r0+r1*r1+r2*r2);
                 boolean b1=(x>=te[k].x[0]) && (x<=te[k].x[2]);
                 boolean b2=(y>=te[k].y[0]) && (y<=te[k].y[1]);
                 boolean b= b1 && b2;
     if(dr<min) {min=dr;kmin=k;}
 }
 return kmin;
}

public void ContourPlot()
{
ContourPlot pp=new ContourPlot( x,y,T,16,true,"Finite element","x","y");
pp.plot();
}
public void Plot3D(double L0,double L,double W0,double W) throws RemoteException, VisADException
{
Plot3D.plot3D_CD1(T,L0,L,W0,W,200,"x","y","T degree C");
}
public void GridPlot(double L0,double L,double W0,double W)
{Plot3DPanel plot = new Plot3DPanel("SOUTH");
 double dx=(L-L0)/M;
 double dy=(W-W0)/N;
 double[] xx = increment(L0, dx,L); // x = 0.0:0.1:1.0
 double[] yy = increment(W0,dy,W);
 plot.addGridPlot("Finite element Grid", xx, yy, T);
 JFrame frame = new JFrame("a plot panel");
 frame.setSize(600, 600);
 frame.setContentPane(plot);
 frame.setVisible(true);
}
public double getT(double x,double y)
{double z[]={x,y};
 int n=find_element(z);
 double T1=te[n].T(x,y);
 return T1;
}

public static void main(String arg[]) throws RemoteException, VisADException
{double L=1.0;
 double W=1.0;
 int N=2,M=2;
 double TT[][]={{80.0,90.0,100.0},{90.0,100.0,110.0},{100.0,110.0,120.0}};
 double hi=20.0;
 triangular_grid1 tg=new triangular_grid1(L,W,M,N,TT);
 double z[]={0.65,0.55};
 int n=tg.find_element(z);
 System.out.println("element = "+n);
 double T=tg.getT(z[0],z[1]);
 System.out.println("T = "+T);
}
```

```
}
```

```
---------- Capture Output ----------
> "C:\java\bin\java.exe" triangular_grid1
element = 6
T = 104.00000000000001

> Terminated with exit code 0.
```

6.17 ARTIFICIAL NEURAL NETWORKS
(Note : This section and its programs is prepared by Dr. Mert Sinan Turgut)

Artificial Neural Networks (ANN) is a mathematical model inspired from animal central nervous systems, particularly brain.

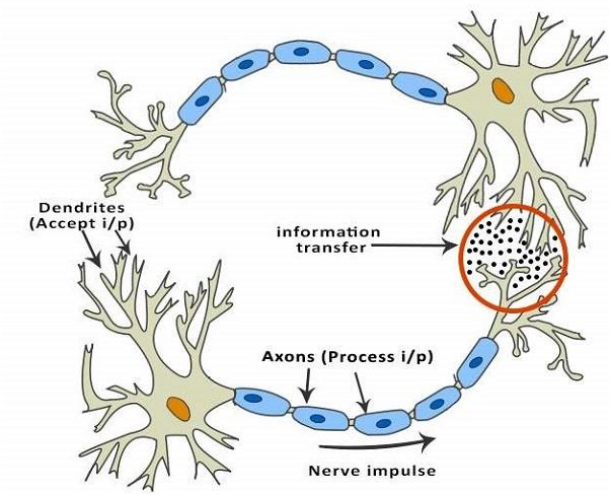


Figure 6.17.1 Brain neuron structure

ANNs generally used to estimate or approximate functions for large number of inputs. Most ANNs consist of one input layer, one hidden layer and one output layer. An ANN with at least one hidden layer, in which each neuron is represented with a non-linear differentiable activation function is called “universal approximator” since it is shown that it can approximate any function. A one layer artificial neural network is only consisted input and outpu layers. The simlest one layer ANN is called ADALINE neoral Network. In figüre 6.7.2 such a network is shown.

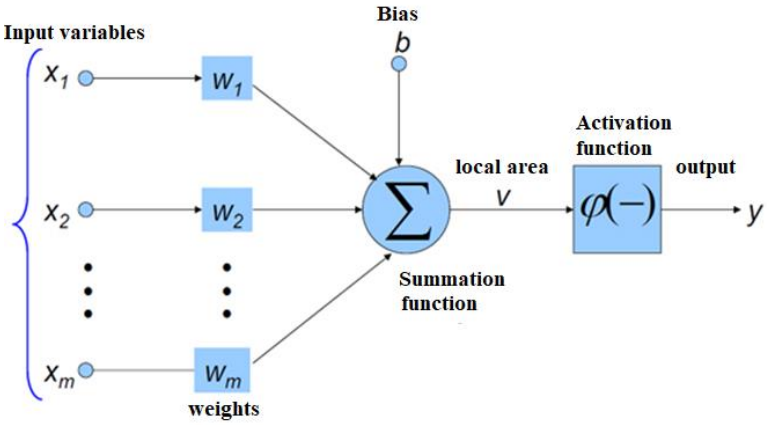
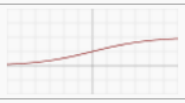
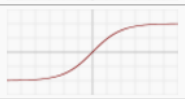
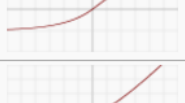


Figure 6.17.2 An ADELINE network

The output y can be expressed as

$v = \sum_{j=1}^{n} x_j w_j + b$ $\qquad\qquad$ $y = \varphi(v) = \varphi\left(\sum_{j=1}^{n} x_j w_j + b\right)$

Different activation functions can be defined in artificical neural networks. some of them are:

Name	Plot	Equation	Derivative
Identity		$f(x) = x$	$f'(x) = 1$
Binary step		$f(x) = \begin{cases} 0 & \text{for } x < 0 \\ 1 & \text{for } x \geq 0 \end{cases}$	$f'(x) = \begin{cases} 0 & \text{for } x \neq 0 \\ ? & \text{for } x = 0 \end{cases}$
Logistic (a.k.a Soft step)		$f(x) = \frac{1}{1+e^{-x}}$	$f'(x) = f(x)(1-f(x))$
TanH		$f(x) = \tanh(x) = \frac{2}{1+e^{-2x}} - 1$	$f'(x) = 1 - f(x)^2$
ArcTan		$f(x) = \tan^{-1}(x)$	$f'(x) = \frac{1}{x^2+1}$
Rectified Linear Unit (ReLU)		$f(x) = \begin{cases} 0 & \text{for } x < 0 \\ x & \text{for } x \geq 0 \end{cases}$	$f'(x) = \begin{cases} 0 & \text{for } x < 0 \\ 1 & \text{for } x \geq 0 \end{cases}$
Parameteric Rectified Linear Unit (PReLU)[2]		$f(x) = \begin{cases} \alpha x & \text{for } x < 0 \\ x & \text{for } x \geq 0 \end{cases}$	$f'(x) = \begin{cases} \alpha & \text{for } x < 0 \\ 1 & \text{for } x \geq 0 \end{cases}$
Exponential Linear Unit (ELU)[3]		$f(x) = \begin{cases} \alpha(e^x - 1) & \text{for } x < 0 \\ x & \text{for } x \geq 0 \end{cases}$	$f'(x) = \begin{cases} f(x) + \alpha & \text{for } x < 0 \\ 1 & \text{for } x \geq 0 \end{cases}$
SoftPlus		$f(x) = \log_e(1+e^x)$	$f'(x) = \frac{1}{1+e^{-x}}$

A graphical reprsentation of a Multi Layer Perceptron (MLP), a neural network with at least one hidden layer, consisting of one input, one hidden and one output layers is provided in Figure 6.17.1.

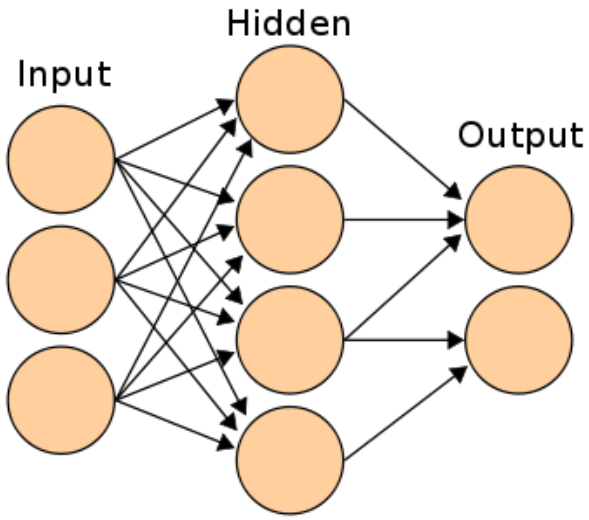


Figure 6.17.1. A Multi Layer Perceptron [1]

To approximate a function, neural network modifies the biases, a real number without a variable parameter, of each and neuron weights of connections between each neuron in different layers. This modifying process is called “training”. Various training procedures have been studied in the literature. One of the most common training procedure is called “backpropagation” [1].

Backpropagation means backpropagation of errors occuring in the output layer. Backpropagation is used with optimization methods such as gradient descent and the method employs the chain rule, which requires a non-linear differentiable function, which is called activation function, is assigned to the each neuron at least in the hidden layer. Hyperbolic tangent function and logistic sigmoid function are commonly used as activation functions in ANNs. Mathematical representations of logistic sigmoid, hyperbolic tangent functions and their derivatives, to be used later, are given below:

Logistic Sigmoid: $\varphi(x) = \frac{1}{1+e^{-x}}$

Derivative of logistic sigmoid: $\varphi(x)(1-\varphi(x))$

Hyperbolic Tangent: $\varphi(x) = \frac{e^{x}-e^{-x}}{e^{x}+e^{-x}}$

Derivative of hyperbolic tangent: $(1-\varphi(x))^2$

Backpropagation learning algorithm mainly consists of two phases: Propagation and weight update. In propagation phase, firstly, training input is forward propagated through the neurons in each layer to generate outputs. Mathematical representation of forward propagation is given below.

$net = x_1 w_1 + x_2 w_2$

y=f(net)

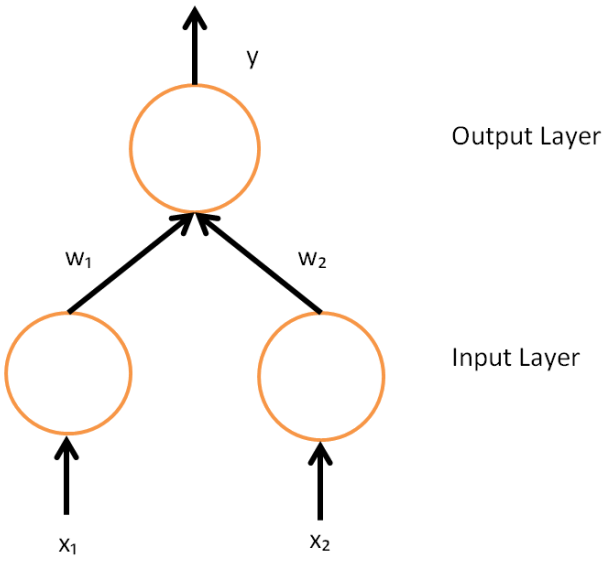
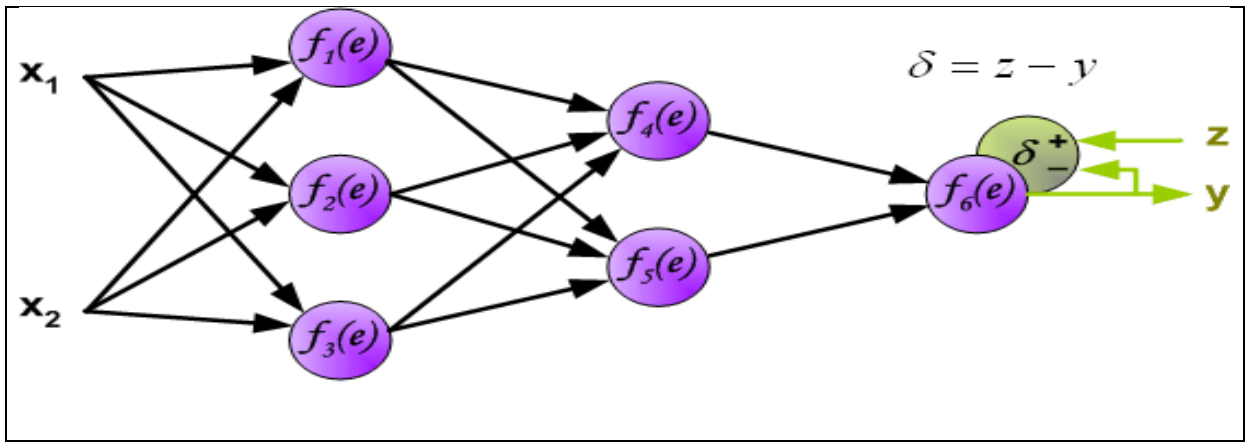


Figure 6.17.2. Forward propagation example [1]

where *x* represents in the inputs to the network, *y* shows the output, *w* represents the weights and *f()* represents the activation of the neuron. After the forward propagation finished and outputs of the network are generated, second step of the propagation phase begins, namely backpropagation. In the backpropagation step, difference between the desired output and calculated output, error, is minimized by a derivative-based optimization algorithm. One of the most commonly used derivative-based algorithm for **backpropagation** is gradient descent algorithm.



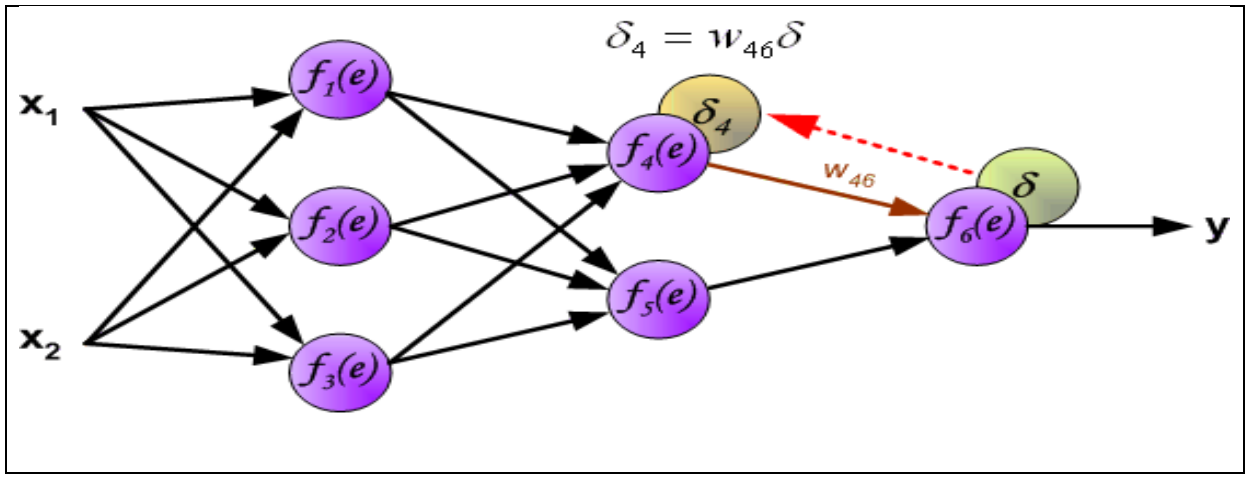


Figure 6.17.3 backpropogation

Generally, to measure the difference between the outputs, squared error function is used:

$$E = \frac{1}{2}(t-y)^2$$

where *t* is the target output, *y* is the actual output and *E* is the squared error. In backpropagation, partial derivative of the error with respect to a weight is calculated by using the chain rule twice.

$$\frac{\partial E}{\partial w_{ij}} = \frac{\partial E}{\partial o_j}\frac{\partial o_j}{\partial net_j}\frac{\partial net_j}{\partial w_{ij}}$$

$$\frac{\partial net_j}{\partial w_{ij}} = \frac{\partial}{\partial w_{ij}}\left(\sum_{k=1}^{n} w_{kj}\, o_k\right) = o_i$$

$$\frac{\partial o_j}{\partial net_j} = \frac{\partial}{\partial net_j}\varphi(net_j) = (1-\varphi(net_j))^2 \quad \text{(assumed sigmoid function)}$$

$$\frac{\partial E}{\partial o_j} = \frac{\partial E}{\partial y} = \frac{\partial E}{\partial}\left[\frac{1}{2}(t-y)^2\right] = y$$

Since *E* is a function of all neurons:

$$\frac{\partial E}{\partial o_j} = \sum_{l\in L}\left(\frac{\partial E}{\partial net_l}\frac{\partial net_j}{\partial o_j}\right) = \sum_{l\in L}\left(\frac{\partial E}{\partial o_l}\frac{\partial o_l}{\partial net_j}w_{jl}\right)$$

After backpropagation of error finishes, the weight updating phase, the second phase, starts. For all iterations, each weight is updated by the following rule. α is the learning rate which determines the step size of each update.

$$\Delta w_{ij} = -\alpha\frac{\partial E}{\partial w_{ij}} = \begin{cases} -\alpha o_i(o_j - t_j)o_j(1-o_j) \; if \; j \; is \; an \; output \; neuron \\ -\alpha o_i\left(\sum_{l\in L}(\delta_l w_{jl})\right)o_j(1-o_j) \; if \; j \; is \; an \; inner \; neuron \end{cases}$$

This procedure continues after a desired error value is reached or maximum number of iterations is accomplished. After the training finishes, test inputs can be forward propagated with the calculated weights and fitted, predicted, results can be found.

As a first programming example here an ADELINE network is given. Please note that result of one layer ADELINE is similar to linear interpolation process

Program 6.17-1 ADELINE network of single layer

```
import java.io.*;
import java.util.*;
import javax.swing.*;
import javax.swing.table.*;
import java.awt.*;
class adalinenetwork{
int numout,sizeinput,numinput;
double error;
double weights[];
double bias[];
double x[][];
double trainedweights[];
double predicted;
double trainedbias[];
double d[];
double totalerror[];
double total[];
int imaxiteration;
double ilearningrate,output;

public adalinenetwork(double ix[][],double id[], int maxiteration, double learningrate)
{
x=ix;
d=id;
ilearningrate=learningrate;
imaxiteration=maxiteration;
numout=1;  //Number of output
sizeinput=x.length;
numinput=x[0].length;
weights=new double[numinput];
total=new double[numinput+1];
trainedweights=new double[numinput];
trainedbias=new double[numout];
bias=new double[numout];
totalerror=new double[imaxiteration];
initWeights();
}
public void initWeights()
{
for(int i=0;i<numinput;i++)
{
weights[i]=Math.random();
}
for(int i=0;i<numout;i++)
{
bias[i]=Math.random();
}
}
public double tanjanth(double x)
{
double p=(Math.exp(x)-Math.exp(-x))/(Math.exp(x)+Math.exp(-x));
return p;
}

public double sigmoid(double x)
{
double p=1.0/(1.0+Math.exp(-x));
return p;
}
```

```
public double[] train()
{
for(int i=0;i<imaxiteration;i++)
{
		error=0.0;
		for(int j=0;j<sizeinput;j++)
		{
			output=0.0;
			for(int k=0;k<numinput;k++)
			{
				output+=weights[k]*x[j][k];
			}
			output+=bias[0];
			error+=(d[j]-output);
		}
		totalerror[i]=error;
		for(int j=0;j<sizeinput;j++)
		{
			for(int k=0;k<numinput;k++)
			{
				weights[k]+=ilearningrate*error*x[j][k];
			}
		}
		bias[0]+=ilearningrate*error;
}
for(int k=0;k<numinput;k++)
{
		total[k]=weights[k];
}
total[numinput]=bias[0];
System.out.println("Final error is="+totalerror[imaxiteration-1]);
return total;
}


void writeMatrix(String filename) {
double[] total=train();
try {
BufferedWriter bw = new BufferedWriter(new FileWriter(filename));
for (int i = 0; i < total.length; i++) {
bw.write(total[i] + " ");
bw.newLine();
}
bw.flush();
} catch (IOException e) {}
}


public static double[][] readDouble(String filename)
{
//read data in matrix form
// data length is not limited
Vector<String> xi=new Vector<String>();
int nn;
int m=2;
int i=0;
BufferedReader fin;
try{
fin=new BufferedReader(new FileReader(filename));
try {
try{
try{
try{
```

```
String s1=Text.readStringLine(fin);
StringTokenizer token=new StringTokenizer(s1);
int n=token.countTokens()-1;
m=n+1;
while(token.hasMoreTokens())
{
String s2=token.nextToken();
xi.add(s2);
}
i=0;
while(fin != null)
{
i++;
String s11=Text.readStringLine(fin);
if(s11=="") break;
StringTokenizer token1=new StringTokenizer(s11);
while(token1.hasMoreTokens())
{
String s2=token1.nextToken();
xi.add(s2);
}
}
} catch(ArrayIndexOutOfBoundsException e_eof) { }
} catch(StringIndexOutOfBoundsException e_eof) { }
} catch(EOFException e_eof) {System.out.println("end of file"); }
} catch(NumberFormatException e_nfe) {System.out.println("non-number input"); }
} catch(IOException e_io) {System.out.println("file name is not existed"); }
nn=i;
double aa[][]=new double[nn][m];
Enumeration<String> nx=xi.elements();
for(i=0;i<nn;i++)
{
for(int j=0;j<m;j++)
{
String s1=nx.nextElement();
Double ax=new Double(s1);
aa[i][j]=ax.doubleValue();
}
}
return aa;
}


public double predict(double x[][])
{
double[][] totall=readDouble("weightsadaline.txt");
for(int k=0;k<numinput;k++)
{
        trainedweights[k]=totall[k][0];
}
trainedbias[0]=totall[numinput][0];
output=0.0;
for(int k=0;k<numinput;k++)
{
        output+=trainedweights[k]*x[0][k];
}
output+=trainedbias[0];
return output;
}
}
```

Program 6.17-2 ADELINE network of single layer a test case

```
class adalinenetworktest
{
public static void main(String args[])
```

```
{
double[] d={2,5,7,8};
double[][] x={{1,1},{2,2},{3,3},{4,4}};
double[][] test={{2.5,2.5}};
int maxiteration=100;
double learningrate=0.001;
adalinenetwork nn=new adalinenetwork(x,d,maxiteration,learningrate);
nn.writeMatrix("weightsadaline.txt");
double predicts=nn.predict(test);
System.out.println("Predicted value is="+predicts);
}
}
```

```
---------- Capture Output ----------
> "E:\co\java\bin\java.exe" adalinenetworktest
Final error is=2.2556898571224338E-9
Predicted value is=5.499999999551117

> Terminated with exit code 0.
```

In the second example we will have hidden layers in between given layers, therefore propagation and backpropogation will applied for all this layers.

Program 6.17-1 neuralnetworkbackpropogation

```
import java.io.*;
import java.util.*;
class neuralnetworkbackprop{
int numout,sizeinput,numinput,numhid;
double error;
double weights[][];
double bias[];
double a[];
double totalerror[];
double fnet[];
double fnet1[];
double H1[];
double delta[];
double x[][];
double trainedweights[][];
double predicted[];
double trainedbias[];
double d[];
double H[];
double total[][];
int u,r,imaxiteration,inumhid,y,s;
double ilearningrate;
public neuralnetworkbackprop(double ix[][],double id[], int maxiteration, double learningrate, int numhid)
{
x=ix;
d=id;
ilearningrate=learningrate;
inumhid=numhid;
imaxiteration=maxiteration;
numout=1;
sizeinput=x.length;
numinput=x[0].length;
predicted=new double[numout];
fnet=new double[inumhid+numout];
fnet1=new double[inumhid+numout];
H1=new double[inumhid+numout];
delta=new double[inumhid+numout];
H=new double[inumhid+numout];
weights=new double[numout+numinput][inumhid];
```

```
trainedweights=new double[numout+numinput][inumhid];
trainedbias=new double[inumhid+numout];
a=new double[inumhid+numout];
bias=new double[inumhid+numout];
total=new double[inumhid+numout+numout+numinput][inumhid];
totalerror=new double[imaxiteration];
initWeights();
}
public void initWeights()
{
for(int i=0;i<numout+numinput;i++)
{
for(int j=0;j<inumhid;j++)
{
weights[i][j]=Math.random()-0.5;
}
}
for(int i=0;i<numout+inumhid;i++)
{
bias[i]=Math.random()-0.5;
}
}
public double tanjanth(double x)
{
double p=(Math.exp(x)-Math.exp(-x))/(Math.exp(x)+Math.exp(-x));
return p;
}
public double sigmoid(double x)
{
double p=1.0/(1.0+Math.exp(-x));
return p;
}
public double[][] train()
{
for(int i=0;i<imaxiteration;i++)
{
error=0.0;
for(int j=0;j<sizeinput;j++)
{
for(int k=0;k<inumhid+numout;k++)
{
delta[k]=0.0;
H[k]=0.0;
}
//hidden layer forward propagation
for(int k=0;k<inumhid;k++)
{
for(int l=0;l<numinput;l++)
{
H[k]+=x[j][l]*weights[l][k];
}
H[k]+=bias[k];
fnet[k]=tanjanth(H[k]);
}
//output layer forward propagation
u=0;
for(int k=inumhid;k<inumhid+numout;k++)
{
for(int l=0;l<inumhid;l++)
{
H[k]+=fnet[l]*weights[numinput+u][l];
}
H[k]+=bias[k];
fnet[k]=H[k];
```

```
u++;
}
//backpropagation starts here
//backpropagation in output layer
for(int k=inumhid;k<inumhid+numout;k++)
{
delta[k]=(d[j]-fnet[k]);
}
//backpropagation in hidden layer
for(int k=0;k<inumhid;k++)
{
for(int l=0;l<numout;l++)
{
delta[k]+=weights[numinput+l][k]*delta[inumhid+l];
}
delta[k]=(1.0-(fnet[k]*fnet[k]))*delta[k];
}
//bias update
for(int k=0;k<inumhid+numout;k++)
{
bias[k]+=ilearningrate*delta[k];
}
//hidden layer update
for(int k=0;k<numinput;k++)
{
for(int l=0;l<inumhid;l++)
{
weights[k][l]+=ilearningrate*x[j][k]*delta[l];
}
}
//output layer update
for(int k=0;k<numout;k++)
{
for(int l=0;l<inumhid;l++)
{
weights[numinput+k][l]+=ilearningrate*fnet[l]*delta[inumhid+k];
}
}
for(int k=0;k<numout;k++)
{
error+=(0.5*(d[j]-fnet[inumhid+k])*(d[j]-fnet[inumhid+k]));
}
}
totalerror[i]=error;
}
for(int i=0;i<numout+numinput;i++)
{
for(int j=0;j<inumhid;j++)
{
total[i][j]=weights[i][j];
}
}
y=0;
for(int i=numout+numinput;i<numout+numout+numinput+inumhid;i++)
{
total[i][0]=bias[y];
y++;
}
System.out.println("Final MSE error is="+totalerror[imaxiteration-1]);
return total;
}
void writeMatrix(String filename) {
double[][] total=train();
try {
```

```
BufferedWriter bw = new BufferedWriter(new FileWriter(filename));
for (int i = 0; i < total.length; i++) {
for (int j = 0; j < total[i].length; j++) {
bw.write(total[i][j] + " ");
}
bw.newLine();
}
bw.flush();
} catch (IOException e) {}
}
public static double[][] readDouble(String filename)
{
//read data in matrix form
// data length is not limited
Vector<String> xi=new Vector<String>();
int nn;
int m=2;
int i=0;
BufferedReader fin;
try{
fin=new BufferedReader(new FileReader(filename));
try {
try{
try{
try{
String s1=Text.readStringLine(fin);
StringTokenizer token=new StringTokenizer(s1);
int n=token.countTokens()-1;
m=n+1;
while(token.hasMoreTokens())
{
String s2=token.nextToken();
xi.add(s2);
}
i=0;
while(fin != null)
{
i++;
String s11=Text.readStringLine(fin);
if(s11=="") break;
StringTokenizer token1=new StringTokenizer(s11);
while(token1.hasMoreTokens())
{
String s2=token1.nextToken();
xi.add(s2);
}
}
} catch(ArrayIndexOutOfBoundsException e_eof) { }
} catch(StringIndexOutOfBoundsException e_eof) { }
} catch(EOFException e_eof) {System.out.println("end of file"); }
} catch(NumberFormatException e_nfe) {System.out.println("non-number input"); }
} catch(IOException e_io) {System.out.println("file name is not existed"); }
nn=i;
double aa[][]=new double[nn][m];
Enumeration<String> nx=xi.elements();
for(i=0;i<nn;i++)
{
for(int j=0;j<m;j++)
{
String s1=nx.nextElement();
Double ax=new Double(s1);
aa[i][j]=ax.doubleValue();
}
}
```

```
return aa;
}
public double[] predict(double x[][])
{
double[][] totall=readDouble("weights.txt");
for(int i=0;i<numout+numinput;i++)
{
for(int j=0;j<inumhid;j++)
{
trainedweights[i][j]=totall[i][j];
}
}
s=0;
for(int i=numout+numinput;i<numout+numout+numinput+inumhid;i++)
{
trainedbias[s]=totall[i][0];
s++;
}
for(int k=0;k<inumhid;k++)
{
for(int l=0;l<numinput;l++)
{
H1[k]+=x[0][l]*trainedweights[l][k];
}
H1[k]+=trainedbias[k];
fnet1[k]=tanjanth(H1[k]);
}
//output layer forward propagation
u=0;
for(int k=inumhid;k<inumhid+numout;k++)
{
for(int l=0;l<inumhid;l++)
{
H1[k]+=fnet1[l]*trainedweights[numinput+u][l];
}
H1[k]+=trainedbias[k];
fnet1[k]=H1[k];
u++;
}
u=0;
for(int k=inumhid;k<inumhid+numout;k++)
{
predicted[u]=fnet1[k];
u++;
}
return predicted;
}
}
```

Program 6.17-2 neuralnetworkbackpropogationtest

```
class neuralnetworkbackproptest
{
public static void main(String args[])
{
double[] d={2,5,7,8};
double[][] x={{1,1},{2,2},{3,3},{4,4}};
double[][] test={{2,2}};
int maxiteration=10000;
int numhid=4;
double learningrate=0.1;
neuralnetworkbackprop nn=new neuralnetworkbackprop(x,d,maxiteration,learningrate,numhid);
nn.writeMatrix("weights.txt");
```

```
double[] predicts=nn.predict(test);
System.out.println("Predicted value is="+predicts[0]);
}
}
```

```
Result:

Final MSE error is=0.0
Predicted value is=5.0
```

Another way to find the desired weights and biases can be directly optimizing the error function with a metaheuristic algorithm after the forward propagation. Crow Search (CS) is a powerful metaheuristic algorithm good at global search, inspired by food searching mechanism of crow swarms [2]. In the following example crow search is used to train a neural network.

Program 6.17-3 crow_search_algorithm

```
import java.io.*;
import java.util.*;
abstract class f_xj
{abstract double func(double x[]);}
public class crow_search_algorithm
{
double[] Lower;
double[] Upper;
double[] BEST;
double[][] X;
double[][] MEM;
double[][] XNEW;
double[] fitness;
double[] fitmem;
double[] VALS;
double AP;
double fl;
int D;
int N;
int Maxiter;
int[] num;
f_xj ff;
public crow_search_algorithm(f_xj iff,double[] iLower,double[] iUpper, int iMaxiter,int iN)
{
ff=iff;
Lower=iLower;
Upper=iUpper;
D=Lower.length;
Maxiter=iMaxiter;
N=iN;
BEST=new double[Maxiter];
X=new double[N][D];
MEM=new double[N][D];
XNEW=new double[N][D];
fitness=new double[N];
fitmem=new double[N];
num=new int[N];
VALS=new double[D];
AP=0.1;
fl=2.0;
}
void init()
{
```

```
for(int i=0;i<N;i++)
{for(int j=0;j<D;j++)
{X[i][j]=Lower[j]+((Upper[j]-Lower[j])*Math.random());}}
for(int i=0;i<N;i++)
{fitness[i]=ff.func(X[i]);}
}
double[] getminval_index(double[] a)
{
double m=0.0;
double b[]=new double[a.length];
for(int i=0;i<a.length;i++)
{b[i]=a[i];}
double minval=a[0];
for(int i=0;i<a.length;i++)
{if(a[i]<minval){minval=a[i];}}
for(int i=0;i<a.length;i++)
{if(b[i]==minval){m=i;break;}};
double[] dep=new double[2];
dep[0]=minval;
dep[1]=m;
return dep;
}
double[][] solution()
{
init();
for(int i=0;i<N;i++)
{for(int j=0;j<D;j++)
{MEM[i][j]=X[i][j];}
fitmem[i]=fitness[i];}
int iter=1;
while(iter<Maxiter)
{
for(int i=0;i<N;i++)
{num[i]=(int)Math.floor(Math.random()*(double)N);}
for(int i=0;i<N;i++)
{
if(Math.random()>AP)
{for(int j=0;j<D;j++)
{XNEW[i][j]=X[i][j]+fl*Math.random()*(MEM[num[i]][j] - X[i][j]);}
}
else
{
for(int j=0;j<D;j++)
{XNEW[i][j]=Lower[j]-((-Upper[j]+Lower[j])*Math.random());}
}
}
for(int i=0;i<N;i++)
{for(int j=0;j<D;j++)
{if((XNEW[i][j]>Upper[j])||(XNEW[i][j]<Lower[j]))
{XNEW[i][j]=Lower[j]+((Upper[j]-Lower[j])*Math.random());}}}
for(int i=0;i<N;i++)
{
fitness[i]=ff.func(XNEW[i]);
if( fitness[i]<fitmem[i] )
{
fitmem[i]=fitness[i];
for(int j=0;j<D;j++)
{MEM[i][j]=XNEW[i][j];X[i][j]=XNEW[i][j];}
}
}
double[] AA2=getminval_index(fitmem);
double fitval2=AA2[0];
BEST[iter]=fitval2;
iter++;
```

```
}
double[] plott=new double[Maxiter];
for(int i=0;i<Maxiter;i++)
{plott[i]=i;}
Plot pp=new Plot(plott,BEST);
// pp.plot();
double[] AA=getminval_index(fitmem);
double fitval=AA[0];
int mm=(int)AA[1];
for(int i=0;i<D;i++)
{VALS[i]=MEM[mm][i];}
double out[][]=new double[2][D];
out[0][0]=fitval;
for(int i=0;i<D;i++)
{out[1][i]=VALS[i];}
return out;
}
void trainandwrite()
{
double[][] out=solution();
double[] result=new double[D];
System.out.println("Optimized value = "+out[0][0]);
for(int i=0;i<D;i++)
{result[i]=out[1][i];
System.out.println("x["+i+"] = "+out[1][i]);}
writeMatrix("weightscrow.txt",result);
}
void writeMatrix(String filename,double[] total) {
try {
BufferedWriter bw = new BufferedWriter(new FileWriter(filename));
for (int i = 0; i < total.length; i++) {
bw.write(total[i] + " ");
}
bw.flush();
} catch (IOException e) {}
}
void toTxtfile()
{   String named="crow_f27_sche.txt";
String line="";
int lineNo;
try
{
FileWriter fw=new FileWriter(named,true);
BufferedWriter bw=new BufferedWriter(fw);
LineNumberReader lnr=new LineNumberReader(new FileReader(named));
lnr.setLineNumber(1);
for(int i=1;i<=lnr.getLineNumber();i++)
{bw.newLine();}
double[][] dd=solution();
Double d1d=new Double(dd[0][0]);
String s1=d1d.toString();
int nn=dd[1].length;
Double[] val=new Double[nn];
String[] str=new String[nn];
for(int i=0;i<nn;i++)
{val[i]=new Double(dd[1][i]);}
for(int i=0;i<nn;i++)
{str[i]=val[i].toString();}
bw.write(s1+" ");
for(int i=0;i<nn;i++)
{bw.write(str[i]+" ");}
bw.close();
lnr.close();
}catch(IOException e)
```

```
{e.printStackTrace();}
}
}
```

Program 6.17-4 neuralnetwork

```
import java.io.*;
import java.util.*;
class neuralnetwork extends f_xj
{
int numout,sizeinput,numinput;
double error;
double weights[][];
double fnet[];
double s[][];
double d[];
double H[];
double predicted[];
double fnet1[];
double H1[];
int u,r,o,inumhid;
public neuralnetwork(double is[][],double id[],int numhid)
{
s=is;
d=id;
inumhid=numhid;
numout=1;
sizeinput=s.length;
numinput=s[0].length;
predicted=new double[numout];
fnet=new double[inumhid+numout];
fnet1=new double[inumhid+numout];
H1=new double[inumhid+numout];
H=new double[inumhid+numout];
}
public double tanjanth(double x)
{
double p=(Math.exp(x)-Math.exp(-x))/(Math.exp(x)+Math.exp(-x));
return p;
}
public double func(double x[])
{
error=0.0;
for(int j=0;j<sizeinput;j++)
{
for(int k=0;k<inumhid+numout;k++)
{
H[k]=0.0;
}
r=0;
//hidden layer forward propagation
for(int k=0;k<inumhid;k++)
{
for(int l=0;l<numinput;l++)
{
H[k]+=s[j][l]*x[r];
r++;
}
H[k]+=x[k+(numinput*inumhid)+(inumhid*numout)];
fnet[k]=tanjanth(H[k]);
}
//output layer forward propagation
u=0;
```

```
for(int k=inumhid;k<inumhid+numout;k++)
{
for(int l=0;l<inumhid;l++)
{
H[k]+=fnet[l]*x[l+(inumhid*numinput)];
}
H[k]+=x[(numinput*inumhid)+(inumhid*numout)+inumhid+u];
fnet[k]=H[k];
u++;
}
for(int k=0;k<numout;k++)
{
error+=(0.5*(d[j]-fnet[inumhid+k])*(d[j]-fnet[inumhid+k]));
}
}
return error;
}
public static double[][] readMatrix(String filename)
{
//read data in matrix form
// data length is not limited
Vector<String> xi=new Vector<String>();
int nn;
int m=2;
int i=0;
BufferedReader fin;
try{
fin=new BufferedReader(new FileReader(filename));
try {
try{
try{
try{
String s1=Text.readStringLine(fin);
StringTokenizer token=new StringTokenizer(s1);
int n=token.countTokens()-1;
m=n+1;
while(token.hasMoreTokens())
{
String s2=token.nextToken();
xi.add(s2);
}
i=0;
while(fin != null)
{
i++;
String s11=Text.readStringLine(fin);
if(s11=="") break;
StringTokenizer token1=new StringTokenizer(s11);
while(token1.hasMoreTokens())
{
String s2=token1.nextToken();
xi.add(s2);
}
}
} catch(ArrayIndexOutOfBoundsException e_eof) { }
} catch(StringIndexOutOfBoundsException e_eof) { }
} catch(EOFException e_eof) {System.out.println("end of file"); }
} catch(NumberFormatException e_nfe) {System.out.println("non-number input"); }
} catch(IOException e_io) {System.out.println("file name is not existed"); }

nn=i;
double aa[][]=new double[nn][m];
Enumeration<String> nx=xi.elements();
for(i=0;i<nn;i++)
```

```
{
for(int j=0;j<m;j++)
{
String s1=nx.nextElement();
Double ax=new Double(s1);
aa[i][j]=ax.doubleValue();
}
}
return aa;
}
public double[] predict(double s[][],double x[][])
{
r=0;
for(int k=0;k<inumhid;k++)
{
for(int l=0;l<numinput;l++)
{
H1[k]+=s[0][l]*x[0][r];
r++;
}
H1[k]+=x[0][k+(numinput*inumhid)+(inumhid*numout)];
fnet1[k]=tanjanth(H1[k]);
}
//output layer forward propagation
o=0;
for(int k=inumhid;k<inumhid+numout;k++)
{
for(int l=0;l<inumhid;l++)
{
H1[k]+=fnet1[l]*x[0][l+(inumhid*numinput)];
}
H1[k]+=x[0][(numinput*inumhid)+(inumhid*numout)+inumhid+o];
fnet1[k]=H1[k];
o++;
}
u=0;
for(int k=inumhid;k<inumhid+numout;k++)
{
predicted[u]=fnet1[k];
u++;
}
return predicted;
}
}
public class crow_search_algorithm_test
{
public static void main(String args[])
{
double[] d={2,5,7,8};
double[][] x={{1,1},{2,2},{3,3},{4,4}};
int numout=1;
int numinput=x[0].length;
double[][] test={{4,4}};
int numhid=4;
neuralnetwork nn=new neuralnetwork(x,d,numhid);
int num=((numout+numinput)*numhid)+(numhid+numout);
double[] Upper=new double[num];
double[] Lower=new double[num];
int Maxiter=100000; // Maximum iteration number
int N=50; // Number of individuals in the swarm
for(int i=0;i<num;i++)
{
Lower[i]=-20;
Upper[i]=20;
```

```
}
crow_search_algorithm mo=new crow_search_algorithm(nn,Lower, Upper,Maxiter,N);
mo.trainandwrite();
double[][] weights=nn.readMatrix("weightscrow.txt");
double predictedval[]=nn.predict(test,weights);
System.out.println("Predicted value is="+predictedval[0]);
}
}
```

```
Optimized value = 0.0
x[0] = -0.5364913717025673
x[1] = -5.599577914584505
x[2] = 0.2746411516819427
x[3] = -8.500643573331972
x[4] = 0.9461278732704546
x[5] = 5.2128555702178305
x[6] = 0.08398781880606487
x[7] = 2.008159771957661
x[8] = -9.347666061384377
x[9] = 2.1777320165608804
x[10] = 3.575094965521974
x[11] = 1.557195226098894
x[12] = -15.622863320944523
x[13] = 9.745906789502728
x[14] = -8.378299501090064
x[15] = -5.918923693975516
x[16] = -4.2791888524335775
Predicted value is=8.0
```

6.18 PROBLEMS

PROBLEM 1

Fit a linear line by using polynomial least square curve fitting

X	2	3	4	7	8	9	5	5
Y	9	6	5	10	9	11	2	3

PROBLEM 2

For R22 refrigerant liquid viscosity values are required. The values are defined as the table below. Fit a curve to input viscosity into computer as a function. Calculate the amount of error.

-101.1111111	8.6519780084E-04	-28.88888889	3.0093836551E-04	32.22222222	1.5253606713E-04
-95.55555556	7.7466826506E-04	-26.11111111	2.9143069737E-04	35	1.4798892150E-04
-90	6.9943367368E-04	-23.33333333	2.8233640610E-04	37.77777778	1.4302839899E-04
-84.44444444	6.3536025795E-04	-20.55555556	2.7365549171E-04	40.55555556	1.3848125336E-04
-78.88888889	5.8079451036E-04	-17.77777778	2.6538795420E-04	43.33333333	1.3393410773E-04
-73.33333333	5.3325616965E-04	-15	2.5712041668E-04	46.11111111	1.2938696209E-04
-70.55555556	5.1176057211E-04	-12.22222222	2.4926625605E-04	48.88888889	1.2483981646E-04
-67.77777778	4.9150510520E-04	-9.444444444	2.4182547228E-04	51.66666667	1.2070604770E-04
-65	4.7290314580E-04	-6.666666667	2.3479806540E-04	54.44444444	1.1615890207E-04
-62.22222222	4.5512794014E-04	-3.888888889	2.2777065851E-04	57.22222222	1.1202513331E-04
-59.44444444	4.3817948824E-04	-1.111111111	2.2074325162E-04	60	1.0747798768E-04
-56.66666667	4.2205779009E-04	1.666666667	2.1412922161E-04	62.77777778	1.0334421892E-04
-53.88888889	4.0717622256E-04	4.444444444	2.0792856848E-04	65.55555556	9.9210450167E-05
-51.11111111	3.9312140879E-04	7.222222222	2.0172791534E-04	68.33333333	9.5076681410E-05
-48.33333333	3.7947997189E-04	10	1.9552726220E-04	71.11111111	9.0529535778E-05
-45.55555556	3.6666528874E-04	12.77777778	1.8973998595E-04	73.88888889	8.6395767021E-05
-42.77777778	3.5426398247E-04	15.55555556	1.8395270969E-04	76.66666667	8.1848621388E-05
-40.81111111	3.4599644496E-04	18.33333333	1.7857881030E-04	79.44444444	7.7714852631E-05
-40	3.4268942995E-04	21.11111111	1.7320491092E-04	82.22222222	7.2754330123E-05
-37.22222222	3.3152825431E-04	23.88888889	1.6783101153E-04	85	6.8207184490E-05
-34.44444444	3.2078045554E-04	26.66666667	1.6287048902E-04	87.77777778	6.2833285106E-05
-31.66666667	3.1044603365E-04	29.44444444	1.5749658964E-04	90.55555556	5.7046008846E-05

		93.33333333	5.0018601959E-05

PROBLEM 3

Saturated vapor pressure for refrigerant R134a is required. These values are given below as table format. Fit a curve to this values that will minimize error.

-99	0.01	-49	2.06	0	32.73	50	212.61	100	786.93	150	2100.93
-98	0.01	-48	2.21	1	34.24	51	219.24	101	804.64	151	2137.75
-97	0.01	-47	2.37	2	35.81	52	226.02	102	822.65	152	2175.06
-96	0.01	-46	2.54	3	37.43	53	232.96	103	840.96	153	2212.87
-95	0.01	-45	2.72	4	39.12	54	240.07	104	859.56	154	2251.19
-94	0.02	-44	2.91	5	40.86	55	247.33	105	878.47	155	2290.01
-93	0.02	-43	3.11	6	42.67	56	254.77	106	897.69	156	2329.35
-92	0.02	-42	3.32	7	44.54	57	262.38	107	917.22	157	2369.21
-91	0.03	-41	3.54	8	46.48	58	270.15	108	937.07	158	2409.61
-90	0.03	-40	3.78	9	48.49	59	278.1	109	957.23	159	2450.54
-89	0.04	-39	4.03	10	50.57	60	286.23	110	977.71	160	2492.01
-88	0.04	-38	4.29	11	52.72	61	294.54	111	998.52	161	2534.04
-87	0.05	-37	4.57	12	54.94	62	303.03	112	1019.66	162	2576.63
-86	0.06	-36	4.86	13	57.23	63	311.7	113	1041.12	163	2619.8
-85	0.07	-35	5.17	14	59.61	64	320.57	114	1062.92	164	2663.54
-84	0.08	-34	5.5	15	62.06	65	329.62	115	1085.05	165	2707.88
-83	0.09	-33	5.84	16	64.59	66	338.86	116	1107.52	166	2752.81
-82	0.1	-32	6.2	17	67.2	67	348.3	117	1130.35	167	2798.36
-81	0.11	-31	6.58	18	69.9	68	357.93	118	1153.52	168	2844.54
-80	0.13	-30	6.98	19	72.68	69	367.77	119	1177.04	169	2891.36
-79	0.14	-29	7.4	20	75.55	70	377.81	120	1200.91	170	2938.83
-78	0.16	-28	7.84	21	78.52	71	388.05	121	1225.15	171	2986.97
-77	0.18	-27	8.3	22	81.57	72	398.5	122	1249.75	172	3035.81
-76	0.2	-26	8.78	23	84.71	73	409.17	123	1274.71	173	3085.29
-75	0.23	-25	9.29	24	87.95	74	420.04	124	1300.04	174	3135.59
-74	0.25	-24	9.82	25	91.29	75	431.13	125	1325.74	175	3186.64
-73	0.28	-23	10.38	26	94.73	76	442.45	126	1351.82	176	3238.47
-72	0.31	-22	10.96	27	98.27	77	453.98	127	1378.28	177	3291.12
-71	0.34	-21	11.58	28	101.92	78	465.74	128	1405.13	178	3344.62
-70	0.37	-20	12.22	29	105.66	79	477.72	129	1432.36	179	3399.02
-69	0.41	-19	12.88	30	109.52	80	489.93	130	1459.98	180	3454.38
-68	0.45	-18	13.58	31	113.49	81	502.38	131	1488	181	3510.78
-67	0.49	-17	14.32	32	117.57	82	515.06	132	1516.41	182	3568.29
-66	0.54	-16	15.08	33	121.76	83	527.98	133	1545.23	183	3627.05
-65	0.59	-15	15.88	34	126.07	84	541.15	134	1574.46		
-64	0.64	-14	16.71	35	130.49	85	554.55	135	1604.1		
-63	0.7	-13	17.58	36	135.04	86	568.21	136	1634.15		
-62	0.76	-12	18.48	37	139.71	87	582.11	137	1664.62		
-61	0.83	-11	19.43	38	144.51	88	596.27	138	1695.52		
-60	0.9	-10	20.41	39	149.43	89	610.68	139	1726.84		
-59	0.97	-9	21.44	40	154.48	90	625.35	140	1758.6		
-58	1.05	-8	22.5	41	159.66	91	640.28	141	1790.79		
-57	1.14	-7	23.61	42	164.98	92	655.47	142	1823.42		
-56	1.23	-6	24.77	43	170.43	93	670.94	143	1856.5		
-55	1.33	-5	25.97	44	176.03	94	686.67	144	1890.03		
-54	1.43	-4	27.22	45	181.76	95	702.68	145	1924.01		
-53	1.54	-3	28.52	46	187.63	96	718.96	146	1958.46		
-52	1.66	-2	29.87	47	193.65	97	735.53	147	1993.37		
-51	1.79	-1	31.27	48	199.82	98	752.38	148	2028.74		
-50	1.92			49	206.14	99	769.51	149	2064.6		

PROBLEM 4

Fit a parametric curve to the values given in the table P4

Table P.4 Girdi verisi

İ	0	1	2	3	4
T	0	1	2	3	4
X	-1	0	1	0	1
Y	0	1	0.5	0	-1

PROBLEM 5

X	y
0	-2.20
0.1	-1.90
0.2	0
0.3	-0.1
0.4	-0.01
0.5	-0.3
0.6	-0.2
0.7	-0.09
0.8	0.4
0.9	-0.1
1.0	-1.1
1.1	-1.3
1.2	-2
1.3	-1.8
1.4	-0.02
1.5	1

Data in the above table is given

a) Fit a polynomial least square curve
b) Use interpolation polynomials to fit the data

PROBLEM 6

X	y
0	0
1	1
2	4
3	8

Fit cubic spline to the data.

PROBLEM 7

X	y
0	0
1	1

2	0
1	-1

Fit cubic spline to the data.

PROBLEM 8

Ethylene gylicol-water mixture density values are given in the table below for different mixture percentages. By using formula

$\rho(x,t)= a_0+a_1*x+a_2*t+a_3*x^2+a_4*t^2+a_5*xt+a_6*x^3+a_7*t^3+a_8*x^2t$
$+a_9*xt^2+a_{10}*x^4+a_{11}*t^4+a_{12}*x^2t^2+a_{13}x^3t+a_{14}*xt^3$

	x %volume ratio ethylene gylicol concentration									
t	0%	10%	20%	30%	40%	50%	60%	70%	80%	90%
-35							1090	1105	1119	1132
-30							1089	1104	1117	1131
-25						1088	1102	1116	1129	1142
-20					1072	1087	1101	1115	1128	1140
-15					1071	1086	1100	1113	1126	1138
-10				1054	1070	1084	1098	1111	1124	1136
-5			1037	1053	1068	1083	1096	1109	1122	1134
0	999.8	1019	1036	1052	1067	1081	1095	1108	1120	1132
5	999.9	1018	1034	1050	1065	1079	1093	1105	1118	1129
10	999.7	1016	1033	1049	1063	1077	1091	1103	1115	1127
15	999.1	1015	1031	1047	1062	1075	1089	1101	1113	1124
20	998.2	1013	1030	1045	1060	1073	1086	1098	1110	1121
25	997	1012	1028	1043	1058	1071	1084	1096	1107	1118
30	995.6	1010	1026	1041	1055	1069	1081	1093	1105	1115
35	994	1008	1024	1039	1053	1066	1079	1090	1102	1112
40	992.2	1006	1022	1037	1051	1064	1076	1088	1098	1109
45	990.2	1004	1020	1034	1048	1061	1073	1084	1095	1106
50	988	1002	1017	1032	1045	1058	1070	1081	1092	1102
55	985.7	999.2	1015	1029	1043	1055	1067	1078	1088	1098
60	983.2	996.7	1012	1026	1040	1052	1064	1075	1085	1095
65	980.5	994.1	1009	1023	1037	1049	1060	1071	1081	1091
70	977.7	991.4	1006	1020	1033	1045	1057	1067	1077	1087
75	974.8	988.5	1003	1017	1030	1042	1053	1064	1073	1083
80	971.8	985.5	1000	1014	1027	1038	1049	1060	1069	1078
85	968.6	982.4	997	1011	1023	1035	1046	1056	1065	1074
90	965.3	979.2	993.6	1007	1019	1031	1042	1052	1061	1070
95	961.9	975.8	990.1	1003	1016	1027	1038	1047	1056	1065
100	958.3	972.3	986.5	999.7	1012	1023	1033	1043	1052	1060
105	954.7	968.7	982.8	995.8	1008	1019	1029	1038	1047	1055
110	950.9	965	978.9	991.8	1004	1014	1025	1034	1042	1050
115	947.1	961.2	974.9	987.7	999.3	1010	1020	1029	1038	1045
120	943.1	957.2	970.8	983.4	994.9	1005	1015	1024	1032	1040
125	939	953.1	966.6	979.1	990.4	1001	1010	1019	1027	1035

PROBLEM 9

Liquid flows through a sphere. Friction drag coefficient C_D is given as a function of Reynolds number. Fit a cubic spline interpolation polynomial to the data. Find C_D values for *Re* = 5, 50, 500 and 5000 . Note: Logarithmic scale can be used.

Re	0.2	2	20	200	2000	20 000
C_D	103	13.9	2.72	0.800	0.401	0.433

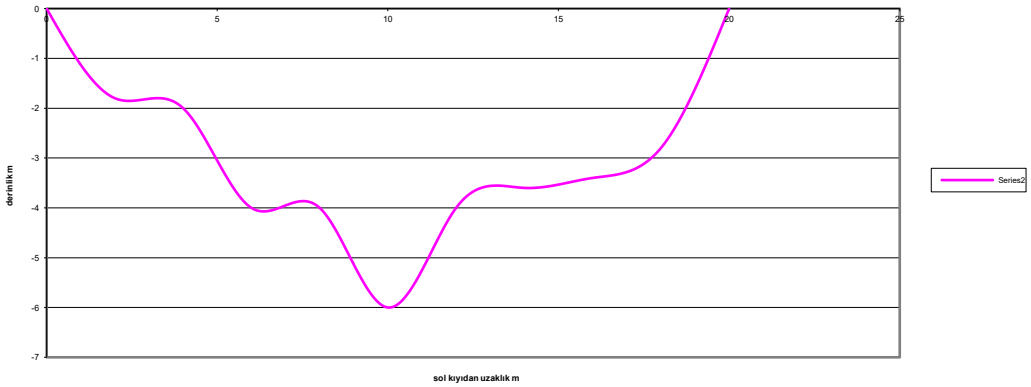
PROBLEM 10

Solve problem 9 by using 4 point Lagrange and Newton interpolation formula

PROBLEM 11

The depth and profile of a channel is given below

x, m	0	2	4	6	8	10	12	14	16	18	20	
H, m	0	1.8	2	4	4	6	4	3.6	3.4	2.8	0	
U m/s	0	0.03	0.045	0.055	0.065	0.12	0.07	0.06	0.05	0.04	0	



Find H(x) and U(x) curve fitting functions by using

a) Cubic spline interpolation formula
b) Polynomial least square
c) Lagrange interpolation formula

PROBLEM 12

Temperature distribution of a solid substance is given as

Time , t, s	**0**	**5**	**10**	**15**	**20**	**25**	**30**	**35**
Temperature T C	**90**	**55**	**40**	**28**	**24**	**22**	**21.5**	**20.6**

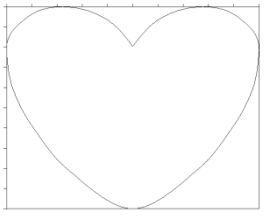
Find the temperature distribution at t=13 sn and t=23 sn t=33 sn, t=43sn ve t=53 sn deki sicaklik dağilimini bulunuz.

PROBLEM 13

A hard shape coordinates are given below, curvefit data to these coordinates

Xi	Yi	Xi	yi	Xi	yi	xi	Yi
241	127	331	157	215	246	149	120
258	107	323	178	199	232	160	106
283	96	312	197	184	217	175	97
307	97	298	217	170	197	199	96
322	106	283	232	159	178	224	107
333	120	267	246	151	157	241	127
334	138	241	258	148	138		

Note: the same values are repeated.



PROBLEM 14

Distance required to stop a car is function of velocity. This relation is given in the table below

Velocity (km/h)	**24**	**32**	**40**	**48**
Stopping distance (m)	**4.8**	**6.0**	**10**	**12**

If the vehicle travels with a velocity of 38 km/h, calculate stopping distance by using Newton interpolation formula.

PROBLEM 15

Thermal conductivity coefficient as a function of temperature is given in the table below.

Sicaklik ***T*** . [K]	275	290	305	320	340	360
Thermal conductivity coefficient ***k***. [W/(m·K)]	0.574	0.598	0.620	0.640	0.660	0.674

a-) Curve fit a quadratic polynomial to data

$$k = A \cdot T^2 + BT + C$$

b-) Estimate the thermal conductivity coefficient at 300 K

PROBLEM 16

For Etylenegylicol Prandtl (Pr) is given as a function of temperature in the below table. Derive a polynomial least square curve fitting and calculate Pr number for T= 42 °C

Temperature (K)	280	300	320	340
Pr number	400	151	73.5	42.8

PROBLEM 17

Saturated water temperature of water as a function of temperature t ^oC will be curvefit to the following equation

$$h_f = A + Bt + Ct^2 + Dt^3$$

Following table is taken from the thermodynamic property table

t, ^oC	10	30	50	70	90
h_f , kJ/kg	41.99	125.66	209.26	292.97	376.94

Calculate the coefficients by using the least square curve fitting

PROBLEM 18

According to Hooke's law Force applied to a linear spring is F=kx. In this equation k is spring constant, x is the displacement in the spring. Displacement versus force in a real spring is given by the below table find the k spring constant

X, m	0.02	0.04	0.06	0.08	0.1
F, kN	3.1	6.1	9.2	12	15.1

PROBLEM 19

Fit below values into function $f(x) = (Ax+B)^{-2}$

X	-1	0	1	2
Y	13.46	3.02	0.68	0.14

PROBLEM 20

2.7 x 10^7 W/m^3 heat is generated in 5 mm thick plate by a heat souce. Thermal conductivity of the plate is k=25 W/m K. Surfce temperatures of the plate (at x=0 mm) 161.5 °C, and (x=5 mm) 166.9 °C. Temperature distribution insde the plate is given in the table below. Give o polinomial function for the temperature distribution inside the plate. Use Newton interpolation Formula. Calculate temperature at x=4 mm.

x (mm)	0	0.5	2.5	4.5	5.0
T (°C)	161.5	166.9	177.7	171.22	166.9

PROBLEM 21

For very low pressure values (P<10 kPa) establish a Formula for the enthalpy for dry steam as h=A+BT

where A and B constants , h ,(kJ/kg) enthalpy and T , (°C) is temperature.

h (kJ/kg)	2510.6	2519.8	2528.9	2538.1	2547.2	2556.3	2565.3
T (°C)	5	10	15	20	25	30	35

PROBLEM 22

In a direct current electrical motor, Moment(F) and electrical current(I) is related with the relation

$I=A\,(F^B)$

Experimental measurements of F and I is given in the table below. Calculate A and B coefficients.

F (Nm)	100	200	300	400
I (Amper)	8	11.31	13.85	16

Bu ilişkiyi en küçük kareler yöntemi ile belirleyiniz.

PROBLEM 23

Derive and equation given the density of dry air ρ (kg/ m^3) at one athmosphere pressure by using Lagrange interpolation formula and then calculate the density at 22.5 °C.

t (°C)	0	20	40	60
ρ (kg/ m^3)	1,2930	1,2045	1,1267	1,0595

PROBLEM 24

For variable z the required function is given as

z=ax+by+cxy

By using least square method find a,b,c values and the function.

z	**x**	**y**
0.1	1	1
-0.9	1	2
2.0	2	2
-1.8	3	1

PROBLEM 25

Saturation pressure of water vapor(steam) changes as a function of temperature. One of the relation used for this rlation is known as "*Dupperet Formula"* is

$$P_{saturation} = A\left(\frac{t_{saturation}}{100}\right)^4$$

In this equation $t_{saturation}$:°C is saturation temperature, $P_{saturation}$, kPa is saturation pressure. Find coefficient A.

t, °C	10	30	60	90	110	140	180	200	250	300	340	370
P kPa	1.2276	4.246	19.940	70.14	143.27	361.3	1002.1	1553.8	3973	8581	14586	21030

PROBLEM 26

The vibration(x) of a machine as a function of time(t) is measured and results are given as the table below.

i	**1**	**2**	**3**	**4**
t_i	0	2	4	6
x_i	5,0	3,7	2,7	2,0

For the vibration of this machine find an equation is in the form of

$$x(t) = ae^{-bt}$$

Where a and b are constants.

PROBLEM 27

In order to develop a new machine, shear stress (τ) is required to be measured. In the experiment the plate above is moved with a velocity of 2 m/s while the plate below remained constant. The distance between the plates are (h) 0.025 m . The shear stres is related to viscosity as follows:

$$\tau = \mu \frac{V}{h}$$

Viskosity (μ) N-s/m^2, is function of temperature and viscosity temperature relation is obtained in the laboratory as :

T (°C)	5	20	30	50	55
μ (N-s/m^2)	0,08	0,015	0,009	0,006	0,005

a) By using Lagrange interpolation find the shear stres at 38 °

b) By using a polynomial least square, obtain an eqaution for sheat stress

PROBLEM 28

A pump manufacturer is measured for the new pump the following data between volumetric flow rate(Q) and pump pressure (P)

Q (lt/s)	0,5	1,0	1,5	2,0	2,5	3,0	3,5
P (kPa)	275	290	280	255	229	205	178

b) Fit a second degree least square polynomial [$P=f(Q)=aQ^2+bQ+c$] b-) Pompa 2,2 lt/s debide su basarken oluşacak basinci tahmin ediniz.

b) Find the flow rate fort he maximum pressure?

PROBLEM 29

Yanma gazlari yoğunluğunun (ρ: kg/m^3), sicaklikla (T:°C) değişimi aşağida verilmiştir.

ρ: kg/m^3	0,950	0,748	0,617	0,525
T:°C	100	200	300	400

Newton enterpolasyon yöntemini kullanarak 1'den 4'ü dereceye kadar polinom Denklemlerini bulunuz. 285 °C'da yanma gazlarinin yoğunluğunu hesaplayiniz.

PROBLEM 30

An orifice is used to measure volumetric flow rate. Such a device is shown in the figure. Following data is used to design the orifice-meter.

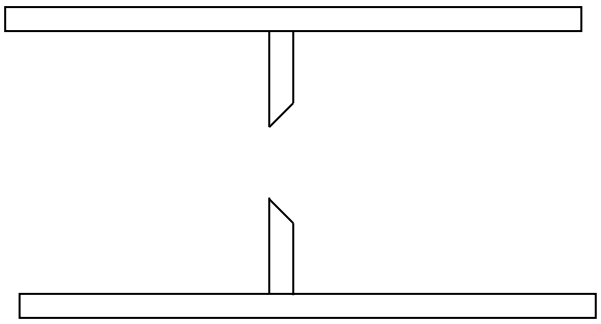
A : Orifice cross sectional area.

A_1 : pipe cross sectional area<.

A_2 : C_cA (shrinked area behind the orifice).

C_c is shrinkage coefficient is a function of (A/A_1) ratio. Experimental results fort his coefficient is given below: Find a least sqaure cirve fiitng equation for [$C_c=f(A/A_1)$]

A/A_1	**C_c**
0.1	0.62
0.2	0.63
0.3	0.64
0.4	0.66
0.5	0.68
0.6	0.71
0.7	0.76
0.8	0.81
0.9	0.89
1.00	1.00



PROBLEM 31

The life of the cutting tool for a lathe is T(minutes) and cutting speed V(m/minutes). The life test is carried out and the following data is obtained.

Cutting speed, V (m/minutes)	Life of the cutting tool,T (minutes)
140	80
160	70
180	55
200	50
220	35

If the general relation between cutting speed and life of the cutting tool is:

$$\mathbf{V=(b/T)^{(1/a)}}$$

Find the coefficients by using least square curve fitting

PROBLEM 32

Fit the data below the following equation by using least square curve fitting

$$y=ax^b$$

by using least square curve fitting

X	1	2	3	4	5
Y	1	1	2	4	4

PROBLEM 33

By using data in the following table answer the quaestions below.

X	1	2	3	5	6
f(X)	4,75	4	5,25	19,75	36

a) Obtain fourth order polynomial equation by using Newton interpolation formula
b) Obtain cubic spline for the second subregion
c) PROBLEM 34

Fit a line to following data by using least square method.

x	5	10	15	20	25	30	35	40	45	50
y	16	25	32	33	38	36	3	40	42	42

PROBLEM 35

Kinematic viscosities of oil SAE30 is given in the table below.

Sicaklik T(C)	5	10	15	20	25	30	35	40	45	50
Vizkozite μ (m^2/s)	16	25	32	33	38	36	3	40	42	42

a) Fit the data into $\mu=ae^{(b/T)}$ equation

b) Fit the data into $\mu = \frac{a\sqrt{T}}{1+\frac{b}{T}}$ equation

PROBLEM 36

For $f(x) = \frac{20}{1+2x^2}$ function in the range of -3<=x<=3 by using points x_i=-3,-2,-1,0,1,2,3

a) Find cubic spline interpolation formulas
c) Find quadratic B-spline formulas

PROBLEM 37

In a smooth sphere, the following relation is existed between Reynolds number(Re) and friction coefficient (C)

Re	0.1	1	10	100	1000	10000
C	210	30	4	1	0.5	0.4

Find an equation to represent the data properly

PROBLEM 38

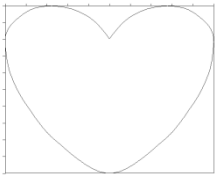
By using bezier curve construct the letter α.

PROBLEM 39

By using bezier curve construct the letter γ .

PROBLEM 40

By using two symmetric bezier curve construct the hard shape.



PROBLEM 41

The tuition for 12 units at *St. Anford* University has been increasing from

1998 to 2008 as shown in the table below:

Year Tuition per year

1998 $21,300
1999 $23,057
2000 $24,441
2001 $25,917
2002 $27,204
2003 $28,564
2004 $29,847
2005 $31,200
2006 $32,994
2007 $34,800
2008 $36,030

(a) Plot the given data points and intuitively interpolate (draw) a smooth curve through them.
(b) Interpolate the data with the Lagrange polynomial. Plot the polynomial and the data points. Use the polynomial to predict the tuition in 2010. This is an extrapolation problem; discuss the utility of Lagrange polynomials for extrapolation.
(c) Repeat with a cubic spline interpolation and compare your results.

PROBLEM 42

The concentration of a certain toxin in a system of lakes downwind of an industrial area has been monitored very accurately at intervals from 1993 to 2007 as shown in the table below. It is believed that the concentration has varied smoothly between these data points.

Year Toxin Concentration

1993 12.0
1995 12.7
1997 13.0
1999 15.2
2001 18.2
2003 19.8
2005 24.1
2007 28.1
2009 ???

(a) Interpolate the data with the Lagrange polynomial. Plot the polynomial and the data points. Use the polynomial to predict the condition of the lakes in 2009. Discuss this prediction.
(b) Interpolation may also be used to fill "holes" in the data. Say the data from 1997 and 1999 disappeared. Predict these values using the Lagrange polynomial fitted through the other known data points.
(c) Repeat (b) with a cubic spline interpolation. Compare and discuss your results.

PROBLEM 43

Fit Pade rational approximation to the function sin(x) between [-π,π] with polynomial

$$r(x) = \frac{p(x)}{q(x)} = \frac{p_0 + p_1 x + p_2 x^2 + p_3 x^3}{q_0 + q_1 x + q_2 x^2}$$

$p(x)$ is 3th degree polynomial and q(x) is 2nd degree polynomial.

PROBLEM 44

Fit Pade rational approximation to the function |x| between [-1,1] with polynomial

$$r(x) = \frac{p(x)}{q(x)} = \frac{p_0 + p_1 x + p_2 x^2 + p_3 x^3}{q_0 + q_1 x + q_2 x^2}$$

p(x) is 3th degree polynomial and q(x) is 3rd degree polynomial.

PROBLEM 45

Fit Chebyschev rational approximation to the function sin(x) between [-π,π] with polynomial

$$r(x) = \frac{p(x)}{q(x)} = \frac{p_0T_0(x) + p_1T_1(x) + p_2T_2(x) + p_3T_3(x)}{q_0T_0(x) + q_1T_1(x) + q_2T_2(x)}$$

p(x) is 3th degree polynomial and q(x) is 2nd degree polynomial.

PROBLEM 46

Fit Chebyschev rational approximation to the function sin(x) between [-1,1] with polynomial

$$r(x) = \frac{p(x)}{q(x)} = \frac{p_0T_0(x) + p_1T_1(x) + p_2T_2(x) + p_3T_3(x)}{q_0T_0(x) + q_1T_1(x) + q_2T_2(x) + q_3T_3(x)}$$

p(x) is 3th degree polynomial and q(x) is 3th degree polynomial.

PROBLEM 47

Fit Trigonometric(Fourier) approximation to the function $f(x) = x^2 - 2x + 5$between [-π,π] with degree 5 function $S_5(x)$

PROBLEM 48

Fit Trigonometric(Fourier) approximation to the function $f(x) = |x|$ between [-1,1] with degree 10 function $S_{10}(x)$

PROBLEM 49

Average temperature in a city is measured as follows

Months x_i	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**
Average Temperature T_i	**-6**	**-3**	**2**	**9**	**15**	**21**	**23**	**22**	**18**	**12**	**5**	**-2**

Fit a discrete Trigonometric (Fourier) least square curve to this data.

PROBLEM 50

For CO2 Specific heat , Cp (KJ/kmolK) data is given as a function of temperature (degree K) as

T (K)	**Cp(KJ/kmolK)**
298.15	37.132
300	37.217
400	41.326
500	44.625
600	47.323
700	49.563
800	51.434
900	52.999
1000	54.308
1100	55.412
1200	56.342
1300	57.13
1400	57.803
1500	58.381
1600	58.883
1700	59.321
1800	59.705
1900	60.046
2000	60.349
2100	60.62
2200	60.864
2300	61.086
2400	61.288
2500	61.473
2600	61.644
2700	61.804

2800	61.953
2900	62.094
3000	62.228

Find an equation for Cp. Investigate different kind of curve fitting equations for a best result.

PROBLEM 51

Saturation vapor pressure equation for HCFC-123 Refrigerant(2,2 dichloro-1,1,1-trifloroethane) is given as: $log_{10} P_{sat} = A + B/T + C\, log_{10} T + DT + E((F - T)/T)\, log_{10}(F - T)$

Curve fit values are given in the book as:

A=1.656333e03 B=-2.480583e06,C=1.792522e1,D=-8.86838e-2,E=4.61781e2,F=1.666667e3

The data in the tables are given as T (degree K) and P_{sat}(kPa):

-80	0.13	18	69.95	50	212.46	84	540.19
-70	0.34	20	75.61	52	225.84	86	567.2
-60	0.81	22	81.63	54	239.85	88	595.2
-50	1.77	24	88.02	56	254.51	90	624.23
-40	3.58	26	94.8	58	269.85	92	654.3
-30	6.75	27.82	101.33	60	285.89	94	685.44
-20	12	28	101.98	62	302.64	96	717.68
-10	20.25	30	109.58	64	320.13	98	751.03
0	32.65	32	117.62	66	338.38	100	785.53
2	35.74	34	126.11	68	357.4	110	976.03
4	39.07	36	135.07	70	377.22	120	1199
6	42.64	38	144.52	72	397.87	130	1457.8
8	46.47	40	154.47	74	419.36	140	1756.3
10	50.57	42	164.95	76	441.71	150	2098.7
12	54.95	44	175.97	78	464.94	160	2490.1
14	59.63	46	187.55	80	489.09	170	2937.2
16	64.63	48	199.71	82	514.16	180	3450.6
						183.68	3661.8

Make a non-linear curve fit for a given data and check if the given coefficients are correct.

7.0 INTEGRATION AND DERIVATIVES

7.1 NEWTON-COTES FORMULATIONS : TRAPEZOIDAL, BOLE, SİMPSON 1/3 VE 3/8 METHODS

If function f(x) is given and ,it is desired to calculate integral of the function between integration limits a and b. Definition of the integral is the area inder the function between integration limits. The simplest approximation is to connect f(a) and f(b) points with a linear line and calculate the area of the trapezoidal under the line. This approach is called trapezoidal rule of Newton-Cotes formulations (Area finding formulations)

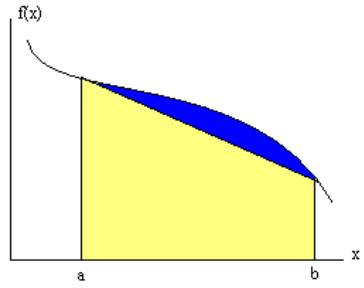


Figure 7.1-1 Finding integration with trapezoidal rule (yellow area) and error created by the method(blue area)

Integration formula :

$$I = \int_a^b f(x)dx = (b-a)\frac{f(b)+f(a)}{2}$$

$$I = \int_0^1 3x^2 dx = \frac{3x^3}{3}\Big|_0^1 = \frac{3}{3} = 1$$

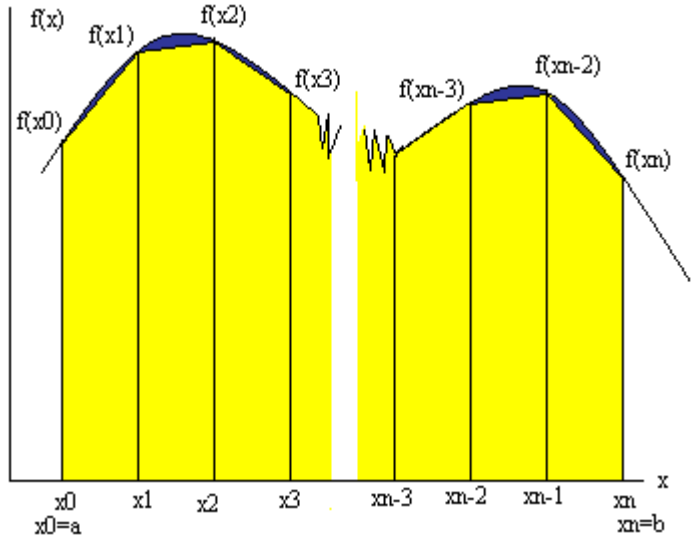


Figure 7.1-1 Finding integration with trapezoidal rule (yellow area) with multiple trapezoidal rule

If total area is divided into smaller areas amount of error will be reduced due to the fact that smaller piece of functions are closer to the line compare to a big piece of function. The same integration Formula for multipart equation will be:

$$I = \int_a^b f(x)dx = \frac{(b-a)}{2}\left[\frac{f(x_0) + 2\sum_{i=1}^{n-1} f(x_i) + f(x_n)}{n}\right]$$

Now assume an example integration function of

$$I = \int_0^1 \frac{4}{\pi}\sqrt{1-x^2}dx = 1.0$$

Since the value of the function is known exactly, it will be easy to evaluate the error of numerical function evaluation process.

Program 7.1-1 Trapezoidal intergral rule of Newton-Cotes approximation n=1

```
//=======================================================
// Numerical Analysis package in java
// Newton-Cotes integral formula n points
// trapezoidal rule
// Dr. Turhan Coban
// =======================================================
import java.io.*;


class f1 extends f_x
{
  double func(double x)
  {return 4.0/Math.PI*Math.sqrt(1.0-x*x);}
}

class NA51
{

public static double newton_cotes2(f_x ff,double a,double b, int n)
{       //Newton-cotes integral 2 points
        //trapezoidal rule
        // ff integrating function
        // a,b integral limits
        // n : number of sub division h=(b-a)/n;
        double h=(b-a)/n;
        double h1=h;
        double sum=0;
        int p,i;
        double x[]=new double[3];
        double f[]=new double[3];
        for(p=0;p<n;p++)
        {  {for(i=0;i<2;i++) {x[i]=a+i*h1+p*h;f[i]=ff.func(x[i]);}}
           sum+=h*(f[0]+f[1])/2.0;
        }
        return sum;
}
```

```
  public static void main(String args[]) throws IOException
  {
  f1 b1=new f1();
  double i1= newton_cotes2(b1,0.0,1.0,1);
  double error=1-i1;
  System.out.println("integral of class         f1 : "+i1+"error = "+error);
}
}
```

```
---------- Capture Output ----------
> "C:\turhan\java\bin\javaw.exe" NA51
integral of class         f1 : 0.6366197723675814error = 0.3633802276324186
> Terminated with exit code 0.
```

In order to see the error happened in the function visiually a graphical output form of the same program is created.

Program 7.1-2 Trapezoidal integral rule of Newton-Cotes approximation with plotting n=1

```
//=====================================================
// Numerical Analysis package in java
// Newton-Cotes integral formula 4 points , n=3
// trapezoidal rule
// Dr. Turhan Coban
// =====================================================
import java.io.*;

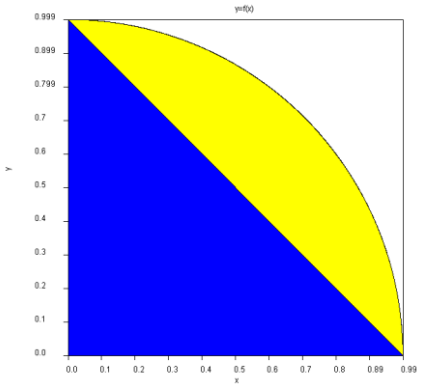

class f1 extends f_x
{
  double func(double x)
  {return 4.0/Math.PI*Math.sqrt(1.0-x*x);}
}


class NA51A
{

public static double newton_cotes2(f_x ff,double a,double b, int n)
{       //Newton-cotes integral 2 points
        //trapezoidal rule
        double h=(b-a)/n;
        double h1=h;
        double sum=0;
        int p,i;
        int nn=2*n;
        double x[]=new double[2];
        double f[]=new double[2];
        double xi[]=new double[nn];
        double fi[]=new double[nn];
        int j=0;
        for(p=0;p<n;p++)
        {  {for(i=0;i<2;i++) {x[i]=a+i*h1+p*h;xi[j]=x[i];f[i]=ff.func(x[i]);fi[j]=f[i];j++;}}
           sum+=h*(f[0]+f[1])/2.0;
        }
        Plot pp=new Plot(xi,fi);
        pp.addFunction(ff,0.0,1.0,2000,2);
        pp.plot();
        return sum;
}
```

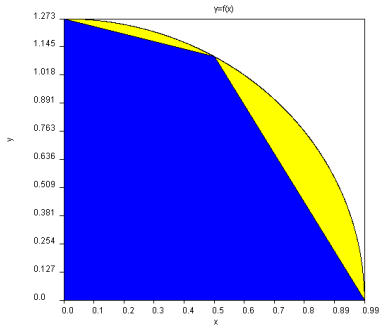
```
  public static void main(String args[]) throws IOException
  {
  f1 b1=new f1();
  double i1= newton_cotes2(b1,0.0,1.0,1);
  double error=1-i1;
  System.out.println("integral of class        f1 : "+i1+"error = "+error);

  }
}
```



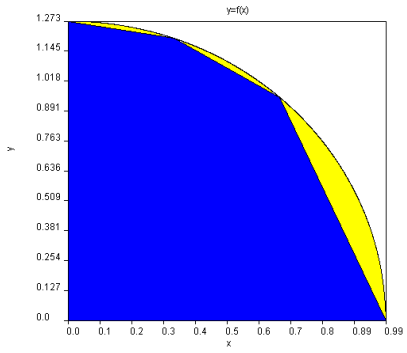
If the integral zone (0 to 1) is divided into two equal regions

```
---------- Capture Output ----------
> "C:\turhan\java\bin\javaw.exe" NA51A
integral of class        f1 : 0.8696387816055828 error = 0.13036121839441717
```



If the integral zone (0 to 1) is divided into three equal regions

```
---------- Capture Output ----------
> "C:\turhan\java\bin\javaw.exe" NA51A
integral of class        f1 : 0.9286860839333196error = 0.07131391606668036
```



If the integral zone (0 to 1) is divided into ten equal regions

```
---------- Capture Output ----------
> "C:\turhan\java\bin\javaw.exe" NA51A
integral of class       f1 : 0.9881988750835945error = 0.011801124916405503
```

If the integral zone (0 to 1) is divided into hundred equal regions

```
---------- Capture Output ----------
> "C:\turhan\java\bin\javaw.exe" NA51A
integral of class       f1 : 0.9996257879552262error = 3.742120447738362E-4
```

As it is seen from the example when trapezoidal steps are increased, approximation is getting better but even with 100 steps there are stil some error in the approximation. The next step in the approach is to use higher degrees of polinomials to connect points of integration. Quadratic (second degree) polinomial rule is called **Simpson 1/3 rule**. If step size of formulation is taken as h

$$h = \frac{b-a}{n}$$

Quadratic polynomial Newton-Cotes formula(Simpson 1/3 rule)

$$\boldsymbol{I = \int_a^b f(x)dx = \frac{b-a}{6}[f(x_0) + 4f(x_1) + f(x_2)]}$$

or

$$h = \frac{b-a}{n} = \frac{b-a}{2}$$

$$\boldsymbol{I = \int_a^b f(x)dx = \frac{h}{3}[f(x_0) + 4f(x_1) + f(x_2)]}$$

(n=2 or three points are given in this Formula)

Quadratic polynomial Newton-Cotes formula(Simpson 1/3 rule) with multipoint approach

$$h = \frac{b-a}{n} = \frac{b-a}{2}$$

$$\boldsymbol{I = \int_a^b f(x)dx = \frac{h}{3}\left[f(x_0) + 4\sum_{i=1,3,5}^{n-1} f(x_i) + \sum_{i=2,4,6}^{n-2} f(x_i) + f(x_n)\right]}$$

Cubic polynomial Newton-Cotes formula(Simpson 3/8 rule)

$$h = \frac{b-a}{n} = \frac{b-a}{3}$$

$$\boldsymbol{I = \int_a^b f(x)dx = \frac{3h}{8}[f(x_0) + 3f(x_1) + 3f(x_2) + f(x_3)]}$$

(n=3 or four points are required for this formula)

Fourth order polynomial Newton-Cotes formula(Bode rule)

$$h = \frac{b-a}{n} = \frac{b-a}{4}$$

$$\boldsymbol{I = \int_a^b f(x)dx = \frac{2h}{45}[7f(x_0) + 32f(x_1) + 12f(x_2) + 32f(x_3) + 7f(x_4)]}$$

(n=4 or five points are used in this Formula)

Fifth order polynomial Newton-Cotes formula(Boole rule)

$$h = \frac{b-a}{n} = \frac{b-a}{5}$$

$$\boldsymbol{I = \int_a^b f(x)dx = \frac{5h}{288}[10f(x_0) + 75f(x_1) + 50f(x_2) + 50f(x_3) + 75f(x_4) + 19f(x_5)]}$$

(n=5 or six points are used in this Formula)

Eight order polynomial Newton-Cotes formula

$$h = \frac{b-a}{n} = \frac{b-a}{8}$$

$$\boldsymbol{I = \int_a^b f(x)dx = \frac{4h}{14175}[989f(x_0) + 5888f(x_1) - 928f(x_2) + 10496f(x_3) - 4540f(x_4)}$$
$$\boldsymbol{+ 10496f(x_3)f(x_5) - 928f(x_6) + 5888f(x_7) + 989f(x_8)]}$$

(n=8 or nine points are used in this Formula)

Programs for this approaches are given below:

Program 7.1-3 quadratic polynomial Newton - Cotes Formula (Simpson 1/3 rule)

```
//=======================================================
//  Numerical Analysis package in java
//  example to show utilisation of integration ( integral )
//  and differentiation (derivative) functions
//  Dr. Turhan Coban
// =======================================================
import java.io.*;

class f1 extends f_x
{
  double func(double x)
  {return 4.0/Math.PI*Math.sqrt(1.0-x*x);}
}

class NA52
{
public static double newton_cotes3(f_x ff,double a,double b, int n)
{      //Newton-cotes integral 3 points
```

```
      //Simpson 1/3 rule

      double h=(b-a)/n;
      double h1=h/2.0;
      double sum=0;
      int p,i;
      double x[]=new double[3];
      double f[]=new double[3];
      for(p=0;p<n;p++)
      {  {for(i=0;i<3;i++) {x[i]=a+i*h1+p*h;f[i]=ff.func(x[i]);}}
        sum+=h*(f[0]+4.0*f[1]+f[2])/6.0;
      }
      return sum;
}

  public static void main(String args[]) throws IOException
  {
   f1 b1=new f1();
  double i1=  newton_cotes3(b1,0.0,1.0,1);
  double error =1-i1;
  System.out.println("integral of class        f1 : "+i1+"error="+error);
}
}
```

If the quadratic Newton-Cotes Formula (Simpson 1/3 rule) applied as one time integral zone (0 to 1)

```
---------- Capture Output ----------
> "C:\turhan\java\bin\javaw.exe" NA52
integral of class        f1 : 0.9473117846849166error=0.052688215315083387
> Terminated with exit code 0.
```

If the quadratic Newton-Cotes Formula (Simpson 1/3 rule) applied as two times integral zone (0 to 1) and divided up into 2

```
---------- Capture Output ----------
> "C:\turhan\java\bin\javaw.exe" NA52
integral of class        f1 : 0.9815388228080559error=0.018461177191944067

> Terminated with exit code 0.
```

Program 7.1-4 cubic polynomial Newton - Cotes Formula (Simpson 3/8 rule)

```
//======================================================
// Numerical Analysis package in java
// Newton-Cotes integral formula 4 points , n=3
// Simpson 3/8 rule
// Dr. Turhan Coban
// ======================================================
import java.io.*;

class f1 extends f_x
{
  double func(double x)
  {
```

```
  return 4.0/Math.PI*Math.sqrt(1.0-x*x);
  }
}

class NA53
{

public static double newton_cotes4(f_x ff,double a,double b, int n)
{       //Newton-cotes integral 4 points
        //Simpson 3/8 rule
        // ff integrating function
        // a,b integral limits
        // n : number of sub division h=(b-a)/n;
                double h=(b-a)/n;
        double h1=h/3.0;
        double sum=0;
        int p,i;
        double x[]=new double[4];
        double f[]=new double[4];
        for(p=0;p<n;p++)
        {  {for(i=0;i<4;i++) {x[i]=a+i*h1+p*h;f[i]=ff.func(x[i]);}}
           sum+=h*(f[0]+3.0*f[1]+3.0*f[2]+f[3])/8.0;
        }
        return sum;
}

public static void main(String args[]) throws IOException
  {
  f1 b1=new f1();
  double i1= newton_cotes4(b1,0.0,1.0,2);
  double error=1-i1;
  System.out.println("integral of class         f1 : "+i1+"error = "+error);
}
}
```

If the cubic Newton-Cotes Formula (Simpson 3/8 rule) applied as one time integral zone (0 to 1)

```
---------- Capture Output ----------
> "C:\turhan\java\bin\javaw.exe" NA53
integral of class         f1 : 0.965194372879037error = 0.03480562712096302


> Terminated with exit code 0.
```

If the cubic Newton-Cotes Formula (Simpson 3/8 rule) applied as two times integral zone (0 to 1) and divided up into 2

```
---------- Capture Output ----------
> "C:\turhan\java\bin\javaw.exe" NA53
integral of class         f1 : 0.9877870530380952error = 0.01221294696190478

> Terminated with exit code 0.
```

Program 7.1-5 fourth order polynomial Newton - Cotes Formula (Boole rule)

```
//=====================================================
// Numerical Analysis package in java
// Newton-Cotes integral formula 5 points , n=4
// Boole formula
// Dr. Turhan Coban
// =====================================================
import java.io.*;


class f1 extends f_x
{
  double func(double x)
  {
  return 4.0/Math.PI*Math.sqrt(1.0-x*x);
  }
}


class NA54
{

public static double boole(f_x ff,double a,double b, int n)
{  return newton_cotes5(ff,a,b,n);}

public static double newton_cotes5(f_x ff,double a,double b, int n)
{       //Newton-cotes integral 5 points
        //Boole rule
        // ff integrating function
        // a,b integral limits
        // n : number of sub division h=(b-a)/n;
                double h=(b-a)/n;
        double h1=h/4.0;
        double sum=0;
        int p,i;
        double x[]=new double[5];
        double f[]=new double[5];
        for(p=0;p<n;p++)
        {  {for(i=0;i<5;i++) {x[i]=a+i*h1+p*h;f[i]=ff.func(x[i]);}}
           sum+=h*(7.0*f[0]+32.0*f[1]+12.0*f[2]+32.0*f[3]+7.0*f[4])/90.0;
        }
        return sum;
}

   public static void main(String args[]) throws IOException
  {
  f1 b1=new f1();
  double i1=  newton_cotes5(b1,0.0,1.0,1);
  double i2 = 1.0;
  double error =i2-i1;
  System.out.println("integral of class         f1 : "+i1+" error="+error);
 }
}
```

If the fourth order polynomial Newton-Cotes Formula (Boolee rule) applied as one time integral zone (0 to 1)

```
---------- Capture Output ----------
> "C:\turhan\java\bin\javaw.exe" NA54
integral of class        f1 : 0.9838206253495985 error=0.016179374650401535


> Terminated with exit code 0.
```

If the fourth order polynomial Newton-Cotes Formula (Boole rule) applied as two times integral zone (0 to 1) and divided up into 2

```
---------- Capture Output ----------
> "C:\turhan\java\bin\javaw.exe" NA54
integral of class        f1 : 0.9943031391613758 error=0.005696860838624196

> Terminated with exit code 0.
```

Program 7.1-6 fifth order polynomial Newton - Cotes Formula

```
//=====================================================
// Numerical Analysis package in java
// Newton-Cotes integral formula 6 points, n=5
//
// Dr. Turhan Coban
// =====================================================
import java.io.*;

class f1 extends f_x
{
  double func(double x)
  {
  return 4.0/Math.PI*Math.sqrt(1.0-x*x);
  }
}

class NA55
{

public static double newton_cotes6(f_x ff,double a,double b, int n)
{       //Newton-cotes integral 6 points, n=5
        // ff integrating function
        // a,b integral limits
        // n : number of sub division h=(b-a)/n;
                double h=(b-a)/n;
        double h1=h/5.0;
        double sum=0;
        int p,i;
        double x[]=new double[6];
        double f[]=new double[6];
        for(p=0;p<n;p++)
        {  {for(i=0;i<6;i++) {x[i]=a+i*h1+p*h;f[i]=ff.func(x[i]);}}
           sum+=h*(19.0*f[0]+75.0*f[1]+50.0*f[2]+50.0*f[3]+75.0*f[4]+19.0*f[5])/288.0;
        }
        return sum;
}

        public static void main(String args[]) throws IOException
```

```
      { f1 b1=new f1();
        double i1=  newton_cotes6(b1,0.0,1.0,1);
        double i2 = 1.0;
        double error =i2-i1;
        System.out.println("integral of class        f1 : "+i1+" error="+error);
      }
}
```

If the fifth order polynomial Newton-Cotes Formula applied as one time integral zone (0 to 1)

```
---------- Capture Output ----------
> "C:\turhan\java\bin\javaw.exe" NA55
integral of class        f1 : 0.9872489399864514 error=0.012751060013548643
> Terminated with exit code 0.
```

If the fifth order polynomial Newton-Cotes Formula applied as two times integral zone (0 to 1) and divided up into 2

```
---------- Capture Output ----------
> "C:\turhan\java\bin\javaw.exe" NA55
integral of class        f1 : 0.9955081956385212 error=0.0044918043614787795

> Terminated with exit code 0.
```

Program 7.1-7 eight order polynomial Newton - Cotes Formula

```
//=====================================================
// Numerical Analysis package in java
// Newton-Cotes integral formula 9 points, n=8
//
// Dr. Turhan Coban
// =====================================================
import java.io.*;


class f1 extends f_x
{
 double func(double x)
 {
 return 4.0/Math.PI*Math.sqrt(1.0-x*x);
 }
}


class NA56
{

public static double newton_cotes9(f_x ff,double a,double b, int n)
{      //Newton-cotes integral 9 points, n=8
       // ff integrating function
       // a,b integral limits
       // n : number of sub division h=(b-a)/n;
                double h=(b-a)/n;
```

```
        double h1=h/8.0;
        double sum=0;
        int p,i;
        double x[]=new double[9];
        double f[]=new double[9];
        for(p=0;p<n;p++)
        {  {for(i=0;i<9;i++) {x[i]=a+i*h1+p*h;f[i]=ff.func(x[i]);}}
           sum+=h/28350.0*(989.0*f[0]+5888.0*f[1]-928.0*f[2]+10496.0*f[3]-4540.0*f[4]+
           10496.0*f[5]-928.0*f[6]+5888.0*f[7]+989.0*f[8]);
        }
        return sum;
}

        public static void main(String args[]) throws IOException
        { f1 b1=new f1();
          double i1=  newton_cotes9(b1,0.0,1.0,1);
          double i2 = 1.0;
          double error =i2-i1;
          System.out.println("integral of class         f1 : "+i1+" error="+error);
        }
}
```

If the eigth order polynomial Newton-Cotes Formula applied as one time integral zone (0 to 1)

```
---------- Capture Output ----------
> "C:\turhan\java\bin\javaw.exe" NA56
integral of class         f1 : 0.9952701383291108 error=0.004729861670889179

> Terminated with exit code 0.
```

If the eigth order polynomial Newton-Cotes Formula applied as two times integral zone (0 to 1) and divided up into 2

```
---------- Capture Output ----------
> "C:\turhan\java\bin\javaw.exe" NA56
integral of class         f1 : 0.998330566008359 error=0.0016694339916409895

> Terminated with exit code 0.
```

The results will be summarised as one table

Table 7.1-1 comparison of Newton-Cotes results for different degree of integrating polnomials

$$I = \int_0^1 \frac{4}{\pi}\sqrt{1-x^2}dx = 1.0$$

N	Integral approximation	Error ($I_{approximation}$-I_{exact})
Region is not divided up calculated as one piece		
1 (Trapezoidal)	0.6366197723675814	0.3633802276324186
2 (Simpson 1/3)	0.9473117846849166	0.052688215315083387
3 (Simpson 3/8)	0.965194372879037	0.03480562712096302
4 (Bode)	0.9838206253495985	0.016179374650401535

5	0.9872489399864514	0.012751060013548643
8	0.9952701383291108	0.004729861670889179
Region is divided into 2 equal pieces (0-0.5 and 0.5-1)		
1 (Trapezoidal)	0.8696387816055828	0.13036121839441717
2 (Simpson 1/3)	0.9815388228080559	0.018461177191944067
3 (Simpson 3/8)	0.9877870530380952	0.01221294696190478
4 (Bode)	0.9943031391613758	0.005696860838624196
5	0.9955081956385212	0.0044918043614787795
8	0.998330566008359	0.0044918043614787795
Region is divided into 10 equal pieces (0-0.1,0.1-0.2,…,0.9-1)		
1 (Trapezoidal)	0.9881988750835945	0.011801124916405503
2 (Simpson 1/3)	0.9983621080451945	0.0016378919548054593
3 (Simpson 3/8)	0.998914915292993	0.001085084707006967
4 (Bode)	0.9994921635232705	5.07836476729473E-4
5	0.9995994322513025	4.005677486974557E-4
8	0.9998508846967017	1.4911530329830835E-4

The same integration calculated by using excel

$$I = \int_0^1 \frac{4}{\pi}\sqrt{1-x^2}dx = 1.0$$

Direct integration Newton-Cotes formulas

x	f(x)	Trapezoidal	Simpson 1/3
0	1.27324	0.158531	0.314961
0.125	1.263253	0.156004	
0.25	1.232809	0.150821	0.294032
0.375	1.180325	0.142686	
0.5	1.102658	0.131036	0.246688
0.625	0.993922	0.114756	
0.75	0.842169	0.091161	0.137824
0.875	0.616404	0.038525	
1	0		
Integral		0.98352	0.993505

Direct integration Newton-Cotes formulas

x	f(x)	Trapezoidal	Simpson 1/3
0	1.27324	0.0795	0.158739
0.0625	1.27075	0.079188	
0.125	1.263253	0.07856	0.156223
0.1875	1.250658	0.077608	
0.25	1.232809	0.076321	0.151063
0.3125	1.209473	0.074681	
0.375	1.180325	0.072664	0.142972
0.4375	1.14492	0.070237	
0.5	1.102658	0.067355	0.131405
0.5625	1.052711	0.063957	
0.625	0.993922	0.059954	0.115302
0.6875	0.924607	0.055212	
0.75	0.842169	0.049513	0.092241
0.8125	0.742249	0.042458	
0.875	0.616404	0.033109	0.049764
0.9375	0.443069	0.013846	
1	0		
Integral		0.994162	0.99771

Direct integration Newton-Cotes formulas

x	f(x)	Trapezoidal	Simpson 1/3
0	1.27324	0.039779	0.079526
0.03125	1.272618	0.03974	
0.0625	1.27075	0.039662	0.079214
0.09375	1.267632	0.039545	
0.125	1.263253	0.039388	0.078587
0.15625	1.257601	0.039192	
0.1875	1.250658	0.038954	0.077636
0.21875	1.242403	0.038675	
0.25	1.232809	0.038354	0.076351
0.28125	1.221845	0.037989	
0.3125	1.209473	0.03758	0.074712
0.34375	1.19565	0.037125	
0.375	1.180325	0.036621	0.072698
0.40625	1.163438	0.036068	
0.4375	1.14492	0.035463	0.070274
0.46875	1.124692	0.034802	
0.5	1.102658	0.034084	0.067398
0.53125	1.078708	0.033303	
0.5625	1.052711	0.032457	0.064007
0.59375	1.024512	0.031538	
0.625	0.993922	0.030541	0.060014
0.65625	0.960715	0.029458	
0.6875	0.924607	0.028279	0.055289
0.71875	0.885243	0.026991	
0.75	0.842169	0.025577	0.04962
0.78125	0.794779	0.024016	
0.8125	0.742249	0.022276	0.042627
0.84375	0.683394	0.020309	
0.875	0.616404	0.018042	0.033463
0.90625	0.538252	0.015333	
0.9375	0.443069	0.011858	0.017774
0.96875	0.315813	0.004935	
1	0		0
Integral		0.997934	0.999191

EXAMPLE :

$$I = \int_0^1 \sin\left(1 - x + x^2\right) dx = 1.0$$

solve with bole rule by hand. Use n=2

Note: Bole rule for n=1

$$\boldsymbol{I = \int_a^b f(x)dx = \frac{(b-a)}{90}\left[7f(x_0) + 32f(x_1) + 12f(x_2) + 32f(x_3) + 7f(x_4)\right]}$$

Step size h=(b-a)/n

x0	0	0.841471	7	5.890297
x1	0.125	0.777465	32	24.87888
x2	0.25	0.726009	12	8.712104
x3	0.375	0.692988	32	22.17561
x4	0.5	0.681639	7	4.771471
				66.42836 0.005556 0.369046
x4	0.5	0.681639	7	4.771471

x5	0.625	0.692988	32	22.17561
x6	0.75	0.726009	12	8.712104
x7	0.875	0.777465	32	24.87888
x8	1	0.841471	7	5.890297
				66.42836

0.005556 0.369046

I= 0.738093

7.2 RİCHARDSON AND AITKEN INTERPOLATION FORMULAS

7.2.1 RİCHARDSON EXTRAPOLATION (ROMBERG INTEGRATION) FORMULA

Richardson extrapolation formula can be applied to decrease integration error. Richardson extrapolation Formula has the folloving form :

$$I \cong I(h_2) + \frac{1}{\left(\frac{h_1}{h_2}\right)^2 - 1}[I(h_2) - I(h_1)]$$

In this Formula h_2 should be smaller than h_1 . If$h_2=h_1/2$ equation becomes:

$$I \cong \frac{4}{3}I(h_2) - \frac{1}{3}I(h_1)$$

As a starting interpolation any of the Newton-Cotes formulations can be taken. A General form of the above equation can be written as

$$I_{j,k} \cong \frac{4I_{j+1,k-1} - I_{j,k-1}}{4^{k-1} - 1}$$

If trapezoidal rule is used as starting point of this iterative formulation, it is called **Romberg Integration**

Consider the previous integral that solved by excel, if the results are integrated into the romberg integration formula:

$I = \int_0^1 \frac{4}{\pi}\sqrt{1 - x^2}dx = 1.0$

Romberg: Trapezoidal

2*h	I0	I1	I2
0.125	0.98352	0.99771	0.999685
0.0625	0.994162	0.999191	
0.03125	0.997934		

Romberg: Simpson 1/3

2*h	I0	I1	I2
0.125	0.993505	0.999111	0.999877
0.0625	0.99771	0.999685	
0.03125	0.999191		

Program 7.2-1 Romberg Integration formula

```
//=======================================================
// Numerical Analysis package in java
// example to show utilisation of integration (integral)
// and differentiation (derivative) functions
//Romberg Integration Formula
// Dr. Turhan Coban
// =======================================================
import java.io.*;
class f1 extends f_x
{
  double func(double x)
  {
  return 2.0/(1.0+2.0*x*x);
  }
}
```

```
class NA57
{
public static double integral_romberg(f_x ff,double a,double b)
{
//integral f(x)dx
//romberg integration
int n=8;//increase n to increase accuracy
double R[][];
R=new double[n+1][n+1];
int m=1;
double h=b-a;
double close=1;
double tol=1e-40;
int j=0,k=0;
double ret=0;
R[0][0]=h/2.0*(ff.func(a)+ff.func(b));
do
{
 j++;
 h/=2.0;
 trap(ff,a,h,j,m,R);
 m*=2;
 for(k=1;k<=j;k++)
 {
  R[j][k]=R[j][k-1]+(R[j][k-1]-R[j-1][k-1])/(Math.pow(4,k)-1.0);
  close=Math.abs(R[j-1][j-1]-R[j][j]);
  ret=R[j][k];
 }
}while(close>tol && j<n);
return ret;
}

public static void trap(f_x ff,double a,double h, int j,int m,double R[][])
{
   //Newton-cotes integral 2 points
            //  successive trapezoidal integration rule
   //  this program will be utilised in romberg integration
      double sum=0;
      int p;
      for(p=1;p<=m;p++)
        { sum+=ff.func(a+h*(2*p-1)); }
      R[j][0]=R[j-1][0]/2.0+h*sum;
}

 public static void main(String args[]) throws IOException
 {
 //Romberg integrali
 f1 b1=new f1();
 System.out.println("integral of class        f1 : "+
 integral_romberg(b1,-3.0,3.0));
}}
```

Romberg integration f(x)= 2.0/(1.0+2.0*x*x)

```
---------- Capture Output ----------
> "C:\co\java\bin\java.exe" integ7
```

```
integral of class       f1 : 3.7881660832213675

> Terminated with exit code 0.
```

If the iterative results of Romberg integration is given as a table

Table 7.2-1 Romberg integration of $I = \int_{-3}^{3} \frac{2}{1+2x^2} dx$

6.3157895	8.2105263							
4.2488038	3.5598086	3.2497608						
3.8058295	3.6581714	3.664729	3.6713158					
3.7866756	3.7802909	3.7884322	3.7903958	3.7908628				
3.7877768	3.7881439	3.7886674	3.7886711	3.7886644	3.7886622			
3.7880687	3.788166	3.7881675	3.7881596	3.7881576	3.7881571	3.7881569		
3.7881417	3.7881661	3.7881661	3.7881661	3.7881661	3.7881661	3.7881661	3.7881661	
3.78816	3.7881661	3.7881661	3.7881661	3.7881661	3.7881661	3.7881661	3.7881661	3.7881661

Solving of another integral $I = \int_0^{\pi} sin(x)dx = 2$ by romberg Integration Formula

Table 8.2-2 Romberg integration $I = \int_0^{\pi} sin(x)dx = 2$

1.5707963267948900	2.0943951023931900								
1.8961188979370300	2.0045597549844200	1.9985707318238300							
1.9742316019455500	2.0002691699483800	1.9999831309459800	2.0000055499796700						
1.9935703437723300	2.0000165910479300	1.9999997524545700	2.0000000162880400	1.9999999945872900					
1.9983933609701400	**2.0000010333694100**	**1.9999999961908400**	**2.0000000000596700**	**1.9999999999960300**	**2.0000000000013200**				
1.9995983886400300	2.0000000645300000	1.9999999999407000	2.0000000000002200	1.9999999999999900	1.9999999999999900	1.9999999999999900			
1.9998996001842000	2.0000000040322500	1.9999999999990700	2.0000000000000000	2.0000000000000000	2.0000000000000000	2.0000000000000000	2.0000000000000000		
1.9999749002350500	2.0000000002520000	1.9999999999999800	1.9999999999999900	1.9999999999999900	1.9999999999999900	1.9999999999999900	1.9999999999999900	1.9999999999999900	

7.2.2 AITKEN EXTRAPOLATION WITH NEWTON –COTES INTEGRATION FORMULAS

Another extrapolation Formula is te Aitken extrapolation. This equation is basically derived from Euler-McLaurin Formula. Aitken interpolation has the following form:

$$I = I_{4n} - \frac{(I_{4n} - I_{2n})^2}{(I_{4n} - I_{2n}) - (I_{2n} - I_n)}$$

By combining this equation with Simpson 1/3 formulation given above, more accurate results can be obtain

Program 7.2-2 SIMPSON 1/3 formula with aitken extrapolation

```
//=====================================================
// Numerical Analysis package in java
// example to show utilisation of integration (integral)
// and differentiation (derivative) functions
// Simpson integration with Aitken extrapolation
// Dr. Turhan Coban
// =====================================================
import java.io.*;


class f1 extends f_x
{
```

```
  double func(double x)
  {return x*Math.sqrt(x);}
}


class NA58
{

public static double newton_cotes3(f_x ff,double a,double b, int n)
{       //Newton-cotes integral 3 points
        //Simpson 1/3 rule

        double h=(b-a)/n;
        double h1=h/2.0;
        double sum=0;
        int p,i;
        double x[]=new double[3];
        double f[]=new double[3];
        for(p=0;p<n;p++)
        {  {for(i=0;i<3;i++) {x[i]=a+i*h1+p*h;f[i]=ff.func(x[i]);}}
           sum+=h*(f[0]+4.0*f[1]+f[2])/6.0;
        }
        return sum;
}

public static double Aitken_simpson1_3(f_x ff,double a,double b, int n)
{   //Aitken interpolation formula
    double I4n=newton_cotes3(ff,a,b,4*n);
    double I2n=newton_cotes3(ff,a,b,2*n);
    double In=newton_cotes3(ff,a,b,n);
    double I=I4n-(I4n-I2n)*(I4n-I2n)/((I4n-I2n)-(I2n-In));
    //System.out.println("I4n="+I4n+"I2n="+I2n+"In="+In+"I="+I);
    return I;
}

  public static void main(String args[]) throws IOException
  {
  f1 b1=new f1();
  for(int n=2;n<257;n*=2)
    System.out.println("n : "+n+"I = "+Aitken_simpson1_3(b1,0.0,1.0,n));
}
}
```

```
---------- Capture Output ---------- ---------- Capture Output ----------
> "D:\java\bin\javaw.exe" NA58
n : 2I = 0.39999984745800915
n : 4I = 0.3999999902487645
n : 8I = 0.3999999993877008
n : 16I = 0.3999999999617067
n : 32I = 0.3999999999976067
n : 64I = 0.3999999999998505
n : 128I = 0.3999999999999909
n : 256I = 0.4

> Terminated with exit code 0.
```

If Aitken intrepolation is combined with 8th degree polynomial Formula:

Program 7.2-3 Newton_Cotes 8th degree polynomial Formula with Aitken extrapolation

```
//=====================================================
// Numerical Analysis package in java
// Newton-Cotes integral formula 9 points, n=8
// with Aitken extrapolation formula
// Dr. Turhan Coban
// =====================================================
import java.io.*;

class f1 extends f_x
{
  double func(double x)
  {return x*Math.sqrt(x);}
}

class NA59
{
public static double newton_cotes9(f_x ff,double a,double b, int n)
{       //Newton-cotes integral 9 points, n=8
        // ff integrating function
        // a,b integral limits
        // n : number of sub division h=(b-a)/n;
                double h=(b-a)/n;
        double h1=h/8.0;
        double sum=0;
        int p,i;
        double x[]=new double[9];
        double f[]=new double[9];
        for(p=0;p<n;p++)
        {  {for(i=0;i<9;i++) {x[i]=a+i*h1+p*h;f[i]=ff.func(x[i]);}}
           sum+=h/28350.0*(989.0*f[0]+5888.0*f[1]-928.0*f[2]+10496.0*f[3]-4540.0*f[4]+
           10496.0*f[5]-928.0*f[6]+5888.0*f[7]+989.0*f[8]);
        }
        return sum;
}

public static double Aitken_newton_cotes9(f_x ff,double a,double b, int n)
{   //Aitken interpolation formula
              double I4n=newton_cotes9(ff,a,b,4*n);
    double I2n=newton_cotes9(ff,a,b,2*n);
    double In=newton_cotes9(ff,a,b,n);
    double I=I4n-(I4n-I2n)*(I4n-I2n)/((I4n-I2n)-(I2n-In));
    //System.out.println("I4n="+I4n+"I2n="+I2n+"In="+In+"I="+I);
    return I;
}

  public static void main(String args[]) throws IOException
  {
  f1 b1=new f1();
  for(int n=2;n<128;n*=2)
    System.out.println("n : "+n+"I = "+Aitken_newton_cotes9(b1,0.0,1.0,n));
}
}
```

```
---------- Capture Output ----------
> "D:\java\bin\javaw.exe" NA59
n : 2I = 0.39999999999998676
```

```
n : 4I = 0.39999999999999986
n : 8I = 0.39999999999999997
n : 16I = 0.39999999999999986
n : 32I = 0.4000000000000001
n : 64I = 0.4

> Terminated with exit code 0.
```

7.3 QUADRATURES: GAUSS-LEGENDRE, GAUSS - CHEBYCHEV, GAUSS -JACOBİ, GAUSS -HERMİT, GAUSS -LEQUERRE INTEGRATİON FORMULAS

A problem known from antique times, is to create a rectangle with the same area with a polynomial. IT is called a quadrature problem.

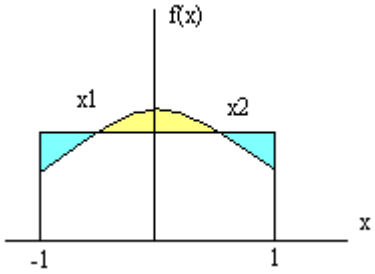


Figure 8.4-1a) a rectangle with the same area with a polynomial

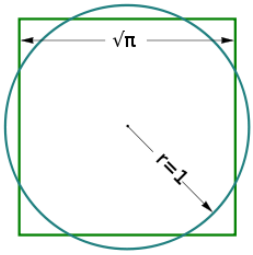


Figure 8.4-1b) A rectangle with the same area with a circle

If point x1 and x2 points intersecting the polynomial and rectangle is known, area calculation of the polynomial can be interchange with the area calculation of the rectangle. Or in more general means instead of integration process of summation can be substituted. As a general definition:

$$I_w(-1,1) = \int_{-1}^{1} w(x)f(x)dx \cong \sum_{k=1}^{n} c_k f(x_k)$$

Can be written. In this equation w(x) is the weight factor. In the first of a such formulation, Gauss-Legendre integral formulation, weight factor can be taken as 1.

$$I_w(-1,1) = \int_{-1}^{1} f(x)dx \cong \sum_{k=1}^{n} c_k f(x_k)$$

$$I_w(-1,1) = f(x)$$

If this integration is solved for a general polynamial equation.

$$f(x) = \sum_{i=0}^{n} a_i x^i$$

Following relation is found:

$$\int_{-1}^{1} x^k dx = \frac{1-(-1)^{k+1}}{k+1}$$

$\int_{-1}^{1} x^k dx \cong \frac{1-(-1)^{k+1}}{k+1}$

This equation can be written in open form as:

$$c_1 x_1^k + c_2 x_2^k + \cdots + c_n x_n^k = \frac{1-(-1)^{k+1}}{k+1} \quad 0 \le k \le 2n$$

For a special case of n=2 equation becomes :

$$f(x) = a_0 + a_1 x + a_2 x^2 + a_3 x^3$$

$$I_w(-1,1) = \int_{-1}^{1} f(x)dx \cong c_1 f(x_1) + c_2 f(x_2) = \int_{-1}^{1} (a_0 + a_1 x + a_2 x^2 + a_3 x^3)dx$$

$$c_1(a_0 + a_1 x_1 + a_2 x_1^2 + a_3 x_1^3) + c_2(a_0 + a_1 x_2 + a_2 x_2^2 + a_3 x_2^3)$$
$$= \int_{-1}^{1} (a_0)dx + \int_{-1}^{1} (a_1 x)dx + \int_{-1}^{1} (a_2 x^2)dx + \int_{-1}^{1} (a_3 x^3)dx$$

$$c_1 + c_2 = 2$$
$$c_1 x_1 + c_2 x_2 = 0$$
$$c_1 x_1^2 + c_2 x_2^2 = 2/3$$
$$c_1 x_1^3 + c_2 x_2^3 = 0$$

If this system of equation is solved roots and coefficients can be found as:

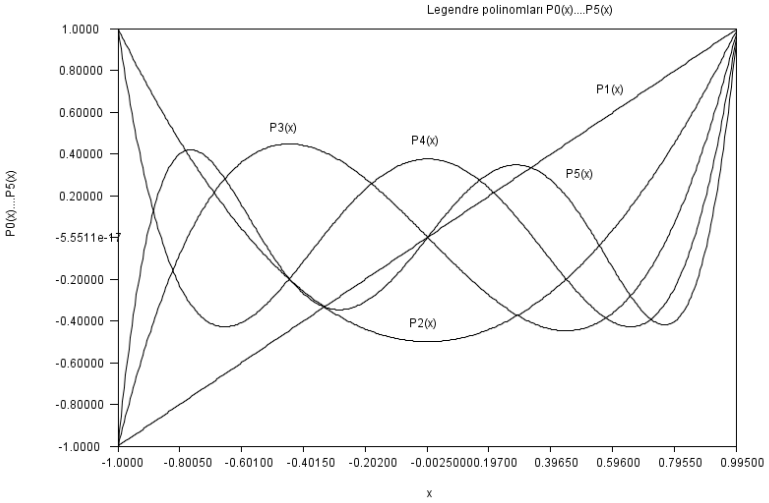
Roots: $x_{1,2} = \pm\frac{1}{\sqrt{3}}$ and Coefficients : $c_{1,2} = 1$

The general solution of this problem can be defined with the Legendre polynomials. Legendre polynomials has a general definition as:

$(k+1)P_{k+1}(x) = (2k+1)xP_k(x)$-k$P_{k-1}(x)$, $k \ge 1$

and $P_0(x)=1$,$P_1(x)=x$

The first 6 Legendre polynomials are giving in graphic form.



The roots of Legendre Polynomials are the roots of the Gauss-Legendre integration Formula. For example if $P_2(x)$ value is calculated from the above general form:

$$P_2(x) = \frac{3x^2 - 1}{2}$$

The root of this is equal to$x_{1,2} = \pm\frac{1}{\sqrt{3}}$. In general roots can be calculated from Legendre polynomials as follows:

$$\theta_{n,k} = \frac{n-k+3/4}{n+1/2}\pi$$

$$x_{n,k} = \left[1 - \frac{1}{8n^2} + \frac{1}{8n^3} - \frac{1}{384n^4}\left(39 - \frac{29}{sin^2\theta_{n,k}}\right)\right] cos\theta_{n,k} + E(n^{-5})$$

This equation contains an error level of $\mathbf{n}^{-5}$. In order to decrease the error Newton-Raphson root finding method can be usefull.

$$x_{v+1} = x_v - \frac{1 - x_v^2}{P_{n-1}(x_v) - x_v P_n(x_v)}$$

After finding the roots coeficients can be calculated as

$$c_1 x_1^{2j} + c_2 x_2^{2j} + \cdots + c_n x_n^{2j} = \frac{2}{2j+1} \quad 0 \le j \le n$$

Or

$$c_k = \frac{2(1-x_k^2)}{[nP_{n-1}(x_k)]^2}$$

As a last concept, consider that Gauss-Legendre integration limits are -1 and 1
Region x=[-1,1] can be converted to z=[a,b] by changing the variables

$$z = \left(\frac{b-a}{2}\right)x + \left(\frac{b+a}{2}\right) = \alpha x + \beta$$

In this case, integration Formula will became :

$$I_w(a,b) = \int_a^b f(x)dx \cong \alpha \sum_{k=1}^{n} c_k f(\alpha x_k + \beta)$$

First 10 Gauss-Legendre integration coefficients are given in the table

Table 8.4-1 Gauss – Legendre integration roots and coefficients

N	x_k	c_k
2	-0.577350269189625	1.000000000000000
	0.577350269189625	1.000000000000000
3	-0.774596669241483	0.555555555555552
	0.000000000000000	0.888888888888888
	0.774596669241483	0.555555555555552
4	-0.861136311594052	0.347854845137447
	-0.339981043584856	0.652145154862546
	0.339981043584856	0.652145154862546
	0.861136311594052	0.347854845137447
5	-0.906179845938664	0.236926885056181
	-0.538469310105683	0.478628670499366
	0.000000000000000	0.568888888888888
	0.538469310105683	0.478628670499366
	0.906179845938664	0.236926885056181
6	-0.932469514203152	0.171324492379162
	-0.661209386466264	0.360761573048138
	-0.238619186083196	0.467913934572691
	0.238619186083196	0.467913934572691
	0.661209386466264	0.360761573048138
	0.932469514203152	0.171324492379162
7	-0.949107912342758	0.129484966168862
	-0.741531185599394	0.279705391489276
	-0.405845151377397	0.381830050505119
	0.000000000000000	0.417959183673469
	0.405845151377397	0.381830050505119
	0.741531185599394	0.279705391489276
	0.949107912342758	0.129484966168862

8	**-0.960289856497536**	**0.101228536290369**
	-0.796666477413626	**0.222381034453374**
	-0.525532409916329	**0.313706645877887**
	-0.183434642495649	**0.362683783378362**
	0.183434642495649	**0.362683783378362**
	0.525532409916329	**0.313706645877887**
	0.796666477413626	**0.222381034453374**
	0.960289856497536	**0.101228536290369**
9	**-0.968160239507626**	**0.081274388361569**
	-0.836031107326635	**0.180648160694857**
	-0.613371432700590	**0.260610696402935**
	-0.324253423403808	**0.312347077040002**
	0.000000000000000	**0.330239355001259**
	0.324253423403808	**0.312347077040002**
	0.613371432700590	**0.260610696402935**
	0.836031107326635	**0.180648160694857**
	0.968160239507626	**0.081274388361569**
10	**-0.973906528517171**	**0.066671344308684**
	-0.865063366688984	**0.149451349150580**
	-0.679409568299024	**0.219086362515982**
	-0.433395394129247	**0.269266719309996**
	-0.148874338981631	**0.295524224714752**
	0.148874338981631	**0.295524224714752**
	0.433395394129247	**0.269266719309996**
	0.679409568299024	**0.219086362515982**
	0.865063366688984	**0.149451349150580**
	0.973906528517171	**0.066671344308684**

EXAMPLE : Calculate integral of $\boldsymbol{f(x) = \frac{\pi}{4}\sqrt{1 - x^2}}$ function between limits 0 and 1 with three points Gauss-Legendre integration formula by hand and by computer program

For n=3 α=(1-0)/2=0.5 β=(1+0)/2=0.5

I=0.5*(0.555555555555553 *f(0.5*-0.774596669241483+0.5)+ 0.888888888888889*f(0.5*0+0.5)+ 0.555555555555553 *f(0.5*0.774596669241483+0.5))

I=1.004609

excel versions of the same integral

For n=4

Gauss Integral Formula

a		0.0000000000
b		1.0000000000
α	(b-a)/2	0.5000000000
β	(b+a)/2	0.5000000000

rk	**Ck**	**xk=a+b*rk**	**yk=f(xk)**	**z=a*Ck*yk**
-0.861136312	**0.347854845**	0.0694318442	1.2701668325	0.2209168434
-0.339981044	**0.652145155**	0.3300094782	1.2019096333	0.3919097720
0.339981044	**0.652145155**	0.6699905218	0.9452143982	0.3082084950
0.861136312	**0.347854845**	0.9305681558	0.4661568155	0.0810774534
			Integral	1.002112564

For n=6

Gauss Integral Formula

a		0.0000000000
b		1.0000000000
a	(b-a)/2	0.5000000000
b	(b+a)/2	0.5000000000

rk	**Ck**	**xk=a+b*rk**	**yk=f(xk)**	**z=a*Ck*yk**
-0.932469514	**0.171324492**	0.0337652429	1.2725135329	0.1090063675
-0.661209386	**0.360761573**	0.1693953068	1.2548389257	0.2263488324
-0.238619186	**0.467913935**	0.3806904070	1.1773678593	0.2754534138
0.238619186	**0.467913935**	0.6193095930	0.9996800599	0.2338821151
0.661209386	**0.360761573**	0.8306046932	0.7090193556	0.1278934690

0.932469514	**0.171324492**	0.9662347571	0.3280671473	0.0281029687
			Integral	1.000687166

For n=10

Gauss Integral Formula

a		0.0000000000
b		1.0000000000
a	(b-a)/2	0.5000000000
b	(b+a)/2	0.5000000000

rk	Ck	xk=a+b*rk	yk=f(xk)	z=a*Ck*yk
-0.973906529	**0.066671344**	0.0130467357	1.2731311764	0.0424406835
-0.865063367	**0.149451349**	0.0674683167	1.2703383629	0.0949268911
-0.679409568	**0.219086363**	0.1602952159	1.2567754345	0.1376711792
-0.433395394	**0.269266719**	0.2833023029	1.2210757598	0.1643975319
-0.148874339	**0.295524225**	0.4255628305	1.1521912348	0.1702502107
0.148874339	**0.295524225**	0.5744371695	1.0422085462	0.1539989363
0.433395394	**0.269266719**	0.7166976971	0.8879368571	0.1195459222
0.679409568	**0.219086363**	0.8397047841	0.6914240280	0.0757407876
0.865063367	**0.149451349**	0.9325316833	0.4597517249	0.0343552578
0.973906529	**0.066671344**	0.9869532643	0.2050004799	0.0068338288
			Integral	1.000161229

For n=60

Gauss Integral Formula $f(x) = \frac{\pi}{4}\sqrt{1-x^2}$

a		0.0000000000
b		1.0000000000
a	(b-a)/2	0.5000000000
b	(b+a)/2	0.5000000000

rk	Ck	xk=a+b*rk	yk=f(xk)	z=a*Ck*yk
-0.999210123227436	0.002026811968873	0.0003949384	1.2732394454	0.0012903085
-0.995840525118838	0.004712729926953	0.0020797374	1.2732367912	0.0030002106
-0.989787895222221	0.007389931163346	0.0051060524	1.2732229468	0.0047045150
-0.981067201752598	0.010047557182288	0.0094663991	1.2731824942	0.0063961870
-0.969701788765052	0.012678166476816	0.0151491056	1.2730934350	0.0080702453
-0.955722255839996	0.015274618596784	0.0221388721	1.2729274803	0.0097217409
-0.939166276116423	0.017829901014207	0.0304168619	1.2726504171	0.0113456155
-0.920078476177627	0.020337120729457	0.0399607619	1.2722225443	0.0129366717
-0.898510310810046	0.022789516943998	0.0507448446	1.2715991671	0.0144895654
-0.874519922646898	0.025180477621521	0.0627400387	1.2707311395	0.0159988085
-0.848171984785929	0.027503556749925	0.0759140076	1.2695654443	0.0174587826
-0.819537526162145	0.029752491500789	0.0902312369	1.2680457997	0.0188637609
-0.788693739932264	0.031921219019296	0.1056531300	1.2661132804	0.0202079397
-0.755723775306585	0.034003892724946	0.1221381123	1.2637069454	0.0214854777
-0.720716513355730	0.035994898051084	0.1396417433	1.2607644611	0.0226905441
-0.683766327381355	0.037888867569243	0.1581168363	1.2572227133	0.0238173724
-0.644972828489477	0.039680695452381	0.1775135858	1.2530184000	0.0248603208
-0.604440597048510	0.041365551235585	0.1977797015	1.2480886002	0.0258139365
-0.562278900753944	0.042938892835936	0.2188605496	1.2423713133	0.0266730243
-0.518601400058569	0.044396478795787	0.2406993000	1.2358059656	0.0274327167
-0.473525841761707	0.045734379716114	0.2632370791	1.2283338822	0.0280885441
-0.427173741583078	0.046948988848912	0.2864131292	1.2198987211	0.0286365057
-0.379670056576798	0.048037031819971	0.3101649717	1.2104468712	0.0290731374
-0.331142848268448	0.048995575455757	0.3344285759	1.1999278109	0.0293955768
-0.281722937423261	0.049822035690550	0.3591385313	1.1882944300	0.0296016238
-0.231543551376029	0.050514184532509	0.3842282243	1.1755033149	0.0296897957
-0.180739964873425	0.051070156069856	0.4096300176	1.1615149982	0.0296593761
-0.129449135396945	0.051488451500981	0.4352754323	1.1462941736	0.0295104560
-0.077809333949537	0.051767943174910	0.4610953330	1.1298098787	0.0292439668
-0.025959772301248	0.051907877631221	0.4870201138	1.1120356457	0.0288617051
0.025959772301248	0.051907877631221	0.5129798862	1.0929496235	0.0283663477

0.077809333949537	0.051767943174910	0.5389046670	1.0725346712	0.0277614570
0.129449135396945	0.051488451500981	0.5647245677	1.0507784251	0.0270514770
0.180739964873425	0.051070156069856	0.5903699824	1.0276733412	0.0262417190
0.231543551376029	0.050514184532509	0.6157717757	1.0032167145	0.0253383371
0.281722937423261	0.049822035690550	0.6408614687	0.9774106755	0.0243482948
0.331142848268448	0.048995575455757	0.6655714241	0.9502621669	0.0232793209
0.379670056576798	0.048037031819971	0.6898350283	0.9217829005	0.0221398573
0.427173741583078	0.046948988848912	0.7135868708	0.8919892958	0.0209389978
0.473525841761707	0.045734379716114	0.7367629209	0.8609024020	0.0196864187
0.518601400058569	0.044396478795787	0.7593007000	0.8285478029	0.0183923025
0.562278900753944	0.042938892835936	0.7811394504	0.7949555081	0.0170672547
0.604440597048510	0.041365551235585	0.8022202985	0.7601598294	0.0157222152
0.644972828489477	0.039680695452381	0.8224864142	0.7241992435	0.0143683648
0.683766327381355	0.037888867569243	0.8418831637	0.6871162440	0.0130170282
0.720716513355730	0.035994898051084	0.8603582567	0.6489571814	0.0116795738
0.755723775306585	0.034003892724946	0.8778618877	0.6097720929	0.0103673124
0.788693739932264	0.031921219019296	0.8943468700	0.5696145247	0.0090913950
0.819537526162145	0.029752491500789	0.9097687631	0.5285413466	0.0078627110
0.848171984785929	0.027503556749925	0.9240859924	0.4866125631	0.0066917881
0.874519922646898	0.025180477621521	0.9372599613	0.4438911240	0.0055886953
0.898510310810046	0.022789516943998	0.9492551554	0.4004427432	0.0045629483
0.920078476177627	0.020337120729457	0.9600392381	0.3563357382	0.0036234215
0.939166276116423	0.017829901014207	0.9695831381	0.3116409221	0.0027782634
0.955722255839996	0.015274618596784	0.9778611279	0.2664316177	0.0020348207
0.969701788765052	0.012678166476816	0.9848508944	0.2207839805	0.0013995680
0.981067201752598	0.010047557182288	0.9905336009	0.1747782074	0.0008780470
0.989787895222221	0.007389931163346	0.9948939476	0.1285028199	0.0004748135
0.995840525118838	0.004712729926953	0.9979202626	0.0820735936	0.0001933953
0.999210123227436	0.002026811968873	0.9996050616	0.0357805413	0.0000362602
			Integral	1.000000837

A java version of Gauss-Legendre integration Formula will be written next to use n=10 formula given in the table. Our function is

$$I = \int_0^1 \frac{4}{\pi}\sqrt{1-x^2}dx = 1.0$$

Program 7.3-1 10 Point Gauss-Legendre integration

```
 //=================================================
// Numerical Analysis package in java
// example to show utilisation of integration (integral)
// and differentiation (derivative) functions
// Gauss-Legendre 10 points integration formula
// Dr. Turhan Coban
// =================================================
import java.io.*;


class f1 extends f_x
{
  public double func(double x)
  {
  return 4.0/Math.PI*Math.sqrt(1.0-x*x);
  }
}

class NA60
```

```
{
public static double integral(f_x f_xnt,double a,double b)
{
//integral f(x)dx
//integral of a function by using gauss-legendre quadrature
//coefficients are pre-calculated for 60 terms for [-1,1]
//band then utilises variable transform
 double r[],c[];
  r=new double[10];
 c=new double[10];
 r[0]=-0.973906528517171;
 r[1]=-0.865063366688984;
 r[2]=-0.679409568299024;
 r[3]=-0.433395394129247;
 r[4]=-0.148874338981631;
 r[5]=0.148874338981631;
 r[6]=0.433395394129247;
 r[7]=0.679409568299024;
 r[8]=0.865063366688984;
 r[9]=0.973906528517171;

 c[0]=0.066671344308684;
 c[1]=0.149451349150580;
 c[2]=0.219086362515982;
 c[3]=0.269266719309996;
 c[4]=0.295524224714752;
 c[5]=0.295524224714752;
 c[6]=0.269266719309996;
 c[7]=0.219086362515982;
 c[8]=0.149451349150580;
 c[9]=0.066671344308684;

int n=10;

double z=0;
double x,y;
double k1=(b-a)/2.0;
double k2=(b+a)/2.0;
double y1=0;
for(int i=0;i<n;i++)
{
x=k2+k1*r[i];
y=f_xnt.func(x);
y1=c[i]*y;
z+=y1;
}
return k1*z;
}

  public static void main(String args[]) throws IOException
  {
  //Gauss-Legendre integration formula with n=10
  f1 b1=new f1();
  System.out.println("integral of class         f1 : "+integral(b1,0.0,1.0));
}
}
```

```
---------- Capture Output ----------
> "D:\co\java\bin\javaw.exe" NA60
integral of class        f1 : 1.0001612291825637

> Terminated with exit code 0.
```

Excel version of the program:

		n	xk	ck	$\alpha x_k + \beta$			
a	0	4	-0.8611363	0.3478548	0.069432	1.270167	0.441833687	
b	1		-0.339981	0.6521452	0.330009	1.20191	0.783819544	
α	0.5		0.339981	0.6521452	0.669991	0.945214	0.61641699	
β	0.5		0.8611363	0.3478548	0.930568	0.466157	0.162154907	
								I
							2.004225128	**1.002113**

The same program as a Python version

Program 7.3-1A 10 Point Gauss-Legendre integration as Python code

```
 # Gauss - Legendre integral 10 coefficients
# python version

from math import *
def f1(x):
   return 4.0/pi*sqrt(1-x*x)

def integral(a,b):
  #integral f1(x)dx
  #integral of a function by using gauss-legendre quadrature
  #coefficients are pre-calculated for 10 terms for [-1,1]
  #band then utilises variable transform
  r=[-0.973906528517171,-0.865063366688984,-0.679409568299024,-0.433395394129247,-
0.148874338981631,0.148874338981631,0.433395394129247,0.679409568299024,0.865063366688984,0.973906528517171]
c=[0.066671344308684,0.149451349150580,0.219086362515982,0.269266719309996,0.295524224714752,0.295524224714752,0.2692667
19309996,0.219086362515982,0.149451349150580,0.066671344308684]
  n=10
  z=0
  k1=(b-a)/2.0;
  k2=(b+a)/2.0;
  for i in range(n):
    x=k2+k1*r[i]
    y=f1(x)
    z=z+k1*c[i]*y
  return z

a=float(input("enter lower limit : "))
b=float(input("enter upper limit : "))
r=integral(a,b)
print("integral  : ",r)
```

```
 ---------- Capture Output ----------
> "D:\co\python\pythonw.exe" NA60.py
enter lower limit : 0
enter upper limit : 1
integral  :  1.0001612291825637

> Terminated with exit code 0.
```

The same program as a Octave(Matlab) version

Program 7.3-1B 10 Point Gauss-Legendre integration as Octave(Matlab) code

```
function z=integral(f,a,b)
```

```
% Gauss- Legendre integration n=10
  r=[-0.973906528517171,-0.865063366688984,-0.679409568299024,-0.433395394129247,-
0.148874338981631,0.148874338981631,0.433395394129247,0.679409568299024,0.865063366688984,0.973906528517171];
c=[0.066671344308684,0.149451349150580,0.219086362515982,0.269266719309996,0.295524224714752,0.295524224714752,0.2692667
19309996,0.219086362515982,0.149451349150580,0.066671344308684];
  n=10;
  z=0;
  k1=(b-a)/2.0;
  k2=(b+a)/2.0;
  for i =1:10
    x=k2+k1*r(i);
    y=f(x);
    z=z+k1*c(i)*y;
   endfor
endfunction
```

```
>> f=@(x)4/pi*sqrt(1-x*x);
>> integral(f,0,1)
ans =  1.00016122918256
>>
```

Program 7.3-1C 10 Point Gauss-Legendre integration as Fortran 90 code

```
program GLintegral1
IMPLICIT NONE
REAL(KIND=8) a,b
a=0
b=1
print *,'integral(',a,',',b,') = ',integral(a,b)

CONTAINS
  FUNCTION func(x)
  REAL(KIND=8) func,x,PI
  PI=4.0*atan(1.0)
  func=4.0/PI*sqrt(1.0-x*x)
  END FUNCTION func

  FUNCTION integral(a,b)
  REAL(KIND=8) integral,a,b,z,x,y,y1,k1,k2
  INTEGER I,n
  REAL(KIND=8), DIMENSION(0:9) :: r,c
  r(0)=-0.973906528517171
  r(1)=-0.865063366688984
  r(2)=-0.679409568299024
  r(3)=-0.433395394129247
  r(4)=-0.148874338981631
  r(5)=0.148874338981631
  r(6)=0.433395394129247
  r(7)=0.679409568299024
  r(8)=0.865063366688984
  r(9)=0.973906528517171

  c(0)=0.066671344308684
  c(1)=0.149451349150580
  c(2)=0.219086362515982
  c(3)=0.269266719309996
  c(4)=0.295524224714752
  c(5)=0.295524224714752
  c(6)=0.269266719309996
  c(7)=0.219086362515982
  c(8)=0.149451349150580
  c(9)=0.066671344308684
  n=10
  z=0
```

```
  k1=(b-a)/2.0
  k2=(b+a)/2.0
  y1=0.0
  DO I = 0, n-1
    x=k2+k1*r(i)
    y=func(x)
    y1=c(i)*y
    z=z+y1
  END DO
  integral=k1*z
  END FUNCTION integral
end program GLintegral1
```

```
---------- Capture Output ----------
> "E:\okul\SCO1\run.bat" GLintegral1

E:\okul\SCO1>GLintegral1
 integral(   0.0000000000000000      ,   1.0000000000000000      ) =    1.0001612212606290

> Terminated with exit code 0.
```

More terms can be used to achieve more accurate results. In the next program 60 terms of the Legendre coefficcients and roots are entered tho the program.

Program 7.3-1B 60 Point Gauss-Legendre integration

```
//=====================================================
// Numerical Analysis package in java
// example to show utilisation of integration (integral)
// and differentiation (derivative) functions
// Gauss-Legendre 10 points integration formula
// Dr. Turhan Coban
// =====================================================
import java.io.*;


class f1 extends f_x
{
  public double func(double x)
  {
  return x*Math.sqrt(x);
  }
}

class NA60A
{
public static double integral(f_x f_xnt,double a,double b)
{
//integral f(x)dx
//integral of a function by using gauss-legendre quadrature
//coefficients are pre-calculated for 30 terms for [-1,1]
//band then utilises variable transform

double r[]=
{-0.99921012322743600,
-0.99584052511883800,
-0.98978789522222100,
-0.98106720175259800,
-0.96970178876505200,
-0.95572225583999600,
-0.93916627611642300,
-0.92007847617762700,
-0.89851031081004600,
```

```
-0.87451992264689800,
-0.84817198478592900,
-0.81953752616214500,
-0.78869373993226400,
-0.75572377530658500,
-0.72071651335573000,
-0.68376632738135500,
-0.64497282848947700,
-0.60444059704851000,
-0.56227890075394400,
-0.51860140005856900,
-0.47352584176170700,
-0.42717374158307800,
-0.37967005657679800,
-0.33114284826844800,
-0.28172293742326100,
-0.23154355137602900,
-0.18073996487342500,
-0.12944913539694500,
-0.07780933394953650,
-0.02595977230124780,
0.02595977230124780,
0.07780933394953650,
0.12944913539694500,
0.18073996487342500,
0.23154355137602900,
0.28172293742326100,
0.33114284826844800,
0.37967005657679800,
0.42717374158307800,
0.47352584176170700,
0.51860140005856900,
0.56227890075394400,
0.60444059704851000,
0.64497282848947700,
0.68376632738135500,
0.72071651335573000,
0.75572377530658500,
0.78869373993226400,
0.81953752616214500,
0.84817198478592900,
0.87451992264689800,
0.89851031081004600,
0.92007847617762700,
0.93916627611642300,
0.95572225583999600,
0.96970178876505200,
0.98106720175259800,
0.98978789522222100,
0.99584052511883800,
0.99921012322743600};
double c[]=
{0.00202681196887338,
0.00471272992695334,
0.00738993116334568,
0.01004755718228800,
0.01267816647681590,
0.01527461859678380,
0.01782990101420730,
0.02033712072945710,
0.02278951694399780,
0.02518047762152120,
0.02750355674992480,
0.02975249150078890,
```

```
0.03192121901929630,
0.03400389272494640,
0.03599489805108440,
0.03788886756924340,
0.03968069545238070,
0.04136555123558470,
0.04293889283593560,
0.04439647879578710,
0.04573437971611440,
0.04694898884891210,
0.04803703181997110,
0.04899557545575670,
0.04982203569055010,
0.05051418453250940,
0.05107015606985560,
0.05148845150098080,
0.05176794317491010,
0.05190787763122060,
0.05190787763122060,
0.05176794317491010,
0.05148845150098080,
0.05107015606985560,
0.05051418453250940,
0.04982203569055010,
0.04899557545575670,
0.04803703181997110,
0.04694898884891210,
0.04573437971611440,
0.04439647879578710,
0.04293889283593560,
0.04136555123558470,
0.03968069545238070,
0.03788886756924340,
0.03599489805108440,
0.03400389272494640,
0.03192121901929630,
0.02975249150078890,
0.02750355674992480,
0.02518047762152120,
0.02278951694399780,
0.02033712072945710,
0.01782990101420730,
0.01527461859678380,
0.01267816647681590,
0.01004755718228800,
0.00738993116334568,
0.00471272992695334,
0.00202681196887338};
int n=r.length;
double z=0;
double x,y;
double k1=(b-a)/2.0;
double k2=(b+a)/2.0;
for(int i=0;i<n;i++)
{
x=k2+k1*r[i];
y=f_xnt.func(x);
z+=k1*c[i]*y;
}
return z;
}

  public static void main(String args[]) throws IOException
  {
```

```
  //Gauss-Legendre integration formula with n=10
  f1 b1=new f1();
  System.out.println("integral of class        f1 : "+integral(b1,0.0,1.0));
}
}
```

```
---------- Capture Output ----------
> "C:\java\bin\javaw.exe" NA60A
integral of class        f1 : 0.39999999989833385

> Terminated with exit code 0.
```

In the next example coeeficients are directly calculated from the Legendre polynomials, so storage of any coefficients is unnecessary. Of course it will have additional cost of calculating the coefficients.

Program 7.3-2 variable Point Gauss-Legendre integration

```
 //=====================================================
// Numerical Analysis package in java
// example to show utilisation of integration (integral)
// and differentiation (derivative) functions
// desired Legendre polynomials coefficients are internally calculated
// Gauss-Legendre n points integration formula
// Dr. Turhan Coban
// =====================================================
import java.io.*;
class f1 extends f_x
{
  public double func(double x)
  {
  return x*x;
  }
}
class NA61
{
public static double F1(double h,double r)
 {double H=h/r;
  double F=1.0/(H*H+1);
  return F;
 }
public static double[][] gauss_legendre_coefficients(double x1,double x2,int n)
{
            //calculates legendre gauss-coefficients as coefficients of the integral
            //for n terms
            double EPS=3.0e-15;
            int m,j,i;
            double z1,z,xm,xl,pp,p3,p2,p1;
   //double x[]=new double[n];
   //double w[]=new double[n];
   double a[][]=new double[2][n];//a[0][i]=x[i]   a[1][i]=w[i]
            m=(n+1)/2;
            xm=0.5*(x2+x1);
            xl=0.5*(x2-x1);
            for (i=1;i<=m;i++)
            { z=Math.cos(Math.PI*((i-0.25)/(n+0.5)));
                  do {
                              p1=1.0;
                              p2=0.0;
                              for (j=1;j<=n;j++) {
                                          p3=p2;
                                          p2=p1;
```

```
                                        p1=((2.0*j-1.0)*z*p2-(j-1.0)*p3)/j;
                                    }
                                    pp=n*(z*p1-p2)/(z*z-1.0);
                                    z1=z;
                                    z=z1-p1/pp;
                            } while (Math.abs(z-z1) > EPS);
                            a[0][i-1]=xm-xl*z;
                            a[0][n-i]=xm+xl*z;
                            a[1][i-1]=2.0*xl/((1.0-z*z)*pp*pp);
                            a[1][n-i]=a[1][i-1];
                    }
return a;
}

public static double integral(f_x f_xnt,double x1,double x2,int n)
{
//n : number of integral coefficients
// this routine first generates gauss legendre coefficients
// for [x1,x2] band
// then calculates gauss legendre integral
double a[][]=new double[2][n];
a=gauss_legendre_coefficients(x1,x2,n);
System.out.println(Matrix.toString(a));
Text.print(Text.T(a));
//System.out.println(Matrix.toString(a));
double z=0;
for(int i=0;i<n;i++)
{ z+=a[1][i]*f_xnt.func(a[0][i]);
}
return z;
}

  public static void main(String args[])
  {
  //Gauss-Legendre integral of any desired point (general)
  f1 b1=new f1();

  double F12G=integral(b1,-1.0,1.0,3);
  System.out.println("F12G = "+F12G);
}
}
```

```
 ---------- Capture Output ----------
> "D:\co\java\bin\javaw.exe" NA61
      -0.774596669241483       0.000000000000000       0.774596669241483
       0.555555555555553       0.888888888888889       0.555555555555553

F12G = 0.6666666666666633
```

Program 7.3-2A variable Point Gauss-Legendre integration in Python code

```
 # Gauss - Legendre integral variable coefficients
# python version

from math import *
def f1(x):
   return x*x

def gauss_legendre_coefficients(x1,x2,n):
    #calculates legendre gauss-coefficients as coefficients of the integral
                #for n terms
                eps=3e-15
                m=int((n+1.0)/2.0)
                xm=0.5*(x2+x1)
```

```
        xl=0.5*(x2-x1)
        nmax=100
        a=[[0.0 for i in range(n)] for j in range(2)]
        for i in range(1,m+1):
                z=cos(pi*((i-0.25)/(n+0.5)))
                for ii in range(nmax):
                #while abs(z-z1) > eps:
                        p1=1.0;
                        p2=0.0;
                        for j in range(1,n+1):
                           p3=p2
                           p2=p1
                           p1=(float)(((2.0*j-1.0)*z*p2-(j-1.0)*p3)/j)
                        pp=(float)(n*(z*p1-p2)/(z*z-1.0))
                        z1=z;
                        z=z1-p1/pp
                        if abs(z-z1) < eps: break
                a[0][i-1]=xm-xl*z
                a[0][n-i]=xm+xl*z
                a[1][i-1]=2.0*xl/((1.0-z*z)*pp*pp)
                a[1][n-i]=a[1][i-1]
        return a

def integral(x1,x2,n):
  #integral f1(x)dx
  #integral of a function by using gauss-legendre quadrature
  #between x1 and x2
  a=gauss_legendre_coefficients(x1,x2,n)
  print(a)
  z=0
  for i in range(n):
    z=z+a[1][i]*f1(a[0][i])
  return z

a=-1.0#float(input("enter lower limit : "))
b=1.0#float(input("enter upper limit : "))
r=integral(a,b,3)
print("integral  : ",r)
```

```
 ---------- Capture Output ----------
> "D:\co\python\pythonw.exe" NA61.py
[[-0.7745966692414834, 0.0, 0.7745966692414834], [0.5555555555555527, 0.8888888888888888, 0.5555555555555527]]
integral  :  0.6666666666666633

> Terminated with exit code 0.
```

Writing integral formulation into an abstract class and then using lambda variables are also practical in python language following code is such an example utilizing Gauss-Legendre code:

Abstract class if_x:

```
from abc import ABC, abstractmethod
from math import *

class if_x(ABC):

   @abstractmethod
   def func(self,x):
      pass;

   def dfunc(self,x,n):
      dx=0.001;
      if n==0: df=self.func(x);
      elif n==1: df=(3.0*self.func(x-4.0*dx)-32.0*self.func(x-3.0*dx)+168.0*self.func(x-2.0*dx)-672.0*self.func(x-
```

```
dx)+672.0*self.func(x+dx)-168.0*self.func(x+2.0*dx)+32.0*self.func(x+3.0*dx)-3.0*self.func(x+4.0*dx))/840.0/dx;
      elif n==2: df=(-14350.0*self.func(x)-9.0*self.func(x-4.0*dx)+128.0*self.func(x-3.0*dx)-1008.0*self.func(x-
2.0*dx)+8064.0*self.func(x-dx)+8064.0*self.func(x+dx)-1008.0*self.func(x+2.0*dx)+128.0*self.func(x+3.0*dx)-
9.0*self.func(x+4.0*dx))/5040.0/dx/dx;
      elif n==3: df=(-7.0*self.func(x-4.0*dx)+72.0*self.func(x-3.0*dx)-338.0*self.func(x-2.0*dx)+488.0*self.func(x-dx)-
488.0*self.func(x+dx)+338.0*self.func(x+2.0*dx)-72.0*self.func(x+3.0*dx)+7.0*self.func(x+4.0*dx))/240.0/dx/dx/dx;
      elif n==4: df=(2730.0*self.func(x)+7.0*self.func(x-4.0*dx)-96.0*self.func(x-3.0*dx)+676.0*self.func(x-2*dx)-1952.0*self.func(x-
dx)-1952.0*self.func(x+dx)+676.0*self.func(x+2.0*dx)-96.0*self.func(x+3.0*dx)+7.0*self.func(x+4.0*dx))/240.0/dx/dx/dx/dx;
      elif n==5: df=(self.func(x-4.0*dx)-9.0*self.func(x-3.0*dx)+26.0*self.func(x-2.0*dx)-29.0*self.func(x-dx)+29.0*self.func(x+dx)-
26.0*self.func(x+2.0*dx)+9.0*self.func(x+3.0*dx)-self.func(x+4.0*dx))/6.0/dx/dx/dx/dx/dx;
      elif n==6: df=(-150.0*self.func(x)-self.func(x-4.0*dx)+12.0*self.func(x-3.0*dx)-52.0*self.func(x-2.0*dx)+116.0*self.func(x-
dx)+116.0*self.func(x+dx)-52.0*self.func(x+2.0*dx)+12.0*self.func(x+3.0*dx)-self.func(x+4.0*dx))/4.0/dx/dx/dx/dx/dx/dx;
      elif n==7: df=(-self.func(x-4.0*dx)+6.0*self.func(x-3.0*dx)-14.0*self.func(x-2.0*dx)+14.0*self.func(x-dx)-
14.0*self.func(x+dx)+14.0*self.func(x+2.0*dx)-6.0*self.func(x+3.0*dx)+self.func(x+4.0*dx))/2.0/dx/dx/dx/dx/dx/dx/dx;
      elif n==8: df=(70.0*self.func(x)+self.func(x-4.0*dx)-8.0*self.func(x-3.0*dx)+28.0*self.func(x-2.0*dx)-56.0*self.func(x-dx)-
56.0*self.func(x+dx)+28.0*self.func(x+2.0*dx)-8.0*self.func(x+3.0*dx)+self.func(x+4.0*dx))/dx/dx/dx/dx/dx/dx/dx/dx;
      else: df=0;
      return df;

   def gauss_legendre_coefficients(self,x1,x2,n):
      #calculates legendre gauss-coefficients as coefficients of the integral
      #for n terms
      eps=3e-15
      m=int((n+1.0)/2.0)
      xm=0.5*(x2+x1)
      xl=0.5*(x2-x1)
      nmax=100
      a=[[0.0 for i in range(n)] for j in range(2)]
      for i in range(1,m+1):
         z=cos(pi*((i-0.25)/(n+0.5)))
         for ii in range(nmax):
            #while abs(z-z1) > eps:
            p1=1.0;
            p2=0.0;
            for j in range(1,n+1):
               p3=p2
               p2=p1
               p1=(float)(((2.0*j-1.0)*z*p2-(j-1.0)*p3)/j)
            pp=(float)(n*(z*p1-p2)/(z*z-1.0))
            z1=z;
            z=z1-p1/pp
            if abs(z-z1) < eps: break
         a[0][i-1]=xm-xl*z
         a[0][n-i]=xm+xl*z
         a[1][i-1]=2.0*xl/((1.0-z*z)*pp*pp)
         a[1][n-i]=a[1][i-1]
      return a

   def integral(self,x1,x2,n):
      #integral func(x)dx
      #integral of a function by using gauss-legendre quadrature
      #between x1 and x2
      a=self.gauss_legendre_coefficients(x1,x2,n)
      z=0
      for i in range(n):
         z=z+a[1][i]*self.func(a[0][i])
      return z
```

Gauss legendre example usung abstract class if_x and lambda variables

```
# Gauss - Legendre integral variable coefficients
# python version integral methods are embedded to abstract class if_x

from math import *
```

```
#float(input("enter lower limit : "))
#float(input("enter upper limit : "))
class f1(if_x):func=lambda self,x:6.0*x*x-2;
f=f1();
r=f.integral(0.0,1.0,5)
print("integral  : ",r)
```

Program 7.3-2B variable Point Gauss-Legendre integration in Octave(Matlab) code

```
## Copyright (C) 2017 Mustafa Turhan Çoban
## Created: 2017-08-25

function [a] = gauss_legendre_cof (x1,x2,n)
##calculates legendre gauss-coefficients as coefficients of the integral
##for n terms
eps=3e-15;
 nmax=100;
m=(n+1.0)/2.0;
xm=0.5*(x2+x1);
xl=0.5*(x2-x1);
for i=1:m
   z=cos(pi*((i-0.25)/(n+0.5)));
                          for ii=0:nmax
                            p1=1.0;
                            p2=0.0;
                            for j =1:n
                                      p3=p2;
                                     p2=p1;
                                      p1=((2.0*j-1.0)*z*p2-(j-1.0)*p3)/j;
      endfor
                            pp=n*(z*p1-p2)/(z*z-1.0);
                            z1=z;
                            z=z1-p1/pp;
     if abs(z-z1) < eps break; endif
   endfor
                          a(1,i)=xm-xl*z;
                          a(1,n+1-i)=xm+xl*z;
                          a(2,i)=2.0*xl/((1.0-z*z)*pp*pp);
                          a(2,n+1-i)=a(2,i);
   endfor
endfunction
```

```
function z=integGLn(f,a,b,n)
% Gauss- Legendre integration n
a=gauss_legendre_cof(a,b,n)
  z=0;
  for i =1:n
    z=z+a(2,i)*f(a(1,i));
   endfor
endfunction
```

```
Command Window
>>
>> f=@(x)x*x;
>> integGLn(f,-1,1,3)
a =

  -0.774596669241483   0.000000000000000   0.774596669241483
   0.555555555555553   0.888888888888889   0.555555555555553

ans =  0.666666666666663
>>
```

Program 7.3-2C variable Point Gauss-Legendre integration in Fortran 90 code

```
program GLintegral2
IMPLICIT NONE
REAL(KIND=8) a,b
INTEGER n
a=-1
b=1
n=3
print *,'integral(',a,',',b,') = ',integral(a,b,n)

CONTAINS
  FUNCTION func(x)
  REAL(KIND=8) func,x,PI
  func=x*x
  END FUNCTION func

  FUNCTION integral(x1,x2,n)
  REAL(KIND=8) integral,EPS,PI,x1,x2,z1,z,xm,xl,pp,p3,p2,p1
  INTEGER n,m,i,j
  REAL(KIND=8), DIMENSION(0:n) :: a0
  REAL(KIND=8), DIMENSION(0:n) :: a1
  PI=4.0*atan(1.0)
  EPS=3.0e-15
  m=(n+1)/2.0
  xm=0.5*(x2+x1)
  xl=0.5*(x2-x1)
  DO i=1,m
    z=cos(PI*((i-0.25)/(n+0.5)))
    DO
      p1=1
      p2=0
      DO j=1,n
        p3=p2
        p2=p1
        p1=((2.0*j-1.0)*z*p2-(j-1.0)*p3)/j
      END DO
      pp=n*(z*p1-p2)/(z*z-1.0)
                z1=z
                z=z1-p1/pp
    IF(abs(z-z1)<EPS) EXIT
    END DO
    a0(i-1)=xm-xl*z
    a0(n-i)=xm+xl*z
    a1(i-1)=2.0*xl/((1.0-z*z)*pp*pp)
              a1(n-i)=a1(i-1);
  END DO
  DO i=0,n-1
    z=z+a1(i)*func(a0(i))
  END DO
  integral=z
  END FUNCTION integral
end program GLintegral2
```

```
---------- Capture Output ----------
> "E:\okul\SCO1\run.bat" GLintegral2

E:\okul\SCO1>GLintegral2
 integral(  -1.0000000000000000      ,   1.0000000000000000      ) =   0.66666666666666330

> Terminated with exit code 0.
```

In the Gauss-Legendre integral formulation weight function was taken as 1.

$$I_w(-1,1) = \int_{-1}^{1} w(x)f(x)dx \cong \sum_{k=1}^{n} c_k f(x_k)$$

If the weight function is taken as:

$$w(x) = \frac{1}{\sqrt{1-x^2}}$$

Gauss-Chebychev integration formula is created. This equation has the form of

$$I_w(-1,1) = \int_{-1}^{1} \frac{1}{\sqrt{1-x^2}} f(x)dx \cong \sum_{k=1}^{n} c_k f(x_k)$$

x_k in the equation is the roots of chebcychev polynomials. General definition of the Chbychev polynomials are given as:

$P_0(x)=1$

$P_1(x)=x$

$P_2(x)=2x^2-1$

$P_n(x)=2xP_{n-1}(x) – P_{n-2}(x)$, n=3,4,5..

It should be note that Chebychev polinomials are orthogonal polnomials. The roots of the polynomials are defined as:

$$x_k = cos\left(\frac{\left(k+\frac{1}{2}\right)\pi}{n}\right), \quad k = 0,1,2,..n-1$$

Coeeficients of the Formula can be calculated as:

$$c_k = \frac{\pi}{n}, \qquad k = 0,1,2,..n-1$$

x=[-1,1] integration range in Gauss-Chebychev Formula can be change to more general z=[a,b] range by using similar formulations used in Gauss-Legendre formulations.

As an example program if following integral is to be solved:

$$I = \int_{-1}^{1} \frac{e^x}{\sqrt{1-x^2}} dx$$

Program 7.3-3 Gauss-Chebychev integral

```
import java.util.*;
import java.awt.*;
import javax.swing.*;
import java.awt.event.*;

class f1 extends f_x
{
  public double func(double x)
  {
  return Math.exp(x);
  }
}

public class integ_gauss_chebychev
{

public static double[][] gauss_chebychev_coefficients(int n)
{   //Gauss-Chebychev quadratürü katsayilarini hesaplar
    double sgngam=1;
```

```
    double a[][]=new double[2][n];
    for(int j = 1; j <=n; j++)
    {int k=j-1;
     a[0][k]=Math.cos((2.0*j-1.0)*Math.PI/(2.0*n));
     a[1][k]=Math.PI/(double)n;
    }
return a;
}

public static double gauss_chebychev_integral(f_x f_xnt,int n)
{
//n : number of integral coefficients
// this routine first generates gauss chebychev coefficients
// for [xa,xb] band
// then calculates gauss chebychev integral
double a[][]=new double[2][n];
a=gauss_chebychev_coefficients(n);
//System.out.println(Matrix.toString(a));

double toplam=0;
double ck;
double x,y;
double z1=0;
for(int k=0;k<n;k++)
{ toplam+=a[1][k]*f_xnt.func(a[0][k]); }
return toplam;
}

     public static void main(String[] args)
     {
              f1 b1=new f1();
     System.out.println("integral  : "+gauss_chebychev_integral(b1,50));
     }
}
```

```
---------- Capture Output ----------
> "D:\java\bin\javaw.exe" NA62
integral  : 3.9774632605064237

> Terminated with exit code 0.
```

Gauss-Jacobi (Mehler) integration Formula is in the form of

$$I_w(-1,1) = \int_{-1}^{1} w(x)f(x)dx = \int_{-1}^{1} (1-x)^{\alpha}(1+x)^{\beta}f(x)dx = \cong \sum_{k=0}^{n-1} c_k f(x_k)$$

In this equation x_k is the root of jacobi polynomials Jacobi polynomials are in the form of

$$J_n(x;\alpha,\beta) = \sum_{i=0}^{n} A_n x^n$$

Coefficients are defined as:

$$A_n = \frac{1}{2n}\frac{\Gamma(n+\alpha+1)\Gamma(n+\beta+1)}{\Gamma(n+\alpha+\beta+1)}$$

Gauss-Jacobi integration equation coefficients, c_k's are defined as :

$$c_k = -\frac{(2n+\alpha+\beta+2)\Gamma(n+\alpha+1)\Gamma(n+\beta+1)}{(2n+\alpha+\beta+1)\Gamma(n+\alpha+\beta+1)!}\frac{2^{\alpha+\beta}}{J_n(x_k;\alpha,\beta)J_{n+1}(x_k;\alpha,\beta)}$$

In this equation Γ is the gamma function. One of the special form of equation is when α=0 and β=0. It is given Gauss_legendre equation. The second one is when α=-0.5 and β=-0.5. It is given the Gauss-Chebychev formulation. As it seen Gauss-Jacobi equation set is a kind of general solution with different possible solutions depends on α and β coefficients. In our example problem, the previous problem defined will be solved by using Gauss-Legendre polynomial.

$$I = \int_{-1}^{1} \frac{e^x}{\sqrt{1-x^2}}\,dx$$

Program 7.3-4 variable point Gauss-Jacobi integration with α=0 and β=0 (Gauss-Legendre)

```
import java.util.*;
import java.awt.*;
import javax.swing.*;
import java.awt.event.*;

class f1 extends f_x
{
  public double func(double x)
  {
  return Math.exp(x)/Math.sqrt(1-x*x);
  }
}

public class integ_gauss_jacobi
{

   public static double [] lngamma(double x)
   {//logaritmik hata fonksiyonu
   //xx[0] hata fonksiyonunun logaritmasinin değerini
   //xx[1] hata fonksiyonunun kendisinin işaretini verir
            double xx[]=new double[2];
   double result;
   double a;
   double b;
   double c;
   double p;
   double q;
   double u;
   double w;
   double z;
   int i;
   double logpi;
   double ls2pi;
   double tmp;
   double sgngam = 1;
   logpi = 1.14472988584940017414;
   ls2pi = 0.91893853320467274178;
   if( x<-34.0 )
   {
      q = -x;
      double yy[]=lngamma(q);
      w = yy[0];
      tmp=yy[1];
```

```
    p = Math.floor(q);
    i = (int)Math.round(p);
    if( i%2==0 )
    {
       sgngam = -1;
    }
    else
    {
       sgngam = 1;
    }
    z = q-p;
    if( z>0.5 )
    {
       p = p+1;
       z = p-q;
    }
    z = q*Math.sin(Math.PI*z);
    result = logpi-Math.log(z)-w;
    xx[0]=result;
    xx[1]=sgngam;
    return xx;
 }
 if( x<13 )
 {
    z = 1;
    p = 0;
    u = x;
    while(u>=3)
    {
       p = p-1;
       u = x+p;
       z = z*u;
    }
    while(u<2)
    {
       z = z/u;
       p = p+1;
       u = x+p;
    }
    if( z<0 )
    {
       sgngam = -1;
       z = -z;
    }
    else
    {
       sgngam = 1;
    }
    if( u==2 )
    {
       result = Math.log(z);
       //return result;
       xx[0]=result;
       xx[1]=sgngam;
       return xx;
    }
    p = p-2;
```

```
      x = x+p;
      b = -1378.25152569120859100;
      b = -38801.6315134637840924+x*b;
      b = -331612.992738871184744+x*b;
      b = -1162370.97492762307383+x*b;
      b = -1721737.00820839662146+x*b;
      b = -853555.664245765465627+x*b;
      c = 1;
      c = -351.815701436523470549+x*c;
      c = -17064.2106651881159223+x*c;
      c = -220528.590553854454839+x*c;
      c = -1139334.44367982507207+x*c;
      c = -2532523.07177582951285+x*c;
      c = -2018891.41433532773231+x*c;
      p = x*b/c;
      result = Math.log(z)+p;
      //return result;
      xx[0]=result;
      xx[1]=sgngam;
      return xx;
   }
   q = (x-0.5)*Math.log(x)-x+ls2pi;
   if( x>100000000 )
   {
      result = q;
      //return result;
      xx[0]=result;
      xx[1]=sgngam;
      return xx;
   }
   p = 1/(x*x);
   if( x>=1000.0 )
   {
      q = q+((7.9365079365079365079365*0.0001*p-
2.777777777777777777778*0.001)*p+0.083333333333333333333)/x;
   }
   else
   {
      a = 8.11614167470508450300*0.0001;
      a = -5.95061904284301438324*0.0001+p*a;
      a = 7.9365034045771694 3945*0.0001+p*a;
      a = -2.77777777730099687205*0.001+p*a;
      a = 8.33333333333331927722*0.01+p*a;
      q = q+a/x;
   }
   xx[0]= q;
   xx[1]=sgngam;
   return xx;
   }
//====================

   public static double[][] gauss_jacobi_coefficients(int n,double alpha,double beta)
   {
      double eps=1.0e-30;
            int i = 0;
      int j = 0;
      double r = 0;
```

```
double r1 = 0;
double t1 = 0;
double t2 = 0;
double t3 = 0;
double p1 = 0;
double p2 = 0;
double p3 = 0;
double pp = 0;
double an = 0;
double bn = 0;
double a = 0;
double b = 0;
double c = 0;
double tmpsgn = 0;
double alfbet = 0;
double temp = 0;
int its = 0;
double A[][]=new double[2][n];
for(i=0; i<=n-1; i++)
{   if( i==0 )
   {
      an = alpha/n;
      bn = beta/n;
      t1 = (1+alpha)*(2.78/(4+n*n)+0.768*an/n);
      t2 = 1+1.48*an+0.96*bn+0.452*an*an+0.83*an*bn;
      r = (t2-t1)/t2;
   }
   else
   {
      if( i==1 )
      {
         t1 = (4.1+alpha)/((1+alpha)*(1+0.156*alpha));
         t2 = 1+0.06*(n-8)*(1+0.12*alpha)/n;
         t3 = 1+0.012*beta*(1+0.25*Math.abs(alpha))/n;
         r = r-t1*t2*t3*(1-r);
      }
      else
      {
         if( i==2 )
         {
            t1 = (1.67+0.28*alpha)/(1+0.37*alpha);
            t2 = 1+0.22*(n-8)/n;
            t3 = 1+8*beta/((6.28+beta)*n*n);
            r = r-t1*t2*t3*(A[0][0]-r);
         }
         else
         {
            if( i<n-2 )
            {
               r = 3*A[0][i-1]-3*A[0][i-2]+A[0][i-3];
            }
            else
            {
               if( i==n-2 )
               {
                  t1 = (1+0.235*beta)/(0.766+0.119*beta);
                  t2 = 1/(1+0.639*(n-4)/(1+0.71*(n-4)));
```

```
                    t3 = 1/(1+20*alpha/((7.5+alpha)*n*n));
                    r = r+t1*t2*t3*(r-A[0][i-2]);
                }
                else
                {
                    if( i==n-1 )
                    {
                        t1 = (1+0.37*beta)/(1.67+0.28*beta);
                        t2 = 1/(1+0.22*(n-8)/n);
                        t3 = 1/(1+8*alpha/((6.28+alpha)*n*n));
                        r = r+t1*t2*t3*(r-A[0][i-2]);
                    }
                }
            }
        }
    }
    alfbet = alpha+beta;
    do
    {
        temp = 2+alfbet;
        p1 = (alpha-beta+temp*r)*0.5;
        p2 = 1;
        for(j=2; j<=n; j++)
        {
            p3 = p2;
            p2 = p1;
            temp = 2*j+alfbet;
            a = 2*j*(j+alfbet)*(temp-2);
            b = (temp-1)*(alpha*alpha-beta*beta+temp*(temp-2)*r);
            c = 2*(j-1+alpha)*(j-1+beta)*temp;
            p1 = (b*p2-c*p3)/a;
        }
        pp = (n*(alpha-beta-temp*r)*p1+2*(n+alpha)*(n+beta)*p2)/(temp*(1-r*r));
        r1 = r;
        r = r1-p1/pp;
    }
    while( Math.abs(r-r1)>=eps*(1+Math.abs(r))*100 );
    A[0][i] = r;
    double yy[]=lngamma(alpha+n);
    double w1 = yy[0];
    tmpsgn=yy[1];
    yy=lngamma(beta+n);
    double w2 = yy[0];
    tmpsgn=yy[1];
    yy=lngamma(n+1);
    double w3 = yy[0];
    tmpsgn=yy[1];
    yy=lngamma(n+alfbet+1);
    double w4 = yy[0];
    tmpsgn=yy[1];
    A[1][i] = Math.exp(w1+w2-w3-w4)*temp*Math.pow(2, alfbet)/(pp*p2);
  }
return A;
}
```

```
public static double gauss_jacobi_integral(f_x f_xnt,int n,double alpha,double beta)
{
//n : number of integral coefficients
// this routine first generates gauss legendre coefficients
// for [x1,x2] band
// then calculates gauss hermite integral
double a[][]=new double[2][n];
a=gauss_jacobi_coefficients(n,alpha,beta);
System.out.println(Matrix.toString(a));
double z=0;
for(int i=0;i<n;i++)
{ z+=a[1][i]*f_xnt.func(a[0][i]);}
return z;
}

    public static void main(String[] args)
    {
            f1 b1=new f1();
    System.out.println(gauss_jacobi_integral(b1,5,0.0,0.0));
    }
}
```

Gauss - Jacobi integration with n=5 points

```
---------- Capture Output ----------
> "D:\java\bin\javaw.exe" NA63
     0.906179845938664     0.538469310105683     0.000000000000000    -0.538469310105683    -0.906179845938664
     0.236926885056189     0.478628670499367     0.568888888888889     0.478628670499367     0.236926885056189

3.4865363609790476

> Terminated with exit code 0.
```

As it is seen Gauss-Legendre coefficients are obtained here correctly. Since n = 5 only accuracy is not very high. If it is solved with more coefficients :

Gauss - Jacobi integration with n=20 points α=0 and β=0

```
---------- Capture Output ----------
> "D:\java\bin\javaw.exe" NA63

3.846342341165288

> Terminated with exit code 0.
```

Gauss - Jacobi integration with n=50 points α=0 and β=0

```
---------- Capture Output ----------
> "D:\java\bin\javaw.exe" NA63
n=50 integral = 3.9242400049441444

> Terminated with exit code 0.
```

Gauss - Jacobi integration with n=100 points α=0 and β=0

```
---------- Capture Output ----------
```

```
> "D:\java\bin\javaw.exe" NA63
n=100 integral = 3.9506992376105177

> Terminated with exit code 0.
```

If we change function and parameter to solve the same program as α=-0.5 and β=-0.5 (Gauss-Chebychev formula)

Program 7.3-5 variable point Gauss-Jacobi integration with α=-0.5 and β=-0.5 (Gauss-Legendre)

```
class f1 extends f_x
{
  public double func(double x)
  {  return Math.exp(x); }
}
……
System.out.println("n="+n+" integral = "+gauss_jacobi_integral(b1,n,-0.5,-0.5));
……
```

10 point variable point Gauss-Jacobi integration with α=-0.5 and β=-0.5 (Gauss-Legendre)

```
---------- Capture Output ----------
> "D:\java\bin\javaw.exe" NA63
n=10 integral = 3.97746265002148

> Terminated with exit code 0.
```

Gauss-Hermit formula : this formulation uses the weight function and calculates integrals from minus infinity to plus infinity.

$$w(x) = e^{-x^2}$$

$$I_w(-1,1) = \int_{-\infty}^{\infty} w(x)f(x)dx = \int_{-\infty}^{\infty} e^{-x^2}f(x)dx = \cong \sum_{k=0}^{n-1} c_k f(x_k)$$

In this equation x_k is the roots of the Hermite polynomials. Hermit Polinomials defined as:

$P_{-1}(x)=0$

$P_0(x)=1$

$P_1(x)=2x$

$P_2(x)=4x^2-2$

$P_3(x)=8x^3-12x$

$P_n(x)=2xP_{n-1}(x) – 2(n-1)P_{n-2}(x)$, n=1,2,3,4,5..

Coefficients are given as:

$$c_k = \frac{2^{n-1}n!\sqrt{\pi}}{n^2[P_{n-1}(x_k)]^2}$$

Coefficients for the first 7 sets are given in the following table

Table 7.3-6 Gauss –Hermit integration roots and coefficients

N	x_k	c_k
2	+/-0.707167811	0.886226926
3	0	1.181635901
	+/- 1.224744871391589	0.295408975
4	+/- 0.524647623275290	0.80491409
	+/- 1.650680123885784	0.081312835
5	0	0.945308721
	+/-0.9585724646	0.393619323
	+/-2.0201828705	0.019953242
6	+/- 2.350604973674492	0.00453001
	+/- 1.335849074013697	0.15706732
	+/- 0.436077411927616	0.724629595
7	+/- 2.651961356835233	0.000971781
	+/- 1.673551628767471	0.054515583
	+/- 0.816287882858965	0.425607253
	0	0.810264618

Gauss_Hermite integral equation example code :

Program 7.3-7 Gauss-Hermite integration

```
import java.util.*;
import java.awt.*;
import javax.swing.*;
import java.awt.event.*;

class f1 extends f_x
{
  public double func(double x)
  {
  return 1/(1+x*x);
  }
}

public class NA64
{

public static double[][] gauss_hermite_coefficients(int n)
{
   double a[][]=new double[2][n];
   int i;
   int j;
   double r=0;
   double r1;
   double p1;
   double p2;
   double p3;
   double dp3;
   double pipm4;
   double eps=1e-10;
   pipm4 = Math.pow(Math.PI, -0.25);
   for(i = 0; i <= (n+1)/2-1; i++)
   {
      if( i==0 )
      { r =Math.sqrt((double)(2*n+1))-1.85575*Math.pow((double)(2*n+1), -1.0/6.0);}
      else if( i==1 )
         { r = r-1.14*Math.pow((double)(n), 0.426)/r; }
      else if( i==2 )
            { r = 1.86*r-0.86*a[0][0];}
      else if( i==3 )
            { r = 1.91*r-0.91*a[0][1];}
      else
```

```
            {r = 2*r-a[0][i-2];}
   do {
         p2 = 0;
         p3 = pipm4;
         for(j = 0; j <= n-1; j++)
         {   p1 = p2;
            p2 = p3;
            p3 = p2*r*Math.sqrt(2.0/(double)(j+1))-p1*Math.sqrt((double)(j)/(double)(j+1));
         }
         dp3 = Math.sqrt((double)(2.0*j))*p2;
         r1 = r;
         r = r-p3/dp3;
      }
      while(Math.abs(r-r1)>=eps*(1+Math.abs(r))*100.0);
   a[0][i] = r;
   a[1][i] = 2.0/(dp3*dp3);
   a[0][n-1-i] = -a[0][i];
   a[1][n-1-i] = a[1][i];
  }
  return a;
}

public static double gauss_hermite_integral(f_x f_xnt,int n)
{
//n : number of integral coefficients
// this routine first generates gauss hermite coefficients
// for [x1,x2] band
// then calculates gauss hermite integral
double a[][]=new double[2][n];
a=gauss_hermite_coefficients(n);
System.out.println(Matrix.toString(a));
double z=0;
for(int i=0;i<n;i++)
{ z+=a[1][i]*f_xnt.func(a[0][i])*Math.exp(a[0][i]*a[0][i]);}
return z;
}

    public static void main(String[] args)
    {
            f1 b1=new f1();
    System.out.println("integral  : "+gauss_hermite_integral(b1,20));
    }
}
```

Output 7.3-8 Gauss-Hermite integrasyonu

```
---------- Capture Output ----------
> "D:\java\bin\javaw.exe" NA64
      5.387480890011233      4.603682449550744      3.944764040115625      3.347854567383216      2.788806058428131
2.254974002089276      1.738537712116586      1.234076215395323      0.737473728545394      0.245340708300901      -
0.245340708300901      -0.737473728545394      -1.234076215395323      -1.738537712116586      -2.254974002089276      -
2.788806058428131      -3.347854567383216      -3.944764040115625      -4.603682449550744      -5.387480890011233
      0.000000000000223      0.000000000439934      0.000000108606937      0.000007802556261      0.000228338599065
0.003243773339370      0.024810520887341      0.109017206020022      0.286675503681034      0.462243669596553
0.462243669596553      0.286675503681034      0.109017206020022      0.024810520887341      0.003243773339370
0.000228338599065      0.000007802556261      0.000000108606937      0.000000000439934      0.000000000000223

integral  : 2.804683301721163

> Terminated with exit code 0.
```

Gauss-Laguerre formulation has the weight factor in the form of

$$w(x) = x^{\alpha}e^{-x}$$

The integral limits of this formulation is in between o and infinity. The equation is in the form of

$$I_w(0,\infty) = \int_0^\infty f(x)w(x)dx = \int_0^\infty x^\alpha e^{-x} f(x)dx \cong \sum_{k=0}^{n-1} c_k f(x_k)$$

In this equation x_k is the root of Laguerre polinomials. Leguerre polynomial has the form of:

$P_{-1}(x)=0$

$P_0(x)=1$

$(n+1)P^{\alpha}_{n+1}(x)=(-x+2*n+\alpha+1)P^{\alpha}_{n}(x) - (n+\alpha)P^{\alpha}_{n-1}(x)$, n=1,2,3,4,5...

Coefficients,

$$c_k = \frac{1}{x_k \left[\frac{dP_n(x_k)}{dx_k}\right]^2}$$

For the first 5 values, roots and coefficient of the equation is given in **Table 8.4-3** (α=0), **Table 8.4-4** (α=1) and **Table 8.4-5** (α=2) .

Table 7.3-9 Gauss – Laquerre integral roots and coefficients (α=0)

N	x_k	c_k
2	0.585786437626905	0.853553390592534
	3.414213562373090	0.146446609406723
3	0.415774556783479	0.711093009915089
	2.294280360279040	0.278517733568676
	6.289945082937470	0.010389256497354
4	0.322547689619392	0.603154104288259
	1.745761101158340	0.357418692437802
	4.536620296921120	0.038887908507759
	9.395070912301130	0.000539294705556
5	0.263560319718141	0.521755610467045
	1.413403059106510	0.398666821407258
	3.596425771040720	0.075942449681708
	7.085810005858830	0.003611758663878
	12.640800844275700	0.000023369972386

Table 7.3-10 Gauss – Laquerre integral roots and coefficients (α=1)

N	x_k	c_k
2	1.267949192431120	0.788675133123411
	4.732050807568870	0.211324865405185
3	0.935822227524088	0.588681472855445
	3.305407289332270	0.391216059222280
	7.758770483143630	0.020102459732679
4	0.743291927981432	0.446870579513482
	2.571635007646270	0.477635774492073
	5.731178751689090	0.074177784726293
	10.953894312683100	0.001315849683447
5	0.617030853278271	0.348014523429866
	2.112965958578520	0.502280674138866
	4.610833151017530	0.140915919102187
	8.399066971204840	0.008719893025972

Table 7.3-11 Gauss – Laquerre integral roots and coefficients (α=2)

N	x_k	c_k
2	2.000000000000000	1.500000000000000
	6.000000000000000	0.500000000000000
3	1.517387080677410	1.037494961490390
	4.311583133719520	0.905750004703039
	9.171029785603060	0.056755033772202
4	1.226763263500300	0.725524997698604
	3.412507358696940	1.063424292391060
	6.902692605851610	0.206696130999709
	12.458036771951100	0.004354579188558
5	1.031109144093380	0.520917396835042
	2.837212823953820	1.066705933159050
	5.620294272598700	0.383549720007093

	9.682909837664020	**0.028564233510280**
	15.828473921690000	**0.000262712802303**

If we investigate Gauss-Leguerre Formula with an example. We will calculate the integral

$$I = \int_0^\infty \frac{x}{e^x + 1} dx$$

Exact integral value I= 0.822467.

Program 7.3-8 Gauss-Laguerre integration

```
import java.util.*;
import java.awt.*;
import javax.swing.*;
import java.awt.event.*;

class f1 extends f_x
{
  public double func(double x)
  {
  return x/(Math.exp(x)+1.0);
  }
}

public class NA65
{
   public static double [] lngamma(double x)
   {//logaritmik hata fonksiyonu
   //xx[0] hata fonksiyonunun logaritmasinin değerini
   //xx[1] hata fonksiyonunun kendisinin işaretini verir
            double xx[]=new double[2];
   double result;
   double a;
   double b;
   double c;
   double p;
   double q;
   double u;
   double w;
   double z;
   int i;
   double logpi;
   double ls2pi;
   double tmp;
   double sngam = 1;
   logpi = 1.14472988584940017414;
   ls2pi = 0.91893853320467274178;
   if( x<-34.0 )
   {
      q = -x;
      double yy[]=lngamma(q);
      w = yy[0];
      tmp=yy[1];
      p = Math.floor(q);
      i = (int)Math.round(p);
      if( i%2==0 )
      {
         sngam = -1;
```

```
    }
    else
    {
       sgngam = 1;
    }
    z = q-p;
    if( z>0.5 )
    {
       p = p+1;
       z = p-q;
    }
    z = q*Math.sin(Math.PI*z);
    result = logpi-Math.log(z)-w;
    xx[0]=result;
    xx[1]=sgngam;
    return xx;
 }
 if( x<13 )
 {
    z = 1;
    p = 0;
    u = x;
    while(u>=3)
    {
       p = p-1;
       u = x+p;
       z = z*u;
    }
    while(u<2)
    {
       z = z/u;
       p = p+1;
       u = x+p;
    }
    if( z<0 )
    {
       sgngam = -1;
       z = -z;
    }
    else
    {
       sgngam = 1;
    }
    if( u==2 )
    {
       result = Math.log(z);
       //return result;
       xx[0]=result;
       xx[1]=sgngam;
       return xx;
    }
    p = p-2;
    x = x+p;
    b = -1378.25152569120859100;
    b = -38801.6315134637840924+x*b;
    b = -331612.992738871184744+x*b;
    b = -1162370.97492762307383+x*b;
```

```
      b = -1721737.00820839662146+x*b;
      b = -853555.664245765465627+x*b;
      c = 1;
      c = -351.815701436523470549+x*c;
      c = -17064.2106651881159223+x*c;
      c = -220528.590553854454839+x*c;
      c = -1139334.44367982507207+x*c;
      c = -2532523.07177582951285+x*c;
      c = -2018891.41433532773231+x*c;
      p = x*b/c;
      result = Math.log(z)+p;
      //return result;
      xx[0]=result;
      xx[1]=sgngam;
      return xx;
   }
   q = (x-0.5)*Math.log(x)-x+ls2pi;
   if( x>100000000 )
   {
      result = q;
      //return result;
      xx[0]=result;
      xx[1]=sgngam;
      return xx;
   }
   p = 1/(x*x);
   if( x>=1000.0 )
   {
      q = q+((7.9365079365079365079365*0.0001*p-
2.7777777777777777777778*0.001)*p+0.0833333333333333333333)/x;
   }
   else
   {
      a = 8.1161416747050845030 0*0.0001;
      a = -5.95061904284301438324*0.0001+p*a;
      a = 7.9365034045771694394 5*0.0001+p*a;
      a = -2.77777777730099687205*0.001+p*a;
      a = 8.33333333333319277 22*0.01+p*a;
      q = q+a/x;
   }
   xx[0]= q;
   xx[1]=sgngam;
   return xx;
   }

public static double[][] gauss_laguerre_coefficients(int n,double alpha)
{   //Gauss-Laguerre quadratürü katsayilarini hesaplar
   double sgngam=1;
   double a[][]=new double[2][n];
   int i;
   int j;
   double r=0;
   double r1;
   double p1;
   double p2;
   double p3;
   double dp3;
```

```
    double tsg;
    double eps=1.0e-10;
    for(i = 0; i <= n-1; i++)
    {
       if( i==0 )
       {
          r = (1.0+alpha)*(3.0+0.92*alpha)/(1+2.4*n+1.8*alpha);
       }
       else
       {
          if( i==1 )
          {r = r+(15.0+6.25*alpha)/(1.0+0.9*alpha+2.5*n);}
          else
          {r = r+((1.0+2.55*(i-1))/(1.9*(i-1))+1.26*(i-1)*alpha/(1+3.5*(i-1)))/(1.0+0.3*alpha)*(r-a[0][i-2]);}
       }
       do
       {
          p2 = 0;
          p3 = 1;
          for(j = 0; j <= n-1; j++)
          {
             p1 = p2;
             p2 = p3;
             p3 = ((-r+2*j+alpha+1)*p2-(j+alpha)*p1)/(j+1);
          }
          dp3 = (n*p3-(n+alpha)*p2)/r;
          r1 = r;
          r = r-p3/dp3;
       }
       while(Math.abs(r-r1)>=eps*(1+Math.abs(r))*100);
       a[0][i] = r;
       double lng[]=lngamma((double)n);
       double lng1[]=lngamma((double)(alpha+n));
       a[1][i] = -Math.exp(lng1[0]-lng[0])/(dp3*n*p2);
       tsg=lng[1];
    }
return a;
}

public static double gauss_laquerre_integral(f_x f_xnt,double xa,int n,double alpha)
{
//n : number of integral coefficients
// this routine first generates gauss legendre coefficients
// for [x1,x2] band
// then calculates gauss hermite integral
double a[][]=new double[2][n];
a=gauss_laguerre_coefficients(n,alpha);
//System.out.println(Matrix.toString(a));
double z=0;
for(int i=0;i<n;i++)
{ z+=a[1][i]*f_xnt.func(a[0][i]+xa)*Math.exp(a[0][i])*Math.pow(a[0][i],-alpha);}
return z;
}

public static double gauss_laquerre_integral(f_x f_xnt,double xa,int n)
{
double alpha=1.0;
```

```
double a[][]=new double[2][n];
a=gauss_laguerre_coefficients(n,alpha);
//System.out.println(Matrix.toString(a));
double z=0;
for(int i=0;i<n;i++)
{ z+=a[1][i]*f_xnt.func(a[0][i]+xa)*Math.exp(a[0][i])/a[0][i];}
return z;
}

    public static void main(String[] args)
    {
            f1 b1=new f1();
    System.out.println("integral  : "+gauss_laquerre_integral(b1,0,30)+" exact= "+0.822467);
    }
}
```

Gauss-Laguerre integration

```
---------- Capture Output ----------
> "D:\java\bin\javaw.exe" NA65
integral  : 0.8224670275951408 exact= 0.822467

> Terminated with exit code 0.
```

7.4 INTEGRATION FORMULATION WITH ADJUSTABLE ERROR

If integration in region **[a,b]** to be defined for a continious integrable function f(x), Integration can be written as:

$$I = \int_a^b f(x)dx$$

If a sub-region is defined for the integration. For example,

$$m = \frac{a-b}{2}$$

definition is given integral can be redefined as

$$I = \int_a^m f(x)dx + \int_m^b f(x)dx$$

For numerical integration it is generally assumed that the error of the second definition will be less than the first definition. It is possible to continue this process by dividing each integral to subregions again. The important concept in here is to stop dividing integral into smaller parts when the desired accuracy is achieved. In this section several integration methods with error control will be investigated.

7.4.1 Adjustable Simpson 1/3 integral

If simpson 1/3 integration equation is written in [a,b] region

$$I(a,b) = I(x_0, x_4) = \frac{h}{3}[f(x_0) + 4f(x_2) + f(x_4)]$$

$$\int_{x_0}^{x_4} f(x)dx = I(x_0, x_4) - h^5 \frac{f^{(4)}(d_1)}{90}$$

Where h =(b-a)/2, x_0=a,x_2=(a+b)/2,x_4=b and d_1 is a number between a and b. and $h^5 \frac{f^{(4)}(d_1)}{90}$ equation gives amount of error in the integral equation. $f^{(4)}$ term in the error equation is the fourth derivative of the function. If the integration region is divided into two subregion, equation becomes

$$I(x_0, x_2) = \frac{h}{6}[f(x_0) + 4f(x_1) + f(x_2)]$$

$$I(x_2, x_4) = \frac{h}{6}[f(x_2) + 4f(x_3) + f(x_4)]$$

$$\int_{x_0}^{x_2} f(x)dx + \int_{x_2}^{x_4} f(x)dx = I(x_0, x_2) + I(x_2, x_4) - \frac{h^5}{16}\frac{f^{(4)}(d_2)}{90}$$

In this equation

$$x_1 = \frac{a+b}{4}$$

$$x_3 = \frac{3(a+b)}{4}$$

d_2 is stil a value in between a and b.

If it is assumed that

$$f^{(4)}(d_1) = f^{(4)}(d_2)$$

Equation becomes

$$h^5 \frac{f^{(4)}(d_1)}{90} \approx \frac{16}{15}[I(x_0, x_2) + I(x_2, x_4) - I(x_0, x_4)]$$

$$\frac{h^5}{16}\frac{f^{(4)}(d_1)}{90} \approx \frac{1}{15}[I(x_0, x_2) + I(x_2, x_4) - I(x_0, x_4)]$$

If we substitute this into integration equation

$$\left| \int_{x_0}^{x_4} f(x)dx - I(x_0, x_2) - I(x_2, x_4) \right| \approx \frac{1}{15}|I(x_0, x_2) + I(x_2, x_4) - I(x_0, x_4)|$$

In this case for error ε>0 , it can be written the equation

$$\frac{1}{15}|I(x_0, x_2) + I(x_2, x_4) - I(x_0, x_4)| < \varepsilon$$

If this condition is met, then approximately

$$\int_{x_0}^{x_4} f(x)dx \cong I(x_0, x_2) + I(x_2, x_4)$$

can be written. If the given condition can not be met, division process will be continue till the condition met. Described algorithm is used in the next program.

Program 7.4-1 Simson 1/3 integration with adjustable error

```
// ================================================
// Numerical Analysis package in java
// example to show utilisation of integration (integral)
// and differentiation (derivative) functions
// Adaptive simpson 1/3 integration
```

```
// Dr. Turhan Coban
// ================================================
import java.io.*;

class f1 extends f_x
{
  double func(double x)
  {return x*Math.sqrt(x);}
}

class NA66
{
public static double adaptive_simpson_integral(f_x ff,double a,double b)
{return adaptive_simpson_integral(ff,a,b,1.0e-20,30);}

public static double adaptive_simpson_integral(f_x ff,double a,double b,double eps)
{return adaptive_simpson_integral(ff,a,b,eps,30);}

public static double adaptive_simpson_integral(f_x ff,double a,double b,int MAX_ITERATION)
{return adaptive_simpson_integral(ff,a,b,1.0e-12,MAX_ITERATION);}

public static double adaptive_simpson_integral(f_x ff,double a,double b,double eps,int MAX_ITERATION)
{   int i;
    double h=(b-a);
    double h2=h/4.0;
            double x[]=new double[5];
    double f[]=new double[5];
    for(i=0;i<5;i+=2) {x[i]=a+i*h2;f[i]=ff.func(x[i]);}
            double sum=h*(f[0]+4.0*f[2]+f[4])/6.0;
            double Q=0;
    Q=simpson1_3(ff,x,f,sum,eps,0,MAX_ITERATION,Q);
    return Q;
}

public static double simpson1_3(f_x ff,double x[],double f[],double sum,double eps,int k,int MAX_ITERATION,double Q)
{       //Newton-cotes integral 3 points
        //adaptive Simpson 1/3 rule
        double h=(x[4]-x[0]);
        double h1=h/2.0;
        double h2=h/4.0;
        int p,i;
        double x1[]=new double[5];
        double f1[]=new double[5];
        double x2[]=new double[5];
        double f2[]=new double[5];
        k++;
        for(i=1;i<5;i+=2) {x[i]=x[0]+i*h2;f[i]=ff.func(x[i]);}
        double sum1=h1*(f[0]+4.0*f[1]+f[2])/6.0;
        double sum2=h1*(f[2]+4.0*f[3]+f[4])/6.0;
        x1[0]=x[0];f1[0]=f[0];
        x1[2]=x[1];f1[2]=f[1];
        x1[4]=x[2];f1[4]=f[2];
        x2[0]=x[2];f2[0]=f[2];
        x2[2]=x[3];f2[2]=f[3];
        x2[4]=x[4];f2[4]=f[4];
        if(1.0/15.0*Math.abs(sum1+sum2-sum)<eps || k>MAX_ITERATION) Q+=sum1+sum2;
        else
```

```
Q+=simpson1_3(ff,x1,f1,sum1,eps,k,MAX_ITERATION,Q)+simpson1_3(ff,x2,f2,sum2,eps,k,MAX_ITERATION,Q);
      return Q;
}

  public static void main(String args[]) throws IOException
  {
  f1 b1=new f1();
  System.out.println("integral of class        f1 : "+adaptive_simpson_integral(b1,0.0,1.0));
}
}
```

Simson 1/3 integration with the adaptive simpson integration

```
---------- Capture Output ----------
> "C:\turhan\java\bin\javaw.exe" NA66
integral of class        f1 : 0.4

> Terminated with exit code 0.
```

If a version of the same program with actual plots of the calculating points are given, automatic adjustment process of the routine can be seen more clearly.

Program 7.4-2 Simson 1/3 integration with adjustable error (version with plot output)

```
// ======================================================
// Numerical Analysis package in java
// example to show utilisation of integration (integral)
// and differentiation (derivative) functions
// Adaptive simpson 1/3 integration with plotting
// Dr. Turhan Coban
// ======================================================
import java.io.*;

class f1 extends f_x
{
  double func(double x)
  {return 100.0/(x*x)*Math.sin(10.0/x); }
}

class NA67
{
public static double adaptive_simpson_integral(f_x ff,double a,double b)
{return adaptive_simpson_integral(ff,a,b,1.0e-20,30);}

public static double adaptive_simpson_integral(f_x ff,double a,double b,double eps)
{return adaptive_simpson_integral(ff,a,b,eps,30);}

public static double adaptive_simpson_integral(f_x ff,double a,double b,int MAX_ITERATION)
{return adaptive_simpson_integral(ff,a,b,1.0e-12,MAX_ITERATION);}

public static double adaptive_simpson_integral(f_x ff,double a,double b,double eps,int MAX_ITERATION)
{   int i;
    double h=(b-a);
    double h2=h/4.0;
            double x[]=new double[5];
    double f[]=new double[5];
```

```
   double w1[]=new double[3];
   double z1[]=new double[3];
   int j;
   for(i=0;i<5;i+=2) {j=i/2;x[i]=a+i*h2;f[i]=ff.func(x[i]);w1[j]=x[i];z1[j]=f[i];}
   double sum=h*(f[0]+4.0*f[2]+f[4])/6.0;
            double Q=0;
            Plot pp=new Plot(ff,a,b,100);
            pp.setPlotType(0,0);//
            pp.addData(w1,z1,28);
            int k=0;
   Q=simpson1_3(ff,x,f,sum,eps,k,MAX_ITERATION,Q,pp);
   pp.plot();
   return Q;
}

public static double simpson1_3(f_x ff,double x[],double f[],double sum,double eps,int k,int MAX_ITERATION,double
Q,Plot pp)
{
      //adaptive Simpson 1/3 rule

      double h=(x[4]-x[0]);
      double h1=h/2.0;
      double h2=h/4.0;
      int p,i;
      double x1[]=new double[5];
      double f1[]=new double[5];
      double x2[]=new double[5];
      double f2[]=new double[5];
      double w1[]=new double[2];
      double z1[]=new double[2];
      int j;
      for(i=1;i<5;i+=2) {j=i/2;x[i]=x[0]+i*h2;w1[j]=x[i];f[i]=ff.func(x[i]);z1[j]=f[i];}
      for(i=0;i<5;i++) {System.out.println(i+" "+x[i]+" "+f[i]);}
      {System.out.println("==================");}
      double sum1=h1*(f[0]+4.0*f[1]+f[2])/6.0;
      double sum2=h1*(f[2]+4.0*f[3]+f[4])/6.0;
      if(1.0/15.0*Math.abs(sum1+sum2-sum)<eps || k>MAX_ITERATION) Q+=sum1+sum2;
      else
      { x1[0]=x[0];f1[0]=f[0];
        x1[2]=x[1];f1[2]=f[1];
        x1[4]=x[2];f1[4]=f[2];
        x2[0]=x[2];f2[0]=f[2];
        x2[2]=x[3];f2[2]=f[3];
        x2[4]=x[4];f2[4]=f[4];
               k++;
               Q+=simpson1_3(ff,x1,f1,sum1,eps,k,MAX_ITERATION,Q,pp)+
          simpson1_3(ff,x2,f2,sum2,eps,k,MAX_ITERATION,Q,pp);
      }
      pp.addData(w1,z1,28);
      return Q;
}

 public static void main(String args[]) throws IOException
 {
 f1 b1=new f1();
 System.out.println("integral of class        f1 : "+adaptive_simpson_integral(b1,1.0,3.0,1.0e-5,2));
 }
```

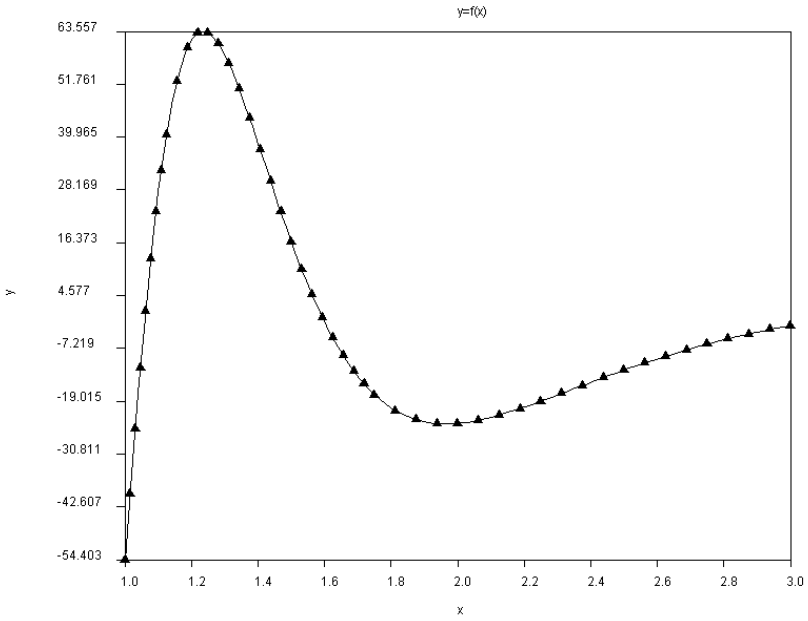
```
}
```

Adaptive Simson 1/3 integration (version with plot output)

```
---------- Capture Output ----------
> "C:\turhan\java\bin\javaw.exe" NA67
0 1.0 -54.40211108893698
1 1.5 16.62894358094311
2 2.0 -23.973106866578462
3 2.5 -12.108839924926851
4 3.0 -2.1174218097276154
==================
0 1.0 -54.40211108893698
1 1.25 63.318927783896434
2 1.5 16.62894358094311
3 1.75 -17.590376763629592
4 2.0 -23.973106866578462
==================
0 1.0 -54.40211108893698
1 1.125 40.34415237327668
2 1.25 63.318927783896434
3 1.375 44.20625835648312
4 1.5 16.62894358094311
==================
0 1.0 -54.40211108893698
1 1.0625 1.1526986941820327
2 1.125 40.34415237327668
3 1.1875 59.814502478779694
4 1.25 63.318927783896434
==================
0 1.25 63.318927783896434
1 1.3125 56.45523901688928
2 1.375 44.20625835648312
3 1.4375 30.17788460030764
4 1.5 16.62894358094311
==================
0 1.5 16.62894358094311
1 1.625 -4.884405926189118
2 1.75 -17.590376763629592
3 1.875 -23.13470269347783
4 2.0 -23.973106866578462
==================
0 1.5 16.62894358094311
1 1.5625 4.773855430676219
2 1.625 -4.884405926189118
3 1.6875 -12.280554478542038
4 1.75 -17.590376763629592
==================
0 1.75 -17.590376763629592
1 1.8125 -21.101511468402688
2 1.875 -23.13470269347783
3 1.9375 -23.999652018301266
4 2.0 -23.973106866578462
==================
0 2.0 -23.973106866578462
1 2.25 -19.048239346741298
2 2.5 -12.108839924926851
```

```
3 2.75 -6.278745267070935
4 3.0 -2.1174218097276154
==================
0 2.0 -23.973106866578462
1 2.125 -22.144859946836526
2 2.25 -19.048239346741298
3 2.375 -15.542391685948601
4 2.5 -12.108839924926851
==================
0 2.0 -23.973106866578462
1 2.0625 -23.29043436266789
2 2.125 -22.144859946836526
3 2.1875 -20.69068321637011
4 2.25 -19.048239346741298
==================
0 2.25 -19.048239346741298
1 2.3125 -17.309323548876932
2 2.375 -15.542391685948601
3 2.4375 -13.79721050171754
4 2.5 -12.108839924926851
==================
0 2.5 -12.108839924926851
1 2.625 -8.988488771999702
2 2.75 -6.278745267070935
3 2.875 -3.9966027013802914
4 3.0 -2.1174218097276154
==================
0 2.5 -12.108839924926851
1 2.5625 -10.500944535981427
2 2.625 -8.988488771999702
3 2.6875 -7.579894531793721
4 2.75 -6.278745267070935
==================
0 2.75 -6.278745267070935
1 2.8125 -5.085116345062419
2 2.875 -3.9966027013802914
3 2.9375 -3.0091045529709124
4 3.0 -2.1174218097276154
==================
integral of class        f1 : -1.416364288717352
```

Graphic output of Adaptive simpson 1/3 integration



7.4.2 Adaptive Gauss-Legendre Integration

The error for the Gauss-Legendre formulas is specified generally by

$$E_t = \frac{2^{2n+3}}{(2n+3)[(2n+2)!]^3} f^{(2n+2)}(\xi)$$

where n = the number of points minus one and $f^{(2n+2)}(\xi)$ = the $(2n + 2)$th derivative of the function after the change of variable with ξ located somewhere on the interval from −1 to 1. It is clear that error level of Gauss-Legendre integration is far smaller than the Newton-Cotes formulas, therefore it will take less iteration to reach the desired error levels. Adjustable error version of the Gauss-Legendre integration is given below. The main idea here is to increase terms in the Gauss-Legendre integration untill the error level between two levels decrease below the desired error level.

Program 7.4-3 Gauss-Legendre integration with the adjustable error

```
// =======================================================
// Numerical Analysis package in java
// example to show utilisation of integration (integral)
// and differentiation (derivative) functions
// Adaptive Gauss-legendre integration
// Dr. Turhan Coban
// =======================================================
import java.io.*;


class f1 extends f_x
{
  public double func(double x)
  {return x*Math.sqrt(x);}
}


class NA68
{
public static double[][] gauss_legendre_coefficients(double x1,double x2,int n)
{
            //calculates legendre gauss-coefficients as coefficients of the integral
            //for n terms
            double EPS=3.0e-15;
```

```
        int m,j,i;
        double z1,z,xm,xl,pp,p3,p2,p1;
         double a[][]=new double[2][n];
        m=(n+1)/2;
        xm=0.5*(x2+x1);
        xl=0.5*(x2-x1);
        for (i=1;i<=m;i++)  {
                z=Math.cos(Math.PI*((i-0.25)/(n+0.5)));
                do {
                        p1=1.0;
                        p2=0.0;
                        for (j=1;j<=n;j++) {
                                p3=p2;
                                p2=p1;
                                p1=((2.0*j-1.0)*z*p2-(j-1.0)*p3)/j;
                        }
                        pp=n*(z*p1-p2)/(z*z-1.0);
                        z1=z;
                        z=z1-p1/pp;
                } while (Math.abs(z-z1) > EPS);
                a[0][i-1]=xm-xl*z;
                a[0][n-i]=xm+xl*z;
                a[1][i-1]=2.0*xl/((1.0-z*z)*pp*pp);
                a[1][n-i]=a[1][i-1];
                        }
return a;
}

public static double gauss_legendre_integral(f_x f,double x1,double x2,int n)
{
//n : number of integral coefficients
// this routine first generates gauss legendre coefficients
// for [x1,x2] band
// then calculates gauss legendre integral
double a[][]=new double[2][n];
a=gauss_legendre_coefficients(x1,x2,n);
//Text.print(Text.T(a));
//System.out.println(Matrix.toString(a));
double z=0;
for(int i=0;i<n;i++)
{ z+=a[1][i]*f.func(a[0][i]);}
return z;
}

public static double adaptive_gauss_legendre_integral(f_x f,double x1,double x2)
{return adaptive_gauss_legendre_integral(f,x1,x2,1.0e-12,100);}

public static double adaptive_gauss_legendre_integral(f_x f,double x1,double x2,double eps)
{return adaptive_gauss_legendre_integral(f,x1,x2,eps,100);}

public static double adaptive_gauss_legendre_integral(f_x f,double x1,double x2,double eps,double MAXITER)
{
int n=1;
double ans1=gauss_legendre_integral(f,x1,x2,n);
double ans2;
do
{ans2=ans1;n++;ans1=gauss_legendre_integral(f,x1,x2,n);}
```

```
while(Math.abs(ans2-ans1)>eps && n<MAXITER);
if(n==MAXITER) System.out.println("istenilen hata seviyesi tutturulamadi n="+n );
return ans2;
}

  public static void main(String args[]) throws IOException
  {
  //Gauss-Legendre integrali
  f1 b1=new f1();
  System.out.println("integral  : "+adaptive_gauss_legendre_integral(b1,0.0,1.0));
}
}
```

Adaptive Gauss-Legendre integration

```
---------- Capture Output ----------
> "C:\co\java\bin\java.exe" integ13
integral  : 0.3999999999829868
> Terminated with exit code 0.
```

7.4.3 Gauss-Kronrod integration formula

$$I_w(-1,1) = \int_{-1}^{1} f(x)dx \cong \sum_{k=1}^{n} c_k f\left(x_k^G\right) + \sum_{k=1}^{n+1} c_k f\left(x_k^K\right)$$

Gauss Kronrod integration formula uses two terms. First term is standard Gauss-Legendre integration term. The second term coefficients are taken as same as the Gauss-Legendre term, but roots are different. It is called Kronrod integration. In our example code constant integration of 15 terms will be used for each equation (total of 30 terms). Program utilises Gauss-Legendre terms as first, and then uses Gauss-Kronrod terms. To improve error reduction equation can be continued with half dividing technique. Gauss-Kronrod Formula is prefferred method for many package programs such as Mathlab.

Program 7.4-4 Gauss-Kronrod integration formula

```
class f1 extends f_x
{
  double func(double x)
  {return x*Math.sqrt(x);}
}
public class NA69
{
   public static double gauss_kronrad_integral(f_x ff,double a,double b,double eps,int n)
   {
              //eps hata miktari
      int Aused=0;
              boolean sonuc;
      double integral=0;
      double A[][];
      int Aw = 0;
      double toplamhata = 0;
      double[] c;
      double[] xg;
      double[] xk;
      int nn = 0;
      int ng = 0;
      int i = 0;
```

```
int j = 0;
int h = 0;
double v = 0;
double k1 = 0;
double k2 = 0;
double intg = 0;
double intk = 0;
double ta = 0;
double tb = 0;
Aw = 4;
A = new double[n-1+1][Aw-1+1];
nn = 61;
ng = 15;
c = new double[nn];
xk = new double[nn];
xg = new double[nn];
//Gauss katsayilari
xg[0] = 0.00796819249616660561546588347 4674;
xg[1] = 0.01846646831109095914230213191 2047;
xg[2] = 0.02878470788332336934971917961 1292;
xg[3] = 0.03879919256962704959680193644 6348;
xg[4] = 0.04840267283059405290293814042 2808;
xg[5] = 0.05749315621761906648172168940 2056;
xg[6] = 0.06597422988218049512812851511 5962;
xg[7] = 0.07375597473770520626824385002 2191;
xg[8] = 0.08075589522942021535469493846 0530;
xg[9] = 0.08689978720108297980238753071 5126;
xg[10] = 0.09212252223778612871763270708 7619;
xg[11] = 0.09636873717464425963946862635 1810;
xg[12] = 0.09959342058679526706278028210 3569;
xg[13] = 0.10176238974840550459642895216 8554;
xg[14] = 0.10285265289355884034128563670 5415;
c[0] = 0.99948441005049063757132589570 5811;
c[1] = 0.99689348407464954027163005091 8695;
c[2] = 0.99163099687040459485862836610 9486;
c[3] = 0.98366812327974720997003258160 5663;
c[4] = 0.97311632250112626837469386842 3707;
c[5] = 0.96002186496830751221687102558 1798;
c[6] = 0.94437444474855997941583132403 7439;
c[7] = 0.92620004742927432587932427708 0474;
c[8] = 0.90557330769990779854652255892 5958;
c[9] = 0.88256053579205268154311646253 0226;
c[10] = 0.85720523354606109895865851065 8944;
c[11] = 0.82956576238276839744289811973 2502;
c[12] = 0.79972783582183908301366894232 2683;
c[13] = 0.76777743210482619491797734097 4503;
c[14] = 0.73379006245322680472617113136 9528;
c[15] = 0.69785049479331579693229238802 6640;
c[16] = 0.66006106412662696137005366814 9271;
c[17] = 0.62052618298924286114047755643 1189;
c[18] = 0.57934523582636169175602493217 2540;
c[19] = 0.53662414814201989926416979331 1073;
c[20] = 0.49248046786177857499369306120 7709;
c[21] = 0.44703376953808917678060990032 2854;
c[22] = 0.40040125483039439253547621154 2661;
c[23] = 0.35270472553087811347103720708 9374;
c[24] = 0.30407320227362507737267710719 9257;
```

```
c[25] = 0.25463692616788984643980512981780 5;
c[26] = 0.20452511668230989143895767100202 5;
c[27] = 0.15386991360858354696379467274325 6;
c[28] = 0.10280693796673703014709675131800 1;
c[29] = 0.05147184255531769583302521316672 3;
c[30] = 0.00000000000000000000000000000000 0;
//Kronrod katsayilari
xk[0] = 0.00138901369867700762455159122676 0;
xk[1] = 0.00389046112709988405126720184451 6;
xk[2] = 0.00663070391593129217331982636975 0;
xk[3] = 0.00927327965951776342844114689202 4;
xk[4] = 0.01182301525349634174223289885325 1;
xk[5] = 0.01436972950704580481245143244358 0;
xk[6] = 0.01692088918905327262757228942032 2;
xk[7] = 0.01941414119394238117340895105012 8;
xk[8] = 0.02182803582160919229716748573833 9;
xk[9] = 0.02419116207808060136568637072523 2;
xk[10] = 0.02650995488233310161060170933507 5;
xk[11] = 0.02875404876504129284397878535433 4;
xk[12] = 0.03090725756238776247288425294309 2;
xk[13] = 0.03298144705748372603181419101685 4;
xk[14] = 0.03497933802806002413749967073146 8;
xk[15] = 0.03688236465182122922391106561713 6;
xk[16] = 0.03867894562472759295034865153228 1;
xk[17] = 0.04037453895153595911199527975246 8;
xk[18] = 0.04196981021516424614714754128597 0;
xk[19] = 0.04345253970135606931683172811707 3;
xk[20] = 0.04481480013316266319235555161672 3;
xk[21] = 0.04605923827100698811627173555937 4;
xk[22] = 0.04718554656929915394526147818109 9;
xk[23] = 0.04818586175708712914077949229830 5;
xk[24] = 0.04905543455502977888752816536723 8;
xk[25] = 0.04979568342707420635781156937994 2;
xk[26] = 0.05040592140278234684089308565358 5;
xk[27] = 0.05088179589874960649229747304980 5;
xk[28] = 0.05122154784925877217065628260494 4;
xk[29] = 0.05142612853745902593386287921578 1;
xk[30] = 0.05149472942945156755834043364709 9;
for(i=nn-1; i>=nn/2; i--)
{c[i] = -c[nn-1-i];}
for(i=nn-1; i>=nn/2; i--)
{ xk[i] = xk[nn-1-i]; }
for(i=ng-1; i>=0; i--)
{ xg[nn-2-2*i] = xg[i];
  xg[1+2*i] = xg[i];
}
for(i=0; i<=nn/2; i++)
{ xg[2*i] = 0; }
k1 = 0.5*(b-a);
k2 = 0.5*(b+a);
intg = 0;
intk = 0;
for(i=0; i<=nn-1; i++)
{ v = ff.func(k1*c[i]+k2);
  intk = intk+v*xk[i];
  if( i%2==1 )
  {intg = intg+v*xg[i];}
```

```
    }
    intk = intk*(b-a)*0.5;
    intg = intg*(b-a)*0.5;
    A[0][0] = Math.abs(intg-intk);
    A[0][1] = intk;
    A[0][2] = a;
    A[0][3] = b;
    toplamhata = A[0][0];
    if( toplamhata<eps )
    {
       sonuc = true;
       integral = intk;
       Aused = 1;
       return integral;
    }
    Aused = 1;
    for(h=1; h<=n-1; h++)
    {
       Aused = h+1;
       gir(A, h, Aw);
       toplamhata = toplamhata-A[h-1][0];
       ta = A[h-1][2];
       tb = A[h-1][3];
       A[h-1][2] = ta;
       A[h-1][3] = 0.5*(ta+tb);
       A[h][2] = 0.5*(ta+tb);
       A[h][3] = tb;
       for(j=h-1; j<=h; j++)
       {
          k1 = 0.5*(A[j][3]-A[j][2]);
          k2 = 0.5*(A[j][3]+A[j][2]);
          intg = 0;
          intk = 0;
          for(i=0; i<=nn-1; i++)
          {   v = ff.func(k1*c[i]+k2);
             intk = intk+v*xk[i];
             if( i%2==1 )
             { intg = intg+v*xg[i];}
          }
          intk = intk*(A[j][3]-A[j][2])*0.5;
          intg = intg*(A[j][3]-A[j][2])*0.5;
          A[j][0] = Math.abs(intg-intk);
          A[j][1] = intk;
          toplamhata = toplamhata+A[j][0];
       }
       cik(A, h-1, Aw);
       cik(A, h, Aw);
       if( toplamhata<eps )
       {break;}
    }
    sonuc = toplamhata<eps;
    integral = 0;
    for(j=0; j<=Aused-1; j++)
    { integral = integral+A[j][1];}
    return integral;
 }
```

```
public static void gir(double A[][],int n,int Awidth)
{
   int i = 0;
   int p = 0;
   double t = 0;
   int maxcp = 0;
   if( n==1 )
   { return; }
   for(i=0; i<=Awidth-1; i++)
   {   t = A[n-1][i];
      A[n-1][i] = A[0][i];
      A[0][i] = t;
   }//end of for
   p = 0;
   while( 2*p+1<n-1 )
   {   maxcp = 2*p+1;
      if( 2*p+2<n-1 )
      {   if( A[2*p+2][0]>A[2*p+1][0] )
         { maxcp = 2*p+2; }
      }
      if( A[p][0]<A[maxcp][0] )
      {   for(i=0; i<=Awidth-1; i++)
         {   t = A[p][i];
            A[p][i] = A[maxcp][i];
            A[maxcp][i] = t;
         }
         p = maxcp;
      }
      else
      { break; }
   } //end of while
}

public static void cik(double A[][],int n,int Awidth)
{
   int i = 0;
   int p = 0;
   double t = 0;
   int kk = 0;
   if( n==0 )
   {return;}
   p = n;
   while( p!=0 )
   {   kk = (p-1)/2;
      if( A[p][0]>A[kk][0] )
      { for(i=0; i<=Awidth-1; i++)
         { t = A[p][i];
           A[p][i] = A[kk][i];
           A[kk][i] = t;
         }
         p = kk;
      }
      else
      {break;}
   }
}
```

```
  public static void main(String axgs[])
  {
  f1 b1=new f1();
  System.out.println("integral of class         f1 : "+gauss_konrad_integral(b1,0.0,1.0,1.0e-20,50));
}
}
```

```
---------- Capture Output ----------
> "C:\co\java\bin\javaw.exe" adaptive_Gauss_Kronrod_integral
integral of class         f1 : 0.4

> Terminated with exit code 0.
```

7.4.4 Adaptive Newton-Cotes integration with Gander ve Gautschi[31] approach

In the previous adaptive integration approaches program follows a stopping criteria such as

if(Math.abs(I1-I2)< tolerance or

if(Math.abs((I1-I2)/I1) < tolerance

This type of stopping criteria might be unsuccessfull for some cases, and Maximum iteration error will be received. If we would like to further decrease the error and reach the machine error level a different type of error control can be used. If an error control such as

if(is+(I1-I2)==is)

is used, this control works only when I1 is equal to I2 with the machine accuracy. If this definition to be used with the halving approach

$$m = \frac{(a-b)}{2}$$

with an additional control definition can even give a better exit criteria

if((is+(I1-I2)==is) || (m<=a) || (b<=m))

Now this criteria will be applied to Newton-Cotes formulas. In this algorithm Newton Cotes Formula with 8th degree polynomial and quadratic polinomial (Simpson 1/3) will be used. 8th degree Newton cotes Formula :

$$h = \frac{(b-a)}{8}$$

$$\boldsymbol{I = \int_a^b f(x)dx = \frac{4h}{14175}[989f(x_0) + 5888f(x_1) - 928f(x_2) + 10496f(x_3) - 4540f(x_4) + 10496f(x_3)f(x_5) - 928f(x_6) + 5888f(x_7) + 989f(x_8)]}$$

Simpson 1/3integral in iterative form will be:

$$\boldsymbol{I_1 = \int_a^b f(x)dx = I[a,b] = \frac{(b-a)}{6}[f(x_0) + 4f(x_2) + f(x_4)]}$$

$$I_2 = \int_a^b f(x)dx = I[a,m] + I[m,b] = \frac{(m-a)}{6}[f(x_0) + 4f(x_1) + f(x_2)]$$
$$+ \frac{(b-m)}{6}[f(x_2) + 4f(x_3) + f(x_4)]$$
$$= \frac{(b-a)}{12}[f(x_0) + 4f(x_1) + 2f(x_2) + 4f(x_3) + f(x_4)]$$

Program 7.4-5 Gander and Gautschi adaptive Simpson 1/3 integral formulation

```
//=======================================================
// Numerical Analysis package in java
// example to show utilisation of integration (integral)
// and differentiation (derivative) functions
// Adaptive Gander and Gautschi simpson 1/3 integration
// Dr. Turhan Coban
// =======================================================
import java.io.*;

class f1 extends f_x
{
  double func(double x)
  {return Math.sin(x);}
}

class NA70
{
public static double adaptive_simpson_integral2(f_x ff,double a,double b)
{   int i;
              Plot pp=new Plot(ff,a,b,100);
    double h=(b-a);
    double h2=h/4.0;
    double h3=h/8.0;
    double m=(a+b)/2.0;
    double x[]=new double[9];
    double y[]=new double[9];
    for(i=0;i<9;i++) {x[i]=a+i*h3;y[i]=ff.func(x[i]);}
    double is=h/28350.0*(989.0*y[0]+5888.0*y[1]-928.0*y[2]+10496.0*y[3]-4540.0*y[4]+
         10496.0*y[5]-928.0*y[6]+5888.0*y[7]+989.0*y[8]);
    double fa = y[0];
    double fm = y[4];
    double fb = y[8];
    if(is==0) is = b-a;
    double Q = adaptsimstp(ff,a,b,fa,fm,fb,is);
    return Q;
}

public static double adaptsimstp(f_x ff,double a,double b,double fa,double fm,double fb,double is)
{
  double m = (a + b)/2;
  double h = (b - a)/4;
  double x[] = {a + h, b - h};
  int n=x.length;
  double y[] = new double[n];
  for(int i=0;i<n;i++) {y[i]=ff.func(x[i]);}
  double fml = y[0];
```

```
  double fmr = y[1];
  double i1 = h/1.5 * (fa + 4*fm + fb);
  double i2 = h/3 * (fa + 4*(fml + fmr) + 2*fm + fb);
  double Q=0;
  i1 = (16.0*i2 - i1)/15.0; //romberg integrasyonu ile iyileştirme
  if (((is + (i1-i2)) == is) || (m <= a) || (b<=m))
  {
    if ((m <= a) || (b<=m))
       { System.out.println("istenilen toleransa ulaşilamadi");}
    Q = i1;
  }
  else
  {
    Q = adaptsimstp(ff,a,m,fa,fml,fm,is)+
        adaptsimstp(ff,m,b,fm,fmr,fb,is);
  }
  return Q;
}

  public static void main(String args[]) throws IOException
  {
  f1 b1=new f1();
  System.out.println("integral of class         f1 : "+adaptive_simpson_integral2(b1,0.0,Math.PI));
  }
}
```

Gander and Gautschi adaptive Simpson 1/3 integral Formula with sample integral

$$\boldsymbol{I = \int_{0}^{\pi} sin(x)dx = 2}$$

```
---------- Capture Output ----------
> "C:\turhan\java\bin\javaw.exe" NA70
integral of class         f1 : 2.0

> Terminated with exit code 0.
```

7.4-5 Adaptive Gauss-Lobatto integral with Gander ve Gautschi[31] approach

Gauss-Lobatto integration Formula is similar to Gauss integration Formula. The most important difference is that Lobatto Formula also covers end points

$$I_W(-1,1) = \int_{-1}^{1} f(x)dx \cong \sum_{k=0}^{n-1} c_k f(x_k)$$

For various n values Gauss-Lobatto coefficients and roots are given

Table 7.4-1 Gauss-Lobatto integration equation coefficients and roots

N	Xk	Ck
4	-1.000000000000000000000000	0.166666666666667000000000
	1.000000000000000000000000	0.166666666666667000000000
	-0.447213595499958000000000	0.833333333333333000000000
	0.447213595499958000000000	0.833333333333333000000000
5	-1.000000000000000000000000	0.100000000000000000000000
	1.000000000000000000000000	0.100000000000000000000000
	-0.654653670707977000000000	0.544444444444440000000000
	0.654653670707977000000000	0.544444444444440000000000
	0.000000000000000000000000	0.711111111111111000000000

6	-1.00000000000000000000000	0.06666666666666660000000
	1.00000000000000000000000	0.06666666666666660000000
	-0.76505532392946500000000	0.37847495629784700000000
	0.76505532392946500000000	0.37847495629784700000000
	-0.28523151648063500000000	0.55485837703548600000000
	0.28523151648063500000000	0.55485837703548600000000
7	-1.00000000000000000000000	0.05238095238095240000000
	1.00000000000000000000000	0.05238095238095240000000
	-0.81649658092772600000000	0.29387755102040800000000
	0.81649658092772600000000	0.29387755102040800000000
	-0.44721359549995800000000	0.42517006802721100000000
	-0.44721359549995800000000	0.42517006802721100000000
	0.00000000000000000000000	0.45714285714285700000000
12	-1.00000000000000000000000	0.01582719197348010000000
	1.00000000000000000000000	0.01582719197348010000000
	-0.94288241569547900000000	0.09427384021885000000000
	0.94288241569547900000000	0.09427384021885000000000
	-0.81649658092772600000000	0.15507198733658500000000
	0.81649658092772600000000	0.15507198733658500000000
	-0.64185334234578100000000	0.18882157396018200000000
	0.64185334234578100000000	0.18882157396018200000000
	-0.44721359549995800000000	0.19977340522685800000000
	0.44721359549995800000000	0.19977340522685800000000
	-0.23638319966214900000000	0.22492646533333900000000
	0.23638319966214900000000	0.22492646533333900000000
	0.00000000000000000000000	0.24261107190140700000000

As the previous simpson 1/3 approach **Gander ve Gautschi[31]** approach will be used as a stopping criteria in this program also. For the interval [a,b]

if((is+(I1 - I2) == is) || (m<=a) || (b<=m))

In this condition m is given as

$$m = \frac{(a-b)}{2}$$

For the is first integration value degree 12 Lobatto integration will be used.

As iteration Formula 4th and 7th degree Gauss-Lobatto formulations will be used.

Program code for this approach is given below.

Program 7.4-5 Gander and Gautschi adaptive Gauss-Lobatto integral formula

```
//=====================================================
// Numerical Analysis package in java
// example to show utilisation of integration (integral)
// and differentiation (derivative) functions
// Gander and Gautschi adaptive Gauss-Lobatto integral
// Dr. Turhan Coban
// =====================================================
import java.io.*;
class f1 extends f_x
{
  double func(double x)
  {//return 13.0*(x-x*x)*Math.exp(-1.5*x);
  return Math.sin(x);
  }
}

class NA71
{

public static double adaptive_Lobatto_integral(f_x ff,double a,double b)
```

```
{   int i;
            double m=(a+b)/2.0; double h=(b-a)/2.0;
   double alpha=Math.sqrt(2.0/3.0);
   double beta=1.0/Math.sqrt(5.0);
   double x1,x2,x3;
   x1=0.94288241569547971905635175843185720232;
   x2=0.64185334234578130578123554132903188354;
   x3=0.23638319966214988028222377349205292599;
   double A,B,C,D,E,F,G;
   A=0.01582719197348018308716998733305510591;
   B=0.09427384021885004553128250507710817196O;
   C=0.15507198733658539625363597980210298680;
   D=0.18882157396018245442000533937297167125;
   E=0.19977340522685852679206802206648840246;
   F=0.22492646533333952701601768799639508076;
   G=0.24261107190140773379964095790325635233;
            double x[]={a,(m-x1*h),(m-alpha*h),(m-x2*h),(m-beta*h),(m-
x3*h),m,(m+x3*h),(m+beta*h),(m+x2*h),(m+alpha*h),(m+x1*h),b};
            int n=x.length;
            double y[]=new double[n];
            for(i=0;i<n;i++) y[i]=ff.func(x[i]);
            double fa=y[0];
            double fb=y[12];
   double i2=(h/6.0)*(y[0]+y[12]+5.0*(y[4]+y[8]));
   double i1=(h/1470.0)*(77.0*(y[0]+y[12])+432.0*(y[2]+y[10])+
    625.0*(y[4]+y[8])+672.0*y[6]);
   double is=h*(A*(y[0]+y[12])+B*(y[1]+y[11])+C*(y[2]+y[10])+D*(y[3]+y[9])+E*(y[4]+y[8])+F*(y[5]+y[7])+G*y[6]);
   int s=(int)(is/Math.abs(is));
   if(s==0) s=1;
   double erri1=Math.abs(i1-is);
   double erri2=Math.abs(i2-is);
   double R;
   R=erri1/erri2;
   //if(R>0 && R<1) tol=tol/R;
   //is=s*Math.abs(is)*tol/eps;
   if(is==0) is=b-a;
  double Q=0;
  Q=Lobatto(ff,a,b,fa,fb,is);
  return Q;
}

public static double Lobatto(f_x ff,double a,double b,double fa,double fb,double is)
{
  double h=(b-a)/2;
  double m=(a+b)/2;
  double alpha=Math.sqrt(2.0/3.0);
  double beta=1.0/Math.sqrt(5.0);
  double mll=m-alpha*h;
  double ml=m-beta*h;
  double mr=m+beta*h;
  double mrr=m+alpha*h;
  double x[]={mll,ml,m,mr,mrr};
  int n=x.length;
  double y[]=new double[n];
  for(int i=0;i<n;i++) y[i]=ff.func(x[i]);
  double fmll=y[0];
  double fml=y[1];
```

```
 double fm=y[2];
 double fmr=y[3];
 double fmrr=y[4];
 double Q;
 double i2=(h/6)*(fa+fb+5*(fml+fmr));
 double i1=(h/1470)*(77*(fa+fb)+432*(fmll+fmrr)+625*(fml+fmr)+672*fm);
 boolean c1,c2,c3;
 double w1=i1-i2;;
 double w2=is;
 double w3=w2+w1;
 c1=(w3==w2);
 c2=(mll<=a);
 c3=(b<=mrr);
 if( c1|| c2 ||c3 )
 {
  if (((m <= a) || (b<=m)))
  {
     System.out.println("istenilen toleransa ulaşilamadi");
  }
  Q=i1;
 }
 else
 {
   Q=Lobatto(ff,a,mll,fa,fmll,is) + Lobatto(ff,mll,ml,fmll,fml,is)+
     Lobatto(ff,ml,m,fml,fm,is)   + Lobatto(ff,m,mr,fm,fmr,is)+
     Lobatto(ff,mr,mrr,fmr,fmrr,is)+Lobatto(ff,mrr,b,fmrr,fb,is);
 }
 return Q;
}

 public static void main(String args[]) throws IOException
 {
 f1 b1=new f1();
 System.out.println("integral of class        f1 : "+adaptive_Lobatto_integral(b1,0.0,Math.PI));
 }
}
```

Gander and Gautschi adaptive Gauss-Lobatto integral formula

```
---------- Capture Output ----------
> "C:\turhan\java\bin\javaw.exe" NA71
integral of class        f1 : 2.0

> Terminated with exit code 0.
```

In order to see how successfull the code will be a mathematical function test will be carried out as the next example. The definition of the error function :

$$\text{erf}(\eta) = \frac{1}{\sqrt{\pi}} \int_0^{\eta} e^{-\eta^2} d\eta$$

Because of the difficulty of calculating this equation with integration Formula, it is generally calculated by serial solutions. In our example both serial solution and Gauss-Lobatto adaptive integration solution will be carried out and compared.

Program 7.4-6 Calculation of error function with Gander and Gautschi adaptive Gauss-Lobatto integral formula

```
//=================================================
// Numerical Analysis package in java
// error function calculation by using
// Adaptive Gauss-Lobatto integral and serial solution
// Referance : Adapted Quadrature - Revisited
// Walter Gander - Walter Gautchi
// ftp.inf.ethz.ch
// Dr. Turhan Coban
// =================================================

import java.io.*;
import javax.swing.*;

class f1 extends f_x
{ double func(double x)
  {return 2.0/Math.sqrt(Math.PI)*Math.exp(-x*x);}
}

class NA72
{

public static double gammp(double a,double x)
{
        //incomplete gamma function P(a,x)
        double gamser,gammcf;
        if (x < 0.0 || a <= 0.0) System.out.println("Invalid arguments in routine GAMMP");
        if (x < (a+1.0)) {
                gamser=gser(a,x);
                return gamser;
        } else {
                gammcf=gcf(a,x);
                return 1.0-gammcf;
        }
}
public static double gser(double a,double x)
{
        int ITMAX=100;
        double EPS=3.0e-7;
        int n;
        double sum,del,ap;
        double gln=gammln(a);
        double gamser=0;
        if (x <= 0.0) {
                if (x < 0.0) System.out.println("x less than 0 in routine GSER");
                gamser=0.0;
                return gamser;
        } else {
                ap=a;
                del=sum=1.0/a;
                for (n=1;n<=ITMAX;n++) {
                        ap += 1.0;
                        del *= x/ap;
                        sum += del;
```

```
                        if (Math.abs(del) < Math.abs(sum)*EPS) {
                                gamser=sum*Math.exp(-x+a*Math.log(x)-(gln));
                                return gamser;
                        }
                }
                System.out.println("a too large, ITMAX too small in routine GSER");
                return gamser;
        }
}

public static double gcf(double a,double x)
{
        int ITMAX=100;
        double EPS=3.0e-7;
        int n;
        double gln;
        double gammcf=0;
        double gold=0.0,g,fac=1.0,b1=1.0;
        double b0=0.0,anf,ana,an,a1,a0=1.0;
        gln=gammln(a);
        a1=x;
        for (n=1;n<=ITMAX;n++) {
                an=(double) n;
                ana=an-a;
                a0=(a1+a0*ana)*fac;
                b0=(b1+b0*ana)*fac;
                anf=an*fac;
                a1=x*a0+anf*a1;
                b1=x*b0+anf*b1;
                if (a1!=0) {
                        fac=1.0/a1;
                        g=b1*fac;
                        if (Math.abs((g-gold)/g) < EPS) {
                                gammcf=Math.exp(-x+a*Math.log(x)-gln)*g;
                                return gammcf;
                        }
                        gold=g;
                }
        }
        System.out.println("a too large, ITMAX too small in routine GCF");
   return gammcf;
}
public static double gammln(double xx)
{
        //gamma function
        double x,y,tmp,ser;
        double cof[]={76.18009172947146,-86.50532032941677,
                24.01409824083091,-1.231739572450155,
                0.1208650973866179e-2,-0.5395239384953e-5};
        int j;
        y=x=xx;
        tmp=x+5.5;
        tmp -= (x+0.5)*Math.log(tmp);
        ser=1.000000000190015;
        for (j=0;j<=5;j++) ser += cof[j]/++y;
        return -tmp+Math.log(2.5066282746310005*ser/x);
}
```

```
public static double  erf(double x)
{
  //error function from series solution
  return x < 0.0 ? -gammp(0.5,x*x) : gammp(0.5,x*x);
}

public static double  erf_int(double x)
{
  //error function from series solution
  f1 b1=new f1();
  return adaptive_Lobatto_integral(b1,0.0,x);
}


public static double adaptive_Lobatto_integral(f_x ff,double a,double b)
{double tol=Double.MIN_VALUE;
 return adaptive_Lobatto_integral(ff,a,b,tol);
}

public static double adaptive_Lobatto_integral(f_x ff,double a,double b,double tol)
{   int i;
            double m=(a+b)/2.0; double h=(b-a)/2.0;
   double alpha=Math.sqrt(2.0/3.0);
   double beta=1.0/Math.sqrt(5.0);
   double x1,x2,x3;
   double eps =Double.MIN_VALUE;
   //System.out.println("eps="+eps);
   x1=0.942882415695479719056351758431857202 32;
   x2=0.641853342345781305781235541329031883 54;
   x3=0.236383199662149880282223773492052925 99;
   double A,B,C,D,E,F,G;
   A=0.015827191973480183087169986733305510591;
   B=0.094273840218850045531282505077108171960;
   C=0.155071987336585396253635979802102986 80;
   D=0.188821573960182454420005339372971671 25;
   E=0.199773405226858526792068022066488402 46;
   F=0.224926465333339527016017687996395080 76;
   G=0.242611071901407733799640957903256352 33;
            double x[]={a,(m-x1*h),(m-alpha*h),(m-x2*h),(m-beta*h),(m-
x3*h),m,(m+x3*h),(m+beta*h),(m+x2*h),(m+alpha*h),(m+x1*h),b};
            int n=x.length;
            double y[]=new double[n];
            for(i=0;i<n;i++) y[i]=ff.func(x[i]);
            double fa=y[0];
            double fb=y[12];
   double i2=(h/6.0)*(y[0]+y[12]+5.0*(y[4]+y[8]));
   double i1=(h/1470.0)*(77.0*(y[0]+y[12])+432.0*(y[2]+y[10])+
    625.0*(y[4]+y[8])+672.0*y[6]);
   double is=h*(A*(y[0]+y[12])+B*(y[1]+y[11])+C*(y[2]+y[10])+D*(y[3]+y[9])+E*(y[4]+y[8])+F*(y[5]+y[7])+G*y[6]);
   int s=(int)(is/Math.abs(is));
   if(s==0) s=1;
   double erri1=Math.abs(i1-is);
   double erri2=Math.abs(i2-is);
   double R;
   R=erri1/erri2;
   if(R>0 && R<1) tol=tol/R;
   //System.out.println("is="+is+"s="+s+"tol="+tol+"eps="+eps);
```

```
    if(is==0) is=b-a;
  double Q=0;
  Q=Lobatto(ff,a,b,fa,fb,is,tol);
  return Q;
}

public static double Lobatto(f_x ff,double a,double b,double fa,double fb,double is,double tol)
{
  double h=(b-a)/2;
  double m=(a+b)/2;
  double alpha=Math.sqrt(2.0/3.0);
  double beta=1.0/Math.sqrt(5.0);
  double mll=m-alpha*h;
  double ml=m-beta*h;
  double mr=m+beta*h;
  double mrr=m+alpha*h;
  double x[]={mll,ml,m,mr,mrr};
  int n=x.length;
  double y[]=new double[n];
  for(int i=0;i<n;i++) y[i]=ff.func(x[i]);
  double fmll=y[0];
  double fml=y[1];
  double fm=y[2];
  double fmr=y[3];
  double fmrr=y[4];
  double Q;
  double i2=(h/6)*(fa+fb+5*(fml+fmr));
  double i1=(h/1470)*(77*(fa+fb)+432*(fmll+fmrr)+625*(fml+fmr)+672*fm);
  double di=Math.abs(i1-i2);
  if (((is + di) == is) || (mll <= a) || (b<=mrr) || di<tol)
  {
   if (((m <= a) || (b<=m)))
   { System.out.println("istenilen toleransa ulaşilamadi delta i = "+di);}
   Q=i1;
  }
  else
  {
    Q=Lobatto(ff,a,mll,fa,fmll,is,tol) + Lobatto(ff,mll,ml,fmll,fml,is,tol)+
      Lobatto(ff,ml,m,fml,fm,is,tol)   + Lobatto(ff,m,mr,fm,fmr,is,tol)+
      Lobatto(ff,mr,mrr,fmr,fmrr,is,tol)+Lobatto(ff,mrr,b,fmrr,fb,is,tol);
  }
  return Q;
}

  public static void main(String args[]) throws IOException
  {
  double x=Double.parseDouble(JOptionPane.showInputDialog(" x = "));
  System.out.println("erf("+x+") = "+erf(x)+"integral erf="+erf_int(x));
  }
}
```

Calculation of error function with Gander and Gautschi adaptive Gauss-Lobatto integral Formula output

```
erf(0.1) = 0.1124629159993309  integral erf=0.1124629160182849
erf(0.4) = 0.4283923519241598  integral erf=0.42839235504666845
erf(1.0) = 0.8427007900291826  integral erf=0.8427007929497149
```

erf(2.0) = 0.995322264754383 integral erf=0.9953222650189527

7.5 CLENSHAW-CURTIS INTEGRATION

Clenshaw-Curtis formulation is based on $x = \cos(\theta)$ transformation. When this transformation applies, in the range of -1 to 1 integration equation becomes:

$$\int_{-1}^{1} f(x)dx = \int_{0}^{\pi} f(\cos(\theta))d(\cos(\theta)) = \int_{0}^{\pi} f(\cos(\theta))\sin(\theta)\,d\theta$$

Form. This integration can be opened as a series as:

$$f(\cos(\theta)) = \frac{a_0}{2} + \sum_{k=1}^{\infty} a_k \cos(k\theta)$$

By applying this transformation integral can be carried out as:

$$\int_{0}^{\pi} f(\cos(\theta))\sin(\theta)\,d\theta = a_0 + \sum_{k=1}^{\infty} \frac{2a_{2k}}{1-(2k)^2}$$

Where

$$a_k = \frac{2}{k}\int_{0}^{\pi} f(\cos(\theta))\cos(k\theta)\,d\theta$$

This integration cam also open to series as:

$$a_k = \frac{2}{N}\left[\frac{f(1)}{2} + \frac{f(-1)}{2}(-1)^k + \sum_{n=1}^{N-1} f(cos\left(\frac{n\pi}{N}\right)cos\left(\frac{nk\pi}{N}\right)\right]$$

When the actual integral is in between limits of x_1 and x_2 values, coordinate conversion should be apply

$$k_2 = \frac{x_2 - x_1}{2}$$

$$k_1 = \frac{x_2 + x_1}{2}$$

$$\int_{a}^{b} f(x)dx = k_1\left[a_0 + \sum_{k=1}^{\infty}\frac{2a_{2k}}{1-(2k)^2}\right]$$

$$x = k_2 + k_1 cos\left(\frac{n\pi}{N}\right)$$

$$a_k = \frac{2}{N}\left[\frac{f(x_1)}{2} + \frac{f(x_2)}{2}(-1)^k + \sum_{n=1}^{N-1} f(x)cos\left(\frac{nk\pi}{N}\right)\right]$$

Program 7.5-1 Clenshaw-Curtis integrasyonu

```
//=================================================
// Numerical Analysis package in java
// example to show utilisation of integration (integral)
// and differentiation (derivative) functions
// Dr. Turhan Coban
// =================================================
import java.io.*;


class f1 extends f_x
{
  public double func(double x)
  {return x*x;}
}
```

```
class integ12
{
public static double[] clenshaw_curtis_coefficients(f_x f,double x1,double x2,int N)
{   double a[]=new double[2*N+1];
              int cc=-1;
              int n,k;
              double x;
              double k1=(x2-x1)/2.0;
   double k2=(x2+x1)/2.0;
              for(k=0;k<2*N;k++)
              {
              a[k]=0.0;
              for (n=1;n<N;n++)
              {x=k2+k1*Math.cos(Math.PI*n/N);
               a[k]+=f.func(x)*Math.cos(n*k*Math.PI/N);}
    cc*=-1;
               a[k]+=f.func(x2)/2.0+f.func(x1)/2.0*cc;
               a[k]*=2.0/n;
   }
   return a;
}
public static double Aitken_clenshaw_curtis(f_x ff,double a,double b, int n)
{   //Aitken interpolation formula
  double I8n=clenshaw_curtis_integral(ff,a,b,8*n);
   double I4n=clenshaw_curtis_integral(ff,a,b,4*n);
   double I2n=clenshaw_curtis_integral(ff,a,b,2*n);
   double In=clenshaw_curtis_integral(ff,a,b,n);
   double I=I4n-(I4n-I2n)*(I4n-I2n)/((I4n-I2n)-(I2n-In));
   return I;
}

public static double clenshaw_curtis_integral(f_x f,double x1,double x2,int N)
{
double a[]=clenshaw_curtis_coefficients(f,x1,x2,N);
double z=a[0];
double k1=(x2-x1)/2.0;
double k2=(x2+x1)/2.0;
for(int k=1;k<N;k+=2)
{ z+=2.0*a[2*k]/(1.0-4.0*k*k);}
return k1*z;
}
  public static void main(String args[]) throws IOException
  {
  //Gauss-Legendre integrali
  f1 b1=new f1();
  double I1=clenshaw_curtis_integral(b1,0.0,1.0,100);
  double I2=Aitken_clenshaw_curtis(b1,0.0,1.0,100);
  System.out.println("integral  : "+I1+"\nAitken interpolation corrrection = "+I2);

}}
```

```
---------- Capture Output ----------
> "E:\co\java\bin\javaw.exe" integ12
integral  : 0.3333301448018433
Aitken interpolation corrrection = 0.33333333158085837

> Terminated with exit code 0.
```

7.6 ONE AND MULTI-DIMENSIONAL MONTE-CARLO INTEGRAL

Monte Carlo integral is based on a very simple principle

$$\int_a^b f(x)dx = (b-a)f_{average}$$

$$x = a + (b-a)random_number(0,1)$$

$$f_{average} = \frac{\sum_{i=0}^{n} f(x)}{n} \qquad n \gg 1$$

In the interval [a,b] a random x number is selected and function is evalueated in this point. When this sequaence is repeated for a very large number of times avarage value of the function can be calculated and directly used in the calculatetions of the integral. Integral is the calculateion of the avarage function value with the interval width. Due to fact that this method requires a huge amount of function evaluation, therefore it is not very practical for one variable integration, but in multivariable form it becomes a practical, sometimes only available approach.

Program 7.6-1 One variable Monte Carlo integration

```
//=======================================================
// Numerical Analysis package in java
// example to show utilisation of integration (integral)
// and differentiation (derivative) functions
// Monte Carlo integration with one variable
// Dr. Turhan Coban
// =======================================================

import java.io.*;

class f1 extends f_x
{
  public double func(double x)
  {return 3.0*x*x;}
}

class NA73
{

public static double monte_carlo(f_x ff,double a,double b, long n)
{       //Monte-carlo integrali
        double h=(b-a);
        double sum=0.0;
        double x=0.0;
        double f;
        for(int i=0;i<n;i++)
        { x=a+(b-a)*Math.random();
          sum+=ff.func(x);
        }
        return (b-a)*sum/(double)n;
}

  public static void main(String args[]) throws IOException
  {
  f1 b1=new f1();
  System.out.println("integral of class         f1 : "+monte_carlo(b1,0.0,1.0,1000000));
}}
```

One variable Monte-carlo integration

```
---------- Capture Output ----------
> "C:\turhan\java\bin\javaw.exe" NA73
integral of class         f1 : 0.9990266020806676
```

```
> Terminated with exit code 0.
```

If Monte Carlo equation is written for two independent variables

$$x = a_x + (b_x - a_x)random_number(0,1)$$
$$y = a_y + (b_y - a_y)random_number(0,1)$$
$$f_{average} = \frac{\sum_{i=0}^{n} f(x,y)}{n},\ \ n \gg 1$$

Integration subfield

$$\mu(D) = \int_{a_y}^{b_y} \int_{a_x=\alpha(x)}^{b_x=\beta(x)} dydx$$

Integration

$$I = \mu(D) f_{average}$$

As it is seen from the equation process fro one variable integration and two variable integraton is almost the same. This is the main factor why it is frequently use as a high dimensional integration tool.

Program 7.6-2 Monte Carlo integration with two independent variables

```
//=====================================================
// Numerical Analysis package in java
// example to show utilisation of integration (integral)
// and differentiation (derivative) functions
// Monte Carlo integration with two variable
// Dr. Turhan Coban
// =====================================================
import java.io.*;

class f1 extends f_xj
{
public double func(double x[])
{
//çözümü istenen fonksiyon
double ff;
ff=2.0*x[0]*x[1]+2.0*x[0]-x[0]*x[0]-2.0*x[1]*x[1];
return -ff; //minimum değil maxsimum istendiğinden - ile çarptik
}
}

class NA74
{
public static double monte_carlo(f_xj ff,double a[],double b[], long n)
{      //Newton-cotes integral 6 points
                double x[]=new double[2];
                double h=1.0;
                double sum=0;
                for(int j=0;j<2;j++) {h*=(b[j]-a[j]);}
                for(int i=0;i<n;i++)
       { for(int j=0;j<2;j++) {x[j]=a[j]+(b[j]-a[j])*Math.random();}
         sum+=ff.func(x);
       }
       return h*sum/(double)n;
```

```
}

  public static void main(String args[]) throws IOException
  {
  f1 b1=new f1();
  double a[]=new double[2];
  double b[]=new double[2];
  a[0]=-1;
  a[1]=-1;
  b[0]=1;
  b[1]=1;
  System.out.println("integral of class        f1 : "+monte_carlo(b1,a,b,1000000));
}}
```

Output of Monte Carlo integration with two independent variables

```
---------- Capture Output ----------
> "D:\java\bin\javaw.exe" NA74
integral of class        f1 : 4.009412425743507

> Terminated with exit code 0.
```

7.7 MULTIDIMENSIONAL INTEGRATION BY PARAMETRIC METHODS

A two dimensional integration

$$I(D) = \int\int_{D} f(x,y)dydx$$

Where D is a region in xy plane. If this region is defined as $a \le x \le b$and $c(x) \le y \le d(x)$, one way two calculate two dimensional integration is to repeat one dimensional integration twice. In the first step intehration function g(x) is calculated as an integration in y dimension and then function g(x) is solved to get integration result.

$$I = \int_{a}^{b}\int_{c(x)}^{d(x)} f(x,y)dydx$$

$$g(x) = \int_{c(x)}^{d(x)} f(x,y)dydx \qquad a \le x \le b$$

$$I(D) = \int_{a}^{b} g(x)dx$$

Any integration formulation can be used for this purpose. All it has to be done is to evaluate integration Formula twice onece for each independent variable. As our first example Gauss-Legendre formulation will be used to evaluate double integration. Two code is defined. In the first one Gauss-Legendre coefficients are defined as constants, in the second one they are calculated through Legendre polynomials. In the two dimensional integration x dimesion limits are still given as constant values, but y dimension limits c(x) and d(x) should be given as functions for proper evaluation of the integral.

$$I = \int_{x=a}^{b}\int_{y=c(x)}^{d(x)} f(x,y)dydx = \int_{x=a}^{b} \frac{d(x)-c(x)}{2}\sum_{j=1}^{n} c_{n,j} f\left(x, \frac{(d(x)-c(x))r_{n,j}+d(x)+c(x)}{2}\right)dx$$

$$g(x) = \frac{d(x)-c(x)}{2}\sum_{j=1}^{n} c_{n,j} f\left(x, \frac{(d(x)-c(x))r_{n,j}+d(x)+c(x)}{2}\right)$$

$$I = \int_{x=a}^{b}\int_{y=c(x)}^{d(x)} f(x,y)dydx = \int_{x=a}^{b} g(x)dx = \frac{b-a}{2}\sum_{i=1}^{m} c_{m,i} f\left(\frac{b-a}{2}r_{m,i}+\frac{b+a}{2}\right)$$

In this equation $r_{n,j}$ root values and $c_{n,j}$ are equation coefficients for integration of y, and $r_{m,i}$ root values and $c_{m,i}$ are equation coefficients for integration of x. If n=m r and c coefficients will be same. These values can be calculated from Legendre polynomials or they can be directly given as an array in computer code.

Program 7.7-1 60 constant points two variable Gauss-Legendre integration formula

```
//=======================================================
// Numerical Analysis package in java
// example to show utilisation of integration (integral)
// and differentiation (derivative) functions
// Gauss-Legendre integration with two variables
// constant number of coefficients (60 coefficients)
// and roots
// Dr. Turhan Coban
// =======================================================
import java.io.*;

abstract class fdi
{
public double a,b;
public fdi(double ai,double bi)
{a=ai;b=bi;}
public fdi()
{a=0;b=1;}
public void setab(double ai,double bi)
{a=ai;b=bi;}
abstract public  double c(double x);
abstract public  double d(double x);
abstract public  double f(double x[]);
}

class f1 extends fdi
{
  public f1(double ai,double bi)
  {super(ai,bi);}
  //lower integration limit function
  public double c(double x)
  {return 1.0;}
  //upper integration limit function
  public double d(double x)
  {return 1.5;}

  public double f(double x[])
  {
  //function required to integrate
  double ff;
  ff=Math.log(x[0]+2.0*x[1]);
```

```
  return ff;
  }
}


class NA75
{

public static double I(fdi f)
{
//integral f(x)dx
//integral of a function by using gauss-legendre quadrature
//coefficients are pre-calculated for 60 terms for [-1,1]
//band then utilises variable transform
 int i,j;
 int n=60;
 double r[],c[];
 r=new double[n];
 c=new double[n];
 r[ 0] = .15532579626752470000E-02;
 r[ 1] = .81659383601264120000E-02;
 r[ 2] = .19989067515846230000E-01;
 r[ 3] = .36899976285362850000E-01;
 r[ 4] = .58719732103973630000E-01;
 r[ 5] = .85217118808615820000E-01;
 r[ 6] = .11611128394758690000E+00;
 r[ 7] = .15107475260334210000E+00;
 r[ 8] = .18973690850537860000E+00;
 r[ 9] = .23168792592899010000E+00;
 r[10] = .27648311523095540000E+00;
 r[11] = .32364763723456090000E+00;
 r[12] = .37268153691605510000E+00;
 r[13] = .42306504319570830000E+00;
 r[14] = .47426407872234120000E+00;
 r[15] = .52573592127765890000E+00;
 r[16] = .57693495680429170000E+00;
 r[17] = .62731846308394490000E+00;
 r[18] = .67635236276543910000E+00;
 r[19] = .72351688476904450000E+00;
 r[20] = .76831207407100990000E+00;
 r[21] = .81026309149462140000E+00;
 r[22] = .84892524739665800000E+00;
 r[23] = .88388871605241310000E+00;
 r[24] = .91478288119138420000E+00;
 r[25] = .94128026789602640000E+00;
 r[26] = .96310002371463720000E+00;
 r[27] = .98001093248415370000E+00;
 r[28] = .99183406163987350000E+00;
 r[29] = .99844674203732480000E+00;
c[ 0] = .39840962480827790000E-02;
c[ 1] = .92332341555455000000E-02;
c[ 2] = .14392353941661670000E-01;
c[ 3] = .19399596284813530000E-01;
c[ 4] = .24201336415292590000E-01;
c[ 5] = .28746578108808720000E-01;
c[ 6] = .32987114941090080000E-01;
c[ 7] = .36877987368852570000E-01;
```

```
c[ 8] = .40377947614710090000E-01;
c[ 9] = .43449893600541500000E-01;
c[10] = .46061261118893050000E-01;
c[11] = .48184368587322120000E-01;
c[12] = .49796710293397640000E-01;
c[13] = .50881194874202750000E-01;
c[14] = .51426326446779420000E-01;
c[15] = .51426326446779420000E-01;
c[16] = .50881194874202750000E-01;
c[17] = .49796710293397640000E-01;
c[18] = .48184368587322120000E-01;
c[19] = .46061261118893050000E-01;
c[20] = .43449893600541500000E-01;
c[21] = .40377947614710090000E-01;
c[22] = .36877987368852570000E-01;
c[23] = .32987114941090080000E-01;
c[24] = .28746578108808720000E-01;
c[25] = .24201336415292590000E-01;
c[26] = .19399596284813530000E-01;
c[27] = .14392353941661670000E-01;
c[28] = .92332341555455000000E-02;
c[29] = .39840962480827790000E-02;
for(i=0;i<30;i++) {r[i+30]=-r[i];c[i+30]=c[i];}
int m=n;
double x[],h1,h2,J,a1,b1,c1,d1,k1,k2,JX,Q;
x=new double[2];
//1==========================
a1=f.a;
b1=f.b;
h1=(b1-a1)/2.0;
h2=(b1+a1)/2.0;
J=0;
//2==========================
for(i=0;i<m;i++)
{
//3==========================
JX=0;
x[0]=h1*r[i]+h2;
d1=f.d(x[0]);
c1=f.c(x[0]);
k1=(d1-c1)/2.0;
k2=(d1+c1)/2.0;
//4==========================
   for(j=0;j<n;j++)
   {
            x[1]=k1*r[j]+k2;
            Q=f.f(x);
            JX+=c[j]*Q;
            }
//System.out.println("JX="+JX);
//==========================
J=J+c[i]*k1*JX;
}
J=h1*J;
return J;
}
```

```
  public static void main(String args[]) throws IOException
  {
  f1 ff=new f1(1.4,2.0);
  System.out.println("integral of class        f1 : "+I(ff));
}
}
```

constant points two variable Gauss-Legendre integration formula output

```
---------- Capture Output ----------
> "D:\java\bin\javaw.exe" NA75
integral of class        f1 : 0.429554527548266

> Terminated with exit code 0.
```

In the next example, instead of constant number of points Legendre transformation defined varible number of points are taken. C and r values are directly calculated from Legendre polynomials.

Program 7.7-2 variable points two variable Gauss-Legendre integration formula

```
//=====================================================
// Numerical Analysis package in java
// example to show utilisation of integration (integral)
// and differentiation (derivative) functions
// Gauss-Legendre integration with two variables
// variable number of coefficients and roots
// calculated by Legendre polynomials
// Dr. Turhan Coban
// =====================================================
import java.io.*;

abstract class fdi
{
public double a,b;
public fdi(double ai,double bi)
{a=ai;b=bi;}
public fdi()
{a=0;b=1;}
public void setab(double ai,double bi)
{a=ai;b=bi;}
abstract public  double c(double x);
abstract public  double d(double x);
abstract public  double f(double x[]);
}

class f1 extends fdi
{
  public f1(double ai,double bi)
  {super(ai,bi);}

  public double c(double x)
  {return 1.0;}

  public double d(double x)
  {return 1.5;}
```

```
 public double f(double x[])
 {
 //çözümü istenen fonksiyon
 double ff;
 ff=Math.log(x[0]+2.0*x[1]);
 return ff;
 }
}

class NA76
{
public static double[][] gauss_legendre_coefficients(double x1,double x2,int n)
{
        //calculates legendre gauss-coefficients as coefficients of the integral
        //for n terms
        double EPS=3.0e-15;
        int m,j,i;
        double z1,z,xm,xl,pp,p3,p2,p1;
         double a[][]=new double[2][n];
        m=(n+1)/2;
        xm=0.5*(x2+x1);
        xl=0.5*(x2-x1);
        for (i=1;i<=m;i++)  {
                z=Math.cos(Math.PI*((i-0.25)/(n+0.5)));
                do {
                        p1=1.0;
                        p2=0.0;
                        for (j=1;j<=n;j++) {
                                p3=p2;
                                p2=p1;
                                p1=((2.0*j-1.0)*z*p2-(j-1.0)*p3)/j;
                        }
                        pp=n*(z*p1-p2)/(z*z-1.0);
                        z1=z;
                        z=z1-p1/pp;
                } while (Math.abs(z-z1) > EPS);
                a[0][i-1]=xm-xl*z;
                a[0][n-i]=xm+xl*z;
                a[1][i-1]=2.0*xl/((1.0-z*z)*pp*pp);
                a[1][n-i]=a[1][i-1];
                        }
return a;
}

public static double I(fdi f,int n,int m)
{
//integral f(x,y)dxdy
//integral of a function by using gauss-legendre quadrature
//coefficients are pre-calculated for n,m terms for [-1,1]
//band then utilises variable transform
 int i,j;
 double rn[],cn[];
 double rm[],cm[];
 rn=new double[n];
 rm=new double[m];
 cn=new double[n];
 double a[][]=new double[2][n];
```

```
 a=gauss_legendre_coefficients(-1.0,1.0,n);
 rn=a[0];
 cn=a[1];
 a=gauss_legendre_coefficients(-1.0,1.0,m);
 rm=a[0];
 cm=a[1];
double x[],h1,h2,J,a1,b1,c1,d1,k1,k2,JX,Q;
x=new double[2];
a1=f.a;
b1=f.b;
h1=(b1-a1)/2.0;
h2=(b1+a1)/2.0;
J=0;
for(i=0;i<m;i++)
{
JX=0;
x[0]=h1*rm[i]+h2;
d1=f.d(x[0]);
c1=f.c(x[0]);
k1=(d1-c1)/2.0;
k2=(d1+c1)/2.0;
//4=========================
   for(j=0;j<n;j++)
   {
            x[1]=k1*rn[j]+k2;
            Q=f.f(x);
            JX+=cn[j]*Q;
            }
//System.out.println("JX="+JX);
//5=========================
J=J+cm[i]*k1*JX;
}
J=h1*J;
return J;
}


public static double I(fdi f,int n)
{return I(f,n,n);}

public static double I(fdi f)
{return I(f,60,60);}

  public static void main(String args[]) throws IOException
  {
  f1 ff=new f1(1.4,2.0);
  System.out.println("integral of class         f1 : "+I(ff));
}
}
```

variable points two variable Gauss-Legendre integration Formula output

```
---------- Capture Output ----------
> "D:\java\bin\javaw.exe" NA76
integral of class         f1 : 0.4295545275482747
```

```
> Terminated with exit code 0.
```

For the two variable integration Newton-Cotes formulas can also be used. As a example a Simpson 1/3 two dimensional integration code is prepared. For the quadratic polynomial Newton-Cotes equation (simpson 1/3 method) equations can be written as follows;
One dimensional form was

$$h = \frac{(b-a)}{2}$$

$$I = \int_a^b f(x)dx \cong \frac{h}{3}[f(x_0) + 4f(x_1) + f(x_2)]$$

If this Formula can be written in composite form.

$$I = \int_a^b f(x)dx \cong \frac{h}{3}\left[f(a) + \sum_{j=1}^{\left(\frac{n}{2}\right)-1} f\left(x_{2j}\right) + 4\sum_{j=1}^{\left(\frac{n}{2}\right)} f\left(x_{2j-1}\right) + f(b)\right]$$

This Formula can be open to the second dimension similar to Gauss-Legendre integration
By defining

$$k = \frac{[d(x) - c(x)]}{m}$$

integration equation becomes

$$I = \int_{c(x)}^{d(x)} f(x,y)dy \cong \frac{k}{3}\left[f(x,y_0) + \sum_{j=1}^{\left(\frac{n}{2}\right)-1} f\left(x,y_{2j}\right) + 4\sum_{j=1}^{\left(\frac{n}{2}\right)} f\left(x,y_{2j-1}\right) + f(x,y_m)\right]$$

From here x integration can be carried out as:

$$I = \int_a^b \left[\int_{c(x)}^{d(x)} f(x,y)dy\right]dx = \int_a^b \frac{k}{3}\left[f(x,y_0) + \sum_{j=1}^{\left(\frac{n}{2}\right)-1} f\left(x,y_{2j}\right) + 4\sum_{j=1}^{\left(\frac{n}{2}\right)} f\left(x,y_{2j-1}\right) + f(x,y_m)\right]dx$$

The same integration equation is then applied to the outer integration. Complete code is given in the program 8.7.3 . In the example the same function as in previous Gauss-Legendre integration is taken to compare the results

Program 7.7-3 two dimensional Simpson 1/3 integration formula

```
//=======================================================
// Numerical Analysis package in java
// simpson integration of two dimensional function
// and differentiation (derivative) functions
// two dimensional Simpson 1/3 integration
// (quadratic Newton-Cotes formula)
// Dr. Turhan Coban
// =======================================================
import java.io.*;

abstract class fdi
{
public double a,b;
```

```
public fdi(double ai,double bi)
{a=ai;b=bi;}
public fdi()
{a=0;b=1;}
public void setab(double ai,double bi)
{a=ai;b=bi;}
abstract public  double c(double x);
abstract public  double d(double x);
abstract public  double f(double x[]);
}

class f1 extends fdi
{
  public f1(double ai,double bi)
  {super(ai,bi);}

  public double c(double x)
  {return 1.0;}

  public double d(double x)
  {return 1.5;}

  public double f(double x[])
  {
  //çözümü istenen fonksiyon
  double ff;
  ff=Math.log(x[0]+2.0*x[1]);
  return ff;
  }
}


class NA77
{

public static double I(fdi f,int m,int n)
{
//integral f(x)dx
//integral of a function by using simpson metod
double h=(f.b-f.a)/n;
double J1=0.0;
double J2=0.0;
double J3=0.0;
double x[]=new double[2];
double HX=0;
double K1,K2,K3;
double d,c;
double Q,L,J;
for(int i=0;i<=n;i++)
{
x[0]=f.a+i*h;
d=f.d(x[0]);
c=f.c(x[0]);
HX=(d-c)/m;
x[1]=c;K1=f.f(x);
x[1]=d;K1+=f.f(x);
K2=0;
K3=0;
  for(int j=1;j<=(m-1);j++)
  {x[1]=f.c(x[0])+j*HX;
  Q=f.f(x);
  if((j/2)*2==j) K2=K2+Q;
  else          K3=K3+Q;
  }
```

```
L=(K1+2.0*K2+4.0*K3)*HX/3.0;
if((i==0)||(i==n)) J1=J1+L;
else if ((i/2)*2==i) J2=J2+L;
else J3=J3+L;
}
J=h*(J1+2.0*J2+4.0*J3)/3.0;
return J;
}

  public static void main(String args[]) throws IOException
  {
  f1 ff=new f1(1.4,2.0);
  System.out.println("integral of class         f1 : "+I(ff,20,20));
}
}
```

two dimensional Simpson 1/3 integration formula

```
---------- Capture Output ----------
> "D:\java\bin\javaw.exe" NA77
integral of class         f1 : 0.4295545273061155

> Terminated with exit code 0.
```

Now the same principle can be used for three dimensional integration process. A three dimensional integral can be defined as:

$$I = \int_{a}^{b} \int_{c(x)}^{d(x)} \int_{\alpha(x,y)}^{\beta(x,y)} f(x,y,z)dzdydx$$

Gauss-Legendre integration will be used here as a solution method.

$$I = \int_{0}^{1} \int_{x}^{1} \int_{0}^{y} y^2 zdzdydx$$

Program 7.7-4 three dimensional Gauss-Legendre integration

```
//=======================================================
// Numerical Analysis package in java
// example to show utilisation of integration (integral)
// and differentiation (derivative) functions
// Gauss-Legendre integration with three independent
// variable number of coefficients and roots
// calculated by Legendre polynomials
// Dr. Turhan Coban
// =======================================================
import java.io.*;

abstract class fdi3
{
public double a,b;
public fdi3(double ai,double bi)
{setab(ai,bi);}
public void setab(double ai,double bi)
{a=ai;b=bi;}
abstract public  double c(double x);
abstract public  double d(double x);
abstract public  double alfa(double x[]);
abstract public  double beta(double x[]);
abstract public  double f(double x[]);
```

```
}

class f1 extends fdi3
{
  public f1(double ai,double bi)
  {super(ai,bi);}

  public double c(double x)
  {return x;}

  public double d(double x)
  {return 1.0;}

  public double alfa(double x[])
  {return 0.0;}

  public double beta(double x[])
  {return x[1];}

  public double f(double x[])
  {
  //sample function
  double ff;
  ff=x[1]*x[1]*x[2];
  return ff;
  }
}

class NA78
{
public static double[][] gauss_legendre_coefficients(double x1,double x2,int n)
{
            //calculates legendre gauss-coefficients as coefficients of the integral
            //for n terms
            double EPS=3.0e-15;
            int m,j,i;
            double z1,z,xm,xl,pp,p3,p2,p1;
    double a[][]=new double[2][n];//a[0][i]=x[i]   a[1][i]=w[i]
            m=(n+1)/2;
            xm=0.5*(x2+x1);
            xl=0.5*(x2-x1);
            for (i=1;i<=m;i++)  {
                        z=Math.cos(Math.PI*((i-0.25)/(n+0.5)));
                        do {
                                    p1=1.0;
                                    p2=0.0;
                                    for (j=1;j<=n;j++) {
                                                p3=p2;
                                                p2=p1;
                                                p1=((2.0*j-1.0)*z*p2-(j-1.0)*p3)/j;
                                    }
                                    pp=n*(z*p1-p2)/(z*z-1.0);
                                    z1=z;
                                    z=z1-p1/pp;
                        } while (Math.abs(z-z1) > EPS);
                        a[0][i-1]=xm-xl*z;
                        a[0][n-i]=xm+xl*z;
```

```
                        a[1][i-1]=2.0*xl/((1.0-z*z)*pp*pp);
                        a[1][n-i]=a[1][i-1];
                                }
return a;
}

public static double I(fdi3 f,int n,int m,int p)
{
//integral f(x)dx
//integral of a function by using gauss-legendre quadrature
//coefficients are pre-calculated for n,m terms for [-1,1]
//band then utilises variable transform
 int i,j;
 double rn[],cn[];
 double rm[],cm[];
 double rp[],cp[];
 rp=new double[p];
 cp=new double[p];
 int nn=Math.max(n,Math.max(m,p));
 int k;
 double x[]=new double[3];
 double a[][]=new double[2][nn];
 int mm=Math.max(Math.max(n,m),p);
 a=gauss_legendre_coefficients(-1.0,1.0,mm);
 rn=rm=rp=a[0];
 cn=cm=cp=a[1];
double h1,h2,J,a1,b1,c1,d1,k1,k2,JX,JY,Q,l1,l2,alfa1,beta1;
a1=f.a;
b1=f.b;
h1=(b1-a1)/2.0;
h2=(b1+a1)/2.0;
J=0;
for(i=0;i<m;i++)
{
JX=0;
x[0]=h1*rm[i]+h2;
d1=f.d(x[0]);
c1=f.c(x[0]);
k1=(d1-c1)/2.0;
k2=(d1+c1)/2.0;
//4=========================
for(j=0;j<n;j++)
{   JY=0;
   x[1]=k1*rn[j]+k2;
   beta1=f.beta(x);
   alfa1=f.alfa(x);
   l1=(beta1-alfa1)/2.0;
   l2=(beta1+alfa1)/2.0;
   for(k=0;k<p;k++)
   {
            x[2]=l1*rp[k]+l2;
            Q=f.f(x);
            JY+=cp[k]*Q;
   }
JX+=cn[j]*l1*JY;
}
//System.out.println("JX="+JX);
```

```
//5=========================
J=J+cm[i]*k1*JX;
}
J=h1*J;
return J;
}

public static double I(fdi3 f,int n)
{return I(f,n,n,n);}

public static double I(fdi3 f)
{return I(f,60,60,60);}

  public static void main(String args[]) throws IOException
  {
  f1 ff=new f1(0.0,1.0);            ;
  System.out.println("İntegral="+I(ff));
  }
}
```

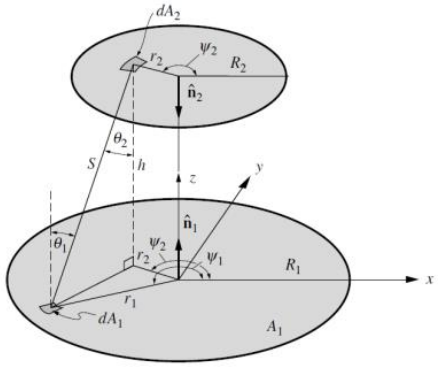
```
---------- Capture Output ----------
> "D:\java\bin\javaw.exe" NA78
Integral=0.08333333333333286

> Terminated with exit code 0.
```

By using the same priciples, a fourth dimensional integral can also be solved.

$$I = \int_{x=a}^{b} \int_{y=c(x)}^{d(x)} \int_{z=\alpha(x,y)}^{\beta(x,y)} \int_{t=\gamma(x,y,z)}^{\theta(x,y,z)} f(x,y,z,t)dtdzdydx$$

As an example problem, let us look at radiation shape factor problem in radiative heat transfer. In this problem shape factor of two parallel round plate will be calculated.



Radiation shape factor is the percentage of radiative energy reached to surface A2, emitted from surface A1. If the Radius of plates are R1 and R2, and the distance in between is h, the radiation shape factor eqaution is given as:

$$F_{A1-A2} = \frac{1}{\pi^2 R_1^2} \int_{r1=0}^{R_1} \int_{r2=0}^{R_2} \int_{\psi_1=0}^{2\pi} \int_{\psi_2=0}^{2\pi} \frac{h^2 r_1 r_2 d\psi_1 d\psi_2 dr_1}{[h^2 + r_1^2 + r_2^2 - 2r_1 r_2 \cos(\psi_1 - \psi_2)]}$$

This equation has an analytical solution as:

$$F_{A1-A2} = \frac{R_2}{R_1}\left(X - \sqrt{X^2 - 1}\right)$$

$$X = \frac{h^2 + R_1^2 + R}{2R_1R_2}$$

In the following program numerical solution by Gauss-Legendre integration is obtained and compared to the analytical solution.

Program 7.7-5 four dimensional Gauss-Legendre integration

```
//=====================================================
// Numerical Analysis package in java
// 4 dimensional Gauss-Legendre integration
// written by: Hamza TAŞ
// =====================================================
import java.io.*;
abstract class fdi4
{
public double a,b;
public fdi4(double ai,double bi)
{a=ai;b=bi;}
public fdi4()
{a=0;b=1;}
public void setab(double ai,double bi)
{a=ai;b=bi;}
abstract public  double c(double x);
abstract public  double d(double x);
abstract public  double alfa(double x[]);
abstract public  double beta(double x[]);
abstract public  double gama(double x[]);
abstract public  double teta(double x[]);
abstract public  double f(double x[]);
}

class f1 extends fdi4
{ double h,R1,R2;
  public f1(double hi,double R1i, double R2i)
  {h=hi;R1=R1i;R2=R2i;a=0;b=R1;}

  public double c(double x)
  {return 0;}

  public double d(double x)
  {return R2;}

   public double alfa(double x[])
  {return 0;}

  public double beta(double x[])
  {return 2.0*Math.PI;}

  public double gama(double x[])
  {return 0.0;}

  public double teta(double x[])
  {return 2.0*Math.PI;}
```

```
 public double f(double x[])
 {
 //çözümü istenen fonksiyon
 double ff;
 double z=(h*h+x[0]*x[0]+x[1]*x[1]-2.0*x[0]*x[1]*Math.cos(x[2]-x[3]));
 ff=1.0/(Math.PI*Math.PI*R1*R1)*h*h*x[0]*x[1]/(z*z);
 return ff;
 }
 public double  F12(double h,double R1,double R2)
 {double X=(h*h+R1*R1+R2*R2)/(2.0*R1*R2);
  double F=R2/R1*(X-Math.sqrt(X*X-1));
  return F;
 }
}

class NA78A
{
public static double[][] gauss_legendre_coefficients(double x1,double x2,int n)
{
        //calculates legendre gauss-coefficients as coefficients of the integral for n terms
        double EPS=3.0e-15;
        int m,j,i;
        double z1,z,xm,xl,pp,p3,p2,p1;
   double a[][]=new double[2][n];//a[0][i]=x[i]   a[1][i]=w[i]
        m=(n+1)/2;
        xm=0.5*(x2+x1);
        xl=0.5*(x2-x1);
        for (i=1;i<=m;i++)  {
                z=Math.cos(Math.PI*((i-0.25)/(n+0.5)));
                do {
                        p1=1.0;
                        p2=0.0;
                        for (j=1;j<=n;j++) {
                                p3=p2;
                                p2=p1;
                                p1=((2.0*j-1.0)*z*p2-(j-1.0)*p3)/j;
                        }
                        pp=n*(z*p1-p2)/(z*z-1.0);
                        z1=z;
                        z=z1-p1/pp;
                } while (Math.abs(z-z1) > EPS);
                a[0][i-1]=xm-xl*z;
                a[0][n-i]=xm+xl*z;
                a[1][i-1]=2.0*xl/((1.0-z*z)*pp*pp);
                a[1][n-i]=a[1][i-1];
                        }
return a;
}
public static double I(fdi4 f,int n,int m,int p,int s)
{
 int i,j,k,h;
 double rn[],cn[];
 double rm[],cm[];
 double rp[],cp[];
 double rs[],cs[];
 rn=new double[n];
 rm=new double[m];
```

```
 rp=new double[p];
 rs=new double[s];
 cn=new double[n];
 cm=new double[m];
 cp=new double[p];
 cs=new double[s];
 double a[][]=new double[2][n];
 a=gauss_legendre_coefficients(-1.0,1.0,n);
 rn=a[0];
 cn=a[1];
 a=gauss_legendre_coefficients(-1.0,1.0,m);
 rm=a[0];
 cm=a[1];
 a=gauss_legendre_coefficients(-1.0,1.0,p);
 rp=a[0];
 cp=a[1];
 a=gauss_legendre_coefficients(-1.0,1.0,s);
 rs=a[0];
 cs=a[1];
double x[],h1,h2,J,a1,b1,c1,d1,k1,k2,JX,JY,JZ,Q,l1,l2,alfa1,beta1,gama1,teta1,m1,m2;
x=new double[4];
a1=f.a;
b1=f.b;
h1=(b1-a1)/2.0;
h2=(b1+a1)/2.0;
J=0;
for(i=0;i<m;i++)
{
JX=0;
x[0]=h1*rm[i]+h2;
d1=f.d(x[0]);
c1=f.c(x[0]);
k1=(d1-c1)/2.0;
k2=(d1+c1)/2.0;
    for(j=0;j<n;j++)
    {
            JY=0;
    x[1]=k1*rn[j]+k2;
    beta1=f.beta(x);
    alfa1=f.alfa(x);
    l1=(beta1-alfa1)/2.0;
    l2=(beta1+alfa1)/2.0;
        for(k=0;k<p;k++)
        {
                JZ=0;
                x[2]=l1*rp[k]+l2;
                teta1=f.teta(x);
                gama1=f.gama(x);
                m1=(teta1-gama1)/2.0;
                m2=(teta1+gama1)/2.0;
                    for (h=0;h<s;h++)
                    {
                            x[3]=m1*rs[h]+m2;
                    Q=f.f(x);
                    JZ+=cs[h]*Q;
            }
      JY+=cp[k]*m1*JZ;
```

```
        }
    JX+=cn[j]*l1*JY;
     }
J=J+cm[i]*k1*JX;
}
J=h1*J;
return J;
}
public static double I(fdi4 f,int n)
{return I(f,n,n,n,n);}
public static double I(fdi4 f)
{return I(f,60,60,60,60);}
  public static void main(String args[]) throws IOException
  {
  double h=1.0;
  double R1=1.0;
  double R2=1.0;
  f1 ff=new f1(h,R1,R2);
  System.out.println("FA1-A2= "+I(ff)+" four dimensional auss Legendre integration ");
  System.out.println("FA1-A2= "+ff.F12(h,R1,R2)+ " analytical solution ");
  }
}
```

```
 ---------- Capture Output ----------
> "c:\java\bin\javaw.exe" NA78A
FA1-A2= 0.3819660112501026 four dimensional Gauss-Legendre integration
FA1-A2= 0.3819660112501051 analytical solution

> Terminated with exit code 0.
```

7.8 DIRECT INTEGRATION OF DATA

If integration of a numerical data set is given two approches and be adapted for this process. The first one is to curve fit a function first and then integrate the fitted function. The second one is to apply an integration equation like Newton-Cotes equation directly to the data to achive integrated result without an intermediate process.

As an example to curve fitting integration a cubic spline integration will be described here. If a cubic polynomial of the following form is given as an interpolation formula:

$$s_k(x) = a_k(x - x_k) + b_k(x_{k+1} - x) + [(x - x_k)^3 c_{k+1} + (x_{k+1} - x)^3 c_k] / (6h_k) \quad 1 \le k \le n \quad (10.6 - 14)$$

Where x_k and y_k are data points. Derivatives of equation becomes

$$s'_k(x) = a_k - b_k + [(x - x_k)^2 c_{k+1} - (x_{k+1} - x)^2 c_k] / h_k \quad 1 \le k \le n \quad (10.6\text{-}15)$$

$$s''_k(x) = [(x - x_k) c_{k+1} - (x_{k+1} - x) c_k] / h_k \qquad 1 \le k \le n$$

a_k ve b_k coefficients can be expressed as a function of c_k

$$b_k = \frac{[6y_k - h_k c_k]}{6h_k} \qquad 1 \le k \le n \qquad (10.6 - 16)$$

$$a_k = \frac{[6y_{k+1} - h_k^2 c_{k+1}]}{6h_k} \qquad 1 \le k \le n \qquad (10.6 - 17)$$

In this case only c_k terms left in the system of equations to be solved.

$$h_{k-1} c_{k-1} + 2(h_{k-1} - h_k) c_{k-1} + h_k c_{k+1} c_{k+1} = 6\left[\frac{y_{k+1} - y_k}{h_k} - \frac{y_k - y_{k-1}}{h_{k-1}}\right] \qquad 1 \le k \le n \qquad (10.6 - 18)$$

This system of equation has only n-2 terms to be solved. By making definition

$w_k = \frac{y_{k+1}-y_k}{h_k}$, $\quad 1 \leq k \leq n$ (10.6-19)

System of equation becomes

$$\begin{bmatrix} 1 & & & & & & & \\ h_1 & 2(h_1+h_2) & h_2 & & & & & \\ & h_2 & 2(h_2+h_3) & \cdots & & & & \\ & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \\ & & & \cdots & 2(h_{n-3}+h_{n-2}) & h_{n-2} & & \\ & & & & h_{n-2} & 2(h_{n-2}+h_{n-1}) & h_{n-1} & \\ & & & & & & 1 & \end{bmatrix} \begin{Bmatrix} c_0 \\ c_1 \\ c_2 \\ \cdots \\ c_{n-2} \\ c_{n-1} \\ c_n \end{Bmatrix} = \begin{Bmatrix} A \\ 6(w_2-w_1) \\ 6(w_3-w_2) \\ \cdots \\ 6(w_{n-2}-w_{n-3}) \\ 6(w_{n-1}-w_{n-2}) \\ B \end{Bmatrix} \quad (10.6-20)$$

Where A and B are the second derivative end conditions. A and B should be defined by user. Another important property of the above matrix is that it is a band matrix, therefore less amount of calculation is required to solve it (by using band matrix algorithms such as Thomas algorithm) compare to gauss elimination type of solution methods.

Integration of the equation could be described with the equation

$$\int_a^b S_k(x)dx = \sum_{x=a}^{b} a_k \frac{(x-x_k)^2}{2} - b_k \frac{(x_{k+1}-x)^2}{2} + c_k \frac{(x-x_k)^4}{24h_k} - d_k \frac{(x_{k+1}-x)^4}{24h_k}$$

For a, x_0, x_1, x_2, ... x_n,. An example program using this approach is given as:

Program 7.8-1 Direct integration of data by using cubic polynomial fitting

```
//=====================================================
// Numerical Analysis package in java
// example to show utilisation of integration (integral)
// and differentiation (derivative) functions
// Direct data integration by using cubic spline
// curve fitting
// Dr. Turhan Coban
// =====================================================
import java.io.*;
import javax.swing.*;

class NA79
{

  public static double [] thomas(double a[][],double r[])
  {
  //
  int n=a.length;
  double f[]=new double[n];
  double e[]=new double[n];
  double g[]=new double[n];
  double x[]=new double[n];
  for(int i=0;i<n;i++)     {f[i]=a[i][i];}
  for(int i=0;i<(n-1);i++) {g[i]=a[i][i+1];}
  for(int i=0;i<(n-1);i++) {e[i+1]=a[i+1][i];}
  for(int k=1;k<n;k++)
     {e[k]=e[k]/f[k-1];
      f[k]=f[k]-e[k]*g[k-1];
     }
  for(int k=1;k<n;k++)
     {r[k]=r[k]-e[k]*r[k-1];
     }
```

```
x[n-1]=r[n-1]/f[n-1];
for(int k=(n-2);k>=0;k--)
  {x[k]=(r[k]-g[k]*x[k+1])/f[k];}
return x;
}

public static double [] thomas(double f[],double e[],double g[],double r[])
{
int n=f.length;
double x[]=new double[n];
for(int k=1;k<n;k++)
  {e[k]=e[k]/f[k-1];
   f[k]=f[k]-e[k]*g[k-1];
  }
for(int k=1;k<n;k++)
  {r[k]=r[k]-e[k]*r[k-1];
  }
x[n-1]=r[n-1]/f[n-1];
for(int k=(n-2);k>=0;k--)
  {x[k]=(r[k]-g[k]*x[k+1])/f[k];}
return x;
}

public static double [][] cubic_spline(double xi[],double yi[],double c0,double cn)
{
// c0 second derivative of the first point
// cn second derivative of the last point
int n=xi.length;
double h[]=new double[n];
double w[]=new double[n];
double f[]=new double[n];
double e[]=new double[n];
double g[]=new double[n];
double d[]=new double[n];
double x[]=new double[n];
double S[][]=new double[4][n];
int k;
for(k=0;k<(n-1);k++)
  {h[k]=xi[k+1]-xi[k];
   w[k]=(yi[k+1]-yi[k])/h[k];
          }
d[0]=c0;
d[n-1]=cn;
for(k=1;k<(n-1);k++)
  {d[k]=6.0*(w[k]-w[k-1]);}
f[0]=1.0;
f[n-1]=1.0;
g[0]=0.0;
g[n-1]=0.0;
e[0]=0.0;
e[n-1]=0.0;
for(k=1;k<(n-1);k++)
{f[k]=2.0*(h[k]+h[k-1]);e[k]=h[k-1];g[k]=h[k];}
S[2]=thomas(f,e,g,d);
S[3]=xi;
for(k=0;k<(n-1);k++)
 {S[0][k]=(6.*yi[k+1]-h[k]*h[k]*S[2][k+1])/(6.0*h[k]);
```

```
    S[1][k]=(6.*yi[k]-h[k]*h[k]*S[2][k])/(6.0*h[k]);
   }
  return S;
  }

  public static double funcSpline(double S[][],double x)
  {
  int n=S[0].length;
  double xx1=0;
  double xx2=0;
  double y=0;
  double hk=0;
  for(int k=0;k<(n-1);k++)
    {if(S[3][k]<=x && x<=S[3][k+1])
     {hk=(S[3][k+1]-S[3][k]);
      xx1=(x-S[3][k]);
      xx2=(S[3][k+1]-x);
      y=S[0][k]*xx1+S[1][k]*xx2+(xx1*xx1*xx1*S[2][k+1]+xx2*xx2*xx2*S[2][k])/(6.0*hk);
      break;
      }
    }
      if(y==0 && S[3][n-2]<=x )
      {
               int k=n-2;
               hk=(S[3][k+1]-S[3][k]);
      xx1=(x-S[3][k]);
      xx2=(S[3][k+1]-x);
      y=S[0][k]*xx1+S[1][k]*xx2+(xx1*xx1*xx1*S[2][k+1]+xx2*xx2*xx2*S[2][k])/(6.0*hk);
      }
  return y;
  }

// Derivative formulations
// ==================
  public static double dfSpline(double xi[],double yi[],double c0,double cn,double x)
  { //kübik şerit türev formülü
           double S[][]=cubic_spline(xi,yi,c0,cn);
   return dfSpline(S,x);
  }

  public static double dfSpline(double S[][],double x)
  {
   //derivative of the cubic spline formula
  int n=S[0].length;
  double xx1=0;
  double xx2=0;
  double y=0;
  double hk=0;
  for(int k=0;k<(n-1);k++)
    {if(S[3][k]<=x && x<=S[3][k+1])
     {hk=(S[3][k+1]-S[3][k]);
      xx1=(x-S[3][k]);
      xx2=(S[3][k+1]-x);
      y=S[0][k]-S[1][k]+(xx1*xx1*S[2][k+1]-xx2*xx2*S[2][k])/(2.0*hk);
      break;
      }
    }
```

```
      if(y==0 && S[3][n-2]<=x )
      {
               int k=n-2;
               hk=(S[3][k+1]-S[3][k]);
      xx1=(x-S[3][k]);
      xx2=(S[3][k+1]-x);
      y=S[0][k]*xx1+S[1][k]*xx2+(xx1*xx1*xx1*S[2][k+1]+xx2*xx2*xx2*S[2][k])/(6.0*hk);
      }
 return y;
 }

 public static double  dfSpline(double xi[],double yi[],double x)
 {
   //natural cubic spline formula
            double S[][]=cubic_spline(xi,yi,0,0);
   return dfSpline(S,x);
 }
// Integration formulas
// ====================
  public static double intSpline(double S[][],double a,double b)
  {
    //cubic spline integration
  int n=S[0].length;
  double xx1=0;
  double xx2=0;
  double y1,y2;
  double hk=0;
  double toplam=0;
  y1=0;y2=0;
  for(int k=0;k<(n-1);k++)
    { hk=(S[3][k+1]-S[3][k]);
      //System.out.println("x1="+S[3][k]+"x2="+S[3][k+1]+"a="+a+"b="+b);
     if(a>S[3][k+1])
     {toplam=0;//System.out.println("şart 1 toplam = "+toplam);
     }//şart1
              else if(S[3][k]<=a && a<=S[3][k+1] && b>S[3][k+1])//şart 2
      {
      xx1=(a-S[3][k]);
      xx2=(S[3][k+1]-a);
      y1=S[0][k]*xx1*xx1/2.0-S[1][k]*xx2*xx2/2.0+(xx1*xx1*xx1*xx1*S[2][k+1]-xx2*xx2*xx2*xx2*S[2][k])/(24.0*hk);
      xx1=hk;
      xx2=0;
      y2=S[0][k]*xx1*xx1/2.0-S[1][k]*xx2*xx2/2.0+(xx1*xx1*xx1*xx1*S[2][k+1]-xx2*xx2*xx2*xx2*S[2][k])/(24.0*hk);
      toplam+=(y2-y1);
      //System.out.println("şart 2toplam = "+toplam+"y1="+y1+"y2="+y2);
      }
      else if(S[3][k]<=a && a<=S[3][k+1] && S[3][k]<=b && b<=S[3][k+1])//şart 3
      {
      xx1=(a-S[3][k]);
      xx2=(S[3][k+1]-a);
      y1=S[0][k]*xx1*xx1/2.0-S[1][k]*xx2*xx2/2.0+(xx1*xx1*xx1*xx1*S[2][k+1]-xx2*xx2*xx2*xx2*S[2][k])/(24.0*hk);
      xx1=(b-S[3][k]);
      xx2=(S[3][k+1]-b);
      y2=S[0][k]*xx1*xx1/2.0-S[1][k]*xx2*xx2/2.0+(xx1*xx1*xx1*xx1*S[2][k+1]-xx2*xx2*xx2*xx2*S[2][k])/(24.0*hk);
      toplam+=(y2-y1);
      //System.out.println("şart 3 toplam = "+toplam+"y1="+y1+"y2="+y2);
      }
```

```
      else if(a<S[3][k] && b>=S[3][k+1]) //şart 4
      {
      xx1=0;
      xx2=hk;
      y1=S[0][k]*xx1*xx1/2.0-S[1][k]*xx2*xx2/2.0+(xx1*xx1*xx1*xx1*S[2][k+1]-xx2*xx2*xx2*xx2*S[2][k])/(24.0*hk);
      xx1=hk;
      xx2=0;
      y2=S[0][k]*xx1*xx1/2.0-S[1][k]*xx2*xx2/2.0+(xx1*xx1*xx1*xx1*S[2][k+1]-xx2*xx2*xx2*xx2*S[2][k])/(24.0*hk);
      toplam+=(y2-y1);
      //System.out.println("şart 4 toplam = "+toplam+"y1="+y1+"y2="+y2);
      }
      else if(S[3][k]<=b && b<=S[3][k+1])//şart 5
      {xx1=0;
       xx2=hk;
       y1=S[0][k]*xx1*xx1/2.0-S[1][k]*xx2*xx2/2.0+(xx1*xx1*xx1*xx1*S[2][k+1]-xx2*xx2*xx2*xx2*S[2][k])/(24.0*hk);
       xx1=(b-S[3][k]);
       xx2=(S[3][k+1]-b);
       y2=S[0][k]*xx1*xx1/2.0-S[1][k]*xx2*xx2/2.0+(xx1*xx1*xx1*xx1*S[2][k+1]-xx2*xx2*xx2*xx2*S[2][k])/(24.0*hk);
       toplam+=(y2-y1);
       //System.out.println("şart 5 toplam = "+toplam+"y1="+y1+"y2="+y2);
       }
       else break;
    }
  return toplam;
  }

  public static double intSpline(double xi[],double yi[],double c0,double cn,double a,double b)
  {double S[][]=cubic_spline(xi,yi,c0,cn);
   return intSpline(S,a,b);
  }

  public static void main(String args[]) throws IOException
  {
  double x[];
  double y[];
  String s1="a.txt";
  //JFileChooser fc=new JFileChooser();
  //if (fc.showOpenDialog(null) == JFileChooser.APPROVE_OPTION) {File file = fc.getSelectedFile();s1=file.getName();
}
  double a[][]=Text.readDoubleT(s1);
  x=a[0];
  y=a[1];
  int n=x.length;
  double S[][]=cubic_spline(x,y,2.0,2.0);
  System.out.println("s="+intSpline(S,0.0,10.0));
  }
  }
```

Following data is taken to integrate

```
0 0
1 1
2 4
5 25
6 36
7 49
```

```
8  64
9 81
10 100
```

Sample data is created from function $f(x)=x^2$. Therefore integration between 0 and 10 will give 333.3333333 as result.

Cubic spline direct data integration

```
---------- Capture Output ----------
> "C:\co\java\bin\java.exe" SCO11FB
s=333.3333333333334

> Terminated with exit code 0.
```

7.9 DERIVATION

In order to approximate derivation difference equations can be used.Difference equations can basically be derived from Taylor formulations

$$f(x_{i+1}) = f(x_i) + f'(x_i)h + \frac{f"(x_i)}{2!}h^2 + \frac{f^{(3)}(x_i)}{3!}h^3 + \frac{f^{(4)}(x_i)}{4!}h^4 + \cdots$$

If the first derivative is solved from this equation it is found as

$$f'(x_i) = \frac{f(x_{i+1}) - f(x_i)}{h} - \frac{f"(x_i)}{2!}h - \frac{f^{(3)}(x_i)}{3!}h^2 - \frac{f^{(4)}(x_i)}{4!}h^3 + O(h^4)$$

Or

$$f'(x_i) = \frac{f(x_{i+1}) - f(x_i)}{h} + O(h)$$

Term O(h) is the order of error. Error is proportional to the difference parameter h. When h is getting smaller, error wil also shrinked. If the second derivative is written the same way:

$$f"(x_i) = \frac{f(x_{i+2}) - 2f(x_{i+1}) + f(x_i)}{h^2} + O(h)$$

In order to decrease the error, if the second derivative difference equation is substituted into the first derivative Taylor equation a first derivative equation with a second order error can be obtained.

$$f'(x_i) = \frac{f(x_{i+1}) - f(x_i)}{h} - \frac{f"(x_i)}{2!}h + O(h^2)$$

$$f'(x_i) = \frac{f(x_{i+1}) - f(x_i)}{h} - \frac{f(x_{i+2}) - 2f(x_{i+1}) + f(x_i)}{2!\,h^2}h + O(h^2)$$

Difference equation can be written as forward or backward way. The backward difference equation:

$$f'(x_i) = \frac{f(x_i) - f(x_{i-1})}{h} + O(h)$$

By taken avarage of forward and bakward formulas can create a central formula

$$f'(x_i) = \frac{\frac{f(x_{i+1}) - f(x_i)}{h} + \frac{f(x_i) - f(x_{i-1})}{h}}{2} = \frac{f(x_{i+1}) - f(x_{i-1})}{2h} + O(h^2)$$

Difference formulas for calculating derivatives are given (as coefficients) in the following tables

Derivative (Difference) equations
Central finite difference

Derivative	Accuracy	−5	−4	−3	−2	−1	0	1	2	3	4	5
1	2					−1/2	0	1/2				
	4				1/12	−2/3	0	2/3	−1/12			
	6			−1/60	3/20	−3/4	0	3/4	−3/20	1/60		
	8		1/280	−4/10	1/5	−4/5	0	4/5	−1/5	4/105	−1/280	
2	2					1	−2	1				
	4				−1/12	4/3	−5/2	4/3	−1/12			
	6			1/90	−3/20	3/2	−49/18	3/2	−3/20	1/90		
	8		−1/560	8/315	−1/5	8/5	−205/72	8/5	−1/5	8/315	−1/560	
3	2				−1/2	1	0	−1	1/2			
	4			1/8	−1	13/8	0	−13/8	1	−1/8		
	6		−7/240	3/10	−169/120	61/30	0	−61/30	169/120	−3/10	7/240	
4	2				1	−4	6	−4	1			
	4			−1/6	2	−13/2	28/3	−13/2	2	−1/6		
	6		7/240	−2/5	169/60	−122/1	91/8	−122/1	169/60	−2/5	7/240	
5	2			−1/2	2	−5/2	0	5/2	−2	1/2		
	4		1/6	−3/2	13/3	−29/6	0	29/6	−13/3	3/2	−1/6	
	6	−13/28	19/36	−87/3	13/2	−323/4	0	323/48	−13/2	87/32	−19/36	13/28
6	2			1	−6	15	−20	15	−6	1		
	4		−1/4	3	−13	29	−75/2	29	−13	3	−1/4	
	6	13/240	−19/24	87/16	−39/2	323/8	−1023/2	323/8	−39/2	87/16	−19/24	13/24

Forward finite difference

Derivative	Accuracy	0	1	2	3	4	5	6	7	8
1	1	−1	1							
	2	−3/2	2	−1/2						
	3	−11/6	3	−3/2	1/3					
	4	−25/12	4	−3	4/3	−1/4				

	5	−137/60	5	−5	10/3	−5/4	1/5			
	6	−49/20	6	−15/2	20/3	−15/4	6/5	−1/6		
2	1	1	−2	1						
	2	2	−5	4	−1					
	3	35/12	−26/3	19/2	−14/3	11/12				
	4	15/4	−77/6	107/6	−13	61/12	−5/6			
	5	203/45	−87/5	117/4	−254/9	33/2	−27/5	137/180		
	6	469/90	−223/10	879/20	−949/18	41	−201/10	1019/180	−7/10	
3	1	−1	3	−3	1					
	2	−5/2	9	−12	7	−3/2				
	3	−17/4	71/4	−59/2	49/2	−41/4	7/4			
	4	−49/8	29	−461/8	62	−307/8	13	−15/8		
	5	−967/120	638/15	−3929/40	389/3	−2545/24	268/5	−1849/120	29/15	
	6	−801/80	349/6	−18353/120	2391/10	−1457/6	4891/30	−561/8	527/30	−469/240
4	1	1	−4	6	−4	1				
	2	3	−14	26	−24	11	−2			
	3	35/6	−31	137/2	−242/3	107/2	−19	17/6		
	4	28/3	−111/2	142	−1219/6	176	−185/2	82/3	−7/2	
	5	1069/80	−1316/15	15289/60	−2144/5	10993/24	−4772/15	2803/20	−536/15	967/240

Backward finite difference

Derivative	Accuracy	0	-1	-2	-3	-4	-5	-6	-7	-8
1	1	1	-1							
	2	3/2	-2	1/2						
	3	11/6	-3	3/2	-1/3					
	4	25/12	-4	3	-4/3	1/4				
	5	137/60	-5	5	-10/3	5/4	-1/5			
	6	49/20	-6	15/2	-20/3	15/4	-6/5	1/6		

2	1	1	−2	1						
	2	2	−5	4	−1					
	3	35/12	−26/3	19/2	−14/3	11/12				
	4	15/4	−77/6	107/6	−13	61/12	−5/6			
	5	203/45	−87/5	117/4	−254/9	33/2	−27/5	137/180		
	6	469/90	−223/10	879/20	−949/18	41	−201/10	1019/180	−7/10	
3	1	1	-3	3	1					
	2	5/2	-9	12	-7	3/2				
	3	17/4	-71/4	59/2	-49/2	41/4	-7/4			
	4	49/8	-29	461/8	-62	307/8	-13	15/8		
	5	967/120	-638/15	3929/40	-389/3	2545/24	-268/5	1849/120	-29/15	
	6	801/80	-349/6	18353/120	-2391/10	1457/6	-4891/30	561/8	-527/30	469/240
4	1	1	−4	6	−4	1				
	2	3	−14	26	−24	11	−2			
	3	35/6	−31	137/2	−242/3	107/2	−19	17/6		
	4	28/3	−111/2	142	−1219/6	176	−185/2	82/3	−7/2	
	5	1069/80	−1316/15	15289/60	−2144/5	10993/24	−4772/15	2803/20	−536/15	967/240

In order to Show utilisation of the table, the derivative of f(x)=sin(x)

y=f(x)=sin(x)

Derivative	Accuracy	-5	-4	-3	-2	-1	0	1	2	3	4	5
1	2	0.00000000	0.00000000	0.00000000	0.00000000	-0.50000000	0.00000000	0.50000000	0.00000000	0.00000000	0.00000000	0.00000000
	4	0.00000000	0.00000000	0.00000000	0.08333333	-0.66666667	0.00000000	0.66666667	-0.08333333	0.00000000	0.00000000	0.00000000
	6	0.00000000	0.00000000	-0.01666667	0.15000000	-0.75000000	0.00000000	0.75000000	-0.15000000	0.01666667	0.00000000	0.00000000
	8	0.00000000	0.00357143	-0.03809524	0.20000000	-0.80000000	0.00000000	0.80000000	-0.20000000	0.03809524	-0.00357143	0.00000000

Δx	0.01	-5	-4	-3	-2	-1	0	1	2	3	4	5		
x	0	-0.05	-0.04	-0.03	-0.02	-0.01	0	0.01	0.02	0.03	0.04	0.05		
y=f(x)	Accuracyy	-0.049979	-0.039989	-0.029996	-0.019998667	-0.01	0	0.009999833	0.0199987	0.0299955	0.0399893	0.049979	Σ	f'(x)
	2	0	0	0	0	0.0049999	0	0.004999917	0	0	0	0	0.01	0.999983
	4	0	0	0	-0.001666556	0.0066666	0	0.006666556	-0.001667	0	0	0	0.01	1
	6	0	0	0.0004999	-0.0029998	0.0074999	0	0.007499875	-0.003	0.000499925	0	0	0.01	1
	8	0	-0.000143	0.0011427	-0.003999733	0.0079999	0	0.007999867	-0.004	0.001142686	-0.000143	0	0.01	1

Derivative	Accuracy	-5	-4	-3	-2	-1	0	1	2	3	4	5
	2					1	-2	1				
	4				-0.083333333	1.3333333	-2.5	1.333333333	-0.083333			
2	6			0.0111111	-0.15	1.5	-2.722222	1.5	-0.15	0.011111111		
	8		-0.001786	0.0253968	-0.2	1.6	-2.847222	1.6	-0.2	0.025396825	-0.001786	

Δx	0.01	-5	-4	-3	-2	-1	0	1	2	3	4	5		
x	0	-0.05	-0.04	-0.03	-0.02	-0.01	0	0.01	0.02	0.03	0.04	0.05		
y=f(x)	Accuracy	-0.049979	-0.039989	-0.029996	-0.019998667	-0.01	0	0.009999833	0.0199987	0.0299955	0.0399893	0.049979	Σ	f''(x)
	2	0	0	0	0	-0.01	0	0.009999833	0	0	0	0	0	0
	4	0	0	0	0.001666556	-0.013333	0	0.013333111	-0.001667	0	0	0	-2.2E-19	-2.2E-17
	6	0	0	-0.000333	0.0029998	-0.015	0	0.01499975	-0.003	0.000333283	0	0	2.71E-19	2.71E-17
	8	0	7.141E-05	-0.000762	0.003999733	-0.016	0	0.015999733	-0.004	0.00076179	-7.14E-05	0	2.17E-19	2.17E-17

Derivative	Accuracy	-5	-4	-3	-2	-1	0	1	2	3	4	5
	2	0	0	0	-0.5	1	0	-1	0.5	0	0	0
3	4	0	0	0.125	-1	1.625	0	-1.625	1	-0.125	0	0
	6	0	-0.029167	0.30000000	-1.40833333	2.03333333	0.00000000	-2.03333333	1.40833333	-0.30000000	0.02916667	0

Δx	0.0001	-5	-4	-3	-2	-1	0	1	2	3	4	5		Third
x	0	-0.05	-0.04	-0.03	-0.02	-0.01	0	0.01	0.02	0.03	0.04	0.05		Derivative
y=f(x)	Accuracy	-0.049979	-0.039989	-0.0299955	-0.019998667	-0.0099998	0	0.009999833	0.01999867	0.0299955	0.03998933	0.049979	Σ	f'''(x)
	2	0	0	0	0.009999333	-0.0099998	0	-0.009999833	0.00999933	0	0	0	-1E-06	-0.999975
	4	0	0	-0.0037494	0.019998667	-0.0162497	0	-0.016249729	0.01999867	-0.003749438	0	0	-1E-06	-1
	6	0	0.0011664	-0.0089987	0.028164789	-0.020333	0	-0.020332994	0.02816479	-0.00899865	0.00116636	0	-1E-06	-1

A derivative calculating program is given as an example

Program 7.9-1 derivative(usage of difference equations)

```
//=====================================================
// Numerical Analysis package in java
// example to show utilisation of integration (integral)
// and differentiation (derivative) functions
//Derivative formulas
// Dr. Turhan Coban
// =====================================================
import java.io.*;

class f1 extends f_x
{ double func(double x)
  {return x*x*x*x*x;}
}

class NA81
{
public static double derivative(f_x f,double x)
{
double h0=0.001;
int i,m;
int n=7;
double T[][]=new double[n][n];
double h[];
h=new double[n];
h[0]=h0;
double r=0.5;
for( i=1;i<n;i++) {h[i]=h0*Math.pow(r,i);}
for(i=0;i<n;i++)  {T[i][0]=dfdx(f,x,h[i]);}
for(m=1;m<n;m++)
{ for(i=0;i<n-m;i++)
  {T[i][m]=(h[i]*h[i]*T[i+1][m-1]-h[i+m]*h[i+m]*T[i][m-1])/(h[i]*h[i]- h[i+m]*h[i+m]);}
}
double xx=T[0][n-1];
```

```
return xx;
}

public static double dfdx_richardson(f_x f,double x)
{
double h0=0.01;
double h=h0;
int i,m;
int n=10;
double T[][]=new double[n][n];
for(i=0;i<n;i++)  {T[i][0]=dfdx(f,x,h);h/=2.0;}
for(m=1;m<n;m++)
{ for(i=0;i<n-m;i++)
  {T[i][m]=(4.0/3.0*T[i+1][m-1]-1.0/3.0*T[i][m-1]);}
}
double xx=T[0][n-1];
return xx;
}


public static double dfdx(f_x f,double x)
{ double h=0.00001;
  return dfdx(f,x,h);
}

public static double dfdx(f_x f,double x,double h)
{ double hh=1.0/h;
  return (-f.func(x+2.0*h)+8.0*f.func(x+h)-8.0*f.func(x-h)+f.func(x-2.0*h))/12.0*hh;
}

public static double d2fdx2(f_x f,double x)
{ double h=0.00001;
  return d2fdx2(f,x,h);
}

public static double d2fdx2(f_x f,double x,double h)
{
  double hh=1.0/h;
            return (-f.func(x+2.0*h)+16.0*f.func(x+h)-30.0*f.func(x)+16.0*f.func(x-h)-f.func(x-2.0*h))/(12.0)*hh*hh;}

public static double d3fdx3(f_x f,double x)
{ double h=0.00001;
  return d3fdx3(f,x,h);}

public static double d3fdx3(f_x f,double x,double h)
{  double hh=1.0/h;
            return (-f.func(x+3.0*h)+8.0*f.func(x+2.0*h)-13.0*f.func(x+h)+13.0*f.func(x-h)-8.0*f.func(x-2.0*h)+f.func(x-
3.0*h))/(8.0)*hh*hh*hh;}

  public static void main(String args[])
  {
  f1 b1=new f1();
  System.out.println("derivative dfdx    : "+dfdx(b1,1.0));
  System.out.println("derivative d2dfx2  : "+d2fdx2(b1,1.0));
  System.out.println("derivative d3dfx3  : "+d3fdx3(b1,1.0));

  System.out.println("derivative dfdx    : "+derivative(b1,1.0));
```

```
  System.out.println("derivative dfdx    : "+dfdx_richardson(b1,1.0));
  }
}
```

derivative(usage of difference equations)

```
---------- Capture Output ----------
> "D:\java\bin\javaw.exe" NA81
derivative dfdx    : 4.999999999989272
derivative d2dfx2  : 20.00000674332961
derivative d3dfx3  : 60.14633235906783
derivative dfdx    : 5.000000000009831
derivative dfdx    : 4.99999999994733
> Terminated with exit code 0.
```

The derivative formula coefficients can be calculated numerically by using Taylor expansion. The general derivation formula is applied into abstract class f_x in order to achieve a general derivation format of any given function. Algorithm used in here is given by Bength Fornberg[47] article "Generation of Finite Difference Formulas on Arbitrary Spaced Grids" an has the following form:

$$\left.\frac{d^m f(x)}{dx^m}_{x=x_0} \approx \sum_{\nu=0}^{n} \delta_{n,\nu}^m f(x+\alpha_\nu h) m = 0,1,\ldots,M; n = m, m+1,\ldots,N\right|$$

Enter M,x_0,α_0 ,α_1 ,α_2 ,α_3 ,... ,α_N

$$\delta_{0,0}^0 := 1$$
$$c1 := 1$$

```
for n:=1 to N do
   c2:=1
   for ν:=0 to n-1 do
      c3:=αn - αν
      c2:=c2*c3
     ifn ≤ Mthenδ^n_{n-1,ν} := 0
      For m:=0 to min(n,M) do
          δ^m_{n,ν} := ((αn − x0)δ^n_{n-1,ν} − mδ^{m-1}_{n-1,ν})/c3
          next m
       next ν
   For m:=0 to min(n,M) do
         δ^m_{n,n} := (c1/c2)[mδ^{m-1}_{n-1,n-1} − (α_{n-1} − x0)δ^m_{n-1,n-1}]
          next m
   c1:=c2
  next n
```

```
 import static java.lang.Math.*;
import java.io.*;
import java.util.*;
import javax.swing.*;
import javax.swing.table.*;
import java.awt.*;
import static java.lang.Math.*;
  // single function single independent variable
  // example f=x*x
```

```
  // includes full set of derivatives
  // Reference :"Generation of Finite Difference Formulas on Arbitrary Spaced Grids",
  // Bength Fornberg, Mathematics of Computation, Volume 51, Number 184, October 1988
  // pages 699-706
  //interface (empty function) version
@FunctionalInterface
interface if_x
{  public double func(double x);
//first order derivative
default double dfunc(double x)
{double h=1.0e-4;
 int n=1;
 int M=10;
 return dfunc(x,n,M,h);
}
//second order derivative
default double dfunc2(double x)
{double h=1.0e-3;
 int n=2;
 int M=10;
 return dfunc(x,n,M,h);
}
default double dfunc(double x,int N,int Mi,double hi)
{// order of the maximum derivative
 // N order of derivative
 // M degree of difference formula
double c[][][];
double alpha[];
double h;
int M=Mi;
double a[]=new double[0];
h=0.01;
double x0=0;
double alphai[]={0,1,-1,2,-2,3,-3,4,-4,5,-5,6,-6,7,-7,8,-8,9,-9,10,-10,11,-11,12,-12,13,-13,14,-14,15,
-15,16,-16,17,-17,18,-18,19,-19,20,-20,21,-21,22,-22,23,-23,24,-24,25,-25,26,-26,27,-27,28,-28,29,-29,30,-30,
31,-31,32,-32,33,-33,34,-34,35,-35,36,-36,37,-37,38,-38,39,-39,40,-40,41,-41,42,-42,43,-43,44,-44,45,-45,46,
-46,47,-47,48,-48,49,-49,50,-50,51,-51,52,-52,53,-53,54,-54,55,-55,56,-56,57,-57,58,-58,59,-59,60,-60,
-61,61,-62,62,-63,63,-64,64,-65,65,-66,66,-67,67,-68,68,-70,70,-71,71,-72,72,-73,73,-74,74,-75,75,
-76,76,-77,77,-78,78,-79,79,-80,80,-81,81,-82,82,-83,83,-84,84,-85,85,-86,86,-87,87,
-88,88,-89,89,-90,90,-91,91,-92,92,-93,93,-94,94,-95,95,-96,96,-97,97,-98,98,-99,99,-100,100};
alpha=alphai;
int N1=alpha.length-1;
// M   degree of highest derivative
// N+1 number of coefficients
double delta[][][]=new double[N1+1][N1+1][M+1];
double c1,c2,c3;
delta[0][0][0]=1.0;
c1=1.0;
for(int n=1;n<=N1;n++)
{  c2=1;
   for(int nu=0;nu<=(n-1);nu++)
   {c3=alpha[n]-alpha[nu];
    c2=c2*c3;
    if(n<=M) delta[n-1][nu][n]=0.0;
    for(int m=0;m<=Math.min(n,M);m++)
    {
       if(m==0)
       { delta[n][nu][m]=((alpha[n]-x0)*delta[n-1][nu][m])/c3;}
       else
       {delta[n][nu][m]=((alpha[n]-x0)*delta[n-1][nu][m]-m*delta[n-1][nu][m-1])/c3;}
    }//next m
   }//next nu
   for(int m=0;m<=Math.min(n,M);m++)
   {   if(m==0)
```

```
                {delta[n][n][m]=c1/c2*(-(alpha[n-1]-x0)*delta[n-1][n-1][m]);}
                else
                {delta[n][n][m]=c1/c2*(m*delta[n-1][n-1][m-1]-(alpha[n-1]-x0)*delta[n-1][n-1][m]);}
   }//next m
   c1=c2;
}//next n
c=delta;
  if(Mi<N) M=N;
  else M=Mi;
  h=hi;
  double deriv=0;
  double h1=1/h;
  double h2=1;
  for(int j=0;j<N;j++)
  {h2*=h1;}
  for(int i=0;i<c[0].length;i++)
  {   deriv+=c[M][i][N]*func(x+alpha[i]*h);}
  return deriv*h2;}
default double dfunc(double x,int N)
{int M=30;double h=0.01*N*N;return dfunc(x,N,M,h);}
default double[][] gauss_legendre_coefficients(double x1,double x2,int n)
{   //calculates legendre gauss-coefficients as coefficients of the integral
            //for n terms
            double EPS=3.0e-18;
            int m,j,i;
            double z1,z,xm,xl,pp,p3,p2,p1;
    //double x[]=new double[n];
    //double w[]=new double[n];
    double a[][]=new double[2][n];//a[0][i]=x[i]   a[1][i]=w[i]
            m=(n+1)/2;
            xm=0.5*(x2+x1);
            xl=0.5*(x2-x1);
            for (i=1;i<=m;i++)
            { z=Math.cos(Math.PI*((i-0.25)/(n+0.5)));
                        do {
                                    p1=1.0;
                                    p2=0.0;
                                    for (j=1;j<=n;j++) {
                                                p3=p2;
                                                p2=p1;
                                                p1=((2.0*j-1.0)*z*p2-(j-1.0)*p3)/j;
                                    }
                                    pp=n*(z*p1-p2)/(z*z-1.0);
                                    z1=z;
                                    z=z1-p1/pp;
                        } while (Math.abs(z-z1) > EPS);
                        a[0][i-1]=xm-xl*z;
                        a[0][n-i]=xm+xl*z;
                        a[1][i-1]=2.0*xl/((1.0-z*z)*pp*pp);
                        a[1][n-i]=a[1][i-1];
            }
return a;
}
default double integral(double x1,double x2)
{ int n=30;
  return integral(x1,x2,n);
}
default double integral(double x1,double x2,int n)
{
//n : number of integral coefficients
// this routine first generates gauss legendre coefficients
// for [x1,x2] band
// then calculates gauss legendre integral
double a[][]=new double[2][n];
```

```
a=gauss_legendre_coefficients(x1,x2,n);
double z=0;
for(int i=0;i<n;i++)
{ z+=a[1][i]*func(a[0][i]);
}
return z;
}
default double bisection(double a,double b)
  {double b1=1.1*b;
   double r=(a+b)/2.0;
   double eps=1.0e-6;
   int nmax=100;
   int i=1;
   while(Math.abs(func(r))>eps && i<nmax)
   {if(func(a)*func(r)<0) b=r;
    else a=r;
    r=(a+b)/2.0;;
    i++;
   }
   if(i>=nmax) r=bisection(a,b1);
   return r;
  }
default double secant(double x)
{ double eps=1.0e-10;
  double y=1.0;
  double dy=0.0;
  int miter=100;
  int i=0;
  while(abs(y)>eps && i<miter)
  {y=func(x);
   dy=dfunc(x);
   x=x-y/dy;
   i++;
  }
  if(i>=miter)
    System.out.println("i="+i+"results may not be valid");
  return x;
}
default double illinois(double xa,double xu)
{
//Modified False position root finding: Illinois method
double test;
double p=0;
double es,ea;
double fa,fp,fu;
int maxit=100,iter=0;
es=0.000001;
ea=1.1*es;
double xold;
int ia=0,iu=0;
fa=func(xa);
fu=func(xu);
p=xu;
double xpold;
while(iter<maxit && ea>es)
{
xpold=p;
p=xu-fu*(xa-xu)/(fa-fu);
fp=func(p);
if(signum(fp)!=signum(fu))
{xa=xu;fa=fu;}
else
{fa=fa/2;}
xu=p;fu=fp;
```

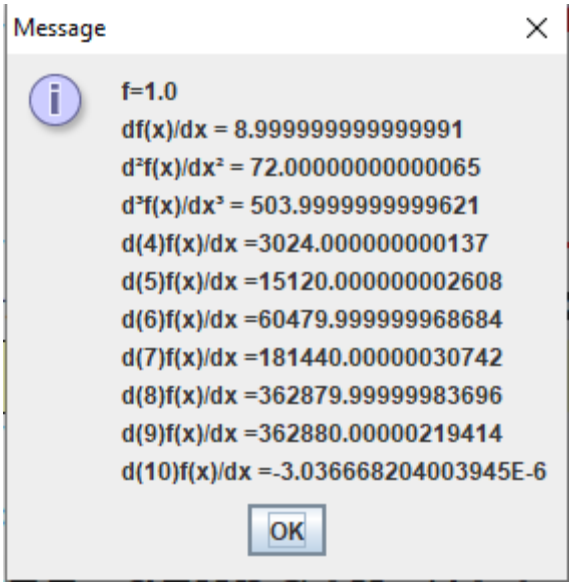
```
iter++;
if(p!=0) ea=abs((p-xpold)/p)*100.0;
}
if(iter>=maxit) System.out.println("Maximum number of iteration is exceeded ");
return p;
}
default double newton_bisection(double x1, double x2)
{
double xacc=1.0e-10; //accuracy
double h=0.0001; //derivative function step
int MAXIT=100;  //Maximum number of iterations
int j;
double dfr,dx,dxold,fr,fh,fl;
double temp,b,a,r;
fl=func(x1);
fh=func(x2);
//Check if a root is existed in the given region
if ((fl > 0.0 && fh > 0.0) || (fl < 0.0 && fh < 0.0))
System.out.println("There are no root in the given region or double root?");
if (fl == 0.0) return x1;
if (fh == 0.0) return x2;
if (fl < 0.0)   { a=x1;b=x2;}
else            { b=x1;a=x2;}
r=0.5*(x1+x2); //midpoint value
dxold=Math.abs(x2-x1);
dx=dxold;
fr=func(r);   //function value at midpoint
dfr=dfunc(r); //derivative of function value at midpoint
for (j=1;j<=MAXIT;j++)
{
if ((((r-b)*dfr-fr)*((r-a)*dfr-fr) > 0.0) || (Math.abs(2.0*fr) > Math.abs(dxold*dfr)))
   { //bisection step
                dxold=dx;
     dx=0.5*(b-a);
     r=a+dx;
     if (a == r) return r; //çözüm!!!!
   }
else{ //Newton-Raphson (Secant) step
                dxold=dx;
     dx=fr/dfr;
     temp=r;
     r -= dx;
     if (temp == r) return r;  //solution!!!!!
   }
if (Math.abs(dx) < xacc) return r; //solution!!!!!
fr=func(r);
dfr=dfunc(r);
if (fr < 0.0)
a=r;
else
b=r;
}
System.out.println("Maximum number of iterations are exceeded");
return 0.0; //program should never reach here
          //dummy return
}
default double brent(double a,double b)
{
double test;
double p=0;
double es,ea;
double f1,f2,f3,fp;
int maxit=500,iter=0;
double tol=1.0e-15;
```

```
es=0.0000001;
double x1=a;f1=func(x1);
double x2=b;f2=func(x2);
double x3=(x1+x2)/2.0;f3=func(x3);
if(f1==0) return x1;
else if(f2==0) return x2;
else if(f3==0) return x3;
if(f1*f2>0) System.out.println("No root is existed in the given region");
p=-(f2*f3*x1*(f2-f3)+f3*f1*x2*(f3-f1)+f1*f2*x3*(f1-f2))/((f1-f2)*(f2-f3)*(f3-f1));
fp=func(p);
ea=Math.abs(x3-p);
while((ea>es)&&(iter<maxit))
{               if(Math.abs(f3)<tol) return x3;
   if((p<x1) && (p>x2))
   {p=(x1+x2)/2.0;
    if(p>x3) {x1=x3;f1=f3;x3=p;f3=fp;}
    else if(p<x3) {x2=x3;f2=f3;x3=p;f3=fp;}
   }
   else
   {
   if(p>x3) {x1=x3;f1=f3;x3=p;f3=fp;}
   else if(p<x3) {x2=x3;f2=f3;x3=p;f3=fp;}
   p=-(f2*f3*x1*(f2-f3)+f3*f1*x2*(f3-f1)+f1*f2*x3*(f1-f2))/((f1-f2)*(f2-f3)*(f3-f1));
   fp=func(p);
   ea=Math.abs(x3-p);
   }
   iter++;
}
if(iter>=maxit) JOptionPane.showMessageDialog(null,"Warning : MAximum number of iteration is exceeded \n"+
  " result may not be valid","MAXIMUM ITERATION NUMBER WARNING",JOptionPane.WARNING_MESSAGE);
return p;
}
default double enlarge(double x0,double dx)
{  //enlarge region untill a root is existed
               double x1=x0;
               double x2=x1+dx;
               int NTRY=200;
               double a[]=new double[2];
               double FACTOR=1.001;
               int j;
               double f1,f2;
               if (x1 == x2) System.out.println("Giriş değerleri yanlış verildi");
               f1=func(x1);
               f2=func(x2);
               for (j=1;j<=NTRY;j++)
               { if (f1*f2 < 0.0) {break;}
                 else {x2+=dx;f2=func(x2);}
               }
               return x2;
}
}
```

```
 import javax.swing.*;
import static java.lang.Math.*;
public class if_xtest {

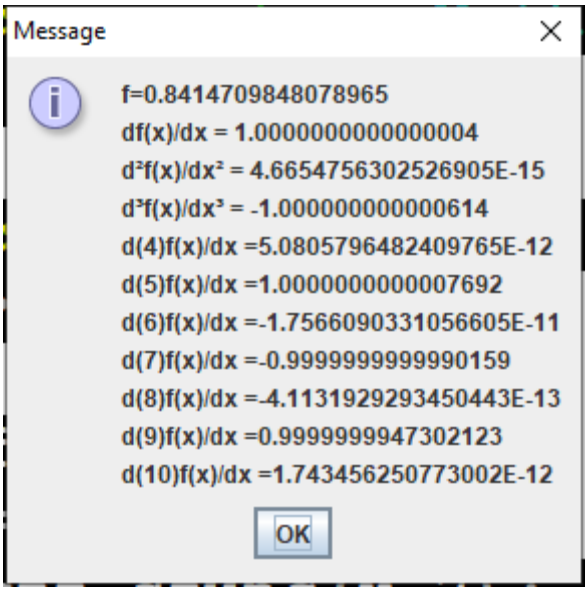
public static void main(String arg[])
{ if_x ff=x->x*x*x*x*x*x*x*x*x;
double x=1;
  double df=ff.dfunc(x,1);
  double d1f=ff.dfunc(x);
  double d2f=ff.dfunc(x,2);
  double d3f=ff.dfunc(x,3);
  double d4f=ff.dfunc(x,4);
```

```
 double d5f=ff.dfunc(x,5);
 double d6f=ff.dfunc(x,6);
 double d7f=ff.dfunc(x,7);
 double d8f=ff.dfunc(x,8);
 double d9f=ff.dfunc(x,9);
 double d10f=ff.dfunc(x,10);
 String s1="f="+ff.func(1.0)+"\ndf(x)/dx = "+df+"\nd"+\u00B2'+"f(x)/dx"+\u00B2'+" = "+d2f+
 "\nd"+\u00B3'+"f(x)/dx"+\u00B3'+" = "+d3f+"\nd(4)f(x)/dx ="+d4f+"\nd(5)f(x)/dx ="+d5f+"\nd(6)f(x)/dx ="+d6f+
 "\nd(7)f(x)/dx ="+d7f+"\nd(8)f(x)/dx ="+d8f+"\nd(9)f(x)/dx ="+d9f+"\nd(10)f(x)/dx ="+d10f;
 JOptionPane.showMessageDialog(null,s1);
}
}
```




```
import javax.swing.*;
import static java.lang.Math.*;
public class if_xtest {

public static void main(String arg[])
{ if_x ff=x->sin(x);
double x=0;
  double df=ff.dfunc(x,1);
  double d1f=ff.dfunc(x);
  double d2f=ff.dfunc(x,2);
  double d3f=ff.dfunc(x,3);
  double d4f=ff.dfunc(x,4);
  double d5f=ff.dfunc(x,5);
  double d6f=ff.dfunc(x,6);
  double d7f=ff.dfunc(x,7);
  double d8f=ff.dfunc(x,8);
  double d9f=ff.dfunc(x,9);
  double d10f=ff.dfunc(x,10);
 String s1="f="+ff.func(1.0)+"\ndf(x)/dx = "+df+"\nd"+\u00B2'+"f(x)/dx"+\u00B2'+" = "+d2f+
 "\nd"+\u00B3'+"f(x)/dx"+\u00B3'+" = "+d3f+"\nd(4)f(x)/dx ="+d4f+"\nd(5)f(x)/dx ="+d5f+"\nd(6)f(x)/dx ="+d6f+
 "\nd(7)f(x)/dx ="+d7f+"\nd(8)f(x)/dx ="+d8f+"\nd(9)f(x)/dx ="+d9f+"\nd(10)f(x)/dx ="+d10f;
 JOptionPane.showMessageDialog(null,s1);
}
}
```



Derivative coefficients for order of the highest derivative M = 4

Δx	df(x)/dx	d²f(x)/dx²	d³f(x)/dx³	d(4)f(x)/dx4
0.0	-0.0	-2.5	-0.0	6.0
1.0	0.6666666666666666	1.3333333333333333	-1.0	-4.0
-1.0	-0.6666666666666666	1.3333333333333335	1.0	-4.0
2.0	-0.08333333333333333	-0.08333333333333333	0.5	1.0
-2.0	0.08333333333333333	-0.08333333333333333	-0.5	1.0

Derivative coefficients for order of the highest derivative M = 6

Δx	df(x)/dx	d²f(x)/dx²	d³f(x)/dx³	d(4)f(x)/dx4	d(5)f(x)/dx5	d(6)f(x)/dx6
0.0	-0.0	-2.7222222222...	-0.0	9.3333333333...	-0.0	-20.0
1.0	0.75	1.5	-1.625	-6.5	2.5	15.0
-1.0	-0.75	1.5	1.625	-6.5	-2.5	15.0
2.0	-0.15	-0.15	1.0	2.0	-2.0	-6.0
-2.0	0.1500000000...	-0.15	-1.0	2.0	2.0	-6.0
3.0	0.0166666666...	0.0111111111...	-0.125	-0.1666666666...	0.5	1.0
-3.0	-0.0166666666...	0.0111111111...	0.125	-0.1666666666...	-0.5	1.0

Derivative coefficients for order of the highest derivative M = 8

Δx	df(x)/dx	d²f(x)/dx²	d³f(x)/dx³	d(4)f(x)/dx4	d(5)f(x)/dx5	d(6)f(x)/dx6	d(7)f(x)/dx7	d(8)f(x)/dx8
0.0	-0.0	-2.84722222222...	-0.0	11.375	-0.0	-37.5	-0.0	70.0
1.0	0.8	1.6	-2.03333333333...	-8.13333333333...	4.833333333333...	29.0	-7.0	-56.0
-1.0	-0.79999999999...	1.599999999999...	2.033333333333...	-8.13333333333...	-4.83333333333...	29.0	7.0	-56.0
2.0	-0.19999999999...	-0.19999999999...	1.408333333333...	2.816666666666...	-4.33333333333...	-13.0	7.0	28.0
-2.0	0.200000000000...	-0.19999999999...	-1.40833333333...	2.816666666666...	4.333333333333...	-13.0	-7.0	28.0
3.0	0.038095238095...	0.025396825396...	-0.3	-0.39999999999...	1.499999999999...	3.0	-3.0	-8.0
-3.0	-0.03809523809...	0.025396825396...	0.3	-0.39999999999...	-1.5	3.0	3.0	-8.0
4.0	-0.00357142857...	-0.00178571428...	0.029166666666...	0.029166666666...	-0.16666666666...	-0.24999999999...	0.5	1.0
-4.0	0.003571428571...	-0.00178571428...	-0.02916666666...	0.029166666666...	0.166666666666...	-0.24999999999...	-0.5	1.0

Derivative coefficients for order of the highest derivative M = 10

Δx	df(x)/dx	d²f(x)/dx²	d³f(x)/dx³	d(4)f(x)/dx4	d(5)f(x)/dx5	d(6)f(x)/dx6	d(7)f(x)/dx7	d(8)f(x)/dx8	d(9)f(x)/dx9	d(10)f(x)/dx10
0.0	-0.0	-2.9272222222...	-0.0	12.741666666...	-0.0	-51.15	-0.0	154.0	-0.0	-252.0
1.0	0.8333333333...	1.6666666666...	-2.3180555555...	-9.2722222222...	6.7291666666...	40.375	-15.75	-126.0	21.0	210.0
-1.0	-0.8333333333...	1.6666666666...	2.3180555555...	-9.2722222222...	-6.7291666666...	40.375	15.75	-126.0	-21.0	210.0
2.0	-0.2380952380...	-0.2380952380...	1.7337301587...	3.4674603174...	-6.5	-19.5	17.0	68.0	-24.0	-120.0
-2.0	0.2380952380...	-0.2380952380...	-1.7337301587...	3.4674603174...	6.5	-19.5	-17.0	68.0	24.0	-120.0
3.0	0.0595238095...	0.0396825396...	-0.4830357142...	-0.6440476190...	2.7187499999...	5.4375	-8.625	-23.0	13.5	45.0
-3.0	-0.0595238095...	0.0396825396...	0.4830357142...	-0.6440476190...	-2.71875	5.4375	8.625	-23.0	-13.5	45.0
4.0	-0.0099206349...	-0.0049603174...	0.0833994708...	0.0833994708...	-0.5277777777...	-0.7916666666...	2.1666666666...	4.3333333333...	-4.0	-10.0
-4.0	0.0099206349...	-0.0049603174...	-0.0833994708...	0.0833994708...	0.5277777777...	-0.7916666666...	-2.1666666666...	4.3333333333...	4.0	-10.0
5.0	7.9365079365...	3.1746031746...	-0.0067791005...	-0.0054232804...	0.0451388888...	0.0541666666...	-0.2083333333...	-0.3333333333...	0.5	1.0
-5.0	-7.9365079365...	3.1746031746...	0.0067791005...	-0.0054232804...	-0.0451388888...	0.0541666666...	0.2083333333...	-0.3333333333...	-0.5	1.0

7.10 RICHARDSON EXTRAPOLATION IN DERIVATION

Richardson extrapolaton process that previously seen in integration can also be applied into derivation. Richardson extrapolation Formula was

$$D \cong D(h_2) + \frac{D(h_2) - D(h_1)}{\left(\frac{h_1}{h_2}\right)^2 - 1}$$

If $h_2=h_1/2$ equation becomes

$$D \cong \frac{4}{3}D(h_2) - \frac{1}{3}D(h_1)$$

This equation can be written in the iterative two dimensional form as:

$$D_{j,k} \cong \frac{4^{k-1}D_{j+1,k-1} - D_{j,k-1}}{4^{k-1} - 1}$$

7.11 PAD'E APPROXIMATIONS

One of the most accurate derivative approximations can be derived by using Taylor series procedures. If the functional values at certain points are given Formula can be include the derivatives as well. For example, we can ask for the most accurate formula that includes f'_j, f'_{j+1} and f'_{j-1}
in addition to the functional values f_j , f_{j+1}, and f_{j-1} in addition to the functional values f_j , f_{j+1}, and f_{j-1}. we would write

$$f'_j + a_0 f_{j+1} + a_1 f_{j+1} + a_2 f_{j-1} + a_1 f'_{j+1} + a_1 f'_{j-1} = O(?)$$

And our task is then find the five coefficients $a_0, a_1, \ldots, a_4$ to maximize
the order of this approximation.
The Taylor table is

TAYLOR TABLE FOR A PAD'E SCHEME

	f_j	f_j'	f_j''	$f_j^{(3)}$	$f_j^{(4)}$	$f_j^{(5)}$
f_j'	0	1	0	0	0	0
$a_0 f_j$	a_0	0	0	0	0	0
$a_1 f_{j+1}$	a_1	$a_1 h$	$a_1 h^2/2$	$a_1 h^3/6$	$a_1 h^4/24$	$a_1 h^5/120$
$a_2 f_{j-1}$	a_2	$-a_2 h$	$a_2 h^2/2$	$-a_2 h^3/6$	$a_2 h^4/24$	$-a_2 h^5/120$
$a_3 f_{j+1}$	0	a_3	$a_3 h$	$a_3 h^2/2$	$a_3 h^3/6$	$a_3 h^4/24$
$a_4 f_{j-1}$	0	a_4	$-a_4 h$	$a_4 h^2/2$	$-a_4 h^3/6$	$a_4 h^4/24$

we now sum all the rows and set as many of the lower order terms to zero as possible. We have five coefficients and can set the sum of the entries in columns 2 to 6 to zero. The linear equations for the coefficients are

$a_0 + a_1 + a_2 = 0$

$a_1 h - a_2 h + a_3 + a_4 = -1$

$$a_1 \frac{h^2}{2} + a_2 \frac{h^2}{2} + a_3 h - a_4 h = 0$$
$$a_1 \frac{h}{3} - a_2 \frac{h}{3} + a_3 + a_4 = 0$$
$$a_1 \frac{h}{4} + a_2 \frac{h}{4} + a_3 - a_4 = 0$$

The solution set is:
$a_0=0$ $a_1 = -\frac{3}{4h}$ $a_2 = \frac{3}{4h}$ $a_3 = a_4 = \frac{1}{4}$
This solution will lead to Padé Formula for numerical differential as

$$f_{j+1}^{'} + f_{j-1}^{'} + 4f_j^{'} = \frac{3}{h}(f_{j+1} - f_{j-1}) + \frac{h^4}{30}f_j^{(5)}$$

This equation requires two more equation to solve complete set of differentiation. Additional equations can be taken as:

$$f_0^{'} + 2f_1^{'} = \frac{1}{h}(-\frac{5}{2}f_0 + 2f_1 + \frac{1}{2}f_2)$$

$$f_n^{'} + 2f_{n-1}^{'} = \frac{1}{h}(\frac{5}{2}f_n + 2f_{n-1} - \frac{1}{2}f_{n-2})$$

In matrix form equations can be written as

$$\begin{bmatrix} 1 & 2 & 0 & 0 & 0 & 0 & 0 \\ 1 & 4 & 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 4 & 1 & 0 & 0 & 0 \\ .. & .. & .. & .. & .. & .. & .. \\ .. & .. & .. & .. & .. & .. & .. \\ & & & & 1 & 4 & 1 \\ & & & & & 2 & 1 \end{bmatrix} \begin{bmatrix} f_0^{'} \\ f_1^{'} \\ f_2^{'} \\ .. \\ .. \\ f_{n-1}^{'} \\ f_n^{'} \end{bmatrix} = \begin{bmatrix} -\frac{5}{2}f_0 + 2f_1 + \frac{1}{2}f_2 \\ 3(f_2 - f_0) \\ 3(f_3 - f_1) \\ .. \\ .. \\ 3(f_n - f_{n-1}) \\ \frac{5}{2}f_n - 2f_{n-1} - \frac{1}{2}f_{n-2} \end{bmatrix}$$

This matrix is a tridiagonal matrix can be solved by using Thomas algorithm.

Program 7.11-1 derivative by Padé Formula

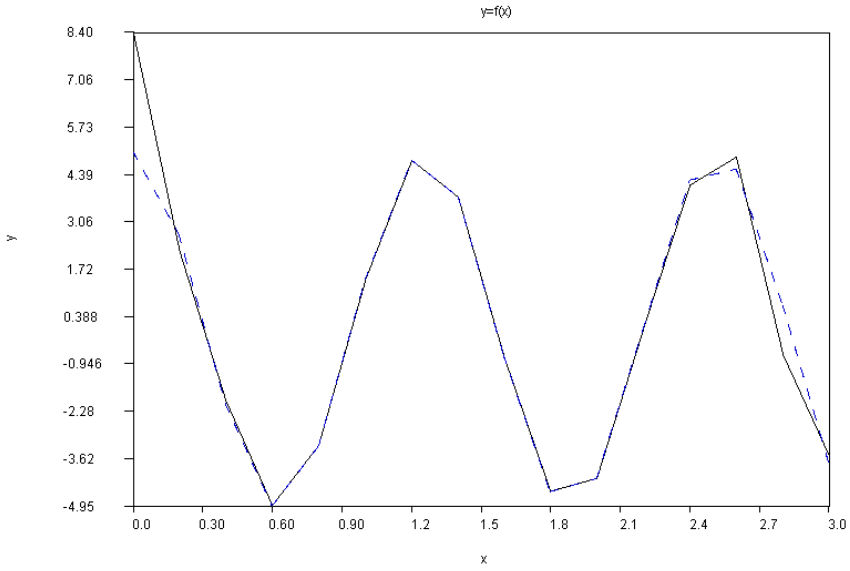
```
import java.io.*;
import javax.swing.*;

class f1 extends f_x
{ double func(double x)
  {return Math.sin(5.0*x);}
}
class f2 extends f_x
{//exact derivative of the function
double func(double x)
  {return 5.0*Math.cos(5.0*x);}
}

class NA81A
{
  public static double [] thomas(double a[][],double r[])
  {
  //Padé formula
  int n=a.length;
  double f[]=new double[n];
  double e[]=new double[n];
  double g[]=new double[n];
  double x[]=new double[n];
  for(int i=0;i<n;i++)     {f[i]=a[i][i];}
  for(int i=0;i<(n-1);i++) {g[i]=a[i][i+1];}
  for(int i=0;i<(n-1);i++) {e[i+1]=a[i+1][i];}
  for(int k=1;k<n;k++)
     {e[k]=e[k]/f[k-1];
      f[k]=f[k]-e[k]*g[k-1];
     }
  for(int k=1;k<n;k++)
     {r[k]=r[k]-e[k]*r[k-1];
     }
  x[n-1]=r[n-1]/f[n-1];
  for(int k=(n-2);k>=0;k--)
     {x[k]=(r[k]-g[k]*x[k+1])/f[k];}
  return x;
  }
```

```
public static double [] thomas(double f[],double e[],double g[],double r[])
{
int n=f.length;
double x[]=new double[n];
for(int k=1;k<n;k++)
   {e[k]=e[k]/f[k-1];
    f[k]=f[k]-e[k]*g[k-1];
   }
for(int k=1;k<n;k++)
   {r[k]=r[k]-e[k]*r[k-1];
   }
x[n-1]=r[n-1]/f[n-1];
for(int k=(n-2);k>=0;k--)
   {x[k]=(r[k]-g[k]*x[k+1])/f[k];}
return x;
}
public static double[][] df(f_x ff,double x0,double xn, int n)
{
int nn=n+1;
double h=(xn-x0)/n;
double y[]=new double[nn];
double x[]=new double[nn];
double f[]=new double[nn];
double g[]=new double[nn];
double e[]=new double[nn];
for(int i=0;i<=n;i++)
{x[i]=x0+h*i;}

for(int i=0;i<=n;i++)
{
 if(i==0) {f[i]=1;}
 else if(i==n) {f[i]=1;}
 else {f[i]=4;}
 if(i==1) {g[i]=2;}
 else if(i==n) {g[i]=0;}
 else {g[i]=1;}
 if(i==(n-1)) {e[i]=2;}
 else if(i==0) {e[i]=0;}
 else {e[i]=1;}
 if(i==0) {y[i]=1.0/h*(-2.5*ff.func(x[0])+2.0*ff.func(x[1])+0.5*ff.func(x[2]));}
 else if(i==n) {y[i]=1.0/h*(2.5*ff.func(x[n])-2.0*ff.func(x[n-1])-0.5*ff.func(x[n-1]));}
 else {y[i]=(1.0/h*3.0*(ff.func(x[i+1])-ff.func(x[i-1])));}
}
 double a[][]=new double[3][nn];
 a[0]=x;
 a[1]=y;
 a[2]=thomas(f,e,g,y);
 return a;
 }
public static void main(String args[]) throws IOException
{
 double a[][]=df(new f1(),0.0,3.0, 15);
 Plot pp=new Plot(a[0],a[2]);
 int n=a[0].length;
 double y[]=new double[n];
 f2 dy=new f2();
 for(int i=0;i<n;i++)
 {y[i]=dy.func(a[0][i]);
  System.out.println(a[0][i]+" "+a[2][i]+" "+y[i]);
 }
 pp.addData(a[0],y);
 pp.plot();
}
}
```



7.12 DIRECT DERIVATION OF THE NUMERICAL DATA

If data is given numerically, two different approach can be used to take the derivative of the function. The first approximation is to curve fit a function into the data and then take the derivative of the function. The second one is to fit difference equations directly to the given data
In order to define a function, interpolation equations can be used. For example if quadratic Lagrange Formula is used:

$$f_2(x) = \frac{(x-x_i)(x-x_{i+1})}{(x_{i-1}-x_i)(x_{i-1}-x_{i+1})} f(x_{i-1}) + \frac{(x-x_{i-1})(x-x_{i+1})}{(x_i-x_{i-1})(x_i-x_{i+1})} f(x_i) + \frac{(x-x_{i-1})(x-x_i)}{(x_{i+1}-x_{i-1})(x_{i+1}-x_i)} f(x_{i+1})$$

Derivative of this function will be:

$$f'_2(x) = \frac{(2x-x_i-x_{i+1})}{(x_{i-1}-x_i)(x_{i-1}-x_{i+1})} f(x_{i-1}) + \frac{(2x-x_{i-1}-x_{i+1})}{(x_i-x_{i-1})(x_i-x_{i+1})} f(x_i) + \frac{(2x-x_{i-1}-x_i)}{(x_{i+1}-x_{i-1})(x_{i+1}-x_i)} f(x_{i+1})$$

The same approach can be applied with different degree of polynomials as well as different interpolation formulations

As another example to function definition (Curve fitting) cubic spline interpolation can be applied. If the cubic polynomial is in the form of:

$$s_k(x)=a_k(x-x_k)+ b_k(x_{k+1}-x)+ [(x-x_k)^3 c_{k+1} +(x_{k+1}-x)^3 c_k]/(6h_k) \quad 1\leq k \leq n$$

The derivative of this equation will be

$$s'_k(x)=a_k- b_k+ [(x-x_k)^2 c_{k+1} -(x_{k+1}-x)^2 c_k]/(2h_k) \quad 1\leq k \leq n$$

Since cubic polynomial has continious first and second polynomials between each spliine function the derivative is also continious. An example program is given below. Data used is generated fron $f(x)=x^2$ fonction..

Program 7.12-1 data derivative with cubic spline interpolation

```
import java.io.*;
import javax.swing.*;
class NA81
{

 public static double [] thomas(double a[][],double r[])
 {
 //
 int n=a.length;
 double f[]=new double[n];
 double e[]=new double[n];
 double g[]=new double[n];
 double x[]=new double[n];
 for(int i=0;i<n;i++)     {f[i]=a[i][i];}
 for(int i=0;i<(n-1);i++) {g[i]=a[i][i+1];}
 for(int i=0;i<(n-1);i++) {e[i+1]=a[i+1][i];}
 for(int k=1;k<n;k++)
   {e[k]=e[k]/f[k-1];
    f[k]=f[k]-e[k]*g[k-1];
   }
 for(int k=1;k<n;k++)
   {r[k]=r[k]-e[k]*r[k-1];
   }
 x[n-1]=r[n-1]/f[n-1];
 for(int k=(n-2);k>=0;k--)
   {x[k]=(r[k]-g[k]*x[k+1])/f[k];}
 return x;
 }

 public static double [] thomas(double f[],double e[],double g[],double r[])
 {
 int n=f.length;
 double x[]=new double[n];
 for(int k=1;k<n;k++)
   {e[k]=e[k]/f[k-1];
    f[k]=f[k]-e[k]*g[k-1];
   }
 for(int k=1;k<n;k++)
   {r[k]=r[k]-e[k]*r[k-1];
   }
 x[n-1]=r[n-1]/f[n-1];
 for(int k=(n-2);k>=0;k--)
   {x[k]=(r[k]-g[k]*x[k+1])/f[k];}
 return x;
 }


 public static double [][] cubic_spline(double xi[],double yi[],double c0,double cn)
 {
 // c0 second derivative of the first point in the first spline
 // cn second derivative of the last point in last spline
 int n=xi.length;
 double h[]=new double[n];
 double w[]=new double[n];
 double f[]=new double[n];
```

```
double e[]=new double[n];
double g[]=new double[n];
double d[]=new double[n];
double x[]=new double[n];
double S[][]=new double[4][n];
int k;
for(k=0;k<(n-1);k++)
  {h[k]=xi[k+1]-xi[k];
   w[k]=(yi[k+1]-yi[k])/h[k];
          }
d[0]=c0;
d[n-1]=cn;
for(k=1;k<(n-1);k++)
  {d[k]=6.0*(w[k]-w[k-1]);}
f[0]=1.0;
f[n-1]=1.0;
g[0]=0.0;
g[n-1]=0.0;
e[0]=0.0;
e[n-1]=0.0;
for(k=1;k<(n-1);k++)
{f[k]=2.0*(h[k]+h[k-1]);e[k]=h[k-1];g[k]=h[k];}
S[2]=thomas(f,e,g,d);
S[3]=xi;
for(k=0;k<(n-1);k++)
 {S[0][k]=(6.*yi[k+1]-h[k]*h[k]*S[2][k+1])/(6.0*h[k]);
  S[1][k]=(6.*yi[k]-h[k]*h[k]*S[2][k])/(6.0*h[k]);
 }
return S;
}

public static double funcSpline(double S[][],double x)
{
int n=S[0].length;
double xx1=0;
double xx2=0;
double y=0;
double hk=0;
for(int k=0;k<(n-1);k++)
  {if(S[3][k]<=x && x<=S[3][k+1])
   {hk=(S[3][k+1]-S[3][k]);
    xx1=(x-S[3][k]);
    xx2=(S[3][k+1]-x);
    y=S[0][k]*xx1+S[1][k]*xx2+(xx1*xx1*xx1*S[2][k+1]+xx2*xx2*xx2*S[2][k])/(6.0*hk);
    break;
    }
  }
    if(y==0 && S[3][n-2]<=x )
    {
            int k=n-2;
            hk=(S[3][k+1]-S[3][k]);
    xx1=(x-S[3][k]);
    xx2=(S[3][k+1]-x);
    y=S[0][k]*xx1+S[1][k]*xx2+(xx1*xx1*xx1*S[2][k+1]+xx2*xx2*xx2*S[2][k])/(6.0*hk);
    }
return y;
}
```

```
 public static double dfSpline(double xi[],double yi[],double c0,double cn,double x)
 { //cubic spline formulations
            double S[][]=cubic_spline(xi,yi,c0,cn);
  return dfSpline(S,x);
 }

 public static double dfSpline(double S[][],double x)
 {
  //cubic spline derivative formulation
 int n=S[0].length;
 double xx1=0;
 double xx2=0;
 double y=0;
 double hk=0;
 for(int k=0;k<(n-1);k++)
   {if(S[3][k]<=x && x<=S[3][k+1])
    {hk=(S[3][k+1]-S[3][k]);
     xx1=(x-S[3][k]);
     xx2=(S[3][k+1]-x);
     y=S[0][k]-S[1][k]+(xx1*xx1*S[2][k+1]-xx2*xx2*S[2][k])/(2.0*hk);
     break;
     }
   }
     if(y==0 && S[3][n-2]<=x )
     {
              int k=n-2;
              hk=(S[3][k+1]-S[3][k]);
     xx1=(x-S[3][k]);
     xx2=(S[3][k+1]-x);
     y=S[0][k]*xx1+S[1][k]*xx2+(xx1*xx1*xx1*S[2][k+1]+xx2*xx2*xx2*S[2][k])/(6.0*hk);
     }
 return y;
 }

public static double  dfSpline(double xi[],double yi[],double x)
{
  //naturaş cubic spline derivative formula
            double S[][]=cubic_spline(xi,yi,0,0);
  return dfSpline(S,x);
}

 public static void main(String args[]) throws IOException
 {
 double x[];
 double y[];
 String s1="a.txt";
 //JFileChooser fc=new JFileChooser();
 //if (fc.showOpenDialog(null) == JFileChooser.APPROVE_OPTION) {File file = fc.getSelectedFile();s1=file.getName();
}
 double a[][]=Text.readDoubleT(s1);
 x=a[0];
 y=a[1];
 int n=x.length;
 double S[][]=cubic_spline(x,y,2.0,2.0);
 System.out.println(dfSpline(S,2.5));
 }
```

```
}
```

Output 7.9-2 data derivative with cubic spline interpolation

```
---------- Capture Output ----------
> "C:\turhan\java\bin\javaw.exe" NA81
5.005944788930356

> Terminated with exit code 0.
```

7.10 PROBLEMS

PROBLEM 1

Calculate integral of **$y=f(x)=1/(1+x^2)$** fonction between limits –1 ile 1 with 3 points Gauss-Legendre integration Formula

Note: Required roots and coefficients :

coefficients	Roots
c_0 0.555555555555553	x_0 0.774596669241483
c_1 0.888888888888889	x_1 0.000000000000000
c_2 0.555555555555553	x_2 0.774596669241483

PROBLEM 2

$$\int_{-3}^{3} \frac{2}{1+2x^2}\,dx$$

solve with bole rule. Use n=2

Note: Bole rule for n=1

I=(b-a)*[7*f(x0)+32*f(x1)+12*f(x2)+32*f(x3)+7*f(x4)]/90

Step size h=(b-a)/n

PROBLEM 3

$$f_{0-\lambda} = \frac{c_1}{\sigma} \int_{x=0}^{\lambda T} \frac{dx}{x^5\left[exp\left(\frac{c_2}{x}\right)-1\right]}$$

Function is known as black body radiation shape function. The values of coefficients in the equation are:

c_1 =3.743 x10^8 W.μm^4/m^2, c_2 =1.4387x10^4 μm.K, σ=5.67x10^{-8} W/(m^2.K^4) dür.

Use Gauss-Legendre integration and for a giving λT calculate black body radiation shape factor.

Create a table for the values of blackbody radiation shape factor from 5000 to infinity(?)

PROBLEM 4

$$I = \int_{x=0}^{1}\int_{y=0}^{1} \frac{dydx}{(1+y)^2}$$

Solve the above integration by using

a) monte-carlo method.
b) Gauss-Legendre integration
c) Simpson 1/3 integration
d) Develop a program for 2 dimensional 4. Order Newton Cotes method(booole)

PROBLEM 5

Solve the integration

$$I = \int_{x=0}^{1} \frac{1}{1+x^2}\,dx$$

a) by analithical methods

b) by Trapezoidal rule with n=2(divide region into 2 parts) and calculate % error

c) Trapezoidal rule with n=2(divide region into 2 parts) and calculate % error

PROBLEM 6

Solve the integration

$$I = \int_{x=0}^{2} \frac{1}{\sqrt{1+x^2}}\,dx$$

By using Gauss-Legendre formulation

PROBLEM 7

For x=1

$$f(x) = ln\left(\frac{1}{1+x^2}\right)$$

a) Calculate analytical derivative
b) Calculate derivative with 3 points central difference formula with h=0.1
c) Calculate derivative with 3 points central difference formula with h=0.05
d) Calculate derivative with Richardson extrapolation formula with h1=0.5 and h2=0.25

PROBLEM 8

$$I = \int_{-1}^{1} xe^x dx$$

Calculate the integration by

a) Trapezoidal rule integrali
b) Simpson 3/8 integration
c) Gauss-Legendre integration
d) Romberg integration

PROBLEM 9

$$I = \int_{0}^{1} sin(1-x+x^2)dx$$

Calculate the integration by

e) Trapezoidal rule integrali
f) Simpson 3/8 integration
g) Gauss-Legendre integration
h) Romberg integration

PROBLEM 10

$$I = \int_{-1}^{1} \frac{ln(1+x^2)}{\sqrt{1-x^2}}\,dx$$

Calculate the integration by

i) Trapezoidal rule integrali
j) Simpson 3/8 integration
k) Gauss-Legendre integration
l) Romberg integration

PROBLEM 11

X	**0.99**	**2.10**	**3.22**	**4.40**	**5.70**	**7.12**	**8.01**	**8.37**	**9.32**	**9.98**
f(x)	**4.90**	**5.70**	**4.20**	**7.04**	**8.31**	**7.82**	**5.97**	**7.01**	**6.68**	**4.79**

Data given in the above table. Find the integration by using cubic spline integration method.

PROBLEM 12

Find the derivative of data given in problem 11 at point x=5

PROBLEM 13

Depth(H) and velocity(U) profile of a channel is given in below table

x, m	0	2	4	6	8	10	12	14	16	18	20
H, m	0	1.8	2	4	4	6	4	3.6	3.4	2.8	0
U m/s	0	0.03	0.045	0.055	0.065	0.12	0.07	0.06	0.05	0.04	0

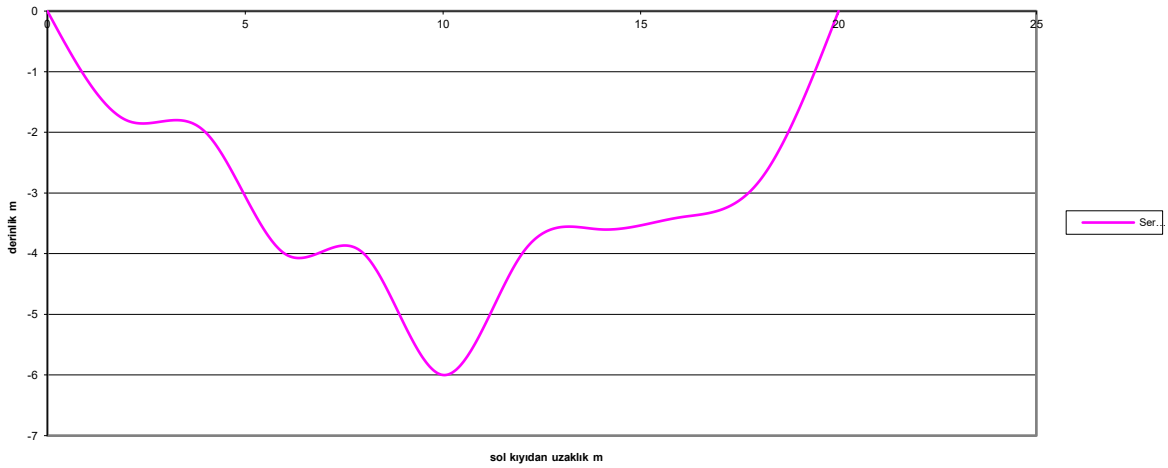
Cross sectiaonal area of the channel can be calculated as :

$$A_c \int_0^B H(x)dx$$

Flow rate of the channel will be

$$Q = \int_0^B U(x)H(x)dx$$

In this equations U is the velocity of the water measured at point x. Calculate cross sectional area and flow rate of the channel.



PROBLEM 14

Temperature profle of a solid body can be given as

$$\frac{dT}{dt} = -k(T - T_a)$$

In this equation T is the temperature, T_a is constant surface temperature.

A solid body with temperatures initially at T=90 C and, T_a =20 C put into a water container at temperature of T_a = 20 C. The temperature profile as a function of time is given at the table.

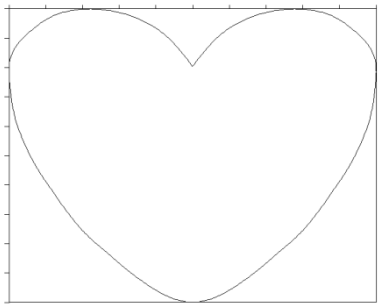
Zaman , t, s	0	5	10	15	20	25	30	35
Sicaklik T C	90	55	40	28	24	22	21.5	20.6

Calculate k thermal conductivity coefficient.

PROBLEM 15

Coordinate of a hard shape is given in the table. Find the area of the hard.

Xi	Yi	xi	yi	xi	yi	xi	Yi
241	127	331	157	215	246	149	120
258	107	323	178	199	232	160	106
283	96	312	197	184	217	175	97
307	97	298	217	170	197	199	96
322	106	283	232	159	178	224	107
333	120	267	246	151	157	241	127
334	138	241	258	148	138		



PROBLEM 16

$$\boldsymbol{I = \int_{0.1}^{0.5} \int_{x^3}^{x^2} e^{y/x} dydx}$$

Solve the integral by Gauss-Legendre method.

PROBLEM 17

$$\boldsymbol{I = \int_{0}^{1} \int_{x}^{2x} \left(x^2 + y^3\right) dydx}$$

Solve the equation with

a) 2 dimensional Gauss-Legendre integration
b) 2 dimensional simpson 1/3 integration
c) Develop an equation for 2 dimensional simpson. 3/8 integration and solve the integral

PROBLEM 18

$$\boldsymbol{I = \int_{x=0}^{1} \int_{y=0}^{1} \frac{dydx}{(1+y)^2}}$$

Solve the integral with Monte carlo method and check the results with Gauss-Legendre method.

PROBLEM 19

Show that equation below is correct

$$\pi = \int_{0}^{2} \sqrt{4 - x^2} dx$$

PROBLEM 20

$$\boldsymbol{I = \int_{0}^{1} e^x dx}$$

Solve the integral for m=25,50,100,500,1000,10000,100000 with monte-carlo method and compare the result with the analytical solution.

PROBLEM 21

An angine of a vehicle with mass of 53400 kg and a velocity of v=30 m/is suddenly stopped at t=0. In this instant equation of the movement can be given as

$$5400v\frac{dv}{dx} = -8276v^2 - 2000$$

In the equation v(m/sn), is the velocity of the vehicle at time t. When velocity is 15 m/s calculate the total distance vehicle moved by using 4 point Gauss integration Formula .

PROBLEM 22

A vehicle is at complete stop. When it is started velocity of the vehicle is recorded for the first 5 minutes. Calculate the total distance vehicle moved by using trapezoidal rule as a direct data integration method.

t(minutes)	0	0,5	1,0	1,5	2,0	2,5	3,0	3,5	4,0	4,5	5,0
v (km/h)	0	4	7	11	15	21	25	30	32	35	40

Note : $\boldsymbol{v(t) = \frac{dx(t)}{dt}}$ $x - x_0 = \int_{t0}^{t} v(t)dt$

PROBLEM 23

Upward velocity of a rocket can be calculated by using the formula

$$V = \frac{dz}{dt} = Uln\frac{m_0}{m_0 - qt} - gt$$

In this equation V is velocity, U is the exit velocity of burning gases from the rocket nozzle, m_o is the mass of the rocket at time t = 0 sn, q is fuel spending rate , g is garavitational acceleration (assume constant as, 9,8 m^2/s). If U = 2000 m/s, m_o = 150 000 kg, q = 2600 kg/s, Calculate the

height rocket will climb in 30 seconds by using Romberg integration with k=4 [$O(h^8)$] order of error.

Note :"

$\boldsymbol{V(t) = \frac{dH(t)}{dt}}$ $H = \int_{t0}^{t=30} V(t)dt$

PROBLEM 24

Calculate velocity and acceleration by using below data with

d) cental difference method [with $O(h^4)$ error level],
e) forward difference method [with $O(h^2)$ error level],
f) backward difference method [with $O(h^2)$ error level],

Time, t (s)	0	1	2	3	4	5	6	7	8
distance, x (m)	0	0.7	1.8	3.4	5.1	6.5	7.3	8.0	8.4

Note : Velocity $\boldsymbol{v(t) = \frac{dx(t)}{dt}}$ Acceleration : $a = \frac{d^2x}{dt^2}$

PROBLEM 25

Integrate data by using trapezoidal rule

x	-3	-1	1	3	5	7	9	11
f(x)	1	-4	-9	2	4	2	6	-3

PROBLEM 26

In a river water height is measured in every 12.5 m distance. Water flow rate can be calculated by Q = ∫ V*dA integration. In this equation Q: water flow rate (m^3/s), V: water flow speed (m/s), A: cross-sectional area (m^2). If avarage water velocity is 0.4 m/s. Calculate water flow rate by using Simpson 1/3 rule

X(m)	0	12,5	25	37,5	50	62,5	75	87,5	100
f(X) m	11	7,5	5,5	0	1,5	4	5,5	8	11

PROBLEM 27

a) $\boldsymbol{\int_0^{10} f(x)dx}$ calculate the integral value from the data given below

b) calculate the derivative of the function at x=3,4,5,6,7,8

X	0.99	2.10	3.22	4.40	5.70	7.12	8.01	8.37	9.32	9.98
f(x)	4.90	5.70	4.20	7.04	8.31	7.82	5.97	7.01	6.68	4.79

PROBLEM 28

By using Gauss-Legendre integral formulation calculate $I = \int_{-4}^{4} \frac{dx}{1+x^2}$ integral for n=2,3,4,5 . Compare the result with analytical solution.

PROBLEM 29

By using **Adaptive** simpson 1/3 solve $I = \int_{-4}^{4} \frac{dx}{1+x^2}$ integral with an error level of 1.0e-5

PROBLEM 30

By using Gauss-Chebychev integration equation solve the integral $I = \int_{-1}^{1} \frac{cos(x)dx}{\sqrt{(1-x^2)}}$ = 2.40394 and compare the result with the exact value.

PROBLEM 31

By using trapezoidal rule calculate the integral $I = \int_0^1 \frac{2dx}{2+sin(10\pi x)} = 1.15470054$ for n=2,4,8 ve 16

And then improve the result by using Richardson extrapolation Formula (Romberg integration).

PROBLEM 32

If velocity profile in a pipe is known, flow rate can be calculated as $Q = \int v\, dA$

In this equation v i the velocity of the fluid A is the cross-sectional area of the pipe. In a circular pipe A=(2 π r) and dA = (2 π r dr). If it is substituted into flow rate equation

$Q = \int v(2\pi r)\, dr$ is obtained. Where r is the radius. If the velocity distribution is given as a function of r as $v = 3[1-(r/r_0)]^{(1/7)}$ and if the radius of the pipe r_0= 0.1 m calculate mass flow rate. Select your own method.

PROBLEM 33

In the integratls below exact solutions are given. By using an adaptive integration method. Chck out the results.

a) $I1 = \int_0^1 \frac{4dx}{1+256(x-0.375)^2} = \frac{1}{4}[tan^{-1}(10) + tan^{-1}(6)] = 0.7191938309210011$

b) $I2 = \int_0^1 x^2\sqrt{x}dx = \frac{2}{7}$

c) $I3 = \int_0^1 \sqrt{x}dx = \frac{2}{3}$

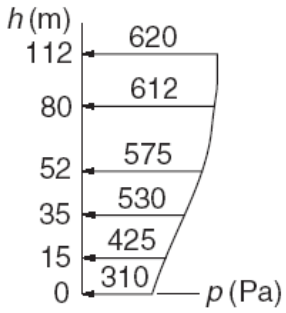
d) $I4 = \int_0^1 log(x)dx = -1$

e) $I5 = \int_0^1 log(|x-0.7|)dx = 0.3 * log(0.3) + 0.7 * log(0.7) - 1.0$

f) $I6 = \int_0^{2\pi} e^{-x} sin(50x)dx = \frac{50}{2501}(1 - e^{-2\pi})$

g) $I6 = \int_{-9}^{10000} \frac{dx}{\sqrt{|x|}} = 206$

PROBLEM 34



Wind pressure in a flat wall is measures as in the graphic. It is desired to find pressure effect center of the wall. It is calculated by using

:

$$\bar{h} = \frac{\int_0^{112} hp(h)dh}{\int_0^{112} p(h)dh}$$

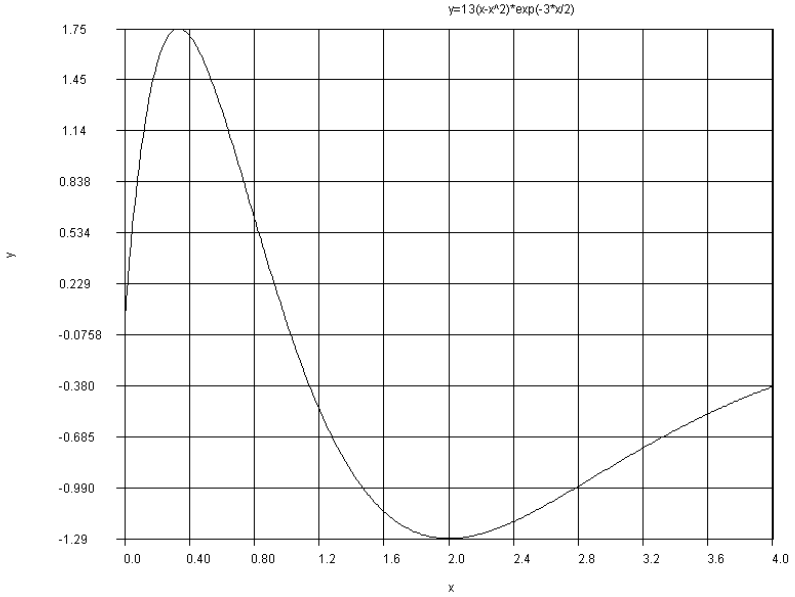
Calculate the value.

PROBLEM 35

For the analytically solved integration that result is given below:
Solve with with

a) Adaptive Simpson 1/3 method
b) Adaptive Gauss-Kronrod integral formula.

$$\int_0^4 13(x-x^2)e^{-3x/2}\,dx = \frac{4108e^{-6}-52}{27}$$
$$= -1.5487883725279481333$$



PROBLEM 36

v(m/s)	0	1	1.8	2.5	3.5	4.4	5.1	6
P(kW)	0	4.7	12.2	19.0	31.8	40.1	43.8	43.2

Power transmitted to the wheels of a vehicle is given in the table as a function of velocty. If the mass of the ar is m=2000 kg, Calculate the required time to accelrate the car from
0 m/s^2 to 6 m/s^2
Note : $\Delta t = m\int_{0s}^{6s}(v/P)dv$

PROBLEM 37

$\boldsymbol{I = \int_0^1 \int_1^2 \int_0^y e^{x+y+z} dzdydx}$ Calculate the integral by using 3 dimensional Gauss-Legendre integration

PROBLEM 38

$\boldsymbol{I = \int_0^{\pi} \int_0^x \int_0^{xy} \frac{1}{y} sin(\frac{z}{y}) dzdydx}$Calculate the integral by using 3 dimensional Gauss-Legendre integration

PROBLEM 39

Blackbody radiation function F(0- λT), λ(μm) T(degree Kelvin) can be calculated as

$$x = \frac{C_2}{\lambda T} = \frac{14387.69\ \mu mK}{\lambda T}$$

$$F(0-\lambda T) = 1 - \frac{15}{\pi^4}\int_0^x \frac{x^3}{e^x-1}dx$$

It can also be calculated from the series solution

$$F(0-\lambda T) = \frac{15}{\pi^4}\sum_{n=1}^{\infty}\left[\frac{e^{-nx}}{n}(x^3+\frac{3x^2}{n}+\frac{6x}{n^2}+\frac{6}{n^3})\right]$$

By creating a program that calculates the result with using series solution and integration. Compare the results. Resulting tables are given as a reference

λ T μ mK	F(0-λT)	λ T	F(0-λT)	λ T μ mK	F(0-λT)	λ T μ mK	F(0-λT)	λ T μ mK	F(0-λT)	λ T μ mK	F(0-λT)
1000	0.00032	2900	0.25056	4800	0.60754	6900	0.8022	10700	0.92709	18000	0.98081
1050	0.00056	2950	0.26191	4850	0.61428	7000	0.80808	10800	0.92872	18200	0.98137
1100	0.00091	3000	0.27323	4900	0.62089	7100	0.81374	10900	0.93031	18400	0.98191
1150	0.00142	3050	0.28453	4950	0.62737	7200	0.81918	11000	0.93185	18600	0.98243
1200	0.00213	3100	0.29578	5000	0.63373	7300	0.82443	11200	0.9348	18800	0.98293
1250	0.00308	3150	0.30697	5050	0.63996	7400	0.82949	11400	0.93758	19000	0.98341
1300	0.00432	3200	0.3181	5100	0.64608	7500	0.83437	11600	0.94021	19200	0.98387
1350	0.00587	3250	0.32915	5150	0.65207	7600	0.83907	11800	0.9427	19400	0.98431
1400	0.00779	3300	0.34011	5200	0.65795	7700	0.8436	12000	0.94505	19600	0.98474
1450	0.01011	3350	0.35097	5250	0.66371	7800	0.84797	12200	0.94728	19800	0.98516
1500	0.01285	3400	0.36173	5300	0.66937	7900	0.85219	12400	0.94939	20000	0.98555
1550	0.01605	3450	0.37238	5350	0.67491	8000	0.85625	12600	0.95139	20500	0.98649
1600	0.01972	3500	0.38291	5400	0.68034	8100	0.86018	12800	0.95329	21000	0.98735
1650	0.02388	3550	0.39332	5450	0.68566	8200	0.86397	13000	0.95509	21500	0.98814
1700	0.02854	3600	0.4036	5500	0.69089	8300	0.86763	13200	0.95681	22000	0.98886
1750	0.03369	3650	0.41375	5550	0.696	8400	0.87116	13400	0.95843	22500	0.98952
1800	0.03934	3700	0.42377	5600	0.70102	8500	0.87457	13600	0.95998	23000	0.99014
1850	0.04549	3750	0.43364	5650	0.70594	8600	0.87787	13800	0.96145	23500	0.9907
1900	0.05211	3800	0.44338	5700	0.71077	8700	0.88105	14000	0.96285	24000	0.99123
1950	0.0592	3850	0.45297	5750	0.7155	8800	0.88413	14200	0.96419	24500	0.99172
2000	0.06673	3900	0.46241	5800	0.72013	8900	0.88711	14400	0.96546	25000	0.99217
2050	0.07469	3950	0.47172	5850	0.72468	9000	0.88999	14600	0.96667	26000	0.99297
2100	0.08306	4000	0.48087	5900	0.72914	9100	0.89278	14800	0.96783	27000	0.99368
2150	0.0918	4050	0.48987	5950	0.73351	9200	0.89547	15000	0.96893	28000	0.99429
2200	0.10089	4100	0.49873	6000	0.73779	9300	0.89808	15200	0.96999	29000	0.99482
2250	0.11031	4150	0.50744	6050	0.74199	9400	0.9006	15400	0.971	30000	0.99529
2300	0.12003	4200	0.516	6100	0.74611	9500	0.90305	15600	0.97196	32000	0.99607
2350	0.13002	4250	0.52442	6150	0.75015	9600	0.90541	15800	0.97289	34000	0.99669
2400	0.14026	4300	0.53269	6200	0.75411	9700	0.9077	16000	0.97377	36000	0.99719
2450	0.15071	4350	0.54081	6250	0.758	9800	0.90992	16200	0.97461	38000	0.99759
2500	0.16136	4400	0.54878	6300	0.76181	9900	0.91207	16400	0.97542	40000	0.99792
2550	0.17217	4450	0.55662	6350	0.76554	10000	0.91416	16600	0.9762	45000	0.99851
2600	0.18312	4500	0.56431	6400	0.76921	10100	0.91618	16800	0.97694	50000	0.9989
2650	0.19419	4550	0.57186	6450	0.7728	10200	0.91814	17000	0.97765	55000	0.99917
2700	0.20536	4600	0.57927	6500	0.77632	10300	0.92004	17200	0.97834	60000	0.99935
2750	0.2166	4650	0.58654	6600	0.78317	10400	0.92188	17400	0.97899		
2800	0.22789	4700	0.59367	6700	0.78976	10500	0.92367	17600	0.97962		
2850	0.23922	4750	0.60067	6800	0.7961	10600	0.9254	17800	0.98023		

PROBLEM 40

Integral of blackbody spectral radiation equation for all wavelengths will give

$$E_b = \left[\frac{C_1}{C_2^4}\int_{x=0}^{\infty}\frac{x^3}{e^x-1}dx\right]T^4 = \sigma T^4$$

In this equation C_1=3.74177489x10^8 Wmm^4/m^2 and C_2=1.438769x10^4 mm K . σ is called Stefan_Boltzman constant. Calculate this integral by using Gauss-Leguerre integral formula. The value of the Stephan Boltsman constant is given σ =5.670x10^{-8} W/m^{2}K^4

PROBLEM 41 Bessel function $J_n(x)$ can be defined as:

$$J_n(x) = \frac{1}{\pi}\int_0^{\pi} cos(x\ sin\ \theta - n\theta)d\theta$$

Write a program to define such a function.

PROBLEM 42 In an AC-DC converter the signal defined with the following function is obtained. Signal frequency is 50 hertz (period T=1.0/50=0.02 second). A true root mean square current value for an alternative current signal is defined as:

$$I_{RMS} = \sqrt{\frac{1}{T}\int_0^{T} i^2(t)dt}$$

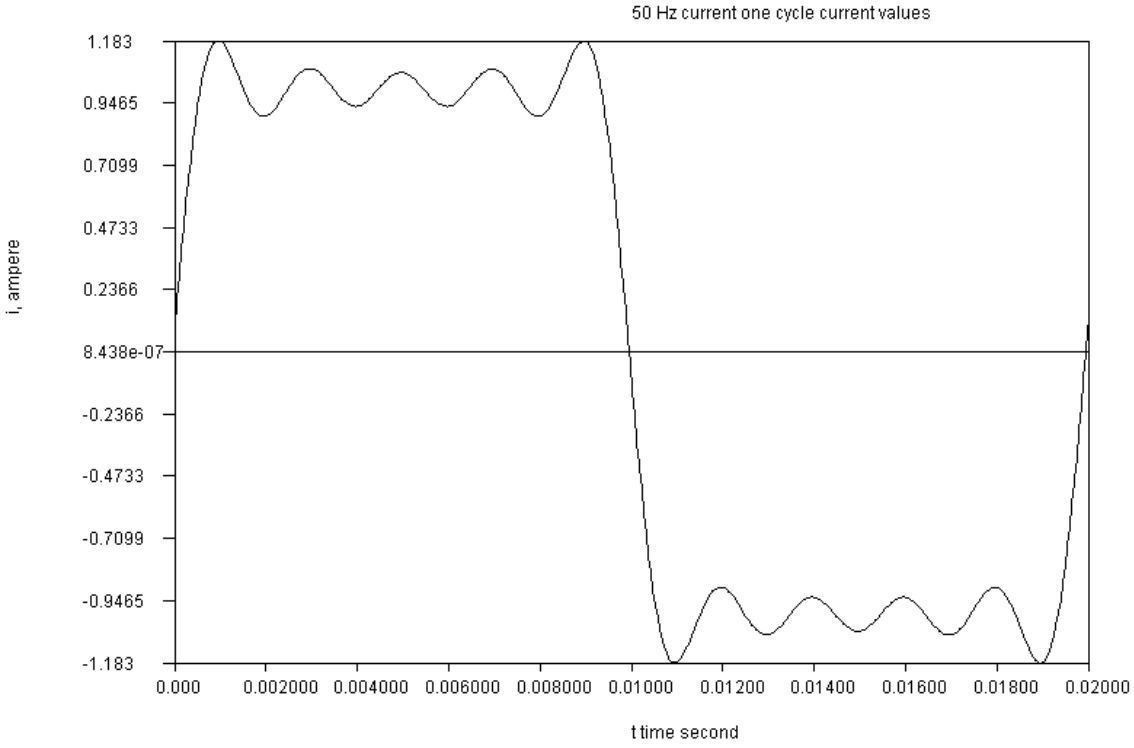
Calculate True RMS current for the given signal.

```
class ff extends f_x
{
double a[]={-8.673617379884035E-17,
-0.02000000000000008,
-3.8163916471489756E-17,
-0.019999999999999792,
-7.45931094670027E-17,
-0.01999999999999994,
-3.122502256758253E-17,
-0.019999999999999858,
1.9255430583342559E-16,
-0.019999999999999983,
9.020562075079397E-17};

double b[]={
0.0,
-1.2731348232574313,
-7.773729576721067E-17,
-0.42409897579377515,
1.1232334506949826E-16,
-0.2541240947234941,
3.447762908503904E-17,
-0.18115773372477864,
4.336808689942018E-18,
-0.14052732458082803,
-2.5110122314764283E-16};
int n=10;
double T=1.0/50.0;
double x0=0;
double xn=T;

public double z_x(double x)
{double zz=2.0*Math.PI*(x-x0)/(xn-x0)-Math.PI;
 return zz;
}
```

```
public  double func(double x)
{
double cc=a[0]/2.0+a[n]*Math.cos(n*z_x(x));
for(int k=1;k<n;k++)
{cc+=a[k]*Math.cos(k*z_x(x))+b[k]*Math.sin(k*z_x(x));}
return cc;
}}
```



PROBLEM 43 Work is defined as

Work =Force*Distance

In a real engineering problem force usually changes, Therefore Work for a variable force should be calculated as:

$$W = \int_{x0}^{xn} F(x)dx$$

In a system Force is measured as in the following table:

x(meter)	**0**	**5**	**10**	**15**	**20**	**25**	**30**	**35**	**40**
F(Newton)	**0**	**9**	**12**	**15**	**18**	**23**	**16**	**13**	**11**

Calculate the work done by the system.

PROBLEM 44 Euler's constant is defined by the following equations:

$$\gamma = \int_{x=0}^{1} \left(\frac{1}{1+x^k} - e^{-x}\right)\frac{dx}{x} = \sum_{n=1}^{\infty}\left(\frac{1}{n} - ln\left(\frac{n+1}{n}\right)\right) \text{ where } \quad k > 0$$

$\gamma = 0.57721566490153286060651209008240243104215933593992...$

Calculate **euler's** constant by integration and compare it with the series solution. You chose different integration methods to see the accuracy of the integration formula.

PROBLEM 45 Catalan's constant is defined by the following equations:

$$K = \int_{x=0}^{1} \left(\frac{ln(x)}{1+x^2}\right) dx$$

$$K = \frac{3}{64}\sum_{k=0}^{\infty} \frac{(-1)^k}{64^k}\left(\frac{32}{(12k+1)^2} - \frac{32}{(12k+2)^2} \frac{32}{(12k+3)^2} \frac{8}{(12k+5)^2} - \frac{16}{(12k+6)^2} - \frac{4}{(12k+7)^2} - \frac{4}{(12k+9)^2} - \frac{2}{(12k+10)^2} + \frac{1}{(12k+11)^2}\right)$$

K=0.915965594177…

Calculate **Catalan's** constant by integration and compare it with the series solution. You chose different integration methods to see the accuracy of the integration formula.

9.0 DIFFERENTIAL EQUATIONS

9.1 EULER EQUATION



Figure 8.1.1 Leonhard Euler

It is desired the find solution of (y values) differential equation

$\frac{dy}{dx} = f(x, y)$

The first simplest method to solve differential equation numerically is the euler equation. Solution set in euler equation can be written as:

$$\frac{y_{i+1} - y_i}{h} \approx f(x, y)$$

$y_{i+1} = y_i + f(x_i, y_i)h$

In the equation h is step size. H can be calculated as $h = \Delta x = x_{i+1} - x_i$

Our example problem is solved for different step sizes as h=1, h=0.5 and h=0.1 The results clearly show that when h is getting smaller, euler solution and exact solution is getting closer to each other.

Program 8.1-1 Euler equation

```
 //=================================================
// Numerical Analysis package in java
// example to show differential equation solution
// and differentiation (derivative) functions
// Dr. Turhan Coban
// =================================================
// Basit diferansiyel denklemler dy/dx=f(x,y)

import java.io.*;

abstract class f_xy
{
  //single function single independent variable
  // example f=x*x
  abstract double func(double x,double y);
}

class fxy extends f_xy
{ //diferansiyeli alinan fonksiyon
  double func(double x,double y)
  {return -2*x*x*x+12*x*x-20*x+8.5;}
}

class fxyKC extends f_x
{ //Gerçek çözüm fonksiyonu
  public double func(double x)
  {return -0.5*x*x*x*x+4.0*x*x*x-10.0*x*x+8.5*x+1.0;}
}

class diferansiyel
```

```
{
 public static double[][] euler(f_xy f,double x0,double y0,double xn,double h)
 {
 int n=(int)((xn-x0)/h);
 System.out.println("n="+n);
 double a[][]=new double[2][n+1];
 int i=0;
 a[0][0]=x0;
 a[1][0]=y0;
 for(double xx=x0;xx<xn;xx+=h)
 for(i=0;i<n;i++)
  { xx=x0+i*h;
  a[0][i+1]=xx+h;
  a[1][i+1]=a[1][i]+f.func(a[0][i],a[1][i])*h;
  }
 return a;
 }

 public static void main(String args[]) throws IOException
 {
  double x0=0;
  double y0=1.0;
  fxy f1=new fxy();
  fxyKC fKC=new fxyKC();
  double a[][]=euler(f1,x0,y0,4.0,0.01);
  Text.print(Text.T(a));
  Plot p1=new Plot(a);
  //p1.setPlotType(0,24);
  //p1.addSpline(a,5);
  p1.addFunction(fKC,0.0,4.0,200);
  p1.plot();
 }
}
```

Output 8.1-1 Euler equation h=1

0.0 1.0

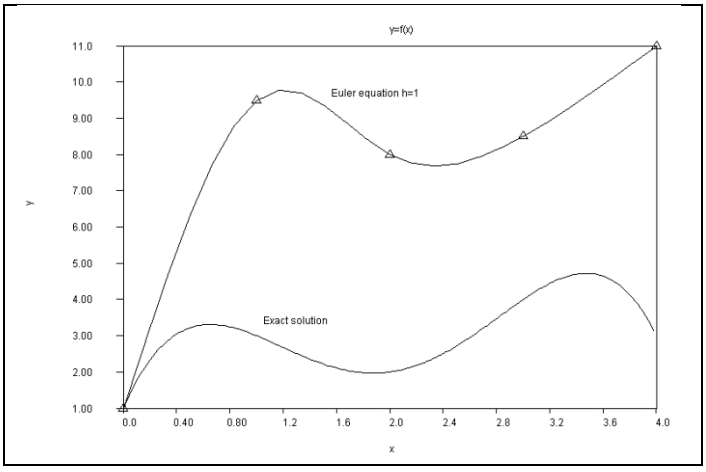
1.0 9.5

2.0 8.0

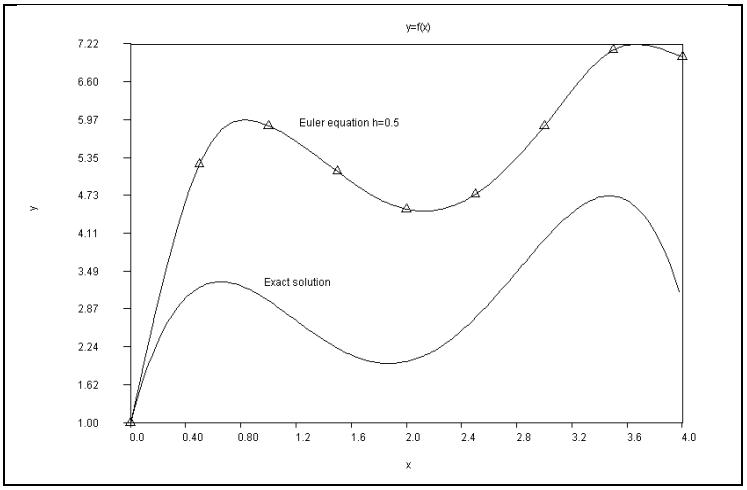
3.0 8.5

4.0 11.0

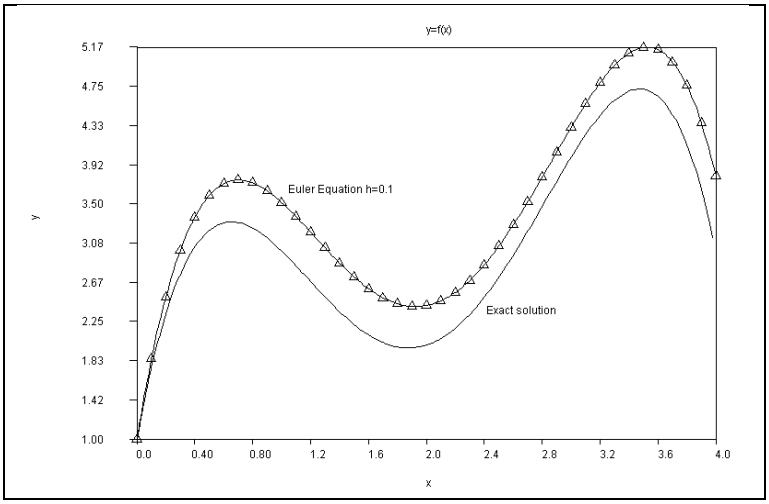
Output 8.1-2 Euler equation h=1



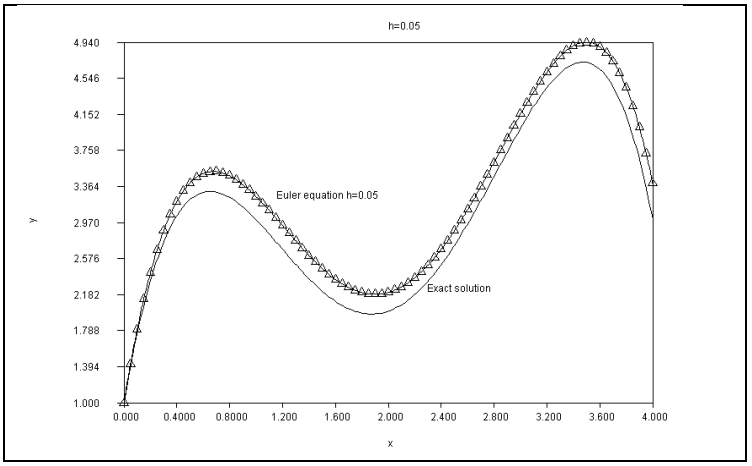
Output 8.1-3 Euler equation h=0.5



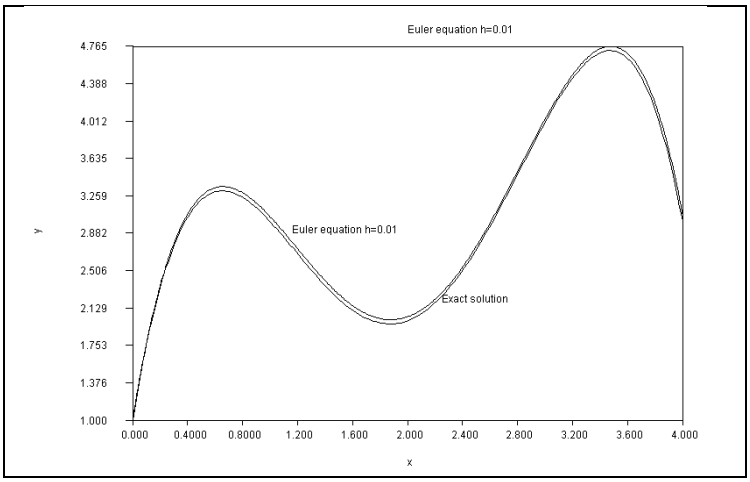
Output 8.1-4 Euler equation h=0.1



Output 8.1-5 Euler equation h=0.05



Output 8.1-6 Euler equation h=0.01



9.2 HEUN EQUATION

Heun equation is similar to euler equation. It is added a correction term into the euler equation

$y^0_{i+1}=y_i+f(x_i,y_i)h$

Corection term

$$y_{i+1}^{k+1} = y_i + \frac{f(x_i,y_i^k)+f(x_{i+1},y_{i+1}^k)}{2} h, k = 0\ldots n$$

The term y^k_{i+1} can be calculated iteratively. Number of iteration can be changed.

Program 8.1-1 Heun equation

```
//=====================================================
// Numerical Analysis package in java
// example to show differential equation solution
// and differentiation (derivative) functions
// Dr. Turhan Coban
// =====================================================
// Basit diferansiyel denklemler dy/dx=f(x,y)

import java.io.*;

abstract class f_xy
{
  //single function single independent variable
  // example f=x*x
  abstract double func(double x,double y);
}


class fxy extends f_xy
{
  double func(double x,double y)
  {return -2*x*x*x+12*x*x-20*x+8.5;}
}

class fxyKC extends f_x
{
  double func(double x)
  {return -0.5*x*x*x*x+4.0*x*x*x-10.0*x*x+8.5*x+1.0;}
}

class fxy1 extends f_xy
{
  double func(double x,double y)
```

```
  {return 4.0*Math.exp(0.8*x)-0.5*y;}
}

class diferansiyelA
{

  public static double[][] heun(f_xy f,double x0,double y0,double xn,double h)
  {
  int n=(int)((xn-x0)/h);
  System.out.println("n="+n);
  double a[][]=new double[2][n+1];
  int i=0;
  a[0][0]=x0;
  a[1][0]=y0;
  double yi0=0;
  for(double xx=x0;xx<xn;xx+=h)
   {a[0][i+1]=xx+h;
    yi0=a[1][i]+f.func(a[0][i],a[1][i])*h;
    for(int j=0;j<5;j++)
    {yi0=a[1][i]+(f.func(a[0][i],a[1][i])+f.func(a[0][i+1],yi0) )/2.0; }
     a[1][i+1]=a[1][i]+(f.func(a[0][i],a[1][i])+f.func(a[0][i+1],yi0) )/2.0;
   i++;
   }
  return a;
  }

  public static void main(String args[]) throws IOException
  {
   double x0=0;
   double y0=1.0;
   fxy f1=new fxy();
   fxyKC fKC=new fxyKC();
   double a[][]=heun(f1,x0,y0,4.0,0.1);
   Text.print(Text.T(a));
   Plot p1=new Plot(a);
   p1.setPlotType(0,24);
   p1.addSpline(a,8);
   p1.addFunction(fKC,0.0,4.0,200);
   p1.plot();
  }
}
```

Output 8.1-1 Heun equation, step h=1 created data

0.0 1.0

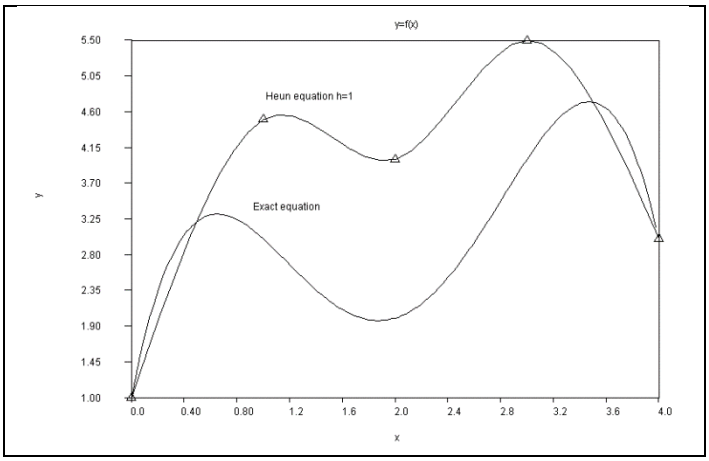
1.0 4.5

2.0 4.0

3.0 5.5

4.0 3.0

Output 8.1-2 Graphic output for Heun equation, step h=1



9.3 MIDPOINT METHOD

In this method a midpoint is calculated first, then by using derivative at the mid point the real point is calculated.

$y_{i+1/2} = y_i + f(x_i , y_i)h/2$

$y_{i+1} = y_i + f(x_{i+1/2} , y_{i+1/2})h$

Program 8.3-1 Mid Point equation

```
/=======================================================
// Numerical Analysis package in java
// example to show differential equation solution
// and differentiation (derivative) functions
// Dr. Turhan Coban
// =======================================================
// Basit diferansiyel denklemler dy/dx=f(x,y)

import java.io.*;

abstract class f_xy
{
  //single function single independent variable
  // example f=x*x
  abstract double func(double x,double y);
}

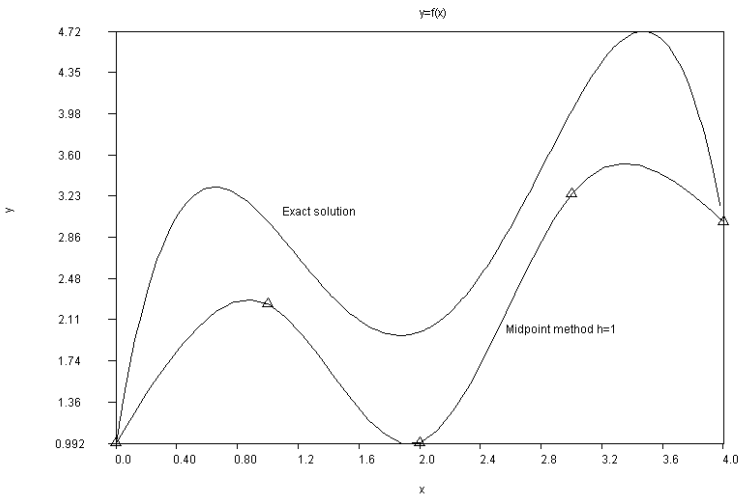
class fxy extends f_xy
{
  double func(double x,double y)
  {return -2*x*x*x+12*x*x-20*x+8.5;}
}


class NA84
{
  public static double[][] polygon(f_xy f,double x0,double y0,double xn,double h)
  {
  int n=(int)((xn-x0)/h);
  System.out.println("n="+n);
  double a[][]=new double[2][n+1];
  int i=0;
  a[0][0]=x0;
  a[1][0]=y0;
  double yi0=0;
  double xi0;
  for(double xx=x0;xx<xn;xx+=h)
   {a[0][i+1]=xx+h;
```

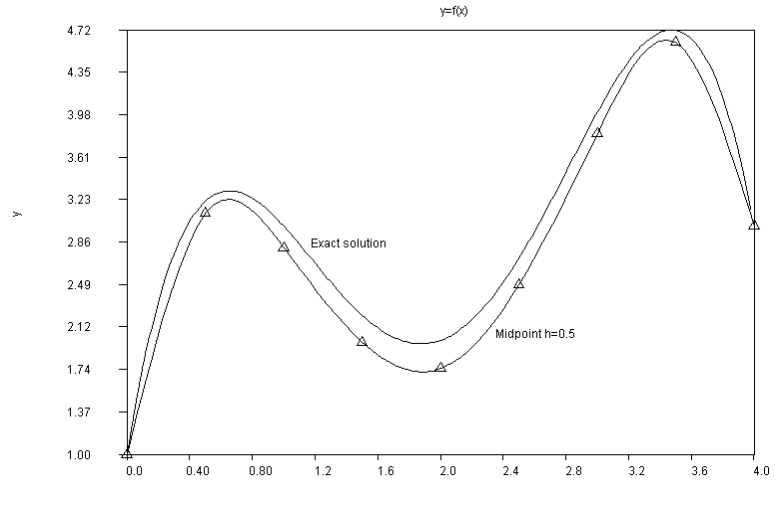
```
   xi0=xx+h/2.0;
  yi0=a[1][i]+f.func(a[0][i],a[1][i])*h/2.0;
  a[1][i+1]=a[1][i]+f.func(xi0,yi0)*h;
  i++;
   }
 return a;
 }

 public static void main(String args[]) throws IOException
 {
  double x0=0;
  double y0=1.0;
  fxy f1=new fxy();
  fxyKC fKC=new fxyKC();
  double a[][]=polygon(f1,x0,y0,4.0,1.0);
  Text.print(Text.T(a));
  Plot p1=new Plot(a);
  p1.setPlotType(0,24);
  p1.addSpline(a,8);
  p1.addFunction(fKC,0.0,4.0,200);
  p1.plot();
 }
}
```

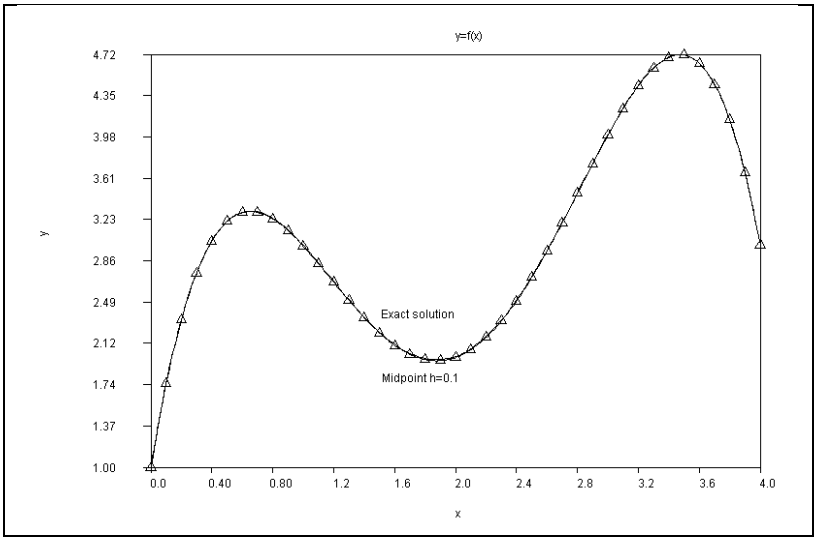
Output 8.3-1 Midpoint method h=1



Output 8.3-2 Midpoint method, h=0.5



Output 8.3-3 Orta nokta ve geliştirilmiş polinom equation, h=0.1



9.4 RUNGE KUTTA METHODS

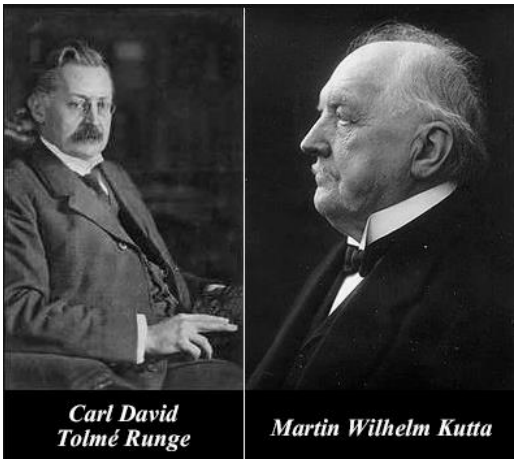


Figure 8.4-1 Carl Davis Tolme Runge and Martin Wilhelm Kutta

Runge-Kutta method as in most general form can be given as:
For differential form

$$\frac{dy}{dx} = f(x, y)$$

$$y_{n+1} = y_n + h\phi(x_n, y_n, h)$$

Where h=Δx is step size and

$$\phi(x_n, y_n, h) = \sum_{i=1}^{m} b_i k_i$$

Where k_i is m dimensional vector calculated as

$$x_i = x_n + c_i h \quad i = 1..m$$

$$y_i = y_n + h \sum_{j=1}^{i-1} a_{ij}\, k_j$$

$$k_i = f(x_i, y_i) = f\left(x_n + c_i h \ , y_n + h \sum_{j=1}^{i-1} a_{ij}\, k_j \right) \quad i = 1..m$$

or in open form

$$k_1 = f(x_i + c_1 h, y_i)$$
$$k_2 = f(x_i + c_2 h, y_i + a_{21} k_1 h)$$
$$k_3 = f(x_i + c_3 h, y_i + a_{31} k_1 h + a_{32} k_2 h)$$
$$\dots\ \dots$$
$$k_m = f\left(x_i + c_m h, y_i + a_{m1} k_1 h + a_{m2} k_2 h + .. + a_{m,m-1} k_{m-1} h\right)$$

In this equation a is constant values depends on polynamial degree of Runge-Kutta equation and coefficients
In numerical analysis books, coefficients are usually shown as Buthcher tableu which puts the coefficients of the method in a table as follows:

$$\begin{array}{c|ccccc} c_1 & a_{11} & a_{12} & a_{13} & \dots & a_{1s} \\ c_2 & a_{21} & a_{22} & a_{23} & \dots & a_{2s} \\ c_3 & a_{31} & a_{32} & a_{33} & \dots & a_{3s} \\ \vdots & & \vdots & \vdots & \dots & \vdots \\ c_s & a_{s1} & a_{s2} & a_{s3} & \dots & a_{s4} \\ \hline & b_1 & b_2 & b_3 & \dots & b_s \end{array}$$

As it is seen from the equation the definition is very general. Depends on polynamial degree of Runge-Kutta equation accuracy will improve. For example Runge-Kutta equation derived by using **a second degree polynomial (RK2)** has the following form

$$y_{n+1} = y_n + h(0.5k_1 + 0.5k_2)$$
$$k_1 = f(x_i, y_i)$$
$$k_2 = f(x_i + h, y_i + k_1 h)$$

This equation can be given as Buthcher tableu as:

$$\begin{array}{c|cc} 0 & 0 & 0 \\ 1 & 1 & 0 \\ \hline & 0.5 & 0.5 \end{array}$$

Second degree Ralston method:

$$y_{n+1} = y_n + h(\frac{1}{3}k_1 + \frac{2}{3}k_2)$$
$$k_1 = f(x_i, y_i)$$
$$k_2 = f\left(x_i + \frac{3}{4}h, y_i + \frac{3}{4}k_1 h\right)$$

This equation can be given as Buthcher tableu as:

$$\begin{array}{c|cc} 0 & 0 & 0 \\ 3/4 & 3/4 & 0 \\ \hline & 1/3 & 2/3 \end{array}$$

Runge-Kutta with third degree polynomial solution RK3:

$$y_{n+1} = y_n + h(\frac{1}{6}k_1 + \frac{4}{6}k_2 + \frac{1}{6}k_3)$$
$$k_1 = f(x_i, y_i)$$
$$k_2 = f(x_i + 0.5h, y_i + 0.5k_1 h)$$
$$k_3 = f(x_i + h, y_i - k_1 h + 2k_2 h)$$

This equation can be given as Buthcher tableu as:

$$\begin{array}{c|ccc} 0 & 0 & 0 & 0 \\ 1/2 & 1/2 & 0 & 0 \\ \underline{1} & -1 & 2 & 0 \\ \hline & 1/6 & 4/6 & 1/6 \end{array}$$

Runge-Kutta with fourth degree polynomial solution RK4:

$$y_{n+1} = y_n + h(\frac{1}{6}k_1 + \frac{2}{6}k_2 + \frac{2}{6}k_3 + \frac{1}{6}k_4)$$
$$k_1 = f(x_i, y_i)$$
$$k_2 = f(x_i + 0.5h, y_i + 0.5k_1h)$$
$$k_3 = f(x_i + 0.5h, y_i + 0.5k_2h)$$
$$k_4 = f(x_i + h, y_i + k_3h)$$

This equation can be given as Buthcher tableu as:

$$\begin{array}{c|cccc} 0 & 0 & 0 & 0 & 0 \\ 1/2 & 1/2 & 0 & 0 & 0 \\ 1/2 & 0 & 1/2 & 0 & 0 \\ \underline{1} & 0 & 0 & 1 & 0 \\ \hline & 1/6 & 2/6 & 2/6 & 1/6 \end{array}$$

Runge-Kutta-Verner methods with different degree of accuricies

For the degrees a general formula of Runge_Kutta equations can be written as:

$x_i = x_n + c_i h$ i=1…s

$$y_i = y_n + h\sum_{j=1}^{i-1} a_{ij}k_j$$
$$k_i = f(x_i, y_i)$$

$y_{n+1} = y_n + h\sum_{j=1}^{s} b_i k_j$or

$$y_{n+1} = y_n + h\sum_{j=1}^{s} bh_i k_j$$

Verner[42-46] given coefficient of a pair of solutions (b and bh). For the constant step size any of these equations can be used, in the coming sections utilisation of these type of equations as a base of variable step size calculations will also be introduced. Due to growing sizes of equations Buthcher tableus will be given directly as table values for the remaining equations.

Table 8.4-1 9th degree Runge-Kutta-Verner coefficients

c[1] = 0
c[2] = 3/50
c[3] = 1439/15000
c[4] = 1439/10000
c[5] = 4973/10000
c[6] = 389/400
c[7] = 1999/2000
c[8] = 1
c[9] = 1

a[2,1] = 3/50

a[3,1] = 519479/27000000
a[3,2] = 2070721/27000000

a[4,1] = 1439/40000
a[4,2] = 0
a[4,3] = 4317/40000

a[5,1] = 109225017611/82828840000
a[5,2] = 0
a[5,3] = -417627820623/82828840000
a[5,4] = 43699198143/10353605000

a[6,1] = -80368152926439073494525521723 69/
191934985946683241245914401600
a[6,2] = 0
a[6,3] = 246134619571490020064824665/154381649665540511760236 8
a[6,4] = -13880495956885686234074067279/113663489566254201783474344
a[6,5] = 755005057777788994734129/136485922925633667082436

a[7,1] = -16632998415661020971805066664988809342302 61/3055842450615617030702095779131138423200 0
a[7,2] = 0
a[7,3] = 130838124195285491799043628811093033/631862949514135618861563657970240
a[7,4] = -32871004538560236341606187871539019628 73/
207243149153767556291357110268514092 00
a[7,5] = 27718267901403321408652425203692 41/396438716042723436917079980147600
a[7,6] = -1799166916139193/96743806114007800

a[8,1] = -83214475003936968389542838643798685392363776 3/152229745500696007487636518446676199452048 87
a[8,2] = 0
a[8,3] = 818622075710363565982285196611368750/393657623790372815185607239534312 9
a[8,4] = -98189851654916584648411945813854634347937418 75/616425979626589940698693709231964635818660 11
a[8,5] = 31796692141848558720425711042548134769375/453025403350004597555785801600630862809 2
a[8,6] = -14064542118843830075/766928748264306853644
a[8,7] = -1424670304836288125/2782839104764768088217

a[9,1] = 382735282417/11129397249634
a[9,2] = 0
a[9,3] = 0
a[9,4] = 5535620703125000/21434089949505429
a[9,5] = 13867056347656250/32943296570459319
a[9,6] = 626271188750/142160006043
a[9,7] = -51160788125000/289890548217
a[9,8] = 163193540017/946795234

b[1] = 382735282417/11129397249634
b[2] = 0
b[3] = 0
b[4] = 5535620703125000/21434089949505429
b[5] = 13867056347656250/32943296570459319
b[6] = 626271188750/142160006043
b[7] = -51160788125000/289890548217
b[8] = 163193540017/946795234
b[9] = 0

bh[1] = 124310637869885675646798613/28900724687894664265968276 70
bh[2] = 0
bh[3] = 0
bh[4] = 26586315173716499036133092187 5/11131972714633723039403193695 79
bh[5] = 30754935571740308065363029531 25/68437499220423238765469496998 76
bh[6] = 67798000008733879813263055/29532792147666737550036372
bh[7] = -10994365851553908462383263 75/15055706496446408859196167
bh[8] = 26171252653086373181571802/36879447889073234603350 5
bh[9] = 1/30

Table 8.4-2 10th degree Runge-Kutta-Verner coefficients

c[1] = 0
c[2] = 1/200
c[3] = 49/450
c[4] = 49/300
c[5] = 91/200
c[6] = 34704460/57271701
c[7] = 167/200
c[8] = 183/200
c[9] = 1
c[10] = 1

a[2,1] = 1/200

a[3,1] = -4361/4050
a[3,2] = 2401/2025

a[4,1] = 49/1200
a[4,2] = 0
a[4,3] = 49/400

a[5,1] = 1781/2800
a[5,2] = 0
a[5,3] = -13689/5600
a[5,4] = 507/224

a[6,1] = -2757769235683215548894853133264601087825440390757974155887649820 99
a[6,2] = 0
a[6,3] = 9576001512158705648097438730000/92976533366731450782554499984 7
a[6,4] = -2217853846564295490229045868960027892960010019435234766349995 41
a[6,5] = 1232368675010928345385491377920015540363434153685345084109283157

a[7,1] = 8221508014471276464993/8206108584945090400000
a[7,2] = 0
a[7,3] = -9852144759099/2364568872400
a[7,4] = 15322550932778398907299/3996134347464283007200
a[7,5] = -183384631218985200 04/36506562121215938675
a[7,6] = 233404755446021251193075113735193834 99/34957329425893779598660175543672800000

a[8,1] = 8108864302274054503373078016 9/29751821102319371401528000 00
a[8,2] = 0
a[8,3] = -2837794586103/67559110640
a[8,4] = -1416757560688131609503834114 1/13447191885934685533378360 00
a[8,5] = 2395552232834276839307772/297600628643880682688 75
a[8,6] = -4076715891031001341580357765362043260356514682697/605358015235585136339810926359877215071864000 00
a[8,7] = 3655152735545995780 8/2801171464968864455

a[9,1] = -3347747115771808477876752833599/1101327591307901549464211073280
a[9,2] = 0
a[9,3] = 1214704878477625125/119815105452943264
a[9,4] = -6558111861786403812445645241 5/10200342297342072539428709672
a[9,5] = -1333730829115754792732984060 95/84070826916189821955373830704
a[9,6] = 6225156836540393863870146375895226744773684105028195013769 3/
3289942188601405845401861424555681176690770946533324320857 60
a[9,7] = 4616918867155144139901376312 5/23436927041568412752689303582 08
a[9,8] = 1888086786587759749309187 5/34696641481969116434755335 04

a[10,1] = -7430981552872219612752603 7/51427190037752065777334640
a[10,2] = 0
a[10,3] = 913722369253875/113761793498234
a[10,4] = -4406582271592926020603968 90/58109996881684093545238767
a[10,5] = 3729088829393576334181438 0/10411746696360295961914239
a[10,6] = -6451528881135810657803603921051793104524963268 47/26473542512180481489813104213136732045148791096 0
a[10,7] = 4737574861900868386818925 00/55632110080294263936045988 7
a[10,8] = 0
a[10,9] = 0

b[1] = 9420080774669597/198627609019792680
b[2] = 0
b[3] = 0
b[4] = 18658605936510000/72821569629535727
b[5] = 296950875175030000/1101802245630054969
b[6] =
1887527698027421268682483556614315118985055389633000 9/14878094713960970610439415759664835799457557703622444 0
b[7] = 18663850606812500/74993696164706319
b[8] = 179884749312500/58508928482581269
b[9] = 349315176247648/7273791403140339
b[10] = 0

bh[1] = 7362904929137/155056681514280
bh[2] = 0
bh[3] = 0
bh[4] = 7505178129270000/29317774785916981
bh[5] = 1851744839320000/6843492208882329
bh[6] = 7508827781898184375128104078396450511280 89/60043632957155362707357899923191763222092 40
bh[7] = 27902602073000000/110704980052661709
bh[8] = 0
bh[9] = 0
bh[10] = 331667036438/6791588611709

Table 8.4-3 13th degree Runge-Kutta-Verner coefficients

c[1] = 0

c[2] = 1/20
c[3] = 341/3200
c[4] = 1023/6400
c[5] = 39/100
c[6] = 93/200
c[7] = 31/200
c[8] = 943/1000
c[9] = 7067558016280/7837150160667
c[10] = 909/1000
c[11] = 47/50
c[12] = 1
c[13] = 1

a[2,1] = 1/20

a[3,1] = -7161/1024000
a[3,2] = 116281/1024000

a[4,1] = 1023/25600
a[4,2] = 0
a[4,3] = 3069/25600

a[5,1] = 4202367/11628100
a[5,2] = 0
a[5,3] = -3899844/2907025
a[5,4] = 3982992/2907025

a[6,1] = 5611/114400
a[6,2] = 0
a[6,3] = 0
a[6,4] = 31744/135025
a[6,5] = 923521/5106400

a[7,1] = 21173/343200
a[7,2] = 0
a[7,3] = 0
a[7,4] = 8602624/76559175
a[7,5] = -26782109/689364000
a[7,6] = 5611/283500

a[8,1] = -1221101821869329/690812928000000
a[8,2] = 0
a[8,3] = 0
a[8,4] = -125/2
a[8,5] = -1024030607959889/168929280000000
a[8,6] = 1501408353528689/265697280000000
a[8,7] = 6070139212132283/92502016000000

a[9,1] = -14725142644862158038813847088772642463460444333307094\
20782905197804453180113305715 5/124689480162003200115705962164398602480330155839\
34879004404536361680460696864366 08
a[9,2] = 0
a[9,3] = 0
a[9,4] = -5172294311085668458375175655246981230039025336933699\
11413831527077231937246928000 0/124619381004809145897278630571215298365257079410\
23625292185093674907648713299519 1
a[9,5] = -1207067925846925480797893644173318794948457151612046\
99665345142964068916526149703 75/272203115476165722171047818453110069949728408 50\
48389015085076961673446140398628096
a[9,6] = 78012515584389364132309055253043103656779559256849718\
27014606748031267701114816 25/183110425412731972197889874507158786859226102980 86\
18595052414430736291431008053 76
a[9,7] = 66411312295991164213478213583910646992814032816057703\
53571553403929500094925118 75/151784655985862481363330231072953491752797651500 89\
07830113994325301687782317 0816
a[9,8] =
10332848184452015604056836767286656859124007796970668046446015775000000/
13127035500360336480738342487407279145379720286389501652495827336793937 83

a[10,1] = -29055573360337415088538618442231036441314060511/
22674759891089577691327962602370597632000000000
a[10,2] = 0
a[10,3] = 0
a[10,4] = -20462749524591049105403365239069/

```
45425191349989346959623126875O
a[10,5] = -18026925980317228116372466322498109 7/
38100922558256871086579832832000000
a[10,6] =  211276702141728028701282869920039408106552214 89/
46794738779978929061458226979767086336737280 00
a[10,7] =  31860723517364931240515126584966086992765341442541 3/
67147167155589653031329380729354654239109120000 00
a[10,8] =  21208320243451908228184224553589 4/
20022426044775672563822865371173879
a[10,9] = -\
26984049294008425187211664850871297985622698482295177937034139512267145 83/
46954567491393431507700044208087114188467603590271755032561672817587500000 0

a[11,1] = -\
23426598458140868369512071400656091790738384762429439 17/
13584809613510567770222314001391587608575321627955200 00
a[11,2] =  0
a[11,3] =  0
a[11,4] = -99628603013253815961393088965 2/
16353068885996164905464325675
a[11,5] = -26053085959256534152588089363841/
43775528045656830610112999424 00
a[11,6] =
20980822345096760292224086794978105312644533925634933539/
37758899920075508038787278391154946419722129621741568 00
a[11,7] =
89072299375637918641892962209583383526432263578229489 9/
13921242001395112657501941955594013822830119803764736
a[11,8] =  16102142614312417838907512192924671083312 5/
10997207722131034650667041364346422894371443
a[11,9] =  30076066976810251783423249756545243494667226619587 64\
96371874262392684852243925359864884962513/46554433375013464555850653366045056 03\
76082477961552128575189281031568049236410667452439828000 0
a[11,10] = -31155237437111730665923206875/
39286214159423051501033895629 1

a[12,1] = -\
28665569918256639717782953291010338875349127877240343 63/
86822671161926270301121392501614361203066923379533824 0
a[12,2] =  0
a[12,3] =  0
a[12,4] = -16957088714171468676387054358954754000/
14369041511965468332636822810157022 1
a[12,5] = -45834939744845729129493146733560335405 75/
45195770365525074715731303427033513574 4
a[12,6] =
23463053885534042586562584734461844191547401725199495 75/
25672671640789540289274497830115148625418318528966246 4
a[12,7] =
16571215593198468021712836909136106985862565734848086626 25/
13431480411255146477259155104956093505361644432088109056
a[12,8] =  34568537955467705221549582547696922637718750 0/
74771167436930077221667203179551347546362089
a[12,9] = -32058909627170725427914343121527275340081027740232 1\
02405713615707572490561670152301603520870486745421960 11/94756954968396581478301\
51244512736049846577471272576153724492059731926573060172391034910747382403325 9\
120
a[12,10] =  40279545832706233433100438588458933210937500/
88964608427994828469169721263773389472151 01
a[12,11] = -61229336010707695916130939939933588772 50/
10505170015102355131982467213020276759 53

a[13,1] = -61867590553548250067280085934453841035866015389963 7
/20354428211821404710011947534066768487429210238976 0
a[13,2] =  0
a[13,3] =  0
a[13,4] = -44111949168047186004784003191229310 00/
40373053902469967450761491269633019
a[13,5] = -16734711409449292534539422531728520225/
18012437152900886693072039272102379 52
a[13,6] =
13513751975705467909804218415274967776125475186563052 5/
16029587794486289597771326361911895112703716593983488
a[13,7] =
```

```
389375683674098760125485519034921961379297104315848 75/
340956454090191606099548798001469306974758443147264
a[13,8] = -67488658550119930377323553358153506672656 25/
700288039571742462121356540671508776477035 7
a[13,9] = -175600552030745092819542276704252509195417829600278\
830892656319352366240473977978973268567 1/348767814578469983605688098046186480 90\
460727802103054073533386208706157493415494283006232 0
a[13,10] = 5338102458923561108401389767418162929687 5/
895935758479569452487496959850859294414 1
a[13,11] = 0
a[13,12] = 0

b[1] = 4490186773775461685197 3/10140464099802310133806 80
b[2] = 0
b[3] = 0
b[4] = 0
b[5] = 0
b[6] = 791638675191615279648100000/22356047250899731264115123 19
b[7] = 384774949086898034811950000 0/155170450621382716181412375 17
b[8] = -13734512432397741476562500000/87513289292499590774692878 3
b[9] = 122747654703131968784288120377406350503192342760069863982944 4\
355496961634227421531633068444820714 1/489345147493715517650385834143510934888 82\
928068660965448289652679652335305216675729945285216604 0
b[10] = -979836368457773944531250000 0/308722986341456031822630699
b[11] = 282035543183190840068750/122954076298730404259 91
b[12] = -306814272936976936753/1299331183183744997286
b[13] = 0

bh[1] = 1083540173940701940657 7/244521829356935137978320
bh[2] = 0
bh[3] = 0
bh[4] = 0
bh[5] = 0
bh[6] = 139081897783218954913750 00/39221135527894265375640567
bh[7] = 734879475270272434876250 00/29650404577334276977339944 3
bh[8] = 68293140641257649609375000/153532086478069457499461 19
bh[9] = 220606479489966786110177113799745788605220182089497215594485\
602033384376260221427763 81/111154200926232587451295918579572721575901057756573 6\
079641376621381577236680929558640
bh[10] = -54797122949564245820312 5000/23237214025700991642563601
bh[11] = 0
bh[12] = 0
bh[13] = -2873545687097896418 9/79783493704265043693
```

Table 8.4-4 16th degree Runge-Kutta-Verner coefficients

```
c[1] = 0
c[2] = 1731/50000
c[3] = 7630049/53810000-983539/53810000*6^(1/2)
c[4] = 22890147/107620000-2950617/107620000*6^(1/2)
c[5] = 561/1000
c[6] = 387/1000-129/2000*6^(1/2)
c[7] = 387/1000+129/2000*6^(1/2)
c[8] = 129/200
c[9] = 387/800
c[10] = 6757/100000
c[11] = 1/4
c[12] = 1427971650951258372/2166662646162554701
c[13] = 4103/5000
c[14] = 2253/2500
c[15] = 1
c[16] = 1

a[2,1] = 1731/50000

a[3,1] = -177968356965557/1002427673820000+14180534491313/
250606918455000*6^(1/2)
a[3,2] = 64021741529527/200485534764000-7504450763411/
100242767382000*6^(1/2)

a[4,1] = 22890147/430480000-2950617/430480000*6^(1/2)
a[4,2] = 0
a[4,3] = 68670441/430480000-8851851/430480000*6^(1/2)
```

```
a[5,1] =  592203994261020339/513126355505556250+
730386990293623641/20525054220222225000*6^(1/2)
a[5,2] =  0
a[5,3] =  -8712153884182794903/2052505422022225000-\
2843421359195851533/20525054220222225000*6^(1/2)
a[5,4] =  1873698362223295443/513126355505556250+
52825859225556973/513126355505556250*6^(1/2)

a[6,1] =  11380823631/157617812000-339148869/39404453000*6^(1/2
)
a[6,2] =  0
a[6,3] =  0
a[6,4] =  -2355345717024309/58864341808507450*6^(1/2)+
16193232887091831/58864341808507450
a[6,5] =  165912282616977/4179075230308000-33181894472511/
2089537615154000*6^(1/2)

a[7,1] =  26523528363/231790900000+863255358/123138915625*6^(1/
2)
a[7,2] =  0
a[7,3] =  0
a[7,4] =  -3820874817801648481778 7/842517966262441068418750-\
8611878855628236982280 7/842517966262441068418750*6^(1/2)
a[7,5] =  923623364074469 13/290322814529044000-\
232039320950012997/246774392349687 4000*6^(1/2)
a[7,6] =  -362925891/1690350537500+857800423623/3380701075000*6
^(1/2)

a[8,1] =  43/600
a[8,2] =  0
a[8,3] =  0
a[8,4] =  0
a[8,5] =  0
a[8,6] =  43/150+43/2400*6^(1/2)
a[8,7] =  -43/2400*6^(1/2)+43/150

a[9,1] =  7353/102400
a[9,2] =  0
a[9,3] =  0
a[9,4] =  0
a[9,5] =  0
a[9,6] =  22833/102400+8901/204800*6^(1/2)
a[9,7] =  -8901/204800*6^(1/2)+22833/102400
a[9,8] =  -3483/102400

a[10,1] =  3767087424722149887008 53/778845602812500000000000 0,
`:`
a[10,2] =  0
a[10,3] =  0
a[10,4] =  0
a[10,5] =  0
a[10,6] =  1879146667539568401952 79/259615200937500000000000 0-\
2104408465562906932689 11/155769120562500000000000000*6^(1/2)
a[10,7] =  2104408465562906932689 11/155769120562500000000000000*
6^(1/2)+1879146667539568401952 79/259615200937500000000000
a[10,8] =  -1855266722189674422664 7/865384003125000000000000 0,
`:`
a[10,9] =  -3167799860072183913409/30423656359863281250000

a[11,1] =  -426968570497/54394415898750-92754382349/
12087647977500*6^(1/2)
a[11,2] =  0
a[11,3] =  0
a[11,4] =  0
a[11,5] =  0
a[11,6] =  1/30
a[11,7] =  -2865012129681958/114898584332330625-\
1296251768765509 9/22979716866466 1250*6^(1/2)
a[11,8] =  4389715333607/309890657317500+92754382349/
11477431752500*6^(1/2)
a[11,9] =  371017529396/9306344041875*6^(1/2)+4990058173976/
83757096376875
a[11,10] =  1099523524595993125000/6257667909869 7560188 91+
```

```
1009573480379896875006257667909869756018891*6^(1/2)

a[12,1] =  1838203110479840386993853900915465658752149857359559\
5063164077882800315372787284683238439478955141517997198007108623761931447163756
/139742569444997243449189609938909336141610253229704500479326889980950085286208\
21239604734608111291769444706187497807869179550841329375+4078857781851586092107\
9389251758259530589647075646761263679625961149140826089641344688345089135162291\
4818800693274034252252905536/28084926388601226073624096169175002956970191576455\
1106332267651411613722940986932751171812393853121981375088465359331278371679268\
75*6^(1/2)
a[12,2] =  0
a[12,3] =  0
a[12,4] =  0
a[12,5] =  0
a[12,6] =  -3338813117898494119715734728681282814382022107217231\
2325174214536773458288757739554777822876017406875808613438995201556340390 4/227\
0872004608103037127689848604039623086639035441372934050180593816493796129405349\
9141489814607142022329887277387784945577276 35+4819272892477768171373308666720 68\
9121421091953625792970278044071549950640195056472955523769829034800621890424847\
009130000000/2316289444700265097870243645576120415548371816150200392731184205 69\
2823672051993456912431961089928486277648502293554064448882187 7*6^(1/2)
a[12,7] =  1698450855653613368055560092963943745276369523793889\
6102606662872515552183276208687563236699647756792865753591219139615556676545782\
6139904/1593979913718173260623679425966077497797978183146596135797795272107637 6\
2009861408743487373293393731509448918731447404529311920099960958687 5*6^(1/2)-13\
6666607496463622270135608863772076443625468798139480390426740993024803946981763\
2093483647161087213128226198457261516936675984376999644 16/37192864653424042747 8\
8585327254180828195282427342057650194855634917821113563432870681372043512520401\
8871414370671061056839448023324223693 75
a[12,8] =  5610987899273278525411960528081442902198567594809764\
3797561956736732657005510768128839255833702537657025532355947644271736376737662\
08/9288159819814403301827880474062633413542335679163959810935886777036160923284\
6012626732332450844264293840456574956036349633197336361875-5587476413495323413 8\
4649167832304925076570507885572072105200355632180011316296456776552672453906332\
7600257543743479921263738432/365303089362201664516413596925286161494473575337 11\
5296250511752859728108868696929614024803255122785403232359817965288739565550625
*6^(1/2)
a[12,9] =  5459853981808361523356614860220324489669695891073433\
9754065270985433507945162707737759469214674480807272210648148477499238783276259\
328/3012479190922988526348868751299593107946629320141844998271450758516372986 98\
3120740305674792395020116934474230264160479447993402458125-652617245096253774\
7372702280281321524894343532103481802188740153783862532174342615150135214261625\
9666371008110923845480360464885 76/864909328430372818360283879213205026685796531\
7662489228456648746817034128576286937426571324705771222895418404433420637223081\
6544375*6^(1/2)
a[12,10] =  -8108825145085088104344721048166325225173729495689 3\
6469642672016111201241422775232896972065898731565417987376035772523573400039944\
0000000/2655582966610640210612741027567977548105720815295466925220401896406182 0\
6693081506133206500662983001047298787619827411375675716583283801757*6^(1/2)+939\
1667348404584010955422210328707125006120661611061908889750805619418785820948002\
4558903609392219121905247310870706451074869134577600000 00/58157266968773020612 4\
1902850373870830351528585497072566232680153129538726578484984317222364519327722\
9358434488742203091272981931739152584783
a[12,11] =  1234617126598879151772713939660686081047902877 7869\
3480148704506062609140195602856612882124981284004760156959603419 52/281629106670\
3206747542452093588407037042351473078388967410755112208260568290472056143249782\
5322617627507892271613246 1
```

```
57576064428027950953189860673178379041842786506645902521O1/
3523191107326604023034227593055748739260487670898437500000*6^(1/2)
a[13,8] =  -\
9311448168593934146015965019904013602133802943325818346622781285907057/
42559708490101242171931354496687399854013133630055761593627929687500000-\
8442137392040976964243665738134631724770749175817/
42101883599465783369768681649661630249023437500000*6^(1/2)
a[13,9] =
8857742338566725902229518676953278164573401303916391530705213354856175781/
30109854138029501101546924846546529011250565614375779993463516235351562 5-\
28140457973469923214145552460448772415902497252 7/
28448191636473798322140232250483030319213867187 5*6^(1/2)
a[13,10] =
31547911672978015395641212405219968509774423938663902378735910795925480218 2/
13448185050650584801258784221551557438021254320089493232912847115474882812 5-\
29403964536478722766460687765922922297376519379346 23/
73454650587819837107958374295307847772452865207031 25*6^(1/2)
a[13,11] =
22509961634065453786165320390188465862176315995 4382254 1/
38249130379709599356330414820427563643350402832031250 0
a[13,12] =  26893409573076918532949023883344540039593781469575 2\
9866233529251986359923360441517089497209588097479705143662934584242721740244 93
/95951638601957880850056911478087170846689475228048283510540802781519489531905 5\
44384278222710212049396080564957556176968750000000 00

a[14,1] =  47342003848024391498707976847688893013083074441159 77\
9465719863625051668939887702630319/44802546873926050730401222636656855760802419\
9938520602646153208014853924560546875000 00-\
8663695309870779911255624028290921871004932096 01/
3325522375873672156017711459173673934936523437500*6^(1/2)
a[14,2] =  0
a[14,3] =  0
a[14,4] =  0
a[14,5] =  0
a[14,6] =  87177932180780244746331003531823876287852715 7/
1344464208233494980590607737196505737304687 50+
1076412684809993960818489752718498579948 18/
1097082793918531904161935913552348681640625*6^(1/2)
a[14,7] =
4961037863518622928000348051141907054848007435133541170 14/
1100997221039563757198196122829921481018902397155761718 75-\
1329938412606197485769312599390307351191540891599374831099/
6605983326237382543189176736979528886113414382934570312 50*6^(1/2)
a[14,8] =  4077407727774763635459845170889116549412313138377723\
5229538611989392175193285994266471/1526429054624816210105898594158807951825674 1\
2553770317363579461257135247039794921875 00+
1237670758552968558750803432612988838714990299 43/
4510916099942762503903787319606603240966796875 00*6^(1/2)
a[14,9] =  -1052203860850055645982864903830206847373574903079 63\
7276496161875197379372479636460698666 4/3899417425005422254034574000397382862235\
8928296533758351973409182715560555076599121093 75+
3465478123948311964502249611316368748401972838404/
2560337247282641848992620902543472728729248046875*6^(1/2)
a[14,10] =  -2784376447126269318936520113562067049032847532328 2\
8202194748516216938957695270943346871089 84/122570410662851642220025943006055939\
2943413919302216631780212141299935702470459626113398437 5+
5747743002719985986838731141054720166992414950552 92/
1049352151254569101542262489932969253892183788671875*6^(1/2)
a[14,11] =  -\
3424113435184824562423280943767688900943193050352985303257641758989851 6/
5613347824358651981100985009024281007603230062439942682713165283203125
a[14,12] =  -3432044375893932378102368568052286501033850910516 9\
9920208853270521163343279392054770280096153243800840188373734185468897263960533\
4600163938610268855705742764072609/1143174106341682260971647690410567292143926 1\
9865092777892082326746111137127590759980171487016581339414751906821093176684449\
4994616580258435518181434575195312500000
a[14,13] =  4746930876023919335079451612726717649218264199984/
1859206553840704975520014438813408934643275559487 7

a[15,1] =  -2518832924925882544374852703814240987992301213373 89\
8531326543093228025085570860 1/1137064132557446931205696187407729855082764230877\
4647316995717036347558064286250+
1234273058981860170179592598535508631343082535549881956/
2105633771469628744518390642968552144069898845895808125*6^(1/2)
```

```
a[15,2] =  0
a[15,3] =  0
a[15,4] =  0
a[15,5] =  0
a[15,6] =  -54821142119685055562477216205428613949905430396088/
39594398370094612890855877467480979473931012780 95-\
15112767538259828560728914695044712566649759250 00/
40386286337496505148672995016830599063409633036569*6^(1/2)
a[15,7] =  -\
60922424274061599918603524049390657305431262635197540405697952/
64848617474890321697745846247599531485315640324174619095168 75+
84558575751635978733109961893984238786929550462615375699341616/
19454585242467096509323753874279859445594692097252385728550625*6^(1/2)
a[15,8] =  -11611814757504516973322287583571995533433479819145 9\
87978212353488939046793510977 2/88106269019548352456722751312958708925037139575 1\
21706814533008149884176424931 25-\
17632472271169431002565608550507266162044036222141170 8/
28561940671982910748577120704204013346542014996455562 5*6^(1/2)
a[15,9] =  17769448722513898342276837490665097286927607247073 33\
56185669871434672949001830332 16/25512170081378896150563421460845618671224851635\
96619283719957742418751029506356875-\
19748368943709762722873481576568138101489320568798111296/
64845542623222590712865459359971291351118136871756506 25*6^(1/2)
a[15,10] =  97659266139124074818193264801929547781659926543786 3\
81510190954184218570746215033823993530000000/18560076654469706205963482908787 05\
68508123082056031273268553609617276082427965511011820800335 99-\
85297084611782122474911131363078900058888025224607913745000000/
69210659450201393843166746722954036326338355649915383851733911*6^(1/2)
a[15,11] =
47338974904975296325611464923135382249291225950964951987086975 0525/
35412440882360341799798842428365422941216508121322622479260846291
a[15,12] =  33351439245158438248073494056784144097872912773415 9\
04536400728387690334563968394114702414108807505158106385116468732853458202899 96\
67484887185315457065591428959031448487646 37/23166110253272874277148020113222528\
86090793904989900621592365627649097578102163572190502232425490606773312310665 59\
34249827457442993712855985882986060885433767420546448189 66
a[15,13] =  -\
38714992656958413389743252726016897599283911682945255636643554687500000/
48540494926971587499294589382572212036169135429877901702347521300421767
a[15,14] =
14800250200940323717124616175641261235119295795768814717803955078125/
33565577125141877760287380588632421223433194078156948298488471160489

a[16,1] =
23057856960863975610808581869398971736456413310850413139443898499865841012 87/
61750824434528226581908737007827512267124616466990046213987605700823944000 0-\
85404623305589712632165905233974183137607899140719/
12482228716908483375841028346952511746054164329250 0*6^(1/2)
a[16,2] =  0
a[16,3] =  0
a[16,4] =  0
a[16,5] =  0
a[16,6] =  10290399696158044826419062526702606265479925908 3/
50463980848900048574816299996733204388194847 30+
41320925487304219313300272052128374567081128125/
51473260465878049546312625996667868475958744246*6^(1/2)
a[16,7] =
62798443349876457506718920843975661399949564598018488144466/
41325534987825733240582635825537152207770513597801413806 25-\
72308807081932961554425711089716771013571419950657300729103/
12397660496347719972174790747661145662331154079340424141875*6^(1/2)
a[16,8] =
17949091421264825643908485229242255532214690197514705449592976146546612933 77/
52596481193994264435601626109752988674679691644275456716633975785978672500+
12200660472227101804595129319139169019658271305817/
16931561456559959115207709344056578263397760602500*6^(1/2)
a[16,9] =  -27752447327801096673428456123947393191156626363714 7\
73004557470224232704759072 56/
22841715367558402972501804542270695582799632820818161943645438344714933375556 25+
34161849322235885052866362093589673255043159656287 6/
96101338378773357469245211954911505447551097205625*6^(1/2)
a[16,10] =  -27680554659769016623530979176727448251292244310769\
99601534219081906897055608306312 5000/32995577774296489605765613822566068446772 5\
84387970729553415813540513750365222314714 37+
```

```
44265521275798953734796703561001797599447665581417303125000/
30771137386673207077488771998046367464949770006589679876776*6^(1/2)
a[16,11] =  -\
29260317192970629105392940215993033073663913625268085362227 5/
15473622826279161150227076887290262443510550964275858143964
a[16,12] =  -98157171295691069885693021932209993438249320845820\
93647596086931754666098662594153095258988516305165794739744873539829069617203 52\
35091366822169330204 31/28647699117093415307614664109440217180193725006859654293\
10286786695017622532876932943976893277973881138545881134300639394050719790925 47\
998950955940992
a[16,13] =
27294911447098379057991487666507825329060502989714065185241699218 75/
21581158886221394731427758121094478029206561492431273092536869514 69
a[16,14] =  0
a[16,15] =  0

b[1] =  81981603662031734111199437115003 31/
56105757938408586016727784712876552 8
b[2] =  0
b[3] =  0
b[4] =  0
b[5] =  0
b[6] =  0
b[7] =  0
b[8] =  -45565549307342883881328144621374000000 0/
11638080111509105612404642258373124978 69
b[9] =  19965163648706008081135075746915614720000000/
86394404190537086868394686205782432516544599
b[10] =  89231107919981418705566970804343750000000000000000000000 0/
69997987098833567444559467985644506056259769358317598539 1
b[11] =  47104273954945906713184913871143492/
20968463912233960193463111349276346 7
b[12] =  20845004421404500464010584740796750650832176798370383084226 3\
51294730731196673647311062330972740734737279503119387627146381678677156136042 52\
41393119074820828440 83/36670849891136373020238225328265100250605144718501926305\
14096658675805484760468146633610316928475598775354232120246237155412059385814 97\
55539878561976786592389608
b[13] =  60530372821423065097959112869091796875000000 00/
10389925735051806345529007757377516273972512698 9
b[14] =  91740110492099349836035840609672546386718750 0/
67242498159113466533157907374536073829895514 63
b[15] =  25854495576652689513716995964939 57/
84574345160764140163208606048427531
b[16] =  0

bh[1] =  55256203120818093931780668425 3/
27669654257734667858523344041464
bh[2] =  0
bh[3] =  0
bh[4] =  0
bh[5] =  0
bh[6] =  0
bh[7] =  0
bh[8] =  22122338863142359758989860169000000 0/
10094613679858709005468507466712746 1
bh[9] =  10183540879130529798465781256192000000 0/
11497638332007437599765066502413121001 39
bh[10] =  13137203090776300144532398437500000000000000000000 0/
11518201923215510989126466531107437037395719117133
bh[11] =  48336112327014405045080861517 28/
19081321241454145230196661524503
bh[12] =  -21296623745823246481069197957033736453531182730667422307 24\
17273102581396471247364714401059920666982538271935911319623885770902551234058 99\
57/10355437392723670808851905462010972188912687281182073325925959875548518829 72\
29267088179417838009771658312306348528743579365742588923308056 8
bh[13] =  10847615917536408558443580639648437500000 0/
31828954860312490719385496913205024887334 23
bh[14] =  0
bh[15] =  0
bh[16] =  18391900710606498871278951007 84/
38045139523510634351420875415397
```

Runge-Kutta-Fehlberg methods

Similar to Verner methods, Fehlberg method is also given a pair of solutions. Later, it will be used in variable step calculations, but it can also utilised as single step solution of a good accuracy. Runge-Kutta-Fehlberg Buthcher tableu is given as:

0	0	0	0	0	0	0
1/4	1/4	0	0	0	0	0
3/8	3/32	9/32	0	0	0	0
12/13	1932/2197	−7200/2197	7296/2197	0	0	0
1	439/2197	−8	3680/513	−845/4104	0	0
1/2	−8/27	2	−3544/2565	1859/4104	−11/40	0
-	25/216	0	1408/2565	2197/4104	−1/5	0
	16/135	0	6656/12825	28561/56430	−9/50	2/55

Table 8.4-5 The same coefficients in table format :

c[1] = 0
c[2] = 1/4
c[3] = 3/8
c[4] = 12/13
c[5] = 1
c[6] = 1/2

a[2,1] = 1/4

a[3,1] = 3/32
a[3,2] = 9/32

a[4,1] = 1932/2197
a[4,2] = -7200/2197
a[4,3] = 7296/2197

a[5,1] = 439/216
a[5,2] = -8
a[5,3] = 3680/513
a[5,4] = -845/4104

a[6,1] = -8/27
a[6,2] = 2
a[6,3] = -3544/2565
a[6,4] = 1859/4104
a[6,5] = -11/40

b[1] = 25/216
b[2] = 0
b[3] = 1408/2565
b[4] = 2197/4104
b[5] = -1/5
b[5] = 0

bh[1] = 16/135
bh[2] = 0
bh[3] = 6656/12825
bh[4] = 28561/56430
bh[5] = -9/50
bh[5] = 2/55

6th degree Runge-Kutta- Cash-Karp coefficients are as below:

Runge-Kutta-Cash-Karp method(RKCK)

Runge-Kutta-Cash-Karp is a modified form of the Runge-Kutta_fehlberg method (different coefficients). Similar to RKF, this method also have a pair of solutions

0	0	0	0	0	0	0
1/5	1/5	0	0	0	0	0
3/10	3/40	9/40	0	0	0	0
3/5	3/10	−9/10	6/5	0	0	0
1	−11/54	5/2	−70/27	35/27	0	0
7/8	1631/55296	175/512	575/13824	44275/110592	253/4096	0
-	37/378	0	250/621	125/594	0	512/1771
	2825/27648	0	18575/48384	13525/55296	277/14336	1/4

The same coefficients is also given in table form below

Table 8.4-6 6th degree Runge-Cash-Karp coefficients

```
c[1] =  0
c[2] =  0.2
c[3] =  0.3
c[4] =  0.6
c[5] =  1
c[6] = 7/8


a[2,1] =  0.2

a[3,1] =  3/40
a[3,2] =  9/40

a[4,1] =  0.3
a[4,2] =  -9/10
a[4,3] =  6/5

a[5,1] =  -11/54
a[5,2] =  5/2
a[5,3] =  -70/27
a[5,4] =  35/27

a[6,1] =  1631/55296
a[6,2] =  175/512
a[6,3] =  575/13824
a[6,4] =  44275/110592
a[6,5] =  253/4096


b[1] =  37/378
b[2] =  250/621
b[3] =  125/591
b[4] =  512/1771
b[5] =  0

bh[1] =  2825/27648
bh[2] =  18575/48384
bh[3] =  13525/55296
bh[4] =  277/14336
bh[5] =  1/4
```

Table 8.4-7 7th degree Runge-Kutta-Dormant-Prince coefficients

```
    c[1]=0.0e0;
    c[2]=0.2e0;
    c[3]=0.3e0;
    c[4]=0.8e0;
    c[5]=8.0e0/9.0e0;
    c[6]=1;
    c[7]=1;
    a[2][1]=0.2e0;
    a[3][1]=3.0e0/40.0e0;
    a[3][2]=9.0e0/40.0e0;
    a[4][1]=44.0e0/45.0e0;
    a[4][2]=-56.0e0/15.0e0;
```

```
    a[4][3]=32.0e0/9.0e0;
    a[5][1]=19372.0e0/6561.0e0;
    a[5][2]=-25360.0e0/2187.0e0;
    a[5][3]=64448.0e0/6561.0e0;
    a[5][4]=-212.0e0/729.0e0;
    a[6][1]=9017.0e0/3168.0e0;
    a[6][2]=-355.0e0/33.0e0;
    a[6][3]=46732.0e0/5247.0e0;
    a[6][4]=49.0e0/176.0e0;
    a[6][5]=-5103.0e0/18656.0e0;
    a[7][1]=35.0e0/384.0e0;
    a[7][3]=500.0e0/1113.0e0;
    a[7][4]=125.0e0/192.0e0;
    a[7][5]=-2187.0e0/6784.0e0;
    a[7][6]=11.0e0/84.0e0;

    b[1]=71.0e0/57600.0e0;
    b[3]=-71.0e0/16695.0e0;
    b[4]=71.0e0/1920.0e0;
    b[5]=-17253.0e0/339200.0e0;
    b[6]=22.0e0/525.0e0;
    b[7]=0.0e0;
    bh[1]=-12715105075.0e0/11282082432.0e0;
    bh[3]=87487479700.0e0/32700410799.0e0;
    bh[4]=-10690763975.0e0/1880347072.0e0;
    bh[5]=701980252875.0e0/199316789632.0e0;
    bh[6]=-1453857185.0e0/822651844.0e0;
    bh[7]=69997945.0e0/29380423.0e0;
```

Program of Runge-Kutta is given in general form. For coefficients of different formulations a Butcher class is created and then a general Runge_Kutta method reads coefficients from this class.

Program 8.4-1 Butcher coefficients for Runge_Kutta class

```
 public class Buthcher
{public double a[][];
 public double b[][];
 public double c[];
 public Buthcher(double ai[][],double bi[][],double ci[])
 {set(ai,bi,ci); }

 public void set(double ai[][],double bi[][],double ci[])
 {a=ai;b=bi;c=ci;}

 public Buthcher(String isim)
 {if(isim.equals("RK2"))
   {double a2[][]={{0,0},{1,0}};
   double b2[][]={{0.5,0.5}};
   double c2[]={0,1};
   set(a2,b2,c2);
   }
  else if(isim.equals("Raltston"))
  {double a2[][]={{0,0},{3.0/4.0,0}};
   double b2[][]={{1.0/3.0,2.0/3.0}};
   double c2[]={1.0/6.0,4.0/6.0,1.0/6.0};
   set(a2,b2,c2);
  }
  else if(isim.equals("RK3"))
  {double a2[][]={{0,0,0},{0.5,0,0},{-1,2,0}};
   double b2[][]={{0,0.5,1}};
   double c2[]={1.0/6.0,4.0/6.0,1.0/6.0};
   set(a2,b2,c2);
  }
  else if(isim.equals("RK4"))
```

```
  {double a2[][]={{0,0,0,0},{0.5,0,0,0},{0,0.5,0,0},{0,0,1,0}};
   double b2[][]={{1.0/6.0,2.0/6.0,2.0/6.0,1.0/6.0}};
   double c2[]={0,0.5,0.5,1};
   set(a2,b2,c2);
  }
  else if(isim.equals("RK6"))
  {double a2[][]={{0,0,0,0,0,0},{0.25,0,0,0,0,0},{0.125,0.125,0,0,0,0},{0,-0.5,1,0,0,0},{3.0/16.0,0,0,9.0/16.0,0,0},{-3.0/7.0,2.0/7.0,12.0/7.0,-
12.0/7.0,8.0/7.0,0}};
   double b2[][]={{7.0/90.0,0,32.0/90.0,12.0/90.0,32.0/90.0,7.0/90.0}};
   double c2[]={0,0.25,0.25,0.5,0.75,1.0};
   set(a2,b2,c2);
  }
  else if(isim.equals("RKV9"))
  { //Runge-Kutta-Verner 9uncu derece sabit adımlı metodu
    int s=10;
    double c[]=new double[s];
    double a[][]=new double[s][s];
    double bh[]=new double[s];
    double bh1[]=new double[s];
c[0]=0;
c[1] =  0.;
c[2] =  .6e-1;
c[3] =  .95933333333333333333333333333333333333333333e-1;
c[4] =  .1439;
c[5] =  .4973;
c[6] =  .9725;
c[7] =  .9995;
c[8] =  1.;
c[9] =  1.;
a[2][1] =  .6e-1;
a[3][1] =  .19239962962962962962962962962962962962962962e-1;
a[3][2] =  .76693370370370370370370370370370370370370370e-1;
a[4][1] =  .35975e-1;
a[4][2] =  0.;
a[4][3] =  .107925;
a[5][1] =  1.3186834152331482609197472764317356128 61;
a[5][2] =  0.;
a[5][3] = -5.042058063628562225427761634715637693344;
a[5][4] =  4.220674648395413964508014358283902080483;
a[6][1] = -41.872591664327514618037577806443468129 05;
a[6][2] =  0.;
a[6][3] =  159.43256216313749177003656690703468304 53;
a[6][4] = -122.11921356501003092025162033892421406 63;
a[6][5] =  5.53174306620005376825263123833299915007 6;
a[7][1] = -54.43015693531650433250642051294142461271;
a[7][2] =  0.;
a[7][3] =  207.06725136501846442736571738665098359 87;
a[7][4] = -158.61081378458999918287424243650585994 69;
a[7][5] =  6.99181658595024232199259728079179390709 6;
a[7][6] = -.18597231062203233977651717995492946236 92e-1;

a[8][1] = -54.66374178728197680241215648050386959351;
a[8][2] =  0.;
a[8][3] =  207.95280625538937345158248166998342442 38;
a[8][4] = -159.28895747449950715089598058714266542 16;
a[8][5] =  7.01874374079694443469817076096425249081 7;
a[8][6] = -.18338785905045723064727820051417382683 61e-1;
a[8][7] = -.51194849978820990778754324972451683958 40e-3;

a[9][1] =  .34389578683570360092788201247283223865 20e-1;
a[9][2] =  0.;
a[9][3] =  0.;
a[9][4] =  .25826245556335034046595580985861208587 67;
a[9][5] =  .42093711896735371506425515140698019670 32;
```

```
a[9][6] =  4.40539646966931017014883681619709566489 1;
a[9][7] = -176.48311902429865761517409424990021250 29;
a[9][8] =  172.36413340141507302940225827119024133 15;

bh1[1] =  .34389578683570360092788201247283223865 20e-1;
bh1[2] =  0.;
bh1[3] =  0.;
bh1[4] =  .25826245556335034046595580985861208587 67;
bh1[5] =  .42093711896735371506425515140698019670 32;
bh1[6] =  4.40539646966931017014883681619709566489 1;
bh1[7] = -176.48311902429865761517409424990021250 29;
bh1[8] =  172.36413340141507302940225827119024133 15;
bh1[9] =  0.;

bh[ 1] =  .49099676483824897309068549279712258364 79e-1;
bh[ 2] =  0.;
bh[ 3] =  0.;
bh[ 4] =  .22511122295165241534013953205398753294 85;
bh[ 5] =  .46946822530295620394319485250473874125 53;
bh[ 6] =  .80657922499888677076341618089952179814 43;
bh[ 7] =  0.;
bh[ 8] = -.60711948917779597976729514652562171224 88;
bh[ 9] =  .56861139440475692411476031787661381535 94e-1;
double b[][]={bh1,bh};
set(a,b,c);
  }
  else if(isim.equals("RKCK45"))
  { // Runge-Kutta-Fehlberg, Cash-Karp, Dormant-Price, Verner
  int s=7;
double dydx=1.0;
double c[]=new double[s];
double a[][]=new double[s][s];
double bh1[]=new double[s];
double bh2[]=new double[s];
c[2] =  0.2;
c[3] =  0.3;
c[4] =  0.6;
c[5] =  1;
c[6] = 7.0/8.0;

a[1][1] = 0;

a[2][1] =  0.2;

a[3][1] =  3.0/40.0;
a[3][2] =  9.0/40.0;

a[4][1] =  0.3;
a[4][2] =  -9.0/10.0;
a[4][3] =  6.0/5.0;

a[5][1] =  -11.0/54.0;
a[5][2] =  0.25;
a[5][3] =  -70.0/27.0;
a[5][4] =  35.0/27.0;

bh1[1] =  37.0/378.0;
bh1[2] =  250.0/621.0;
bh1[3] =  125.0/591.0;
bh1[4] =  512.0/1771.0;
bh1[5] =  0.0;

bh2[1] =  2825.0/27648.0;
bh2[2] =  18575.0/48384.0;
```

```
bh2[3] =  13525.0/55296.0;
bh2[4] =  277.0/14336.0;
bh2[5] =  1.0/4.0;
double b[][]={bh1,bh2};
set(a,b,c);
  }
  else if(isim.equals("RKDP45"))
  {// Runge-Kutta Dormant-Prince order 4-5
  int s=8;
double dydx=1.0;
double c[]=new double[s];
double a[][]=new double[s][s];
double bh1[]=new double[s];
double bh2[]=new double[s];
c[1]=0.0e0;
      c[2]=0.2;
      c[3]=0.3;
      c[4]=0.8;
      c[5]=8.0/9.0;
      c[6]=1;
      c[7]=1;
      a[2][1]=0.2;
      a[3][1]=3.0/40.0;
      a[3][2]=9.0/40.0;
      a[4][1]=44.0/45.0;
      a[4][2]=-56.0/15.0;
      a[4][3]=32.0/9.0;
      a[5][1]=19372.0/6561.0;
      a[5][2]=-25360.0/2187.0;
      a[5][3]=64448.0/6561.0;
      a[5][4]=-212.0/729.0;
      a[6][1]=9017.0/3168.0;
      a[6][2]=-355.0/33.0;
      a[6][3]=46732.0/5247.0;
      a[6][4]=49.0/176.0;
      a[6][5]=-5103.0/18656.0;
      a[7][1]=35.0/384.0;
      a[7][3]=500.0/1113.0;
      a[7][4]=125.0/192.0;
      a[7][5]=-2187.0/6784.0;
      a[7][6]=11.0/84.0;
      bh1[1]=5179.0/57600.0;
      bh1[2]=0;
      bh1[3]=7571.0/16695.0;
      bh1[4]=393.0/640.0;
      bh1[5]=-92097.0/339200.0;
      bh1[6]=187.0/2100.0;
      bh1[7]=1.0/40.0;
      bh2[1]=35.0/384.0;
      bh2[2]=0;
      bh2[3]=500.0/1113.0;
      bh2[4]=125.0/192.0;
      bh2[5]=-2187.0/6784.0;
      bh2[6]=11.0/84.0;
      bh2[7]=0.0;
      double b[][]={bh1,bh2};
      set(a,b,c);
  }
  else if(isim.equals("RKV13"))
  { //Runge-Kutta-Verner 13üncü derece sabit adımlı metodu
int s=14;
double c[]=new double[s];
double a[][]=new double[s][s];
double bh[]=new double[s];
```

```
c[0]=0;
c[1] =  0.;
c[2] =  .5e-1;
a[2][1] =  .5e-1;
c[3] =  .1065625;
a[3][1] = -.69931640625e-2;
a[3][2] =  .1135556640625;
c[4] =  .15984375;
a[4][1] =  .399609375e-1;
a[4][2] =  0.;
a[4][3] =  .1198828125;
c[5] =  .39;
a[5][1] =  .36139756280045751240529407211840283451 29;
a[5][2] =  0.;
a[5][3] = -1.34152406670049277181998778820271583491 7;
a[5][4] =  1.37012650390003525941469371608431300040 4;
c[6] =  .465;
a[6][1] =  .490472027972027972027972027972027972028 0e-1;
a[6][2] =  0.;
a[6][3] =  0.;
a[6][4] =  .23509720422144047398629883354934271431 22;
a[6][5] =  .18085559298135672881090396365345448848 50;
//
c[7] =  .155;
a[7][1] =  .61692890442890442890442890442890442890 44e-1;
a[7][2] =  0.;
a[7][3] =  0.;
a[7][4] =  .11236568314640276622625570351300154423 03;
a[7][5] = -.38850460714513667670490481081112445674 56e-1;
a[7][6] =  .19791887125220458553791887125220458553 79e-1;
c[8] =  .943;
a[8][1] = -1.76763024022232687573559711957214558671 4;
a[8][2] =  0.;
a[8][3] =  0.;
a[8][4] = -62.5;
a[8][5] = -6.06188937737666910082136145965933199975 8;
a[8][6] =  5.65082319822276313856129803060084017420 1;
a[8][7] =  65.62169641937623283799566054863063741227;
c[9] =  .90180204173585695825970794067837214995 60;
a[9][1] = -1.18094506655497079982511628262828297957 882;
a[9][2] =  0.;
a[9][3] =  0.;
a[9][4] = -41.50473441114320841606641502701994225874;
a[9][5] = -4.43443831910372501122516922984610021177 6;
a[9][6] =  4.26040818858613302481219371074469324076 1;
a[9][7] =  43.75364022446171584987676829438379303004;
a[9][8] =  .78714254899123106874464750442263075508 60e-2;
c[10] =  .909;
a[10][1] = -1.28140599944148840545951029118205424626 6;
a[10][2] =  0.;
a[10][3] =  0.;
a[10][4] = -45.04713996013986630220754257136007322267;
a[10][5] = -4.73136206944957647731146426549128281094 3;
a[10][6] =  4.51496701659380784118585158459724099621 4;
a[10][7] =  47.44909557172985134869022392235929015114;
a[10][8] =  .10592282971116611356873939555165428752 28e-1;
a[10][9] = -.57468422638446162544323184782862962320 21e-2;
c[11] =  .94;
a[11][1] = -1.72447013426248519175670981748448186173 1;
a[11][2] =  0.;
a[11][3] =  0.;
a[11][4] = -60.92349008483054016518434619253765246063;
a[11][5] = -5.95151837622239245520283276706185486829 0;
a[11][6] =  5.55652373069845623597979165084359249683 9;
```

```
a[11][7] =  63.983011980333053368375363786359959392 81;
a[11][8] =  .1464202825041496159275921391759452676003e-1;
a[11][9] =  .6460408772358203603621865144977650714892e-1;
a[11][10] = -.7930323169008878984024452548693373291447e-1;
c[12] =  1.;
a[12][1] = -3.3016226677470790163539947897909836255 69;
a[12][2] =  0.;
a[12][3] =  0.;
a[12][4] = -118.011272359752508566692330395789886851 0;
a[12][5] = -10.1414223884561124864278391603451089759 5;
a[12][6] =  9.1393113322320579235440122735568270006 19;
a[12][7] =  123.375942828404268368484718098650189436 4;
a[12][8] =  4.623244378874580474839807625067630924792;
a[12][9] = -3.3832777380682019236525509715368112408 14;
a[12][10] =  4.527592100324618189451265339351129035325;
a[12][11] = -5.828495485811622963193088019162985703755;
c[13] =  1.;
a[13][1] = -3.0395150337663090300401028518212002510 56;
a[13][2] =  0.;
a[13][3] =  0.;
a[13][4] = -109.260868089417625468644419232216462335 2;
a[13][5] = -9.2906424974002934497176655426568975491 58;
a[13][6] =  8.4305049817649111421342992538361678034 54;
a[13][7] =  114.201001037833131355742404109552342747 6;
a[13][8] = -.9637271342145479358162375658987901652762;
a[13][9] = -5.0348840888021897911986803361833323231 18;
a[13][10] =  5.9581308240029231775404021653881720720 94;
a[13][11] =  0.;
a[13][12] =  0.;
bh[1] =  .4431261522908979212486436510209029764893e-1;
bh[2] =  0.;
bh[3] =  0.;
bh[4] =  0.;
bh[5] =  0.;
bh[6] =  .3546095642343226447863179350895055038855;
bh[7] =  .2478480431366653069619986721504458660016;
bh[8] =  4.448134732475784492725128317159648871312;
bh[9] =  19.846886366118733699309323992976879352 91;
bh[10] = -23.581623377465618419695179608703949650 85;
bh[11] =  0.;
bh[12] =  0.;
bh[13] = -.3601679437289775162124536737746202409110;
double b[][]={bh};
set(a,b,c);
  }
else if(isim.equals("RKF6"))
  { //Runge-Kutta-Fehlberg 6ıncı derece sabit adımlı metodu
int s=7;
double c[]=new double[s];
double a[][]=new double[s][s];
double bh[]=new double[s];
c[1] =  1;
c[2] =  0.2;
c[3] =  0.3;
c[4] =  0.6;
c[5] =  1;
c[5] = 7.0/8.0;

a[1][1] =  1;

a[2][1] =  0.2;

a[3][1] =  3.0/40.0;
a[3][2] =  9.0/40.0;
```

```
a[4][1] =  0.3;
a[4][2] =  -9.0/10.0;
a[4][3] =  6.0/5.0;

a[5][1] =  -11.0/54.0;
a[5][2] =  0.25;
a[5][3] =  -70.0/27.0;
a[5][4] =  35.0/27.0;

bh[1] =  2825.0/27648.0;
bh[2] =  18575.0/48384.0;
bh[3] =  13525.0/55296.0;
bh[4] =  277.0/14336.0;
bh[5] =  1.0/4.0;
double b[][]={bh};
set(a,b,c);
  }

}
}
```

Program 8.4-2 Runge_Kutta class

```
import java.io.*;
import java.text.*;
import java.util.Locale;

abstract class f_xy
{
  //single function single independent variable
  // example f=x*x
  abstract double func(double x,double y);
}
abstract class f_xi
{
  // çok fonksiyonlu ve çok denklemli fonksiyon
  // denklemlerin sadece bir tanesi denklem_ref
  // tarafından seçilir.
  // örnek   f[0]=x[0]+sin(x[1])
  //         f[1]=x[0]*x[0]-x[1]
  // func(x,1)  f[1] değerini verir
  // func(x,0)  f[0] değerini verir

  abstract double func(double x[],int denklem_ref);
}
public class Runge_Kutta
{
double a[][];
double b[][];
double c[];

public Runge_Kutta(double ai[][],double bi[][],double ci[])
{a=ai;b=bi;c=ci;}

public Runge_Kutta(Buthcher bu)
{a=bu.a;b=bu.b;c=bu.c;}
public Runge_Kutta(String isim)
{Buthcher bu=new Buthcher(isim);
 a=bu.a;b=bu.b;c=bu.c;
}
public double[][] hesapla(f_xy f,double x0,double xn,double y0,double h)
{return hesapla(f,x0,xn,y0,h,0);}

public double[][] hesapla(f_xy f,double x0,double xn,double y0,double h,int bk)
```

```
{
  int n=(int)((xn-x0)/h);
  double x[]=new double[n+1];
  double y[]=new double[n+1];
  int s=c.length;
  double xi;
  double yi;
  double ki[]=new double[s];
  x[0]=x0;y[0]=y0;
  for(int ni=0;ni<n;ni++)
  {for(int i=0;i<s;i++)
   { xi=x[ni]+c[i]*h;
     yi=y[ni];
    for(int j=0;j<i;j++)
    {yi+=h*a[i][j]*ki[j];//System.out.println("xi(0)="+xi+"xi(j)="+yi+"i="+i+"j="+j+"ki="+ki[j]);
    }
    ki[i]=f.func(xi,yi);
    //System.out.println("ki="+ki[i]);
   }
   y[ni+1]=y[ni];
   for(int i=0;i<s;i++)
    {y[ni+1]+=h*b[bk][i]*ki[i];}
   x[ni+1]=x[ni]+h;
  }
  double XY[][]=new double[2][n+1];
  XY[0]=x;
  XY[1]=y;
  return XY;
}
public static double[] tasi(double x1,double y1[])
{
int n=y1.length;
double z[]=new double[n+1];
z[0]=x1;
for(int i=1;i<=n;i++)
{z[i]=y1[i-1];}
return z;
}
//çok boyutlu RK girişler N(toplam veri sayısı) veya h(step boyu) üzeriinden verilebilmektedir.
public double[][] hesapla(f_xi f,double x0,double xn,double y0[],double h)
{return hesapla(f,x0,xn,y0,h,0);}
public double[][] hesapla(f_xi f,double x0,double xn,double y0[],int ni)
{double h=(xn-x0)/ni;
return hesapla(f,x0,xn,y0,h,0);
}
public double[][] hesapla(f_xi f,double x0,double xn,double y0[],int ni,int bk)
{double h=(xn-x0)/ni;
return hesapla(f,x0,xn,y0,h,bk);
}
public double[][] hesapla(f_xi f,double x0,double xn,double y0[],double h,int bk)
{
  int n=(int)((xn-x0)/h);
  int m=y0.length;
  double x[]=new double[n+1];
  double y[][]=new double[n+1][m];
  int s=c.length;
  double xi;
  double yi;
  double z[]=new double[m+1];
  double ki[]=new double[s];

  x[0]=x0;
  for(int ni=0;ni<=n;ni++)
  {x[ni]=x0+ni*h;}
```

```
  y[0]=y0;
  for(int ni=0;ni<n;ni++)
  { for(int k=0;k<m;k++)
                {
      for(int i=0;i<s;i++)
      { xi=x[ni]+c[i]*h;
        z=tasi(xi,y[ni]);
        for(int j=0;j<i;j++)
          {z[k+1]+=h*a[i][j]*ki[j];//System.out.println("xi(0)="+z[0]+"xi(j)="+z[k+1]+"i="+i+"j="+j+"ki="+ki[j]);
          }
        ki[i]=f.func(z,k);
        //System.out.println("ki="+ki[i]);
      }
    y[ni+1][k]=y[ni][k];
    for(int i=0;i<s;i++)
     {y[ni+1][k]+=h*b[bk][i]*ki[i];}
  }
  }
  double XY[][]=new double[m+1][n+1];
  XY[0]=x;
  for(int k=0;k<m;k++)
   { for(int ni=0;ni<=n;ni++)
     {XY[k+1][ni]=y[ni][k];}
   }
  return XY;
}

public static String toString(double left, int w, int d)
// converts a double to a string with given width and decimals.
{
                NumberFormat df=NumberFormat.getInstance(Locale.US);
                df.setMaximumFractionDigits(d);
    df.setMinimumFractionDigits(d);
    df.setGroupingUsed(false);
    String s = df.format(left);
    while (s.length() < w)
      s = " " + s;
    if (s.length() > w)
    {
       s = "";
       for (int i=0; i<w; i++)
         s = s + "-";
    }
    return s;
}
public String toString()
 {int m=b[0].length;
  int n=c.length;
  int k=b.length;
  String s="Buthcher tablosu n = "+n+"\n";
  for(int i=0;i<n;i++)
  {s+=toString(c[i],15,8)+" | ";
  for(int j=0;j<i;j++)
  {s+=toString(a[i][j],15,8)+" ";}
  s+="\n";
  }
  for(int j=0;j<m;j++)
  {s+="------------------";}
  s+="\n                  ";
  for(int i=0;i<k;i++)
  {
  for(int j=0;j<m;j++)
  {s+=toString(b[i][j],15,8)+" ";}
   s+="\n                  ";
```

```
  }
  return s;
 }

}
```

Now, we can call desired Runge-Kutta formula as a small program

Program 8.4-3 second degree Runge-Kutta (Ralston) equation by using class Runge_Kutta

```
import java.io.*;

abstract class f_xy
{
 //single function single independent variable
 // example f=x*x
 abstract double func(double x,double y);
}

class fxy extends f_xy
{ //diferansiyeli alınan fonksiyon
 public double func(double x,double y)
 {return x*y;}
}

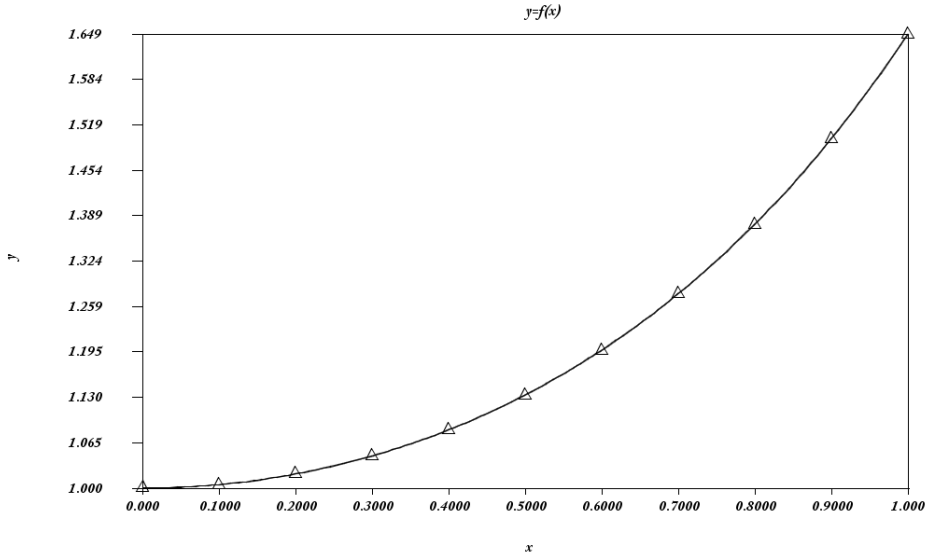
class fxyKC extends f_x
{ //Gerçek çözüm fonksiyonu
 public double func(double x)
 {return Math.exp(x*x/2.0);}
}

class dif1
{
 public static void main(String args[]) throws IOException
 {
  double x0=0;
  double y0=1.0;
  double xn=1.0;
  fxy f1=new fxy();
  fxyKC f2=new fxyKC();
  Runge_Kutta rk=new Runge_Kutta("RK2");
  System.out.println(rk.toString());
  double XY[][]=rk.hesapla(f1,x0,xn,y0,0.1);
  System.out.println(Matrix.toString(Matrix.Transpose(XY)));
  Plot p1=new Plot(XY);
  p1.addFunction(f2,x0,xn,200);
  p1.setPlotType(0,24);
  p1.plot();
 }
}
```

```
---------- Capture Output ----------
> "C:\java\bin\java.exe" dif1
Buthcher table n = 2
    0.00000000 |
    1.00000000 |      1.00000000
---------------------------------------
                   0.50000000      0.50000000

       0.000000000000000       1.000000000000000
       0.100000000000000       1.005000000000000
       0.200000000000000       1.020175500000000
       0.300000000000000       1.045985940150000
       0.400000000000000       1.083223039619340
```

```
0.500000000000000    1.133051299441830
0.600000000000000    1.197068697860293
0.700000000000000    1.277392007486719
0.800000000000000    1.376773105669185
0.900000000000000    1.498755202831475
1.000000000000000    1.647881345513207
```



The same programming can be done by defining the coefficients inside the specific Runge-Kutta program

Program 8.4-4 second degree Runge-Kutta (Ralston) equation

```
 //=====================================================
// Numerical Analysis package in java
// example to show differential equation solution
// A second degree Runge-Kutta RK2 (Ralston equation)
// Dr. Turhan Coban
// =====================================================
//  dy/dx=f(x,y)

import java.io.*;

abstract class f_xy
{
  //single function single independent variable
  // example f=x*x
  abstract double func(double x,double y);
}

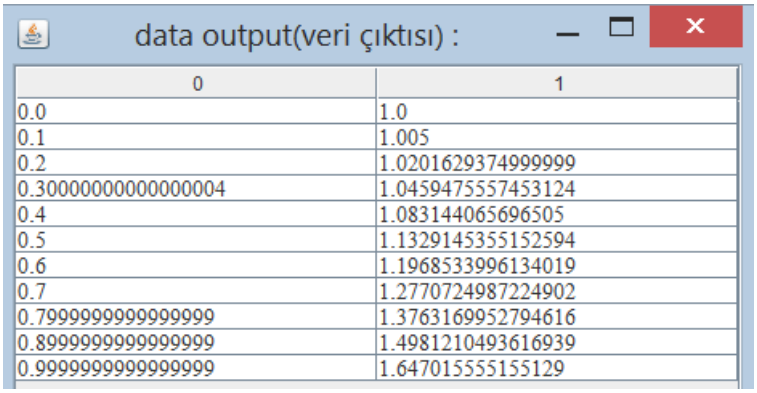
class fxy extends f_xy
{
  double func(double x,double y)
  {return x*y;}
}

class fxyKC extends f_x
{
  public double func(double x)
  {return Math.exp(x*x/2.0);}
}

class NA85
{
```

```
  public static double[][] RK2(f_xy f,double x0,double y0,double xn,double h)
  {
                //Ralston 2/3
  int n=(int)((xn-x0)/h);
  double a[][]=new double[2][n+1];
  int i=0;
  a[0][0]=x0;
  a[1][0]=y0;
  double k1,k2;
  double xx=x0;
  for(i=0;i<n;i++)
   {a[0][i+1]=xx+h;
   k1=f.func(a[0][i],a[1][i]);
   k2=f.func(a[0][i]+3.0/4.0*h,a[1][i]+3.0/4.0*k1*h);
   a[1][i+1]=a[1][i]+(1.0/3.0*k1+2.0/3.0*k2)*h;
   xx+=h;
   }
  return a;
  }


  public static void main(String args[]) throws IOException
  {double x0=0;
   double y0=1.0;
   fxy f1=new fxy();
   fxyKC fKC=new fxyKC();
   double a[][]=RK2(f1,x0,y0,1.0,0.1);
   Text.print(Text.T(a));
  }
}
```



0	1
0.0	1.0
0.1	1.005
0.2	1.0201629374999999
0.30000000000000004	1.0459475557453124
0.4	1.083144065696505
0.5	1.1329145355152594
0.6	1.1968533996134019
0.7	1.2770724987224902
0.7999999999999999	1.3763169952794616
0.8999999999999999	1.4981210493616939
0.9999999999999999	1.647015555155129

Program 8.4-5 Fourth degree Runge-Kutta Equation

```
 import java.io.*;

abstract class f_xy
{
  //single function single independent variable
  // example f=x*x
  abstract double func(double x,double y);
}

class fxy extends f_xy
{ //diferansiyeli alınan fonksiyon
  public double func(double x,double y)
  {return x*y;}
}

class fxyKC extends f_x
{ //Gerçek çözüm fonksiyonu
```

```
  public double func(double x)
  {return Math.exp(x*x/2.0);}
}

class dif1
{
  public static void main(String args[]) throws IOException
  {
   double x0=0;
   double y0=1.0;
   double xn=1.0;
   fxy f1=new fxy();
   fxyKC f2=new fxyKC();
   Runge_Kutta rk=new Runge_Kutta("RK4");
   System.out.println(rk.toString());
   double XY[][]=rk.hesapla(f1,x0,xn,y0,0.1);
   System.out.println(Matrix.toString(Matrix.Transpose(XY)));
   Plot p1=new Plot(XY);
   p1.addFunction(f2,x0,xn,200);
   p1.setPlotType(0,24);
   p1.plot();
  }
}
```

```
---------- Capture Output ----------
> "C:\java\bin\java.exe" dif1
Buthcher table n = 4
    0.00000000 |
    0.50000000 |     0.50000000
    0.50000000 |     0.00000000     0.50000000
    1.00000000 |     0.00000000     0.00000000     1.00000000
--------------------------------------------------------------------------
                  0.16666667     0.33333333     0.33333333     0.16666667

       0.000000000000000       1.000000000000000
       0.100000000000000       1.005012520833333
       0.200000000000000       1.020201339758369
       0.300000000000000       1.046027858885970
       0.400000000000000       1.083287064820495
       0.500000000000000       1.133148446117537
       0.600000000000000       1.197217347412631
       0.700000000000000       1.277621279469110
       0.800000000000000       1.377127694901942
       0.900000000000000       1.499302362448324
       1.000000000000000       1.648721007053397

> Terminated with exit code 0.
```

Program 8.4-6 Sixth order Runge-Kutta (RK6) equation

```
 import java.io.*;

abstract class f_xy
{
  //single function single independent variable
  // example f=x*x
  abstract double func(double x,double y);
}

class fxy extends f_xy
{ //diferansiyeli alınan fonksiyon
  public double func(double x,double y)
  {return x*y;}
}
```

```
class fxyKC extends f_x
{ //Gerçek çözüm fonksiyonu
  public double func(double x)
  {return Math.exp(x*x/2.0);}
}

class dif1
{
  public static void main(String args[]) throws IOException
  {
   double x0=0;
   double y0=1.0;
   double xn=1.0;
   fxy f1=new fxy();
   fxyKC f2=new fxyKC();
   Runge_Kutta rk=new Runge_Kutta("RK6");
   System.out.println(rk.toString());
   double XY[][]=rk.hesapla(f1,x0,xn,y0,0.1);
   System.out.println(Matrix.toString(Matrix.Transpose(XY)));
   Plot p1=new Plot(XY);
   p1.addFunction(f2,x0,xn,200);
   p1.setPlotType(0,24);
   p1.plot();
  }
}
```

```
---------- Capture Output ----------
> "C:\java\bin\java.exe" dif1
Buthcher table n = 6
    0.00000000 |
    0.25000000 |     0.25000000
    0.25000000 |     0.12500000     0.12500000
    0.50000000 |     0.00000000    -0.50000000     1.00000000
    0.75000000 |     0.18750000     0.00000000     0.00000000     0.56250000
    1.00000000 |    -0.42857143     0.28571429     1.71428571    -1.71428571     1.14285714
-------------------------------------------------------------------------------------------------
                  0.07777778     0.00000000     0.35555556     0.13333333     0.35555556     0.07777778

       0.000000000000000       1.000000000000000
       0.100000000000000       1.005012521378270
       0.200000000000000       1.020201341079585
       0.300000000000000       1.046027861534459
       0.400000000000000       1.083287069933695
       0.500000000000000       1.133148456035439
       0.600000000000000       1.197217366887088
       0.700000000000000       1.277621317852590
       0.800000000000000       1.377127769933579
       0.900000000000000       1.499302506626659
       1.000000000000000       1.648721278178056
```

Butcher table and general Runge_Kutta method based on Butcher table and example case is is given below

Program 8.4-7 Butcher table class (Python code)

```
# -*- coding: utf-8 -*-
"""
Created on Fri Aug 31 10:22:54 2018

@author: Mustafa Turhan Çoban
"""
from math import *

class  Buthcher():
```

```
    def set(self,ai,bi,ci):
        self.a=ai;
        self.b=bi;
        self.c=ci;
    def __init__(self,namei):
        self.name = namei;
        if str(self.name) ==("RK2"):
            a=[[0,0],[1,0]];
            b=[[0.5,0.5]];
            c=[0,1];
            self.set(a,b,c);
        elif str(self.name) == ("Raltston"):
            a=[[0,0],[3.0/4.0,0]];
            b=[[1.0/3.0,2.0/3.0]];
            c=[1.0/6.0,4.0/6.0,1.0/6.0];
            self.set(a,b,c);
        elif str(self.name) == ("RK3"):
            a=[[0,0,0],[0.5,0,0],[-1,2,0]];
            b=[[0,0.5,1]];
            c=[1.0/6.0,4.0/6.0,1.0/6.0];
            self.set(a,b,c);
        elif str(self.name) == ("RK4"):
            a=[[0,0,0,0],[0.5,0,0,0],[0,0.5,0,0],[0,0,1,0]];
            b=[[1.0/6.0,2.0/6.0,2.0/6.0,1.0/6.0]];
            c=[0,0.5,0.5,1];
            self.set(a,b,c);
        elif str(self.name) == ("RK6"):
            a=[[0,0,0,0,0,0],[0.25,0,0,0,0,0],[0.125,0.125,0,0,0,0],[0,-0.5,1,0,0,0],[3.0/16.0,0,0,9.0/16.0,0,0],[-3.0/7.0,2.0/7.0,12.0/7.0,-
12.0/7.0,8.0/7.0,0]];
            b=[[7.0/90.0,0,32.0/90.0,12.0/90.0,32.0/90.0,7.0/90.0]];
            c=[0,0.25,0.25,0.5,0.75,1.0];
            self.set(a,b,c);
        elif str(self.name) == ("RKV9"):
            #Runge-Kutta-Verner 9th degree formula
            n=10;
            c=[0.0 for i in range(n)];
            a = [[0 for x in range(n)] for y in range(n)];
            bh=[0.0 for i in range(n)];
            bh1=[0.0 for i in range(n)];
            c[0]=0;
            c[1] =  0.;
            c[2] =  .6e-1;
            c[3] =  .95933333333333333333333333333333333333333333e-1;
            c[4] =  .1439;
            c[5] =  .4973;
            c[6] =  .9725;
            c[7] =  .9995;
            c[8] =  1.;
            c[9] =  1.;
            a[2][1] =  .6e-1;
            a[3][1] =  .19239962962962962962962962962962962962962e-1;
            a[3][2] =  .76693370370370370370370370370370370370370e-1;
            a[4][1] =  .35975e-1;
            a[4][2] =  0.;
            a[4][3] =  .107925;
            a[5][1] =  1.3186834152331482609197472764317356128 61;
            a[5][2] =  0.;
            a[5][3] = -5.042058063628562225427761634715637693344;
            a[5][4] =  4.220674648395413964508014358283902080483;
            a[6][1] = -41.87259166432751461803757780644346812905;
            a[6][2] =  0.;
            a[6][3] =  159.43256216313749177003656690703468304 53;
            a[6][4] = -122.11921356501003092025162033892421406 63;
            a[6][5] =  5.53174306620005376825263123833299915007 6;
```

```
        a[7][1] = -54.430156935316504332506420512941424612 71;
        a[7][2] =  0.;
        a[7][3] =  207.067251365018464427365717386650983 5987;
        a[7][4] = -158.610813784589999182874242436505859 9469;
        a[7][5] =  6.991816585950242321992597280791793907096;
        a[7][6] = -.185972310622032339776517179954929462 3692e-1;
        a[8][1] = -54.663741787281976802412156480503869593 51;
        a[8][2] =  0.;
        a[8][3] =  207.952806255389373451582481669983424423 8;
        a[8][4] = -159.288957474499507150895980587142665421 6;
        a[8][5] =  7.018743740796944434698170760964252490817;
        a[8][6] = -.183387859050457230647278200514173826836 1e-1;
        a[8][7] = -.511948499788209907787543249724516839584 0e-3;
        a[9][1] =  .343895786835703600927882012472832238652 0e-1;
        a[9][2] =  0.;
        a[9][3] =  0.;
        a[9][4] =  .258262455563350340465955809858612085876 7;
        a[9][5] =  .420937118967353715064255151406980196703 2;
        a[9][6] =  4.405396469669310170148836816197095664891;
        a[9][7] = -176.483119024298657615174094249900212502 9;
        a[9][8] =  172.364133401415073029402258271190241331 5;
        bh1[1] =  .343895786835703600927882012472832238652 0e-1;
        bh1[2] =  0.;
        bh1[3] =  0.;
        bh1[4] =  .258262455563350340465955809858612085876 7;
        bh1[5] =  .420937118967353715064255151406980196703 2;
        bh1[6] =  4.405396469669310170148836816197095664891;
        bh1[7] = -176.483119024298657615174094249900212502 9;
        bh1[8] =  172.364133401415073029402258271190241331 5;
        bh1[9] =  0.;
        bh[ 1] =  .490996764838248973090685492797122583647 9e-1;
        bh[ 2] =  0.;
        bh[ 3] =  0.;
        bh[ 4] =  .225111222951652415340139532053987532948 5;
        bh[ 5] =  .469468225302956203943194852504738741255 3;
        bh[ 6] =  .806579224998886770763416180899521798144 3;
        bh[ 7] =  0.;
        bh[ 8] = -.607119489177795979767295146525621712248 8;
        bh[ 9] =  .568611394404756924114760317876613815359 4e-1;
        b=[bh1,bh];
        self.set(a,b,c);
    elif str(self.name) == ("RKCK45"):
        #Runge-Kutta-Fehlberg, Cash-Karp, Dormant-Price, Verner
        n=7;
        dydx=1.0;
        c=[0.0 for i in range(n)];
        a = [[0 for x in range(n)] for y in range(n)];
        bh1=[0.0 for i in range(n)];
        bh2=[0.0 for i in range(n)];
        c[2] =  0.2;
        c[3] =  0.3;
        c[4] =  0.6;
        c[5] =  1;
        c[6] = 7.0/8.0;
        a[1][1] = 0;
        a[2][1] =  0.2;
        a[3][1] =  3.0/40.0;
        a[3][2] =  9.0/40.0;
        a[4][1] =  0.3;
        a[4][2] =  -9.0/10.0;
        a[4][3] =  6.0/5.0;
        a[5][1] =  -11.0/54.0;
        a[5][2] =  0.25;
        a[5][3] =  -70.0/27.0;
```

```
        a[5][4] =  35.0/27.0;
        bh1[1] =  37.0/378.0;
        bh1[2] =  250.0/621.0;
        bh1[3] =  125.0/591.0;
        bh1[4] =  512.0/1771.0;
        bh1[5] =  0.0;
        bh2[1] =  2825.0/27648.0;
        bh2[2] =  18575.0/48384.0;
        bh2[3] =  13525.0/55296.0;
        bh2[4] =  277.0/14336.0;
        bh2[5] =  1.0/4.0;
        b=[bh1,bh2];
        self.set(a,b,c);
    elif str(self.name) == ("RKDP45"):
        #Runge-Kutta Dormant-Prince order 4-5
        n=8;
        dydx=1.0;
        c=[0.0 for i in range(n)];
        a = [[0 for x in range(n)] for y in range(n)];
        bh1=[0.0 for i in range(n)];
        bh2=[0.0 for i in range(n)];
        c[1]=0.0e0;
        c[2]=0.2;
        c[3]=0.3;
        c[4]=0.8;
        c[5]=8.0/9.0;
        c[6]=1;
        c[7]=1;
        a[2][1]=0.2;
        a[3][1]=3.0/40.0;
        a[3][2]=9.0/40.0;
        a[4][1]=44.0/45.0;
        a[4][2]=-56.0/15.0;
        a[4][3]=32.0/9.0;
        a[5][1]=19372.0/6561.0;
        a[5][2]=-25360.0/2187.0;
        a[5][3]=64448.0/6561.0;
        a[5][4]=-212.0/729.0;
        a[6][1]=9017.0/3168.0;
        a[6][2]=-355.0/33.0;
        a[6][3]=46732.0/5247.0;
        a[6][4]=49.0/176.0;
        a[6][5]=-5103.0/18656.0;
        a[7][1]=35.0/384.0;
        a[7][3]=500.0/1113.0;
        a[7][4]=125.0/192.0;
        a[7][5]=-2187.0/6784.0;
        a[7][6]=11.0/84.0;
        bh1[1]=5179.0/57600.0;
        bh1[2]=0;
        bh1[3]=7571.0/16695.0;
        bh1[4]=393.0/640.0;
        bh1[5]=-92097.0/339200.0;
        bh1[6]=187.0/2100.0;
        bh1[7]=1.0/40.0;
        bh2[1]=35.0/384.0;
        bh2[2]=0;
        bh2[3]=500.0/1113.0;
        bh2[4]=125.0/192.0;
        bh2[5]=-2187.0/6784.0;
        bh2[6]=11.0/84.0;
        bh2[7]=0.0;
        b=[bh1,bh2];
        self.set(a,b,c);
```

```
    elif str(self.name) == ("RKV13"):
        #Runge-Kutta-Verner 13th degree
        n=14;
        c=[0.0 for i in range(n)];
        a = [[0 for x in range(n)] for y in range(n)];
        bh=[0.0 for i in range(n)];
        c[0]=0;
        c[1] =  0.;
        c[2] =  .5e-1;
        a[2][1] =  .5e-1;
        c[3] =  .1065625;
        a[3][1] = -.69931640625e-2;
        a[3][2] =  .1135556640625;
        c[4] =  .15984375;
        a[4][1] =  .399609375e-1;
        a[4][2] =  0.;
        a[4][3] =  .1198828125;
        c[5] =  .39;
        a[5][1] =  .36139756280045751240529407211840283 45129;
        a[5][2] =  0.;
        a[5][3] = -1.34152406670049277181998778820271583 4917;
        a[5][4] =  1.37012650390003525941469371608431300 0404;
        c[6] =  .465;
        a[6][1] =  .490472027972027972027972027972027972 0280e-1;
        a[6][2] =  0.;
        a[6][3] =  0.;
        a[6][4] =  .23509720422144047398629883354934271431 22;
        a[6][5] =  .18085559298135672881090396365345448848 50;
        c[7] =  .155;
        a[7][1] =  .616928904428904428904428904428904428904 4e-1;
        a[7][2] =  0.;
        a[7][3] =  0.;
        a[7][4] =  .11236568314640276622625570351300154423 03;
        a[7][5] = -.38850460714513667670490481081112445674 56e-1;
        a[7][6] =  .19791887125220458553791887125220458553 79e-1;
        c[8] =  .943;
        a[8][1] = -1.76763024022232687573559711957214558 6714;
        a[8][2] =  0.;
        a[8][3] =  0.;
        a[8][4] = -62.5;
        a[8][5] = -6.06188937737666910082136145965933199 9758;
        a[8][6] =  5.65082319822276313856129803060084017 4201;
        a[8][7] =  65.62169641937623283799566054863063741227;
        c[9] =  .90180204173585695825970794067837214995 60;
        a[9][1] = -1.18094506655497079982511628262828979 57882;
        a[9][2] =  0.;
        a[9][3] =  0.;
        a[9][4] = -41.50473441114320841606641502701994225874;
        a[9][5] = -4.43443831910372501122516922984610021 1776;
        a[9][6] =  4.26040818858613302481219371074469324 0761;
        a[9][7] =  43.75364022446171584987676829438379303004;
        a[9][8] =  .78714254899123106874464750442263075508 60e-2;
        c[10] =  .909;
        a[10][1] = -1.28140599944148840545951029118205424 6266;
        a[10][2] =  0.;
        a[10][3] =  0.;
        a[10][4] = -45.04713996013986630220754257136007322267;
        a[10][5] = -4.73136206944957647731146426549128281 0943;
        a[10][6] =  4.51496701659380784118585158459724099 6214;
        a[10][7] =  47.44909557172985134869022392235929015114;
        a[10][8] =  .10592282971116611356873939555165428752 28e-1;
        a[10][9] = -.57468422638446162544323184782862960232021e-2;
        c[11] =  .94;
        a[11][1] = -1.72447013426248519175670981748448186 1731;
```

```
        a[11][2] =  0.;
        a[11][3] =  0.;
        a[11][4] = -60.923490084830540165184346192537652460 63;
        a[11][5] = -5.9515183762223924552028327670618548682 90;
        a[11][6] =  5.5565237306984562359797916508435924968 39;
        a[11][7] =  63.9830119803330533683753637863599593928 1;
        a[11][8] =  .1464202825041496159275921391759452676003e-1;
        a[11][9] =  .6460408772358203603621865144977650714892e-1;
        a[11][10] = -.7930323169008878984024452548693373291447e-1;
        c[12] =  1.;
        a[12][1] = -3.3016226677470790163539947897909836255 69;
        a[12][2] =  0.;
        a[12][3] =  0.;
        a[12][4] = -118.011272359752508566692330395789886851 0;
        a[12][5] = -10.141422388456112486427839160345108975 95;
        a[12][6] =  9.139311332232057923544012273556827000619;
        a[12][7] =  123.375942828404268368484718098650189436 4;
        a[12][8] =  4.623244378874580474839807625067630924792;
        a[12][9] = -3.383277738068201923652550971536811240814;
        a[12][10] =  4.527592100324618189451265339351129035325;
        a[12][11] = -5.828495485811622963193088019162985703755;
        c[13] =  1.;
        a[13][1] = -3.039515033766309030040102851821200251056;
        a[13][2] =  0.;
        a[13][3] =  0.;
        a[13][4] = -109.260868089417625468644419232216462335 2;
        a[13][5] = -9.290642497400293449717665542656897549158;
        a[13][6] =  8.430504981764911142134299253836167803454;
        a[13][7] =  114.201001037833131355742404109552342747 6;
        a[13][8] = -.963727134214547935816237565898790165276 2;
        a[13][9] = -5.034884088802189791198680336183332323118;
        a[13][10] =  5.958130824002923177540402165388172072794;
        a[13][11] =  0.;
        a[13][12] =  0.;
        bh[1] =  .443126152290897921248643651020902976489 3e-1;
        bh[2] =  0.;
        bh[3] =  0.;
        bh[4] =  0.;
        bh[5] =  0.;
        bh[6] =  .354609564234322644786317935089505503885 5;
        bh[7] =  .247848043136665306961998672150445866001 6;
        bh[8] =  4.448134732475784492725128317159648871312;
        bh[9] =  19.846886366118733699309323992976879352 91;
        bh[10] = -23.581623377465618419695179608703949650 85;
        bh[11] =  0.;
        bh[12] =  0.;
        bh[13] = -.360167943728977516212453673774620240911 0;
        b=[bh];
        self.set(a,b,c);
    elif str(self.name) == ("RKF6"):
        #Runge-Kutta-Fehlberg 6th degree
        n=7;
        c=[0.0 for i in range(n)];
        a = [[0 for x in range(n)] for y in range(n)];
        bh=[0.0 for i in range(n)];
        c[1] =  1;
        c[2] =  0.2;
        c[3] =  0.3;
        c[4] =  0.6;
        c[5] =  1;
        c[5] = 7.0/8.0;
        a[1][1] =  1;
        a[2][1] =  0.2;
        a[3][1] =  3.0/40.0;
```

```
        a[3][2] =  9.0/40.0;
        a[4][1] =  0.3;
        a[4][2] =  -9.0/10.0;
        a[4][3] =  6.0/5.0;
        a[5][1] =  -11.0/54.0;
        a[5][2] =  0.25;
        a[5][3] =  -70.0/27.0;
        a[5][4] =  35.0/27.0;
        bh[1] =  2825.0/27648.0;
        bh[2] =  18575.0/48384.0;
        bh[3] =  13525.0/55296.0;
        bh[4] =  277.0/14336.0;
        bh[5] =  1.0/4.0;
        b=[bh];
        self.set(a,b,c);
```

Program 8.4-8 Runge_Kutta class in python to calculate single or set differential equations

```
# -*- coding: utf-8 -*-
"""
Created on Fri Aug 31 13:10:28 2018

@author: Mustafa Turhan Çoban
"""
from  Butcher import *;
from f_xi import *;

class  Runge_Kutta():
  def __init__(self,namei):
     self.name=namei;
     self.bu=Buthcher(namei);
     self.a=self.bu.a;
     self.b=self.bu.b;
     self.c=self.bu.c;
  def calc_1(self,f,x0,xn,y0,hi,bk):
     ff=float((xn-x0)/hi);
     n=int(ff);
     self.h=hi;
     xx=[0 for i in range(2)];
     x=[0 for i in range(n+1)];
     y=[0 for i in range(n+1)];
     s=len(self.c);
     ki=[0 for i in range(n+1)];
     x[0]=x0;y[0]=y0;
     for ni in range(n):
        for i in range(s):
           xi=x[ni]+self.c[i]*self.h;
           yi=y[ni];
           for j in range(i):
              yi+=self.h*self.a[i][j]*ki[j];
           xx[0]=xi;
           xx[1]=yi;
           ki[i]=f.func(xx);
        y[ni+1]=y[ni];
        for i in range(s):
           y[ni+1]+=self.h*self.b[bk][i]*ki[i];
        x[ni+1]=x[ni]+self.h;
     XY=[[0 for i2 in range(n+1)] for j2 in range(2)];
     XY[0]=x;
     XY[1]=y;
     return XY;

  def  tasi(self,x1,y1):
     n1=len(y1);
     z=[0 for i in range(n1+1)];
```

```
        z[0]=x1;
        for i in range(1,n1+1):
            z[i]=y1[i-1];
        print("z=",z);
        return z;

    def calc_n(self,f,x0,xn,y0,hi,bk):
        #bk index of k if more than one equation is existed
        self.h=hi;
        ff=float((xn-x0)/hi+1e-30);
        n=int(ff);
        m=len(y0);
        x=[0 for i in range(n+1)];
        y=[[0 for i2 in range(m)] for j2 in range(n+1)];
        print(y[0]);
        s=len(self.c);
        z=[0 for i in range(m+1)];
        ki=[0 for i in range(s)];
        x[0]=x0;
        for ni in range(n+1):
            x[ni]=x0+ni*self.h;
        y[0]=y0;
        for ni in range(n):
            for k in range(m):
                for i in range(s):
                    xi=x[ni]+self.c[i]*self.h;
                    z[0]=xi;
                    for ix in range(m):
                        z[ix+1]=y[ni][ix];
                    for j in range(i):
                        z[k+1]+=self.h*self.a[i][j]*ki[j];
                    ki[i]=f.func(z,k);
                y[ni+1][k]=y[ni][k];
                for i in range(s):
                    y[ni+1][k]+=self.h*self.b[bk][i]*ki[i];
        XY=[[0 for i2 in range(n+1)] for j2 in range(m+1)];
        XY[0]=x;
        for k in range(m):
            for ni in range(n+1):
                XY[k+1][ni]=y[ni][k];
        return XY;
```

Program 8.4-9 Sixth order Runge-Kutta (RK6) equation using Buthher class (Python code)

```
# -*- coding: utf-8 -*-
"""
Created on Fri Aug 31 13:10:28 2018

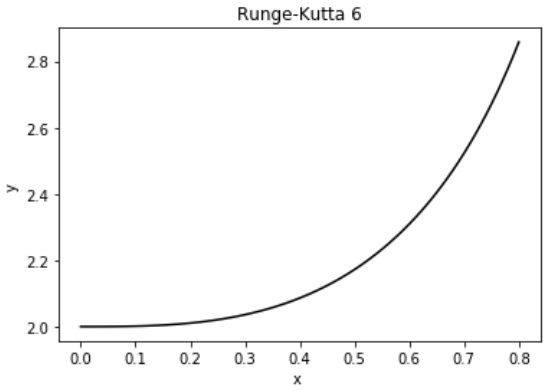
@author: Mustafa Turhan Çoban
"""
from Runge_Kutta import *;
from f_x import *;
import numpy as np;
import matplotlib.pyplot as plt;

class f1(f_x):
    def func(self,X):
        x=X[0];
        y=X[1];
        F=y*y*sin(x)*sin(x);
        return F;


rk=Runge_Kutta("RK6");
```

```
f=f1();
XY=rk.calc_1(f,0.0,0.8,2.0,0.01,0);
print(XY);
x=XY[0];
y=XY[1];
plt.title('Runge-Kutta 6')
plt.xlabel('x ')
plt.ylabel('y ');
plt.plot(x,y,'k');
```

```
runfile('D:/okul/SCO1/Runge_Kutta_test_RK6_1.py', wdir='D:/okul/SCO1')
Reloaded modules: Runge_Kutta, Butcher, f_xi, f_x
[[0.0, 0.01, 0.02, 0.03, 0.04, 0.05, 0.060000000000000005, 0.07, 0.08, 0.09, 0.09999999999999999, 0.10999999999999999,
0.11999999999999998, 0.12999999999999998, 0.13999999999999999, 0.15, 0.16, 0.17, 0.18000000000000002, 0.19000000000000003,
0.20000000000000004, 0.21000000000000005, 0.22000000000000006, 0.23000000000000007, 0.24000000000000007,
0.25000000000000006, 0.26000000000000006, 0.2700000000000001, 0.2800000000000001, 0.2900000000000001, 0.3000000000000001,
0.3100000000000001, 0.3200000000000001, 0.3300000000000001, 0.34000000000000014, 0.35000000000000014,
0.36000000000000015, 0.37000000000000016, 0.38000000000000017, 0.3900000000000002, 0.4000000000000002,
0.4100000000000002, 0.4200000000000002, 0.4300000000000002, 0.4400000000000002, 0.45000000000000023, 0.46000000000000024,
0.47000000000000025, 0.48000000000000026, 0.49000000000000027, 0.5000000000000002, 0.5100000000000002,
0.5200000000000002, 0.5300000000000002, 0.5400000000000003, 0.5500000000000003, 0.5600000000000003, 0.5700000000000003,
0.5800000000000003, 0.5900000000000003, 0.6000000000000003, 0.6100000000000003, 0.6200000000000003, 0.6300000000000003,
0.6400000000000003, 0.6500000000000004, 0.6600000000000004, 0.6700000000000004, 0.6800000000000004, 0.6900000000000004,
0.7000000000000004, 0.7100000000000004, 0.7200000000000004, 0.7300000000000004, 0.7400000000000004, 0.7500000000000004,
0.7600000000000005, 0.7700000000000005, 0.7800000000000005, 0.7900000000000005, 0.8000000000000005], [2.0,
2.0000013333075586, 2.0000106658702514, 2.0000359941683423, 2.0000853096695534, 2.0001665972293488, 2.00028783412938,
2.0004569897517466, 2.0006820258861464, 2.00097089766665, 2.001331555134674, 2.001771945424804, 2.0023000155703943,
2.0029237159263467, 2.0036510042071387, 2.004489850139076, 2.005448240726798, 2.0065341861353527, 2.007755726190629,
2.009120937502593, 2.0106379412176634, 2.0123149114086223, 2.0141600841127394, 2.0161817670313047, 2.0183883499064637,
2.020788315594225, 2.0233902518557123, 2.0262028638922014, 2.029234987653242, 2.032495603951213, 2.035993853420053,
2.0397390523606527, 2.043740709520522, 2.048008543860918, 2.0525525033706464, 2.057382784992311, 2.062509855733902,
2.0679444750463984, 2.07369771855652, 2.0797810032530317, 2.0862061142351456, 2.092985233142692, 2.1001309683999394,
2.1076563874184013, 2.1155750509187783, 2.1239010495485644, 2.1326490429899403, 2.1418343017726333, 2.151472752028696,
2.161581023450873, 2.172176500743812, 2.183277378888052, 2.194902722571101, 2.2070725301782668, 2.2198078027789916,
2.2331306185927375, 2.2470642134728376, 2.2616330680079257, 2.2768630019096614, 2.292781276433536, 2.3094167056680117,
2.3267997776275444, 2.34496278619912, 2.3639399751217804, 2.383767695326836, 2.4044845771358694, 2.4261317190077967,
2.4487528947490995, 2.4723947813578184, 2.4971072099676364, 2.5229434427002473, 2.549960478630286, 2.5782193925271737,
2.60778571057397, 2.638729827888825, 2.6711274734069064, 2.7050602285404257, 2.7406161070467014, 2.77789020472977,
2.8169854290175724, 2.858013320140403]]
```



EXAMPLE :

Solve differential equation

$\frac{dy}{dx} = yx^2 - y$ at the range x=0 ve x=2 , with initial condition y(0)=1.

a) solve analytically from x=0 to x=2.

b) solve numerically by hand with h=0.5 step size with 4th order Runge_Kutte and calculate y(2)

c) solve numerically by computer with h=0.1 step size with 4th order Runge-Kutta

d) Plot to results and compare with analytical solution

Analytical solution :

$$\frac{dy}{dx} = yx^2 - y = y(x^2 - 1)$$

$$\frac{dy}{y} = (x^2 - 1)dx$$

$$\int \frac{dy}{y} = \int (x^2 - 1)dx$$

$$ln(y) = (\frac{x^3}{3} - x) + C$$

Boundary condition x=0 y=1➔ C=0

$$y = exp(\frac{x^3}{3} - x)$$

Solution with h=0.5 and h=0.1 by using spreadsheet:

h	0.5														
	x1	y1	k1	x2	y2	k2	x3	y3	k3	x4	y4	k4	y	y exact	error
	0	1	-1	0.25	0.75	-0.703	0.25	0.8242	-0.773	0.5	0.6136	-0.46	1	1	0
	0.5	0.6323	-0.474	0.75	0.5138	-0.225	0.75	0.5761	-0.252	1	0.5063	0	0.6323	0.6323	-5.4E-06
	1	0.5133	0	1.25	0.5133	0.2888	1.25	0.5855	0.3294	1.5	0.678	0.8475	0.5133	0.5134	7.02E-05
	1.5	0.687	0.8587	1.75	0.9017	1.8597	1.75	1.1519	2.3758	2	1.8749	5.6248	0.687	0.6873	0.000294
	2	1.9332	5.7996	2.25	3.3831	13.744	2.25	5.3692	21.812	2.5	12.839	67.407	1.9332	1.9477	0.01452

h	0.1														
	x1	y1	k1	x2	y2	k2	x3	y3	k3	x4	y4	k4	y	y exact	error
	0	1	-1	0.05	0.95	-0.948	0.05	0.9526	-0.95	0.1	0.905	-0.896	1	1	0
	0.1	0.9051	-0.896	0.15	0.8603	-0.841	0.15	0.8631	-0.844	0.2	0.8208	-0.788	0.9051	0.9051	-7E-08
	0.2	0.8209	-0.788	0.25	0.7815	-0.733	0.25	0.7843	-0.735	0.3	0.7474	-0.68	0.8209	0.8209	-1.2E-07
	0.3	0.7475	-0.68	0.35	0.7135	-0.626	0.35	0.7162	-0.628	0.4	0.6847	-0.575	0.7475	0.7475	-1.5E-07
	0.4	0.6848	-0.575	0.45	0.656	-0.523	0.45	0.6586	-0.525	0.5	0.6322	-0.474	0.6848	0.6848	-1.6E-07
	0.5	0.6323	-0.474	0.55	0.6086	-0.425	0.55	0.6111	-0.426	0.6	0.5897	-0.377	0.6323	0.6323	-1.6E-07
	0.6	0.5898	-0.377	0.65	0.5709	-0.33	0.65	0.5733	-0.331	0.7	0.5567	-0.284	0.5898	0.5898	-1.5E-07
	0.7	0.5567	-0.284	0.75	0.5425	-0.237	0.75	0.5449	-0.238	0.8	0.5329	-0.192	0.5567	0.5567	-1.4E-07
	0.8	0.5329	-0.192	0.85	0.5234	-0.145	0.85	0.5257	-0.146	0.9	0.5184	-0.098	0.5329	0.5329	-1.3E-07
	0.9	0.5184	-0.098	0.95	0.5135	-0.05	0.95	0.5159	-0.05	1	0.5134	0	0.5184	0.5184	-1.2E-07
	1	0.5134	0	1.05	0.5134	0.0526	1.05	0.516	0.0529	1.1	0.5187	0.1089	0.5134	0.5134	-1.2E-07
	1.1	0.5188	0.1089	1.15	0.5242	0.1691	1.15	0.5272	0.17	1.2	0.5358	0.2357	0.5188	0.5187	-1.2E-07
	1.2	0.5358	0.2358	1.25	0.5476	0.308	1.25	0.5512	0.31	1.3	0.5668	0.3911	0.5358	0.5358	-1.2E-07
	1.3	0.5668	0.3911	1.35	0.5864	0.4823	1.35	0.591	0.4861	1.4	0.6155	0.5908	0.5668	0.5668	-1.1E-07
	1.4	0.6155	0.5909	1.45	0.645	0.7112	1.45	0.651	0.7178	1.5	0.6873	0.8591	0.6155	0.6155	-6.9E-08
	1.5	0.6873	0.8591	1.55	0.7302	1.0242	1.55	0.7385	1.0357	1.6	0.7909	1.2337	0.6873	0.6873	7.08E-08
	1.6	0.7908	1.2337	1.65	0.8525	1.4685	1.65	0.8643	1.4887	1.7	0.9397	1.776	0.7908	0.7908	5.25E-07
	1.7	0.9396	1.7758	1.75	1.0284	2.121	1.75	1.0456	2.1566	1.8	1.1552	2.5877	0.9396	0.9396	1.9E-06
	1.8	1.1549	2.5869	1.85	1.2842	3.111	1.85	1.3104	3.1745	1.9	1.4723	3.8428	1.1549	1.1549	5.82E-06
	1.9	1.4716	3.8408	1.95	1.6636	4.6622	1.95	1.7047	4.7773	2	1.9493	5.8479	1.4716	1.4716	1.67E-05
	2	1.9477	5.8431	2.05	2.2398	7.1731	2.05	2.3063	7.3861	2.1	2.6863	9.1603	1.9477	1.9477	4.59E-05

Solution by java:

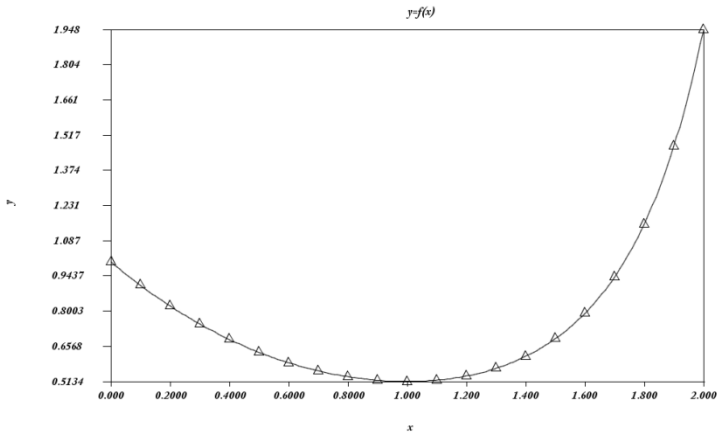
Program 8.4-5 solution of example problem with fourth order Runge-Kutta (RK4) equation

```
 import java.io.*;
abstract class f_xy
{
```

```
  //single function single independent variable
  // example f=x*x
  abstract double func(double x,double y);
}
class fxy extends f_xy
{
  public double func(double x,double y)
  {return y*x*x-y;}
}


class fxyKC extends f_x
{
  public double func(double x)
  {return Math.exp(x*x*x/3.0-x);}
}

class dif1A
{
  public static void main(String args[]) throws IOException
  {
   double x0=0;
   double y0=1.0;
   double xn=2.0;
   fxy f2=new fxy();
   fxyKC f3=new fxyKC();
   Runge_Kutta rk=new Runge_Kutta("RK4");
    System.out.println(rk.toString());
   double XY[][]=rk.hesapla(f2,x0,xn,y0,0.1);
   Plot p1=new Plot(XY);
   p1.addFunction(f3,x0,xn,200);
   p1.setPlotType(0,24);
   p1.plot();
  }
}
```




```
 ---------- Capture Output ----------
> "C:\java\bin\java.exe" dif1A
Buthcher table n = 4
    0.00000000 |
    0.50000000 |     0.50000000
    0.50000000 |     0.00000000     0.50000000
    1.00000000 |     0.00000000     0.00000000     1.00000000
-------------------------------------------------------------------------
                  0.16666667     0.33333333     0.33333333     0.16666667

      0.000000000000000       1.000000000000000
      0.100000000000000       0.905139151280990
      0.200000000000000       0.820917069099996
```

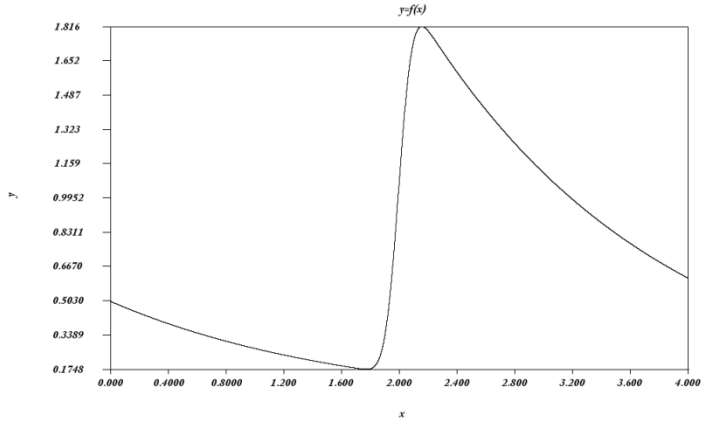
```
0.300000000000000    0.747515827045378
0.400000000000000    0.684773992072904
0.500000000000000    0.632336819274610
0.600000000000000    0.589783505270557
0.700000000000000    0.556734717986668
0.800000000000000    0.532947107134396
0.900000000000000    0.518404336705643
1.000000000000000    0.513417235958808
1.100000000000000    0.518750050786687
1.200000000000000    0.535797072713138
1.300000000000000    0.566846645557574
1.400000000000000    0.615492067374456
1.500000000000000    0.687289208016086
1.600000000000000    0.790833891701643
1.700000000000000    0.939567747881861
1.800000000000000    1.154878288457116
1.900000000000001    1.471558461178071
2.000000000000000    1.947688122952552
```

Program 8.4-6 Runge-Kutta Fehlberg and Runge-Kutta-Verner constant step size codes

```
 import java.io.*;
class fxy2 extends f_xy
{
  double func(double x,double y)
  {return -0.6*y+10.0*Math.exp(-(x-2)*(x-2)/(2*0.075*0.075));}
}

class fxyKC extends f_x
{
  public double func(double x)
  {return Math.exp(x*x*x/3.0-x);}
}

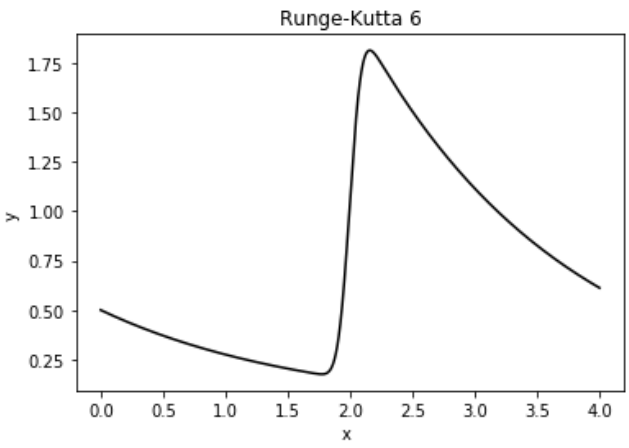
class dif1B
{
  public static void main(String args[]) throws IOException
  {
   double x0=0;
   double y0=0.5;
   double xn=4.0;
   fxy2 f2=new fxy2();
   fxyKC f3=new fxyKC();
   Runge_Kutta rk=new Runge_Kutta("RKF6");
   System.out.println(rk.toString());
   double XY[][]=rk.hesapla(f2,x0,xn,y0,0.001);
   //System.out.println(Matrix.toString(Matrix.Transpose(XY)));
   Plot p1=new Plot(XY);
   p1.plot();
  }
}
```




```
 ---------- Capture Output ----------
> "C:\java\bin\java.exe" dif1B
Buthcher table n = 7
    0.00000000 |
    1.00000000 |    0.00000000
    0.20000000 |    0.00000000    0.20000000
    0.30000000 |    0.00000000    0.07500000    0.22500000
    0.60000000 |    0.00000000    0.30000000   -0.90000000    1.20000000
    0.87500000 |    0.00000000   -0.20370370    0.25000000   -2.59259259    1.29629630
    0.00000000 |    0.00000000    0.00000000    0.00000000    0.00000000    0.00000000    0.00000000
-------------------------------------------------------------------------------------------------------------------
                0.00000000    0.10217737    0.38390790    0.24459274    0.01932199    0.25000000    0.00000000
```

Program 8.4-7 Runge-Kutta Fehlberg constant step size Python codes

```
class f1(f_xj):
    def func(self,X):
        x=X[0];
        y=X[1];
        F=-0.6*y+10.0*exp(-(x-2)*(x-2)/(2*0.075*0.075));
        return F;
rk=Runge_Kutta("RKF6");
f=f1();
XY=rk.calc_1(f,0.0,4.0,0.5,0.001,0);
print(XY);
x=XY[0];
y=XY[1];
plt.title('Runge-Kutta 6')
plt.xlabel('x ')
plt.ylabel('y ');
plt.plot(x,y,'k');
```



9.5 MULTISTEP (OPEN) METHODS (PREDICTOR-CORRECTOR METHODS)

If more than one step of equations are used for calculation of one step solution, it is called multistep method. Some people claims that this type of equations gives more stable solutions. **Milne** Formula is the first multistep method to be investigated.

Milne 1. Step formula (first prediction)

$$y_{i+1}^{p} = y_{i-3} + \frac{4h}{3}[2f(x_i, y_i) - f(x_{i-1}, y_{i-1}) + 2f(x_{i-2}, y_{i-2})]$$

Milne 2. Step Formula (first correction)

$y_{i+1}^{(0)} = y_{j+1}^{p} + \frac{28}{29}[y_i - y_i^p]$

Milne 3. Step formula (second correction)

$y_{i+1}^{(k+1)} = y_{j-1} + \frac{h}{3}[f(x_{i+1}, y_{i+1}^{(k)}) + 4f(x_i, y_i) + f(x_{i-1}, y_{i-1})]$ k=0,1,….

Due to using a first step to predict a guess and then corrected it with one or multistep correctors, Multistep methods are also called **Predictor-Corrector** methods. In order to apply this method, previously calculated points are needed. Single step methods like Runge-Kutta methods can serve us for calculating this first points. As an example problem

$y' = \frac{1}{1+x^2} - 2y^2$diferential equation with the initial conditions y(0)=0 will be solved. Exact analytical solution of this diferential equation

$y = \frac{x}{1+x^2}$

Program 8.5-1 Milne Predictor-corrector formula

```
//=====================================================
// Numerical Analysis package in java
// example to show differential equation solution
// and differentiation (derivative) functions
// Dr. Turhan Coban
// =====================================================
// Basit diferansiyel denklemler dy/dx=f(x,y)

import java.io.*;

abstract class f_xy
{
  //single function single independent variable
  // example f=x*x
  abstract double func(double x,double y);
}
```

```
class fxy extends f_xy
{
  double func(double x,double y)
  {return 1.0/(1.0+x*x)-2.0*y*y;}
}

class fxyKC extends f_x
{
  double func(double x)
  {return x/(1+x*x);}
}

class diferansiyelN
{
  public static double[][] RK6M(f_xy f,double x0,double y0,double xn,double h)
  {
  // Milne method
  int n=(int)((xn-x0)/h);
  double a[][]=new double[2][n+1];
  double yp[]=new double[n+1];
  double y[][]=new double[6][n+1];
  // Runge-Kutta 6
  int i=0;
  a[0][0]=x0;
  a[1][0]=y0;
  double x2=x0+3.0*h;
  double k0,k1,k2,k3,k4,k5,k6;
  for(double xx=x0;xx<x2;xx+=h)
   {a[0][i+1]=xx+h;
   k1=f.func(a[0][i],a[1][i]);
   k2=f.func(a[0][i]+0.25*h,a[1][i]+0.25*k1*h);
   k3=f.func(a[0][i]+0.25*h,a[1][i]+1.0/8.0*k1*h+1.0/8.0*k2*h);
   k4=f.func(a[0][i]+0.5*h,a[1][i]-0.5*k2*h+k3*h);
   k5=f.func(a[0][i]+3.0/4.0*h,a[1][i]+3.0/16.0*k1*h+9.0/16.0*k4*h);
   k6=f.func(a[0][i]+h,a[1][i]-3.0/7.0*k1*h+2.0/7.0*k2*h+12.0/7.0*k3*h-12.0/7.0*k4*h+8.0/7.0*k5*h);
   a[1][i+1]=a[1][i]+(7.0*k1+32.0*k3+12.0*k4+32.0*k5+7*k6)/90.0*h;
   i++;
   }
   for(double xx=x2;xx<xn;xx+=h)
   { a[0][i+1]=xx+h;
   //Milne formülü
   k1=f.func(a[0][i],a[1][i]);
   k2=f.func(a[0][i-1],a[1][i-1]);
   k3=f.func(a[0][i-2],a[1][i-2]);
   yp[i+1]=a[1][i-3]+4.0*h/3.0*(2.0*k1-k2+2.0*k3);
   y[0][i+1]=yp[i+1]+28.0/29.0*(a[1][i]- yp[i]);
   int k;
   for(k=0;k<5;k++)
   {
                  k0=f.func(a[0][i+1],y[k][i+1]);
      y[k+1][i+1]=a[1][i-1]+h/3.0*(k0+4.0*k1+k2);
   }
   a[1][i+1]=y[k][i+1];
   i++;
   }
  return a;
  }

  public static void main(String args[]) throws IOException
  {double x0=0;
   double y0=0.0;
   double yn=10.0;
   double h=0.0625;
```

```
    fxy f2=new fxy();
    fxyKC fKC=new fxyKC();
    double a[][]=RK6M(f2,x0,y0,yn,h);
    Text.print(Text.T(a));
    Plot p1=new Plot(a);
    p1.setPlotType(0,24);
    p1.addFunction(fKC,0.0,10.0,50);
    p1.plot();
   }
}
```

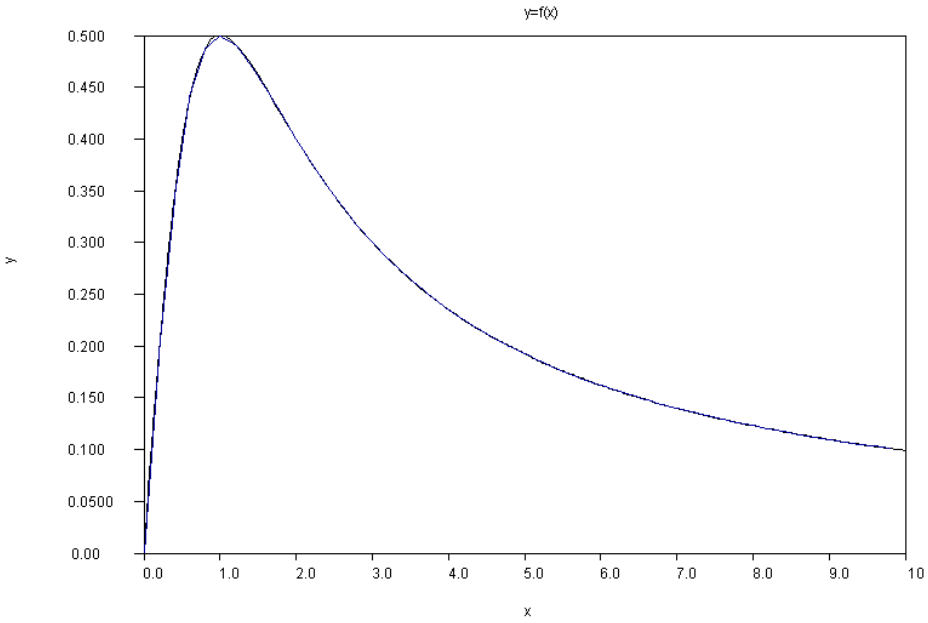
Output 8.5-1 Milne Predictor-Corrector method table output with error term

	Milne solution	Exact solution	Error
0	**0**	**0**	**0**
0.0625	**0.062256809**	**0.062256809**	**1.29392E-10**
0.125	**0.123076922**	**0.123076923**	**6.07481E-10**
0.1875	**0.181132074**	**0.181132075**	**1.36894E-09**
0.25	0.235294614	0.235294118	-4.9588E-07
0.3125	0.284697577	0.284697509	-6.78819E-08
0.375	0.328767367	0.328767123	-2.43534E-07
0.4375	0.367212702	0.367213115	4.1304E-07
0.5	0.399999711	0.4	2.8879E-07
0.5625	0.427298769	0.427299703	9.34563E-07
0.625	0.449437527	0.449438202	6.75003E-07
0.6875	0.466842287	0.466843501	1.2147E-06
0.75	0.479999216	0.48	7.8413E-07
0.8125	0.489410522	0.489411765	1.24273E-06
0.875	0.495574537	0.495575221	6.83803E-07
0.9375	0.498959376	0.498960499	1.1231E-06
1	0.499999514	0.5	4.86142E-07
1.0625	0.499081613	0.499082569	9.56231E-07
1.125	0.496551452	0.496551724	2.72191E-07
1.1875	0.492705844	0.492706645	8.01101E-07
1.25	0.487804796	0.487804878	8.16887E-08
1.3125	0.482065316	0.482065997	6.81141E-07
1.375	0.475675749	0.475675676	-7.3537E-08
1.4375	0.468789209	0.468789809	5.99558E-07
1.5	0.461538657	0.461538462	-1.95371E-07
1.5625	0.454028961	0.454029512	5.51033E-07
1.625	0.446352222	0.446351931	-2.9049E-07
1.6875	0.438578152	0.43857868	5.28107E-07
1.75	0.430769597	0.430769231	-3.6604E-07
1.8125	0.422971217	0.422971741	5.23921E-07
1.875	0.415225342	0.415224913	-4.28026E-07
1.9375	0.40755904	0.407559573	5.33046E-07
2	0.400000481	0.4	-4.8098E-07
2.0625	0.392564504	0.392565056	5.51496E-07
2.125	0.38526965	0.385269122	-5.28128E-07
2.1875	0.378122313	0.37812289	5.76463E-07
2.25	0.371134592	0.371134021	-5.7169E-07
2.3125	0.364307086	0.364307692	6.06014E-07
2.375	0.357647672	0.357647059	-6.13158E-07
2.4375	0.351152991	0.35115363	6.38839E-07
2.5	0.34482824	0.344827586	-6.53519E-07
2.5625	0.338667369	0.338668043	6.74055E-07
2.625	0.332673961	0.332673267	-6.93416E-07
2.6875	0.326840144	0.326840855	7.11066E-07
2.75	0.321168616	0.321167883	-7.33265E-07
2.8125	0.315650281	0.31565103	7.49474E-07
2.875	0.310287451	0.310286678	-7.73331E-07

2.9375	0.305070205	0.305070994	7.89007E-07
3	0.300000814	0.3	-8.13778E-07
3.0625	0.295068798	0.295069627	8.29482E-07
3.125	0.290276617	0.290275762	-8.54704E-07
3.1875	0.28561341	0.285614281	8.70771E-07
3.25	0.281081977	0.281081081	-8.96169E-07
3.3125	0.276671192	0.276672104	9.12786E-07
3.375	0.272384293	0.272383354	-9.38201E-07
3.4375	0.268209956	0.268210911	9.55464E-07
3.5	0.264151924	0.264150943	-9.80814E-07
3.5625	0.260198716	0.260199715	9.98759E-07
3.625	0.256354615	0.256353591	-1.02401E-06
3.6875	0.252608002	0.252609045	1.04264E-06
3.75	0.248963723	0.248962656	-1.06778E-06
3.8125	0.245410027	0.245411114	1.08707E-06
3.875	0.241952332	0.24195122	-1.11212E-06
3.9375	0.23857875	0.238579882	1.13204E-06
4	0.235295275	0.235294118	-1.15702E-06
4.0625	0.232089874	0.232091051	1.17754E-06
4.125	0.228969112	0.22896791	-1.20245E-06
4.1875	0.2259208	0.225922023	1.22353E-06
4.25	0.222952068	0.22295082	-1.24842E-06
4.3125	0.220050554	0.220051824	1.27002E-06
4.375	0.217223948	0.217222653	-1.29491E-06
4.4375	0.214459699	0.214461016	1.31699E-06
4.5	0.211766048	0.211764706	-1.3419E-06
4.5625	0.209130238	0.209131603	1.36443E-06
4.625	0.206561054	0.206559665	-1.38939E-06
4.6875	0.204045518	0.204046931	1.41233E-06
4.75	0.201592949	0.201591512	-1.43735E-06
4.8125	0.199190132	0.199191593	1.46069E-06
4.875	0.196846912	0.196845426	-1.48579E-06
4.9375	0.194549822	0.194551331	1.50949E-06
5	0.192309227	0.192307692	-1.53468E-06
5.0625	0.190111394	0.190112953	1.55873E-06
5.125	0.1879672	0.187965616	-1.58403E-06
5.1875	0.185862632	0.185864241	1.6084E-06
5.25	0.183809074	0.18380744	-1.63382E-06
5.3125	0.181792219	0.181793878	1.65849E-06
5.375	0.179823953	0.179822269	-1.68403E-06
5.4375	0.177889665	0.177891374	1.70899E-06
5.5	0.176001735	0.176	-1.73467E-06
5.5625	0.174145238	0.174146998	1.7599E-06
5.625	0.172333045	0.172331259	-1.78573E-06
5.6875	0.170549905	0.170551716	1.81122E-06
5.75	0.168809177	0.168807339	-1.83719E-06
5.8125	0.167095274	0.167097136	1.86293E-06
5.875	0.165422039	0.16542015	-1.88905E-06
5.9375	0.16377354	0.163775455	1.91503E-06
6	0.162164103	0.162162162	-1.94131E-06
6.0625	0.160577443	0.16057941	1.96752E-06
6.125	0.159028363	0.159026369	-1.99395E-06
6.1875	0.157500217	0.157502237	2.02038E-06
6.25	0.156008287	0.15600624	-2.04697E-06
6.3125	0.154535557	0.15453763	2.07362E-06
6.375	0.153097785	0.153095685	-2.10037E-06
6.4375	0.151677578	0.151679705	2.12722E-06
6.5	0.150291171	0.150289017	-2.15413E-06
6.5625	0.148920787	0.148922968	2.18119E-06
6.625	0.147583134	0.147580926	-2.20826E-06

6.6875	0.146260046	0.146262281	2.23552E-06
6.75	0.144968706	0.144966443	-2.26275E-06
6.8125	0.14369055	0.14369284	2.29019E-06
6.875	0.142443237	0.142440919	-2.31759E-06
6.9375	0.1412078	0.141210146	2.34522E-06
7	0.140002373	0.14	-2.37278E-06
7.0625	0.13880758	0.138809981	2.40059E-06
7.125	0.13764203	0.137639602	-2.42831E-06
7.1875	0.136485935	0.136488391	2.4563E-06
7.25	0.135358377	0.135355893	-2.48418E-06
7.3125	0.134239151	0.134241664	2.51235E-06
7.375	0.133147815	0.133145275	-2.54039E-06
7.4375	0.132063742	0.132066311	2.56873E-06
7.5	0.131006964	0.131004367	-2.59693E-06
7.5625	0.129956427	0.129959052	2.62544E-06
7.625	0.128932641	0.128929987	-2.65379E-06
7.6875	0.12791412	0.127916802	2.68247E-06
7.75	0.126921851	0.12691914	-2.71098E-06
7.8125	0.125933914	0.125936654	2.73981E-06
7.875	0.124971774	0.124969006	-2.76848E-06
7.9375	0.124013071	0.124015868	2.79748E-06
8	0.123079749	0.123076923	-2.8263E-06
8.0625	0.122149006	0.122151861	2.85546E-06
8.125	0.121243267	0.121240382	-2.88443E-06
8.1875	0.120339281	0.120342194	2.91375E-06
8.25	0.119459956	0.119457014	-2.94287E-06
8.3125	0.118581592	0.118584564	2.97234E-06
8.375	0.117727579	0.117724577	-3.00161E-06
8.4375	0.116873761	0.116876792	3.03124E-06
8.5	0.116044016	0.116040956	-3.06066E-06
8.5625	0.11521373	0.11521682	3.09043E-06
8.625	0.114407265	0.114404145	-3.12E-06
8.6875	0.113599547	0.113602697	3.14993E-06
8.75	0.112815428	0.112812248	-3.17964E-06
8.8125	0.112029367	0.112032577	3.20971E-06
8.875	0.111266707	0.111263467	-3.23957E-06
8.9375	0.110501439	0.110504709	3.26979E-06
9	0.109759397	0.109756098	-3.29979E-06
9.0625	0.109014103	0.109017433	3.33015E-06
9.125	0.108291882	0.108288522	-3.3603E-06
9.1875	0.107565784	0.107569174	3.3908E-06
9.25	0.106862627	0.106859206	-3.42109E-06
9.3125	0.106154984	0.106158436	3.45173E-06
9.375	0.105470172	0.10546669	-3.48216E-06
9.4375	0.104780284	0.104783797	3.51294E-06
9.5	0.104113133	0.104109589	-3.5435E-06
9.5625	0.10344033	0.103443905	3.57443E-06
9.625	0.102790189	0.102786584	-3.60513E-06
9.6875	0.102133838	0.102137474	3.63619E-06
9.75	0.101500089	0.101496422	-3.66703E-06
9.8125	0.100859582	0.10086328	3.69823E-06
9.875	0.100241636	0.100237906	-3.72919E-06
9.9375	0.099616398	0.099620159	3.76053E-06
10	0.099013693	0.099009901	-3.79163E-06

Output 8.5-2 Milne Predictor-Corrector method graphic output



As a second method, Hamming Formula will be investigated. This method is very close to Milne method. As amound of error it is a little worst than Milne method, but its stability is much better.

Hamming 1. step formula (first prediction)

$$y_{i+1}^{p} = y_{i-3} + \frac{4h}{3}[2f(x_i, y_i) - f(x_{i-1}, y_{i-1}) + 2f(x_{i-2}, y_{i-2})]$$

Hamming 2. Step formula (first correction)

$$y_{i+1}^{(0)} = y_{j+1}^{p} + \frac{112}{121}\left[y_i - y_i^{p}\right]$$

Hamming 3. Step formula (second correction)

$$y_{i+1}^{(k+1)} = \frac{1}{8}(9y_i - y_{i-2}) + \frac{3h}{8}\left[f(x_{i+1}, y_{i+1}^{(k)}) + 2f(x_i, y_i) - f(x_{i-1}, y_{i-1})\right] \quad \text{k=0,1,....}$$

Program 8.5-2 Hamming Predictor-corrector Formula

```
//=====================================================
// Numerical Analysis package in java
// example to show differential equation solution
// Hamming Predictor-Corrector method
// Dr. Turhan Coban
// =====================================================
// dy/dx=f(x,y)

import java.io.*;

abstract class f_xy
{
  //single function single independent variable
  // example f=x*x
  abstract double func(double x,double y);
}

class fxy extends f_xy
{
  double func(double x,double y)
  {return 1.0/(1.0+x*x)-2.0*y*y;}
}

class fxyKC extends f_x
```

```
{
  double func(double x)
  {return x/(1+x*x);}
}

class NA90
{
  public static double[][] RK6M(f_xy f,double x0,double y0,double xn,double h)
  {
  // Hamming formülü
  int n=(int)((xn-x0)/h);
  double a[][]=new double[2][n+1];
  double yp[]=new double[n+1];
  double y[][]=new double[6][n+1];
  // Runge-Kutta 6
  int i=0;
  a[0][0]=x0;
  a[1][0]=y0;
  double x2=x0+3.0*h;
  double k0,k1,k2,k3,k4,k5,k6;
  for(double xx=x0;xx<x2;xx+=h)
   {a[0][i+1]=xx+h;
   k1=f.func(a[0][i],a[1][i]);
   k2=f.func(a[0][i]+0.25*h,a[1][i]+0.25*k1*h);
   k3=f.func(a[0][i]+0.25*h,a[1][i]+1.0/8.0*k1*h+1.0/8.0*k2*h);
   k4=f.func(a[0][i]+0.5*h,a[1][i]-0.5*k2*h+k3*h);
   k5=f.func(a[0][i]+3.0/4.0*h,a[1][i]+3.0/16.0*k1*h+9.0/16.0*k4*h);
   k6=f.func(a[0][i]+h,a[1][i]-3.0/7.0*k1*h+2.0/7.0*k2*h+12.0/7.0*k3*h-12.0/7.0*k4*h+8.0/7.0*k5*h);
   a[1][i+1]=a[1][i]+(7.0*k1+32.0*k3+12.0*k4+32.0*k5+7*k6)/90.0*h;
   i++;
   }
   for(double xx=x2;xx<xn;xx+=h)
   { a[0][i+1]=xx+h;
   //Milne formülü
   k1=f.func(a[0][i],a[1][i]);
   k2=f.func(a[0][i-1],a[1][i-1]);
   k3=f.func(a[0][i-2],a[1][i-2]);
   yp[i+1]=a[1][i-3]+4.0*h/3.0*(2.0*k1-k2+2.0*k3);
   y[0][i+1]=yp[i+1]+112.0/121.0*(a[1][i]- yp[i]);
   int k;
   for(k=0;k<5;k++)
   {
                k0=f.func(a[0][i+1],y[k][i+1]);
      y[k+1][i+1]=1.0/8.0*(9.0*a[1][i]-a[1][i-2])+3.0*h/8.0*(k0+2.0*k1-k2);
   }
   a[1][i+1]=y[k][i+1];
   i++;
   }
  return a;
  }

  public static void main(String args[]) throws IOException
  {double x0=0;
   double y0=0.0;
   double yn=10.0;
   double h=0.0625;
   fxy f2=new fxy();
   fxyKC fKC=new fxyKC();
   double a[][]=RK6M(f2,x0,y0,yn,h);
   Text.print(Text.T(a));
   Plot p1=new Plot(a);
   p1.setPlotType(0,24);
   p1.addFunction(fKC,0.0,10.0,50);
   p1.plot();
```

```
  }
}
```

Output 8.5-2 Hamming Predictor-corrector Formula calculated values

	Hamming solution	Exact solution	Error
0	**0**	**0**	**0**
0.0625	**0.062256809**	**0.062256809**	**1.29392E-10**
0.125	**0.123076922**	**0.123076923**	**6.07481E-10**
0.1875	**0.181132074**	**0.181132075**	**1.36894E-09**
0.25	0.235295447	0.235294118	-1.32917E-06
0.3125	0.284699363	0.284697509	-1.85455E-06
0.375	0.328768754	0.328767123	-1.63052E-06
0.4375	0.367213781	0.367213115	-6.6612E-07
0.5	0.399999285	0.4	7.14995E-07
0.5625	0.427297497	0.427299703	2.20588E-06
0.625	0.449434654	0.449438202	3.5483E-06
0.6875	0.466838916	0.466843501	4.58534E-06
0.75	0.479994747	0.48	5.25346E-06
0.8125	0.489406203	0.489411765	5.56192E-06
0.875	0.495569657	0.495575221	5.56428E-06
0.9375	0.498955165	0.498960499	5.33375E-06
1	0.499995055	0.5	4.94536E-06
1.0625	0.499078103	0.499082569	4.46539E-06
1.125	0.496547778	0.496551724	3.94663E-06
1.1875	0.492703217	0.492706645	3.42766E-06
1.25	0.487801944	0.487804878	2.93423E-06
1.3125	0.482063515	0.482065997	2.48172E-06
1.375	0.475673598	0.475675676	2.07773E-06
1.4375	0.468788085	0.468789809	1.7244E-06
1.5	0.461537041	0.461538462	1.42031E-06
1.5625	0.45402835	0.454029512	1.16193E-06
1.625	0.446350987	0.446351931	9.44654E-07
1.6875	0.438577917	0.43857868	7.63473E-07
1.75	0.430768617	0.430769231	6.13446E-07
1.8125	0.422971251	0.422971741	4.89946E-07
1.875	0.415224525	0.415224913	3.88796E-07
1.9375	0.407559266	0.407559573	3.06316E-07
2	0.399999761	0.4	2.3933E-07
2.0625	0.392564871	0.392565056	1.85132E-07
2.125	0.38526898	0.385269122	1.41441E-07
2.1875	0.378122784	0.37812289	1.06351E-07
2.25	0.371133942	0.371134021	7.82796E-08
2.3125	0.364307636	0.364307692	5.59194E-08
2.375	0.357647021	0.357647059	3.81956E-08
2.4375	0.351153605	0.35115363	2.42269E-08
2.5	0.344827573	0.344827586	1.3293E-08
2.5625	0.338668039	0.338668043	4.80608E-09
2.625	0.332673269	0.332673267	-1.71236E-09
2.6875	0.326840862	0.326840855	-6.65133E-09
2.75	0.321167894	0.321167883	-1.03267E-08
2.8125	0.315651043	0.31565103	-1.29944E-08
2.875	0.310286693	0.310286678	-1.4862E-08
2.9375	0.30507101	0.305070994	-1.6097E-08
3	0.300000017	0.3	-1.68349E-08
3.0625	0.295069645	0.295069627	-1.7185E-08
3.125	0.290275779	0.290275762	-1.72351E-08
3.1875	0.285614298	0.285614281	-1.70561E-08
3.25	0.281081098	0.281081081	-1.67044E-08
3.3125	0.276672121	0.276672104	-1.62255E-08
3.375	0.27238337	0.272383354	-1.56554E-08
3.4375	0.268210926	0.268210911	-1.50227E-08
3.5	0.264150958	0.264150943	-1.435E-08
3.5625	0.260199728	0.260199715	-1.36552E-08
3.625	0.256353604	0.256353591	-1.29519E-08
3.6875	0.252609057	0.252609045	-1.22508E-08
3.75	0.248962667	0.248962656	-1.15601E-08
3.8125	0.245411125	0.245411114	-1.08858E-08
3.875	0.24195123	0.24195122	-1.02323E-08

3.9375	0.238579891	0.238579882	-9.60278E-09
4	0.235294127	0.235294118	-8.99929E-09
4.0625	0.23209106	0.232091051	-8.42314E-09
4.125	0.228967918	0.22896791	-7.87496E-09
4.1875	0.225922031	0.225922023	-7.35492E-09
4.25	0.222950827	0.22295082	-6.86278E-09
4.3125	0.22005183	0.220051824	-6.39803E-09
4.375	0.217222659	0.217222653	-5.95992E-09
4.4375	0.214461021	0.214461016	-5.54758E-09
4.5	0.211764711	0.211764706	-5.16002E-09
4.5625	0.209131607	0.209131603	-4.79616E-09
4.625	0.206559669	0.206559665	-4.4549E-09
4.6875	0.204046935	0.204046931	-4.13513E-09
4.75	0.201591516	0.201591512	-3.83572E-09
4.8125	0.199191596	0.199191593	-3.55556E-09
4.875	0.196845429	0.196845426	-3.29356E-09
4.9375	0.194551334	0.194551331	-3.04869E-09
5	0.192307695	0.192307692	-2.81991E-09
5.0625	0.190112956	0.190112953	-2.60627E-09
5.125	0.187965618	0.187965616	-2.40683E-09
5.1875	0.185864243	0.185864241	-2.22071E-09
5.25	0.183807442	0.18380744	-2.04707E-09
5.3125	0.18179388	0.181793878	-1.88511E-09
5.375	0.17982227	0.179822269	-1.73409E-09
5.4375	0.177891375	0.177891374	-1.59329E-09
5.5	0.176000001	0.176	-1.46205E-09
5.5625	0.174146999	0.174146998	-1.33975E-09
5.625	0.17233126	0.172331259	-1.22579E-09
5.6875	0.170551717	0.170551716	-1.11963E-09
5.75	0.16880734	0.168807339	-1.02075E-09
5.8125	0.167097137	0.167097136	-9.28673E-10
5.875	0.16542015	0.16542015	-8.42937E-10
5.9375	0.163775456	0.163775455	-7.63125E-10
6	0.162162163	0.162162162	-6.88839E-10
6.0625	0.160579411	0.16057941	-6.19713E-10
6.125	0.15902637	0.159026369	-5.55399E-10
6.1875	0.157502238	0.157502237	-4.95579E-10
6.25	0.156006241	0.15600624	-4.39949E-10
6.3125	0.154537631	0.15453763	-3.88231E-10
6.375	0.153095685	0.153095685	-3.40162E-10
6.4375	0.151679706	0.151679705	-2.95498E-10
6.5	0.150289018	0.150289017	-2.54013E-10
6.5625	0.148922968	0.148922968	-2.15491E-10
6.625	0.147580926	0.147580926	-1.79737E-10
6.6875	0.146262281	0.146262281	-1.46565E-10
6.75	0.144966443	0.144966443	-1.15802E-10
6.8125	0.14369284	0.14369284	-8.72877E-11
6.875	0.142440919	0.142440919	-6.08722E-11
6.9375	0.141210146	0.141210146	-3.64138E-11
7	0.14	0.14	-1.3784E-11
7.0625	0.138809981	0.138809981	7.14004E-12
7.125	0.137639602	0.137639602	2.64724E-11
7.1875	0.136488391	0.136488391	4.43198E-11
7.25	0.135355893	0.135355893	6.07798E-11
7.3125	0.134241664	0.134241664	7.59459E-11
7.375	0.133145275	0.133145275	8.99049E-11
7.4375	0.132066311	0.132066311	1.02736E-10
7.5	0.131004367	0.131004367	1.14516E-10
7.5625	0.129959052	0.129959052	1.25316E-10
7.625	0.128929987	0.128929987	1.35198E-10
7.6875	0.127916802	0.127916802	1.44227E-10
7.75	0.12691914	0.12691914	1.52459E-10
7.8125	0.125936654	0.125936654	1.59948E-10
7.875	0.124969006	0.124969006	1.66743E-10
7.9375	0.124015868	0.124015868	1.72892E-10
8	0.123076923	0.123076923	1.78437E-10
8.0625	0.122151861	0.122151861	1.83421E-10
8.125	0.121240382	0.121240382	1.8788E-10
8.1875	0.120342194	0.120342194	1.91852E-10

8.25	0.119457013	0.119457014	1.95369E-10
8.3125	0.118584564	0.118584564	1.98462E-10
8.375	0.117724577	0.117724577	2.0116E-10
8.4375	0.116876792	0.116876792	2.03492E-10
8.5	0.116040955	0.116040956	2.05481E-10
8.5625	0.11521682	0.11521682	2.07154E-10
8.625	0.114404145	0.114404145	2.0853E-10
8.6875	0.113602697	0.113602697	2.09633E-10
8.75	0.112812248	0.112812248	2.10481E-10
8.8125	0.112032577	0.112032577	2.11091E-10
8.875	0.111263467	0.111263467	2.11482E-10
8.9375	0.110504709	0.110504709	2.1167E-10
9	0.109756097	0.109756098	2.1167E-10
9.0625	0.109017433	0.109017433	2.11494E-10
9.125	0.108288522	0.108288522	2.11156E-10
9.1875	0.107569174	0.107569174	2.1067E-10
9.25	0.106859206	0.106859206	2.10047E-10
9.3125	0.106158436	0.106158436	2.09296E-10
9.375	0.10546669	0.10546669	2.08428E-10
9.4375	0.104783796	0.104783797	2.07452E-10
9.5	0.104109589	0.104109589	2.06378E-10
9.5625	0.103443904	0.103443905	2.05214E-10
9.625	0.102786584	0.102786584	2.03965E-10
9.6875	0.102137474	0.102137474	2.02642E-10
9.75	0.101496421	0.101496422	2.01251E-10
9.8125	0.10086328	0.10086328	1.99795E-10
9.875	0.100237906	0.100237906	1.98283E-10
9.9375	0.099620159	0.099620159	1.96719E-10
10	0.099009901	0.099009901	1.9511E-10

First points calculated by Runge Kutta -6 method is given with bold font. As the third method **Adams-Bashford-Moulton formulas.** This formulas like Runge-Kutta sets can be given in different orders. They are given as two sets. The first set (Adams-Bashford) is used as a predictor, and the second set is used as a corrector step.

Predictor Formula :

Adams-Bashford 1 step formula

$y_{i+1} = y_i + hf(x_i, y_i)$

Adams-Bashford 2 step formula

$$y_{i+1} = y_i + \frac{h}{2}(3f(x_i, y_i) - f(x_{i-1}, y_{i-1}))$$

Adams-Bashford 3 step formula

$$y_{i+1} = y_i + \frac{h}{12}(23f(x_i, y_i) - 16f(x_{i-1}, y_{i-1}) + 5f(x_{i-2}, y_{i-2}))$$

Adams-Bashford 4 step formula

$$y_{i+1} = y_i + \frac{h}{24}(55f(x_i, y_i) - 59f(x_{i-1}, y_{i-1}) + 37f(x_{i-2}, y_{i-2}) - 9f(x_{i-3}, y_{i-3}))$$

Adams-Bashford 5 step formula

$$y_{i+1} = y_i + \frac{h}{720}(1901f(x_i, y_i) - 2774f(x_{i-1}, y_{i-1}) + 2616f(x_{i-2}, y_{i-2}) - 1274f(x_{i-3}, y_{i-3}) + 251f(x_{i-4}, y_{i-4}))$$

Corrector Formula :

Adams-Moulton 0 step formula

$$y_{i+1} = y_i + hf(x_{i+1}, y_{i+1})$$

Adams-Moulton 1 step formula

$$y_{i+1} = y_i + \frac{h}{2}(f(x_{i+1}, y_{i+1}) + f(x_i, y_i))$$

Adams-Moulton 2 step formula

$$y_{i+1} = y_i + \frac{h}{12}(5f(x_{i+1}, y_{i+!}) + 8f(x_i, y_i) - f(x_{i-1}, y_{i-!}))$$

Adams-Moulton 3 step formula

$$y_{i+1} = y_i + \frac{h}{24}(9f(x_{i+1}, y_{i+1}) + 19f(x_i, y_i) - 5f(x_{i-1}, y_{i-1}) + f(x_{i-2}, y_{i-2}))$$

Adams-Moulton 4 step formula

$y_{i+1} = y_i + \frac{h}{720}(251f(x_{i+1}, y_{i+1}) + 646f(x_i, y_i) - 264f(x_{i-1}, y_{i-1}) + 106f(x_{i-2}, y_{i-2}) - 19f(x_{i-3}, y_{i-3}))$Program list is given as the following program. In this program two method is defined. In the first method Adams-Bashford 4 step Formula is used together with Adams-Milton 3 step Formula. In the second method Adams-Bashford 5 step Formula is used together with Adams-Milton 4 step Formula.

Program 8.5-3 Adams-Bashford-Milton Predictor-corrector method.

```
//=====================================================
// Numerical Analysis package in java
// example to show differential equation solution
// Adams-BAshford-Milton Predictor Corrector formula
// with Adams-Bashford 4 step predictor and Adams-Milford 3 step corrector
// also with Adams-Bashford 5 step predictor and Adams-Milford 4 step corrector
// Dr. Turhan Coban
// =====================================================
// dy/dx=f(x,y)

import java.io.*;

abstract class f_xy
{
  //single function single independent variable
  // example f=x*x
  abstract double func(double x,double y);
}

class fxy extends f_xy
{
  double func(double x,double y)
  {return 1.0/(1.0+x*x)-2.0*y*y;}
}

class fxyKC extends f_x
{
  double func(double x)
  {return x/(1+x*x);}
}

class NA91
{
  public static double[][] RK6AB4AM3(f_xy f,double x0,double y0,double xn,double h)
  {
  // Adams-Bashford (4 adim formülü)  Adams-Moulton (3 adim formülü) predictor corrector method
  //with RK6 initial step
  int n=(int)((xn-x0)/h);
  double a[][]=new double[2][n+1];
  // Runge-Kutta 6
  int i=0;
  a[0][0]=x0;
  a[1][0]=y0;
  double x3=x0+3.0*h;
  double k0,k1,k2,k3,k4,k5,k6;
  for(double xx=x0;xx<x3;xx+=h)
   {a[0][i+1]=xx+h;
   k1=f.func(a[0][i],a[1][i]);
   k2=f.func(a[0][i]+0.25*h,a[1][i]+0.25*k1*h);
   k3=f.func(a[0][i]+0.25*h,a[1][i]+1.0/8.0*k1*h+1.0/8.0*k2*h);
   k4=f.func(a[0][i]+0.5*h,a[1][i]-0.5*k2*h+k3*h);
   k5=f.func(a[0][i]+3.0/4.0*h,a[1][i]+3.0/16.0*k1*h+9.0/16.0*k4*h);
   k6=f.func(a[0][i]+h,a[1][i]-3.0/7.0*k1*h+2.0/7.0*k2*h+12.0/7.0*k3*h-12.0/7.0*k4*h+8.0/7.0*k5*h);
```

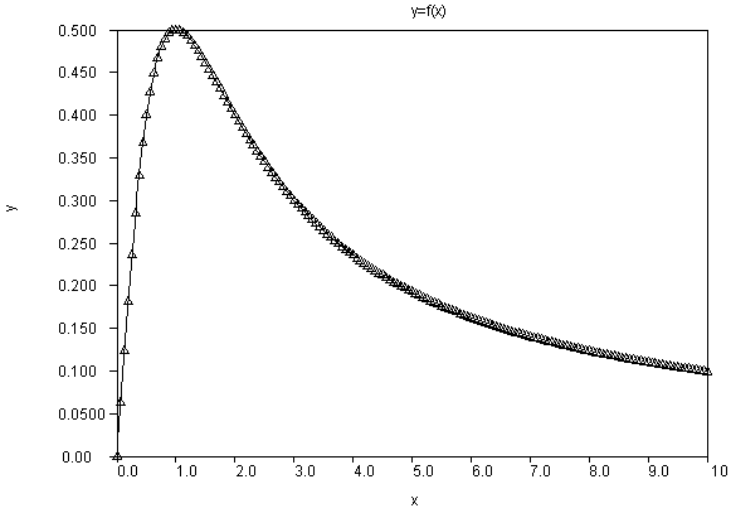
```
 a[1][i+1]=a[1][i]+(7.0*k1+32.0*k3+12.0*k4+32.0*k5+7*k6)/90.0*h;
 i++;
 }
 for(double xx=x3;xx<xn;xx+=h)
 {a[0][i+1]=xx+h;
 //Adams-Bashford
 k1=f.func(a[0][i],a[1][i]);
 k2=f.func(a[0][i-1],a[1][i-1]);
 k3=f.func(a[0][i-2],a[1][i-2]);
 k4=f.func(a[0][i-3],a[1][i-3]);
 a[1][i+1]=a[1][i]+h/24.0*(55.0*k1-59.0*k2+37.0*k3-9.0*k4);
 //Adams-Moulton j 1 den büyük olabilir
 for(int j=0;j<5;j++)
 {k0=f.func(a[0][i+1],a[1][i+1]);
 a[1][i+1]=a[1][i]+h/24.0*(9.0*k0+19.0*k1-5.0*k2+k3);
 }
 i++;
 }
return a;
}

public static double[][] RK6AB5AM4(f_xy f,double x0,double y0,double xn,double h)
{
// Adams-Bashford (5 adim formülü)  Adams-Moulton (4 adim formülü) predictor corrector method
//with RK6 initial step
int n=(int)((xn-x0)/h);
double a[][]=new double[2][n+1];
// Runge-Kutta 6
int i=0;
a[0][0]=x0;
a[1][0]=y0;
double x4=x0+4.0*h;
double k0,k1,k2,k3,k4,k5,k6;
for(double xx=x0;xx<x4;xx+=h)
 {a[0][i+1]=xx+h;
 k1=f.func(a[0][i],a[1][i]);
 k2=f.func(a[0][i]+0.25*h,a[1][i]+0.25*k1*h);
 k3=f.func(a[0][i]+0.25*h,a[1][i]+1.0/8.0*k1*h+1.0/8.0*k2*h);
 k4=f.func(a[0][i]+0.5*h,a[1][i]-0.5*k2*h+k3*h);
 k5=f.func(a[0][i]+3.0/4.0*h,a[1][i]+3.0/16.0*k1*h+9.0/16.0*k4*h);
 k6=f.func(a[0][i]+h,a[1][i]-3.0/7.0*k1*h+2.0/7.0*k2*h+12.0/7.0*k3*h-12.0/7.0*k4*h+8.0/7.0*k5*h);
 a[1][i+1]=a[1][i]+(7.0*k1+32.0*k3+12.0*k4+32.0*k5+7*k6)/90.0*h;
 i++;
 }
 for(double xx=x4;xx<xn;xx+=h)
 {a[0][i+1]=xx+h;
 //Adams-Bashford
 k1=f.func(a[0][i],a[1][i]);
 k2=f.func(a[0][i-1],a[1][i-1]);
 k3=f.func(a[0][i-2],a[1][i-2]);
 k4=f.func(a[0][i-3],a[1][i-3]);
 k5=f.func(a[0][i-4],a[1][i-4]);
 a[1][i+1]=a[1][i]+h/720.0*(1901.0*k1-2774.0*k2+2616.0*k3-1274.0*k4+251*k5);
 //Adams-Moulton j 1 den büyük olabilir
 for(int j=0;j<5;j++)
 {k0=f.func(a[0][i+1],a[1][i+1]);
 a[1][i+1]=a[1][i]+h/720.0*(251.0*k0+646.0*k1-264.0*k2+106.0*k3-19*k4);
 }
 i++;
 }
return a;
}

public static void main(String args[]) throws IOException
```

```
  {double x0=0;
   double y0=0.0;
   double yn=10.0;
   double h=0.0625;
   fxy f2=new fxy();
   fxyKC fKC=new fxyKC();
   double a[][]=RK6AB4AM3(f2,x0,y0,yn,h);
   Text.print(Text.T(a));
   Plot p1=new Plot(a);
   p1.setPlotType(0,24);
   //p1.addSpline(a,10);
   p1.addFunction(fKC,0.0,4.0,200);
   p1.plot();
  }
}
```

Output 8.5-4 Adams-Bashford 4 step Adams-Milton 3 step Predictor-corrector formüla graphic output



In order to check the error created solution and exact solution is given as a table. The first steps calculated with Runge –Kutta 6 method is given as bold font.

Output 8.5-5 Adams-Bashford 4 step Adams-Milton 3 step Predictor-corrector formüla calculated output with error

x	y_calculated	y_exact	Error
0	**0**	**0**	**0**
0.0625	**0.062256809**	**0.062256809**	**1.29392E-10**
0.125	**0.123076922**	**0.123076923**	**6.07481E-10**
0.1875	**0.181132074**	**0.181132075**	**1.36894E-09**
0.25	0.235295557	0.235294118	-1.43921E-06
0.3125	0.284699363	0.284697509	-1.8537E-06
0.375	0.328768525	0.328767123	-1.40145E-06
0.4375	0.367213479	0.367213115	-3.64545E-07
0.5	0.399999069	0.4	9.31283E-07
0.5625	0.427297499	0.427299703	2.20423E-06
0.625	0.449434941	0.449438202	3.26136E-06
0.6875	0.466839495	0.466843501	4.0062E-06
0.75	0.479995578	0.48	4.42175E-06
0.8125	0.489407221	0.489411765	4.54356E-06
0.875	0.495570788	0.495575221	4.43364E-06
0.9375	0.498956338	0.498960499	4.16066E-06
1	0.499996212	0.5	3.78762E-06
1.0625	0.499079203	0.499082569	3.36586E-06
1.125	0.496548791	0.496551724	2.93349E-06
1.1875	0.492704129	0.492706645	2.51644E-06
1.25	0.487802747	0.487804878	2.13058E-06
1.3125	0.482064213	0.482065997	1.78422E-06
1.375	0.475674195	0.475675676	1.48028E-06
1.4375	0.468788591	0.468789809	1.21822E-06
1.5	0.461537466	0.461538462	9.95355E-07
1.5625	0.454028704	0.454029512	8.0791E-07
1.625	0.44635128	0.446351931	6.51661E-07
1.6875	0.438578158	0.43857868	5.22374E-07

1.75	0.430768815	0.430769231	4.16056E-07
1.8125	0.422971412	0.422971741	3.29088E-07
1.875	0.415224655	0.415224913	2.58278E-07
1.9375	0.407559372	0.407559573	2.00868E-07
2	0.399999845	0.4	1.54505E-07

In order to see the error comparisons, the same problem is also solved with Adam Bashford 5 step predictor and Adams-Milton 4 step corrector

Output 8.5-3 Adams-Bashford 5 step predictor and Adams-Milton 4 step corrector and the error created compare to exact solution.

x	y_calculated	y_exact solution	Error
0.00000000000000000000	**0.00000000000000000000**	**0.00000000000000000000**	**0.00000000000000000000E+00**
0.06250000000000000000	**0.06225680920912990000**	**0.06225680933852140000**	**1.29391504455345000000E-10**
0.12500000000000000000	**0.12307692246944200000**	**0.12307692307692300000**	**6.07481079328842000000E-10**
0.18750000000000000000	**0.18113207410275700000**	**0.18113207547169800000**	**1.36894109936669000000E-09**
0.25000000000000000000	**0.23529411533906500000**	**0.23529411764705900000**	**2.30799382383573000000E-09**
0.31250000000000000000	0.28469685475736000000	0.28469750889679700000	6.54139437172852000000E-07
0.37500000000000000000	0.32876594499773400000	0.32876712328767100000	1.17828993723146000000E-06
0.43750000000000000000	0.36721161302808700000	0.36721311475409800000	1.50172601132148000000E-06
0.50000000000000000000	0.39999839935601100000	0.40000000000000000000	1.60064398901927000000E-06
0.56250000000000000000	0.42729819196822300000	0.42729970326409500000	1.51129587194765000000E-06
0.62500000000000000000	0.44943690583998300000	0.44943820224719100000	1.29640720797353000000E-06
0.68750000000000000000	0.46684247928276000000	0.46684350132626000000	1.02204349994306000000E-06
0.75000000000000000000	0.47999925820049200000	0.48000000000000000000	7.41799507986141000000E-07
0.81250000000000000000	0.48941127391795700000	0.48941176470588200000	4.90787925355729000000E-07
0.87500000000000000000	0.49557493472326900000	0.49557522123893800000	2.86515669023490000000E-07
0.93750000000000000000	0.49896036576082800000	0.49896049896049900000	1.33199671015216000000E-07
1.00000000000000000000	0.49999997320325300000	0.50000000000000000000	2.67967469880226000000E-08
1.06250000000000000000	0.49908260956179300000	0.49908256880733900000	-4.07544535341309000000E-08
1.12500000000000000000	0.49655180272928400000	0.49655172413793100000	-7.85913529810500000000E-08
1.18750000000000000000	0.49270674027182300000	0.49270664505672600000	-9.52150969180110000000E-08
1.25000000000000000000	0.48780497575024000000	0.48780487804878000000	-9.77014595338055000000E-08
1.31250000000000000000	0.48206608863152900000	0.48206599713056000000	-9.15009694679725000000E-08
1.37500000000000000000	0.47567575623482700000	0.47567567567567600000	-8.05591512942350000000E-08
1.43750000000000000000	0.46878987649136200000	0.46878980891719700000	-6.75741645417283000000E-08
1.50000000000000000000	0.46153851582225100000	0.46153846153846200000	-5.42837894634829000000E-08
1.56250000000000000000	0.45402955364521000000	0.45402951191827500000	-4.17269353514271000000E-08
1.62500000000000000000	0.44635196178861500000	0.44635193133047200000	-3.04581428967055000000E-08
1.68750000000000000000	0.43857870091491100000	0.43857868020304600000	-2.07118653339222000000E-08
1.75000000000000000000	0.43076924329127700000	0.43076923076923100000	-1.25220462332543000000E-08
1.81250000000000000000	0.42297174691765600000	0.42297174111212400000	-5.80553199958089000000E-09
1.87500000000000000000	0.41522491391281700000	0.41522491349481000000	-4.18007350955207000000E-10
1.93750000000000000000	0.40755956890978500000	0.40755957271980300000	3.81001780125345000000E-09
2.00000000000000000000	0.39999992948837000000	0.40000000000000000000	7.05116304233755000000E-09

9.6 METHODS WITH VARIABLE STEP SIZES

One way of reducing the error, is changing step sizes by using error estimation methods. In order to achive this, two equations with different error level is required. Runge Kutta equations were defined in general form as

$x_i = x_n + c_i h$ i=1…s

$$y_i = y_n + h\sum_{j=1}^{i-1} a_{ij}k_j$$

$$k_i = f(x_i, y_i)$$

$y_{n+1} = y_n + h\sum_{j=1}^{s} b_i k_j$ or

$$y_{n+1} = y_n + h\sum_{j=1}^{s} bh_i k_j$$

Runge Kutta-Fehlberg is one of the best candidate for variable step size due to fact that two different degree equations can be given with the same coefficients. Runge Kutta Fehlberg coefficients were given in Table 8.4-6. If equations are written with coefficients it becomes :

$y_{i+1} = y_i + (\frac{37}{378}k_1 + \frac{250}{621}k_3 + \frac{125}{591}k_4 + \frac{512}{1771}k_6)h$ (4. order Runge-Kutta- Fehlberg)

$y_{i+1} = y_i + (\frac{2825}{27648}k_1 + \frac{18575}{48384}k_3 + \frac{13525}{55296}k_4 + \frac{277}{14336}k_5 + \frac{1}{4}k_6)h$ (5. order Runge-Kutta- Fehlberg)

$k_1=f(x_i,y_i)$
$k_2=f(x_i+0.2h , y_i+0.2k_1h)$
$k_3=f(x_i+0.3h , y_i+(3/40)k_1h+(9/40)k_2h)$
$k_4=f(x_i+(3/5)h , y_i + 0.3k_1h - (9/10)k_2h +(6/5)k_3h)$
$k_5=f(x_i+h , y_i - (11/54)k_1h+0.25k_2h-(70/27)k_3h+(35/27)k_4h)$
$k_6=f(x_i+(7/8)h , y_i + (1631/55296)k_1h+(175/512)k_2h+(575/13824)k_3h + (44275/110592)k_4h+(253/4096)k_5h)$
The error term

$$h_{new} = h_{previous}\left|\frac{\Delta_{new}}{\Delta_{previous}}\right|^{\alpha}$$

$$\Delta_{new} = \varepsilon y_{measure}$$

ε is a general tolerance level. $y_{measure}$ indicates how error will be measured. For example $y_{measure}$ = y can be choosen. In our program, $y_{measure}$ will be taken as $y_{measure} = |y| + \left|y\frac{dy}{dx}\right|$. The value of α can be taken as 0.2 if ($\Delta_{previous} \le \Delta_{new}$); as 0.25 if ($\Delta_{previous} \ge \Delta_{new}$).
As an example differential equation the following equation will be taken

$$\frac{dy}{dx} = -0.6y + 10e^{-(x-2)^2/[2(0.075)^2]}$$

Solution of this differential equation has a jump point around x=2 . Because of this, solving it with constant step size is relatively difficult.

Program 8.6-1 Runge-Kutta-Fehlberg variable step method with variable step size

```
//=====================================================
// Numerical Analysis package in java
// example to show differential equation solution
// Runge-Kutta-Fehlberg, Cash-Karp, Dormant-Price, Verner
//  variable step methods
// Dr. Turhan Coban
// =====================================================
// dy/dx=f(x,y)

import java.io.*;
import java.util.*;

abstract class f_xy
{
  //single function single independent variable
  // example f=x*x
  abstract double func(double x,double y);
}

class fxy2 extends f_xy
{
  double func(double x,double y)
  {return -0.6*y+10.0*Math.exp(-(x-2)*(x-2)/(2*0.075*0.075));}
}

class NA92H1
{

public static double[][] RKCK45(f_xy f,double x0,double y0,double xn,double h,double eps)
{
```

```
   double hmax=(xn-x0)/40;
   double hmin=hmax/100;
// Runge-Kutta Cash-Karp order 4-5
//eps bağıl hata
int s=7;
double dydx=1.0;
double c[]=new double[s];
double a[][]=new double[s][s];
double bh[]=new double[s];
double b[]=new double[s];
c[2] =  0.2;
c[3] =  0.3;
c[4] =  0.6;
c[5] =  1;
c[6] = 7.0/8.0;

a[1][1] = 0;

a[2][1] =  0.2;

a[3][1] =  3.0/40.0;
a[3][2] =  9.0/40.0;

a[4][1] =  0.3;
a[4][2] =  -9.0/10.0;
a[4][3] =  6.0/5.0;

a[5][1] =  -11.0/54.0;
a[5][2] =  0.25;
a[5][3] =  -70.0/27.0;
a[5][4] =  35.0/27.0;

b[1] =  37.0/378.0;
b[2] =  250.0/621.0;
b[3] =  125.0/591.0;
b[4] =  512.0/1771.0;
b[5] =  0.0;

bh[1] =  2825.0/27648.0;
bh[2] =  18575.0/48384.0;
bh[3] =  13525.0/55296.0;
bh[4] =  277.0/14336.0;
bh[5] =  1.0/4.0;

  double alfa;
  //double a[][]=new double[2][n+1];
  int i=0;
  double delta=h;
  double deltayeni=eps*y0;
  double k1,k2,k3,k4,k5,k6;
  Vector<Double> xi=new Vector<Double>();
  Vector<Double> yi=new Vector<Double>();
  double x,y,yp1_5,yp1_6,xeski,yeski;
  x=x0;
  y=y0;
  xeski=x;
  yeski=y;
  Double Xi=new Double(x);
  Double Yi=new Double(y);
  xi.add(Xi);
  yi.add(Yi);
  while((x+h)<=xn)
   {
   xeski=x;
```

```
  yeski=y;
  k1=f.func(x,y);
  k2=f.func(x+c[2]*h,y+a[2][1]*k1*h);
  k3=f.func(x+c[3]*h,y+a[3][1]*k1*h+a[3][2]*k2*h);
  k4=f.func(x+c[4]*h,y+a[4][1]*k1*h+a[4][2]*k2*h+a[4][3]*k3*h);
  k5=f.func(x+c[5]*h,y+a[5][1]*k1*h+a[5][2]*k2*h+a[5][3]*k3*h+a[5][4]*k4*h);
  k6=f.func(x+c[6]*h,y+a[6][1]*k1*h+a[6][2]*k2*h+a[6][3]*k3*h+a[6][4]*k4*h+a[6][5]*k5*h);
  yp1_5=y+(b[1]*k1+b[2]*k2+b[3]*k3+b[4]*k4+b[5]*k5+b[6]*k6)*h;
  yp1_6=y+(bh[1]*k1+bh[2]*k2+bh[3]*k3+bh[4]*k4+bh[5]*k5+bh[6]*k6)*h;
  y=yp1_6;
  x=x+h;
  Xi=new Double(x);
  Yi=new Double(y);
  xi.add(Xi);
  yi.add(Yi);
  delta=Math.abs(yp1_6-yp1_5);
  dydx=f.func(x,y);
  deltayeni=Math.abs(eps*h*dydx);
  if(delta<=deltayeni) alfa=0.2;
  else                 alfa=0.25;
  h=h*0.9437436*Math.pow((deltayeni/delta),alfa);
  if(h>hmax) h=hmax;
  if(h<hmin) h=hmin;
  i++;
  }
  if(x<xn)//son bir step daha
  {
  h=xn-x;
  k1=f.func(x,y);
  k2=f.func(x+c[2]*h,y+a[2][1]*k1*h);
  k3=f.func(x+c[3]*h,y+a[3][1]*k1*h+a[3][2]*k2*h);
  k4=f.func(x+c[4]*h,y+a[4][1]*k1*h+a[4][2]*k2*h+a[4][3]*k3*h);
  k5=f.func(x+c[5]*h,y+a[5][1]*k1*h+a[5][2]*k2*h+a[5][3]*k3*h+a[5][4]*k4*h);
  k6=f.func(x+c[6]*h,y+a[6][1]*k1*h+a[6][2]*k2*h+a[6][3]*k3*h+a[6][4]*k4*h+a[6][5]*k5*h);
  yp1_5=y+(b[1]*k1+b[2]*k2+b[3]*k3+b[4]*k4+b[5]*k5+b[6]*k6)*h;
  yp1_6=y+(bh[1]*k1+bh[2]*k2+bh[3]*k3+bh[4]*k4+bh[5]*k5+bh[6]*k6)*h;
  y=yp1_6;
  //System.out.println("x="+x+"h="+h+"y="+yp1_6);
  x=xn;
  Xi=new Double(x);
  Yi=new Double(y);
  xi.add(Xi);
  yi.add(Yi);
  i++;
  }
  //vektöre aktarılan değerleri double matrise aktar
  int n=i;
      double aa[][]=new double[2][n+1];
      Enumeration nx=xi.elements();
      Enumeration ny=yi.elements();
      for(i=0;i<=n;i++)
      {
                //String ss1=(String)nx.nextElement();
                //String ss2=(String)ny.nextElement();
                Double ax=(Double)nx.nextElement();
                Double ay=(Double)ny.nextElement();
      aa[0][i]=ax.doubleValue();
      aa[1][i]=ay.doubleValue();
      }
  return aa;
 }


public static double[][] RKDP45(f_xy f,double x0,double y0,double xn,double h,double eps)
```

```
{
  double hmax=(xn-x0)/40;
  double hmin=hmax/100;
// Runge-Kutta Dormant-Price order 4-5
// eps relative error
int s=8;
double dydx=1.0;
double c[]=new double[s];
double a[][]=new double[s][s];
double bh[]=new double[s];
double b[]=new double[s];
c[1]=0.0e0;
     c[2]=0.2;
     c[3]=0.3;
     c[4]=0.8;
     c[5]=8.0/9.0;
     c[6]=1;
     c[7]=1;

     a[2][1]=0.2;
     a[3][1]=3.0/40.0;
     a[3][2]=9.0/40.0;
     a[4][1]=44.0/45.0;
     a[4][2]=-56.0/15.0;
     a[4][3]=32.0/9.0;
     a[5][1]=19372.0/6561.0;
     a[5][2]=-25360.0/2187.0;
     a[5][3]=64448.0/6561.0;
     a[5][4]=-212.0/729.0;
     a[6][1]=9017.0/3168.0;
     a[6][2]=-355.0/33.0;
     a[6][3]=46732.0/5247.0;
     a[6][4]=49.0/176.0;
     a[6][5]=-5103.0/18656.0;
     a[7][1]=35.0/384.0;
     a[7][3]=500.0/1113.0;
     a[7][4]=125.0/192.0;
     a[7][5]=-2187.0/6784.0;
     a[7][6]=11.0/84.0;

     b[1]=5179.0/57600.0;
     b[2]=0;
     b[3]=-7571.0/16695.0;
     b[4]=393.0/640.0;
     b[5]=-92097.0/339200.0;
     b[6]=187.0/2100.0;
     b[7]=1.0/40.0;

     bh[1]=35.0/384.0;
     bh[2]=0;
     bh[3]=500.0/1113.0;
     bh[4]=125.0/192.0;
     bh[5]=-2187.0/6784.0;
     bh[6]=11.0/84.0;
     bh[7]=0.0;

  double alfa;
  //double a[][]=new double[2][n+1];
  int i=0;
  //a[0][0]=x0;
  //a[1][0]=y0;
  double delta=h;
  double deltayeni=eps*y0;
  double k1,k2,k3,k4,k5,k6,k7;
```

```
Vector<Double> xi=new Vector<Double>();
Vector<Double> yi=new Vector<Double>();
double x,y,yp1_5,yp1_6,xeski,yeski;
x=x0;
y=y0;
xeski=x;
yeski=y;
Double Xi=new Double(x);
Double Yi=new Double(y);
xi.add(Xi);
yi.add(Yi);
while((x+h)<=xn)
 {
 xeski=x;
 yeski=y;
 k1=f.func(x,y);
 k2=f.func(x+c[2]*h,y+a[2][1]*k1*h);
 k3=f.func(x+c[3]*h,y+a[3][1]*k1*h+a[3][2]*k2*h);
 k4=f.func(x+c[4]*h,y+a[4][1]*k1*h+a[4][2]*k2*h+a[4][3]*k3*h);
 k5=f.func(x+c[5]*h,y+a[5][1]*k1*h+a[5][2]*k2*h+a[5][3]*k3*h+a[5][4]*k4*h);
 k6=f.func(x+c[6]*h,y+a[6][1]*k1*h+a[6][2]*k2*h+a[6][3]*k3*h+a[6][4]*k4*h+a[6][5]*k5*h);
 k7=f.func(x+c[7]*h,y+a[7][1]*k1*h+a[7][2]*k2*h+a[7][3]*k3*h+a[7][4]*k4*h+a[7][5]*k5*h+a[7][6]*k6*h);
 yp1_5=y+(b[1]*k1+b[2]*k2+b[3]*k3+b[4]*k4+b[5]*k5+b[6]*k6+b[7]*k7)*h;
 yp1_6=y+(bh[1]*k1+bh[2]*k2+bh[3]*k3+bh[4]*k4+bh[5]*k5+bh[6]*k6+bh[7]*k7)*h;
 y=yp1_6;
 x=x+h;
 Xi=new Double(x);
 Yi=new Double(y);
 xi.add(Xi);
 yi.add(Yi);
 delta=Math.abs(yp1_6-yp1_5);
 dydx=f.func(x,y);
 deltayeni=Math.abs(eps*h*dydx);
 if(delta<=deltayeni) alfa=0.2;
 else                 alfa=0.25;
 h=h*0.9437436*Math.pow((deltayeni/delta),alfa);
 if(h>hmax) h=hmax;
 if(h<hmin) h=hmin;
 i++;
 }
 if(x<xn)//son bir step daha
 {
 h=xn-x;
 xeski=x;
 yeski=y;
 k1=f.func(x,y);
 k2=f.func(x+c[2]*h,y+a[2][1]*k1*h);
 k3=f.func(x+c[3]*h,y+a[3][1]*k1*h+a[3][2]*k2*h);
 k4=f.func(x+c[4]*h,y+a[4][1]*k1*h+a[4][2]*k2*h+a[4][3]*k3*h);
 k5=f.func(x+c[5]*h,y+a[5][1]*k1*h+a[5][2]*k2*h+a[5][3]*k3*h+a[5][4]*k4*h);
 k6=f.func(x+c[6]*h,y+a[6][1]*k1*h+a[6][2]*k2*h+a[6][3]*k3*h+a[6][4]*k4*h+a[6][5]*k5*h);
 k7=f.func(x+c[7]*h,y+a[7][1]*k1*h+a[7][2]*k2*h+a[7][3]*k3*h+a[7][4]*k4*h+a[7][5]*k5*h+a[7][6]*k6*h);
 yp1_5=y+(b[1]*k1+b[2]*k2+b[3]*k3+b[4]*k4+b[5]*k5+b[6]*k6+b[7]*k7)*h;
 yp1_6=y+(bh[1]*k1+bh[2]*k2+bh[3]*k3+bh[4]*k4+bh[5]*k5+bh[6]*k6+bh[7]*k7)*h;
 y=yp1_6;
 //System.out.println("x="+x+"h="+h+"y="+yp1_6);
 x=xn;
 Xi=new Double(x);
 Yi=new Double(y);
 xi.add(Xi);
 yi.add(Yi);
 i++;
 }
 //vektöre aktarılan değerleri double matrise aktar
```

```
 int n=i;
     double aa[][]=new double[2][n+1];
     Enumeration nx=xi.elements();
     Enumeration ny=yi.elements();
     for(i=0;i<=n;i++)
     {
               //String ss1=(String)nx.nextElement();
               //String ss2=(String)ny.nextElement();
               Double ax=(Double)nx.nextElement();
               Double ay=(Double)ny.nextElement();
     aa[0][i]=ax.doubleValue();
     aa[1][i]=ay.doubleValue();
     }
 return aa;
}

public static double[][] RKF45(f_xy f,double x0,double y0,double xn,double h,double eps)
{
 //  Runge-Kutta Fehlberg order 4-5
 //  eps relative error
 double hmax=(xn-x0)/40;
 double hmin=hmax/1000;
double sh=1;
double dydx=1.0;
double alfa;
int i=0;
double delta=h;
double deltayeni=eps*y0;
double k1,k2,k3,k4,k5,k6;
Vector<Double> xi=new Vector<Double>();
Vector<Double> yi=new Vector<Double>();
double x,y,yp1_5,yp1_6,xeski,yeski;
x=x0;
y=y0;
xeski=x;
yeski=y;
Double Xi=new Double(x);
Double Yi=new Double(y);
xi.add(Xi);
yi.add(Yi);
while((x+h)<=xn)
 {
 xeski=x;
 yeski=y;
 k1=f.func(x,y);
 k2=f.func(x+1.0/4.0*h,y+1.0/4.0*k1*h);
 k3=f.func(x+3.0/8.0*h,y+3.0/32.0*k1*h+9.0/32.0*k2*h);
 k4=f.func(x+12/13*h,y+1932.0/2197.0*k1*h-7200.0/2197.0*k2*h+7296/2197*k3*h);
 k5=f.func(x+h,y+439.0/216.0*k1*h-8.0*k2*h+3680.0/513.0*k3*h-845.0/4104.0*k4*h);
 k6=f.func(x+1.0/2.0*h,y-8.0/27.0*k1*h+2.0*k2*h+-3544.0/2565.0*k3*h+1859.0/4104.0*k4*h-11.0/40.0*k5*h);

 yp1_5=y+(25.0/216.0*k1+1408.0/2565.0*k3+2197.0/4104.0*k4-1.0/5.0*k6)*h;
 yp1_6=y+(16.0/135.0*k1+6656.0/12825.0*k3+28561.0/56430.0*k4-9.0/50.0*k5+2.0/55.0*k6)*h;
 y=yp1_6;
 x=x+h;
 Xi=new Double(x);
 Yi=new Double(y);
 xi.add(Xi);
 yi.add(Yi);
 delta=Math.abs(yp1_6-yp1_5);
 dydx=f.func(x,y);
 deltayeni=Math.abs(eps*h*dydx);
 if(delta<=deltayeni) alfa=0.25;
 else                 alfa=1.0/6.0;
```

```
  h=h*0.98*Math.pow((deltayeni/delta),alfa);
  if(h>hmax) h=hmax;
  if(h<hmin) h=hmin;
  i++;
  }
  if(x<xn)//son bir step daha
  {
  h=xn-x;
  k1=f.func(x,y);
  k2=f.func(x+1.0/4.0*h,y+1.0/4.0*k1*h);
  k3=f.func(x+3.0/8.0*h,y+3.0/32.0*k1*h+9.0/32.0*k2*h);
  k4=f.func(x+12/13*h,y+1932.0/2197.0*k1*h-7200.0/2197.0*k2*h+7296/2197*k3*h);
  k5=f.func(x+h,y+439.0/216.0*k1*h-8.0*k2*h+3680.0/513.0*k3*h-845.0/4104.0*k4*h);
  k6=f.func(x+1.0/2.0*h,y-8.0/27.0*k1*h+2.0*k2*h+-3544.0/2565.0*k3*h+1859.0/4104.0*k4*h-11.0/40.0*k5*h);

  yp1_5=y+(25.0/216.0*k1+1408.0/2565.0*k3+2197.0/4104.0*k4-1.0/5.0*k6)*h;
  yp1_6=y+(16.0/135.0*k1+6656.0/12825.0*k3+28561.0/56430.0*k4-9.0/50.0*k5+2.0/55.0*k6)*h;
  y=yp1_6;
  //System.out.println("x="+x+"h="+h+"y="+yp1_6);
  x=xn;
  Xi=new Double(x);
  Yi=new Double(y);
  xi.add(Xi);
  yi.add(Yi);
  i++;
  }
  //vektöre aktarılan değerleri double matrise aktar
  int n=i;
      double aa[][]=new double[2][n+1];
      Enumeration nx=xi.elements();
      Enumeration ny=yi.elements();
      for(i=0;i<=n;i++)
      {
                //String ss1=(String)nx.nextElement();
                //String ss2=(String)ny.nextElement();
                Double ax=(Double)nx.nextElement();
                Double ay=(Double)ny.nextElement();
      aa[0][i]=ax.doubleValue();
      aa[1][i]=ay.doubleValue();
      }
  return aa;
 }

public static double[][] RKV16(f_xy f,double x0,double y0,double xn,double h,double eps)
{  //Runge-Kutta-Verner 16th degree step constant step size version
double hmax=(xn-x0)/40;
double hmin=hmax/100;
double dydx=1.0;
double delta,deltayeni,alfa;
int s=17;
double c[]=new double[s];
double a[][]=new double[s][s];
double bh[]=new double[s];
double b[]=new double[s];
c[0]=0;
c[1] =  0.;
c[2] =  .6e-1;
c[3] =  .95933333333333333333333333333333333333333e-1;
c[4] =  .1439;
c[5] =  .4973;
c[6] =  .9725;
c[7] =  .9995;
c[8] =  1.;
c[9] =  1.;
```

```
a[2][1] =  .6e-1;
a[3][1] =  .1923996296296296296296296296296296e-1;
a[3][2] =  .7669337037037037037037037037037037037e-1;
a[4][1] =  .35975e-1;
a[4][2] =  0.;
a[4][3] =  .107925;
a[5][1] =  1.3186834152331482609197472764317356128 61;
a[5][2] =  0.;
a[5][3] = -5.042058063628562225427761634715637693344;
a[5][4] =  4.220674648395413964508014358283902080483;
a[6][1] = -41.872591664327514618037577806443468129 05;
a[6][2] =  0.;
a[6][3] =  159.432562163137491770036566907034683045 3;
a[6][4] = -122.119213565010030920251620338924214066 3;
a[6][5] =  5.531743066200053768252631238332999150076;
a[7][1] = -54.430156935316504332506420512941424612 71;
a[7][2] =  0.;
a[7][3] =  207.067251365018464427365717386650983598 7;
a[7][4] = -158.610813784589999182874242436505859946 9;
a[7][5] =  6.991816585950242321992597280791793907096;
a[7][6] = -.185972310622032339776517179954929462369 2e-1;

a[8][1] = -54.663741787281976802412156480503869593 51;
a[8][2] =  0.;
a[8][3] =  207.952806255389373451582481669983424423 8;
a[8][4] = -159.288957474499507150895980587142665421 6;
a[8][5] =  7.018743740796944434698170760964252490817;
a[8][6] = -.183387859050457230647278200514173826836 1e-1;
a[8][7] = -.511948499788209907787543249724516839584 0e-3;

a[9][1] =  .343895786835703600927882012472832238652 0e-1;
a[9][2] =  0.;
a[9][3] =  0.;
a[9][4] =  .258262455563350340465955809858612085876 7;
a[9][5] =  .420937118967353715064255151406980196703 2;
a[9][6] =  4.405396469669310170148836816197095664891;
a[9][7] = -176.483119024298657615174094249900212502 9;
a[9][8] =  172.364133401415073029402258271190241331 5;

b[1] =  .343895786835703600927882012472832238652 0e-1;
b[2] =  0.;
b[3] =  0.;
b[4] =  .258262455563350340465955809858612085876 7;
b[5] =  .420937118967353715064255151406980196703 2;
b[6] =  4.405396469669310170148836816197095664891;
b[7] = -176.483119024298657615174094249900212502 9;
b[8] =  172.364133401415073029402258271190241331 5;
b[9] =  0.;

bh[ 1] =  .490996764838248973090685492797122583647 9e-1;
bh[ 2] =  0.;
bh[ 3] =  0.;
bh[ 4] =  .225111222951652415340139532053987532948 5;
bh[ 5] =  .469468225302956203943194852504738741255 3;
bh[ 6] =  .806579224998886770763416180899521798144 3;
bh[ 7] =  0.;
bh[ 8] = -.607119489177795979767295146525621712248 8;
bh[ 9] =  .568611394404756924114760317876613815359 4e-1;
  // Runge-Kutta Verner 16 variable step
  //eps relative error
  //double a[][]=new double[2][n+1];
  int i=0;
  double x1,y1;
  Vector<Double> xi=new Vector<Double>();
```

```
Vector<Double> yi=new Vector<Double>();
double x,y,yp1=0,yp2=0,xeski,yeski;

x=x0;
y=y0;
xeski=x;
yeski=y;
Double Xi=new Double(x);
Double Yi=new Double(y);
double k[]=new double[s];
xi.add(Xi);
yi.add(Yi);
int n=1;
while((x+h)<=xn)
{
 xeski=x;
 yeski=y;
 double total=0;
 //yp1=y;
 yp2=y;
 double w=1;
 for(int ii=1;ii<s;ii++)
 {  x1=x+c[ii]*h;
   y1=y;
              for(int j=1;j<ii;j++)
   {y1+=h*a[ii][j]*k[j];}
    k[ii]=f.func(x1,y1);
    yp2+=h*bh[ii]*k[ii];
    yp1+=h*b[ii]*k[ii];
 }
 delta=Math.abs(yp2-yp1);
 dydx=f.func(x,y);
 deltayeni=Math.abs(eps*h*dydx);
 if(delta<=deltayeni) alfa=0.2;
 else               alfa=0.25;
 h=h*0.9437436*Math.pow((deltayeni/delta),alfa);
 if(h>hmax) h=hmax;
 if(h<hmin) h=hmin;
 y=yp2;
 x+=h;
 Xi=new Double(x);
 Yi=new Double(y);
 xi.add(Xi);
 yi.add(Yi);
 n++;
 } //end of while
 if(x<xn)//son bir step daha
 {
 h=xn-x;
 xeski=x;
 yeski=y;
 double total=0;
 yp1=y;
 yp2=y;
 for(int ii=1;ii<s;ii++)
 {  x1=x+c[ii]*h;
   y1=y;
              for(int j=1;j<ii;j++)
   {y1+=h*a[i][j]*k[j];}
   k[ii]=f.func(x1,y1);
   yp1+=h*b[ii]*k[ii];
   yp2+=h*bh[ii]*k[ii];
 }
```

```
   y=yp2;
   x+=h;
   Xi=new Double(x);
   Yi=new Double(y);
   xi.add(Xi);
   yi.add(Yi);
   }
      // vektöre aktarilan değerleri double matrise aktar
      double aa[][]=new double[2][n+1];
      Enumeration nx=xi.elements();
      Enumeration ny=yi.elements();
      for(i=0;i<=n;i++)
      {
               Double ax=(Double)nx.nextElement();
               Double ay=(Double)ny.nextElement();
      aa[0][i]=ax.doubleValue();
      aa[1][i]=ay.doubleValue();
      }
   return aa;
}

public static double[][] RKV13(f_xy f,double x0,double y0,double xn,double h,double eps)
{  //Runge-Kutta-Verner 13th degree step constant step size version
   double hmax=(xn-x0)/40;
   double hmin=hmax/1000;
   double dydx=1.0;
   double delta,deltayeni,alfa;
int s=14;
double c[]=new double[s];
double a[][]=new double[s][s];
double bh[]=new double[s];
double b[]=new double[s];
c[0]=0;
c[1] =  0.;
c[2] =  .5e-1;
a[2][1] =  .5e-1;
c[3] =  .1065625;
a[3][1] = -.69931640625e-2;
a[3][2] =  .1135556640625;
c[4] =  .15984375;
a[4][1] =  .399609375e-1;
a[4][2] =  0.;
a[4][3] =  .1198828125;
c[5] =  .39;
a[5][1] =  .36139756280045751240529407211840283451 29;
a[5][2] =  0.;
a[5][3] = -1.34152406670049277181998778820271583491 7;
a[5][4] =  1.37012650390003525941469371608431300040 4;
c[6] =  .465;
a[6][1] =  .490472027972027972027972027972027972028 0e-1;
a[6][2] =  0.;
a[6][3] =  0.;
a[6][4] =  .23509720422144047398629883354934271431 22;
a[6][5] =  .18085559298135672881090396365345448848 50;
//
c[7] =  .155;
a[7][1] =  .61692890442890442890442890442890442890 44e-1;
a[7][2] =  0.;
a[7][3] =  0.;
a[7][4] =  .11236568314640276622625570351300154423 03;
a[7][5] = -.38850460714513667670490481081112445674 56e-1;
a[7][6] =  .19791887125220458553791887125220458553 79e-1;
c[8] =  .943;
a[8][1] = -1.76763024022232687573559711957214558671 4;
```

```
a[8][2] =  0.;
a[8][3] =  0.;
a[8][4] = -62.5;
a[8][5] = -6.06188937737666910082136145965933199975 8;
a[8][6] =  5.65082319822276313856129803060084017420 1;
a[8][7] =  65.62169641937623283799566054863063741227;
c[9] =  .90180204173585695825970794067837214995 60;
a[9][1] = -1.18094506655497079982511628262828297957882;
a[9][2] =  0.;
a[9][3] =  0.;
a[9][4] = -41.50473441114320841606641502701994225874;
a[9][5] = -4.43443831910372501122516922984610021177 6;
a[9][6] =  4.26040818858613302481219371074469324076 1;
a[9][7] =  43.75364022446171584987676829438379303004;
a[9][8] =  .78714254899123106874464750442263075508 60e-2;
c[10] =  .909;
a[10][1] = -1.28140599944148840545951029118205424626 6;
a[10][2] =  0.;
a[10][3] =  0.;
a[10][4] = -45.04713996013986630220754257136007322267;
a[10][5] = -4.73136206944957647731146426549128281094 3;
a[10][6] =  4.51496701659380784118585158459724099621 4;
a[10][7] =  47.44909557172985134869022392235929015114;
a[10][8] =  .10592282971116611356873939555165428752 28e-1;
a[10][9] = -.57468422638446162544323184782862962320 21e-2;
c[11] =  .94;
a[11][1] = -1.72447013426248519175670981748448186173 1;
a[11][2] =  0.;
a[11][3] =  0.;
a[11][4] = -60.92349008483054016518434619253765246063;
a[11][5] = -5.95151837622239245520283276706185486829 0;
a[11][6] =  5.55652373069845623597979165084359249683 9;
a[11][7] =  63.98301198033305336837536378635995939281;
a[11][8] =  .14642028250414961592759213917594526760 03e-1;
a[11][9] =  .64604087723582036036218651449776507148 92e-1;
a[11][10] = -.79303231690088789840244525486933732914 47e-1;
c[12] =  1.;
a[12][1] = -3.30162266774707901635399478979098362556 9;
a[12][2] =  0.;
a[12][3] =  0.;
a[12][4] = -118.01127235975250856669233039578988685 10;
a[12][5] = -10.14142238845611248642783916034510897595;
a[12][6] =  9.13931133223205792354401227355682700061 9;
a[12][7] =  123.37594282840426836848471809865018943 64;
a[12][8] =  4.62324437887458047483980762506763092479 2;
a[12][9] = -3.38327773806820192365255097153681124081 4;
a[12][10] =  4.52759210032461818945126533935112903532 5;
a[12][11] = -5.82849548581162296319308801916298570375 5;
c[13] =  1.;
a[13][1] = -3.03951503376630903004010285182120025105 6;
a[13][2] =  0.;
a[13][3] =  0.;
a[13][4] = -109.26086808941762546864441923221646233 52;
a[13][5] = -9.29064249740029344971766554265689754915 8;
a[13][6] =  8.43050498176491114213429925383616780345 4;
a[13][7] =  114.20100103783313135574240410955234274 76;
a[13][8] = -.96372713421454793581623756589879016527 62;
a[13][9] = -5.03488408880218979119868033618333232311 8;
a[13][10] =  5.95813082400292317754040216538817207279 4;
a[13][11] =  0.;
a[13][12] =  0.;

bh[1] =  .44312615229089792124864365102090297648 93e-1;
bh[2] =  0.;
```

```
bh[3] =  0.;
bh[4] =  0.;
bh[5] =  0.;
bh[6] =  .3546095642343226447863179350895055038855;
bh[7] =  .2478480431366653069619986721504458660016;
bh[8] =  4.448134732475784492725128317159648871312;
bh[9] =  19.846886366118733699309323992976879 35291;
bh[10] = -23.581623377465618419695179608703949 65085;
bh[11] =  0.;
bh[12] =  0.;
bh[13] = -.360167943728977516212453673774620240 9110;

 // Runge-Kutta Verner 16 variable step
 //eps relative error
 //double a[][]=new double[2][n+1];
 int i=0;
 double x1,y1;
 Vector<Double> xi=new Vector<Double>();
 Vector<Double> yi=new Vector<Double>();
 double x,y,yp1,yp2,xeski,yeski;
 x=x0;
 y=y0;
 xeski=x;
 yeski=y;
 Double Xi=new Double(x);
 Double Yi=new Double(y);
 double k[]=new double[s];
 xi.add(Xi);
 yi.add(Yi);
 int n=1;
 while((x+h)<=xn)
 {
  xeski=x;
  yeski=y;
  double total=0;
  yp1=y;
  yp2=y;
  double w=1;
  for(int ii=1;ii<s;ii++)
  {  x1=x+c[ii]*h;
    y1=y;
              for(int j=1;j<ii;j++)
    {y1+=h*a[ii][j]*k[j];}
     k[ii]=f.func(x1,y1);
     yp1+=h*b[ii]*k[ii];
     yp2+=h*bh[ii]*k[ii];
  }
  delta=Math.abs(yp2-yp1);
  dydx=f.func(x,y);
  deltayeni=Math.abs(eps*h*dydx);
  if(delta<=deltayeni) alfa=0.2;
  else               alfa=0.25;
  h=h*0.9437436*Math.pow((deltayeni/delta),alfa);
  if(h>hmax) h=hmax;
  if(h<hmin) h=hmin;

  y=yp2;
  x+=h;
  Xi=new Double(x);
  Yi=new Double(y);
  xi.add(Xi);
  yi.add(Yi);
  n++;
 } //end of while
```

```
   if(x<xn)//son bir step daha
   {
   h=xn-x;
   xeski=x;
   yeski=y;
   double total=0;
   yp1=y;
   yp2=y;
   for(int ii=1;ii<s;ii++)
   {  x1=x+c[ii]*h;
     y1=y;
                for(int j=1;j<ii;j++)
     {y1+=h*a[i][j]*k[j];}
     k[ii]=f.func(x1,y1);
     yp1+=h*b[ii]*k[ii];
     yp2+=h*bh[ii]*k[ii];
   }
   y=yp2;
   x+=h;
   Xi=new Double(x);
   Yi=new Double(y);
   xi.add(Xi);
   yi.add(Yi);
   }
   //vektöre aktarilan değerleri double matrise aktar
       double aa[][]=new double[2][n+1];
       Enumeration nx=xi.elements();
       Enumeration ny=yi.elements();
       for(i=0;i<=n;i++)
       {
                  Double ax=(Double)nx.nextElement();
                  Double ay=(Double)ny.nextElement();
       aa[0][i]=ax.doubleValue();
       aa[1][i]=ay.doubleValue();
       }
   return aa;
}

public static double[][] RKV9(f_xy f,double x0,double y0,double xn,double h,double eps)
{  //Runge-Kutta-Verner 9th degree step constant step size version
   double hmax=(xn-x0)/40.0;
   double hmin=hmax/100.0;int s=10;
   double dydx=1.0;
double c[]=new double[s];
double a[][]=new double[s][s];
double bh[]=new double[s];
double b[]=new double[s];
c[0]=0;
c[1] =  0.;
c[2] =  .6e-1;
c[3] =  .95933333333333333333333333333333333333333333e-1;
c[4] =  .1439;
c[5] =  .4973;
c[6] =  .9725;
c[7] =  .9995;
c[8] =  1.;
c[9] =  1.;
a[2][1] =  .6e-1;
a[3][1] =  .19239962962962962962962962962962962962962962e-1;
a[3][2] =  .76693370370370370370370370370370370370370370e-1;
a[4][1] =  .35975e-1;
a[4][2] =  0.;
a[4][3] =  .107925;
a[5][1] =  1.31868341523314826091974727643173561286 1;
```

```
a[5][2] =  0.;
a[5][3] = -5.042058063628562225427761634715637693344;
a[5][4] =  4.220674648395413964508014358283902080483;
a[6][1] = -41.872591664327514618037577806443468129 05;
a[6][2] =  0.;
a[6][3] =  159.432562163137491770036566907034683045 3;
a[6][4] = -122.119213565010030920251620338924214066 3;
a[6][5] =  5.531743066200053768252631238332999150076;
a[7][1] = -54.430156935316504332506420512941424612 71;
a[7][2] =  0.;
a[7][3] =  207.067251365018464427365717386650983598 7;
a[7][4] = -158.610813784589999182874242436505859946 9;
a[7][5] =  6.991816585950242321992597280791793907096;
a[7][6] = -.185972310622032339776517179954929462369 2e-1;

a[8][1] = -54.663741787281976802412156480503869593 51;
a[8][2] =  0.;
a[8][3] =  207.952806255389373451582481669983424423 8;
a[8][4] = -159.288957474499507150895980587142665421 6;
a[8][5] =  7.018743740796944434698170760964252490817;
a[8][6] = -.183387859050457230647278200514173826836 1e-1;
a[8][7] = -.511948499788209907787543249724516839584 0e-3;

a[9][1] =  .343895786835703600927882012472832238652 0e-1;
a[9][2] =  0.;
a[9][3] =  0.;
a[9][4] =  .258262455563350340465955809858612085876 7;
a[9][5] =  .420937118967353715064255151406980196703 2;
a[9][6] =  4.405396469669310170148836816197095664891;
a[9][7] = -176.483119024298657615174094249900212502 9;
a[9][8] =  172.364133401415073029402258271190241331 5;

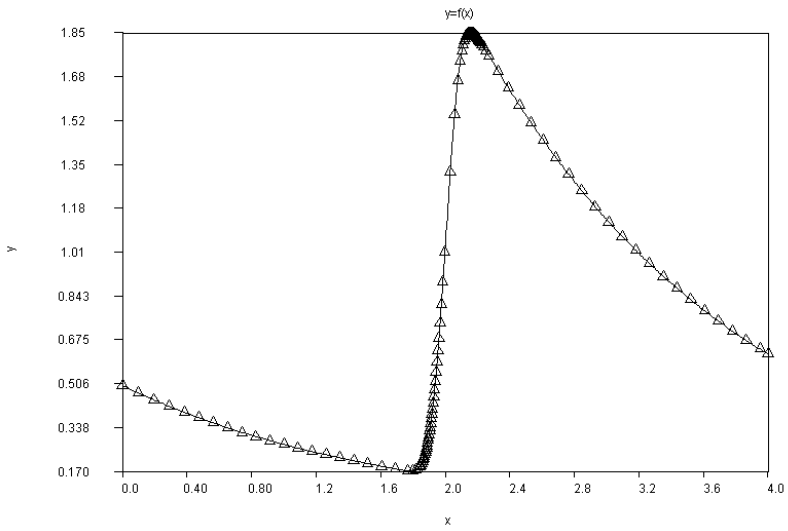
b[1] =  .343895786835703600927882012472832238652 0e-1;
b[2] =  0.;
b[3] =  0.;
b[4] =  .258262455563350340465955809858612085876 7;
b[5] =  .420937118967353715064255151406980196703 2;
b[6] =  4.405396469669310170148836816197095664891;
b[7] = -176.483119024298657615174094249900212502 9;
b[8] =  172.364133401415073029402258271190241331 5;
b[9] =  0.;

bh[ 1] =  .490996764838248973090685492797122583647 9e-1;
bh[ 2] =  0.;
bh[ 3] =  0.;
bh[ 4] =  .225111222951652415340139532053987532948 5;
bh[ 5] =  .469468225302956203943194852504738741255 3;
bh[ 6] =  .806579224998886770763416180899521798144 3;
bh[ 7] =  0.;
bh[ 8] = -.607119489177795979767295146525621712248 8;
bh[ 9] =  .568611394404756924114760317876613815359 4e-1;
  // Runge-Kutta Verner 16 variable step
  //eps relative error
  double alfa;
  //double a[][]=new double[2][n+1];
  int i=0;
  double x1,y1;
  Vector<Double> xi=new Vector<Double>();
  Vector<Double> yi=new Vector<Double>();
  double x,y,yp1,yp2,xeski,yeski;
  x=x0;
  y=y0;
  xeski=x;
  yeski=y;
```

```
double delta,deltayeni;
Double Xi=new Double(x);
Double Yi=new Double(y);
double k[]=new double[s];
xi.add(Xi);
yi.add(Yi);
int n=1;
while((x+h)<=xn)
{
 xeski=x;
 yeski=y;
 double total=0;
 yp1=y;
 yp2=y;
 double w=1;
 for(int ii=1;ii<s;ii++)
 {  x1=x+c[ii]*h;
   y1=y;
              for(int j=1;j<ii;j++)
   {y1+=h*a[ii][j]*k[j];}
    k[ii]=f.func(x1,y1);
    yp2+=h*bh[ii]*k[ii];
 }
 delta=Math.abs(yp2-yp1);
 dydx=f.func(x,y);
 deltayeni=Math.abs(eps*h*dydx);
 if(delta<=deltayeni) alfa=0.2;
 else               alfa=0.25;
 h=h*0.9437436*Math.pow((deltayeni/delta),alfa);
 if(h>hmax) h=hmax;
 if(h<hmin) h=hmin;
 y=yp2;
 x+=h;
 Xi=new Double(x);
 Yi=new Double(y);
 xi.add(Xi);
 yi.add(Yi);
 n++;
 } //end of while
 if(x<xn)//son bir step daha
 {
 h=xn-x;
 xeski=x;
 yeski=y;
 double total=0;
 yp1=y;
 yp2=y;
 for(int ii=1;ii<s;ii++)
 {  x1=x+c[ii]*h;
   y1=y;
              for(int j=1;j<ii;j++)
   {y1+=h*a[i][j]*k[j];}
   k[ii]=f.func(x1,y1);
   yp1+=h*b[ii]*k[ii];
   yp2+=h*bh[ii]*k[ii];
 }
 y=yp2;
 x+=h;
 Xi=new Double(x);
 Yi=new Double(y);
 xi.add(Xi);
 yi.add(Yi);
 }
 //vektöre aktarilan değerleri double matrise aktar
```

```
      double aa[][]=new double[2][n+1];
      Enumeration nx=xi.elements();
      Enumeration ny=yi.elements();
      for(i=0;i<=n;i++)
      {
            Double ax=(Double)nx.nextElement();
            Double ay=(Double)ny.nextElement();
      aa[0][i]=ax.doubleValue();
      aa[1][i]=ay.doubleValue();
      }
  return aa;
}
 public static void main(String args[]) throws IOException
 {
  double x0=0;
  double y0=0.5;
  fxy2 f1=new fxy2();
  double a[][]=RKF45(f1,x0,y0,4.0,0.1,1e-3);
   Text.print(Text.T(a));
   Plot p1=new Plot(a);
   //p1.setPlotType(0,24);
   //p1.addSpline(a,8);
   //p1.addData(a);
   p1.plot();
 }
}
```

Output 8.6-1 Runge-Kutta Fehlberg variable step methods with variable step size



DIFFERENTIAL EQUATIONS WITH MORE THEN ONE SET OF EQUATION

Consider the solution of differential equation set:

$$\frac{dy_1}{dx} = f_1(x, y_1, y_2, \ldots, y_n)$$

$$\frac{dy_2}{dx} = f_2(x, y_1, y_2, \ldots, y_n)$$

….

$$\frac{dy_n}{dx} = f_n(x, y_1, y_2, \ldots, y_n)$$

This set requires n initial conditions to solve it. For example consider differential equation set

$$\frac{dy_1}{dx} = -0.5y_1$$

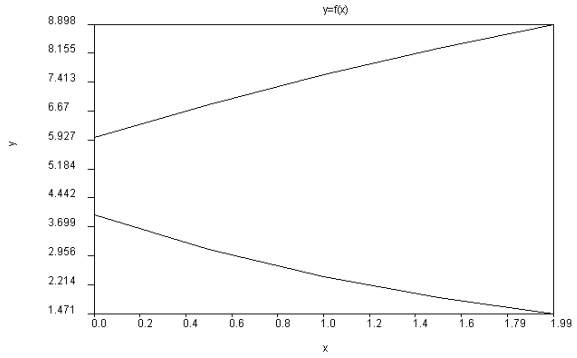
$$\frac{dy_2}{dx} = 4 - 0.3y_2 - 0.1y_1$$

If for x=0, y_1=4 and y_2=6 is given. If this set is solved with 6th order Runge_Kutta method:

Program 8.7-1 2 variable differential equation set with Runge-Kutta6 method

```
 import java.io.*;
import java.text.*;
import java.util.Locale;
class fm1 extends f_xi
{
//multivariable function
 double func(double x[],int x_ref)
 {
   //x[0] is x, x[1] is y
   if(x_ref==0) return -0.5*x[1];
   if(x_ref==1) return 4-0.3*x[2]-0.1*x[1];
   else return 0;
 }}
public class NA93A
{
public static void main(String arg[])
{ Buthcher bb=new Buthcher("RK6");
  Runge_Kutta rk=new Runge_Kutta(bb);
  fm1 f=new fm1();
  double y[]=new double[2];
  y[0]=4.0;
  y[1]=6.0;
  double a[][]=rk.hesapla(f,0.0,2.0,y,0.01);
  Text.printT(a);
  Plot p=new Plot(a);
  p.plot();
}
}
```

Output 8.7-1 Solution of 2 equation set of differential equation by using RK6



Output 8.7-2 Output 8.7-1 Solution of 2 equation set of differential equation by using RK6

0.0	4.0	6.0
0.1	3.804917698014323	6.177326798708166
0.2	3.6193496721656535	6.351334649609415
0.30000000000000004	3.4428319057313583	6.522027912650391
0.4	3.274923012351407	6.689415390177402
0.5	3.1152031323325615	6.853509972700889
0.6000000000000001	2.9632728827804544	7.014328306002212
0.7000000000000001	2.818752358934318	7.171890478421663
0.8	2.681280184207202	7.326219727226854

0.9	2.5505126065562718	7.477342163017759
1.0	2.42612263892365	7.625286511178904
1.1	2.307799241598452	7.770083869440747
1.2000000000000002	2.195246544455496	7.911767480661146
1.3	2.0881831071258756	8.050372519984265
1.4000000000000001	1.9863412152494457	8.185935895578297
1.5	1.8894662110494735	8.318496062195234
1.6	1.7973158565555518	8.448092846835637
1.7000000000000002	1.709659727882498	8.574767285838973
1.8	1.626278639050617	8.698561472755916
1.9000000000000001	1.546964093906585	8.819518416392935
2.0	1.471517764774464	8.937681908451696

Set of differential equation solution concept can be used to solve nth degree differential equations as well.

$$\frac{d^n y}{dx^n} = g(x, y, \frac{dy}{dx}, \frac{d^2 y}{dx^2}, \ldots, \frac{d^{n-1} y}{dx^{n-1}})$$

This equation can be cange to take form of

$x=x_0$

$y=x_1$

$$\frac{dy}{dx} = x_2$$

$$\frac{d^2 y}{dx^2} = x_3$$

..

$$\frac{d^{n-1} y}{dx^{n-1}} = x_{n-1}$$

Our high order differential equation is converted to a set of single order differential equations

$$\frac{dx_k}{dx_0} = \frac{d^k y}{dx_0^k} = x_{k+1}, 1 \leq k < n$$

$$\frac{dx_n}{dx_0} = \frac{d^n y}{dx_0^n} = g(x_0, x_1, x_2, \ldots, x_n)$$

If the initial conditions are given, this type of set of equations can be solved easily. For example consider the diferential equation

$$\frac{d^2 y}{dt^2} + 11\frac{dy}{dt} + 10y = 10u(t)$$

With u(t)=1,and the initial conditions at t=0 y(t)=0 and dy(t)/dt=0 . Exact solution of this differential equation is : y(t)=1-(10/9)exp(-t)+(1/9)exp(-10t) . To solve the same equation numerically, it can be divided into two equations as

$$\frac{dx_1}{dx_0} = x_2$$

$$\frac{dx_2}{dx_0} = 10u - 10x_1 - 11x_2$$

$x_1(0)=0$

$x_2(0)=0$

This form can be easily solved by using the previous RK6 program(NA93)

Program 8.7-2 Solution of set of differential equation with Runge Kutta

```
//=======================================================
// Numerical Analysis package in java
```

```
// example to show differential equation solution
// Runge-Kutta 6 method with multible differential equations
// Dr. Turhan Coban
// =====================================================
// dy1/dx=f(x,y1,y2,..,yn)
// dy2/dx=f(x,y1,y2,..,yn)
// ......
// dyn/dx=f(x,y1,y2,..,yn)
import java.io.*;

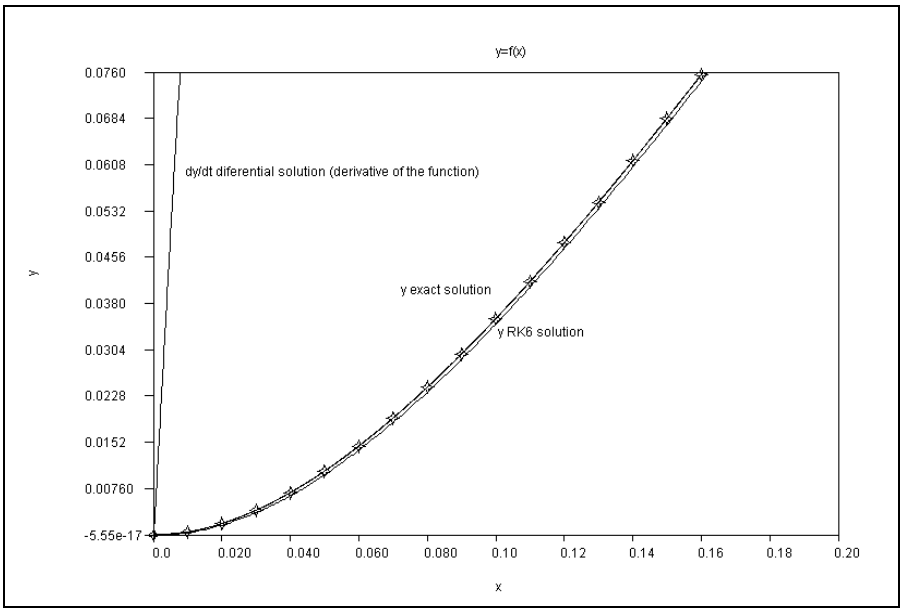

class fm1 extends f_xi
{
//multivariable function
 double func(double x[],int x_ref)
 {
   //x[0] is x, x[1] is y
   if(x_ref==0) return x[2];
   if(x_ref==1) return 10-10.0*x[1]-11*x[2];
   else return 0;
 }
}

class fm2 extends f_x
{// gerçek çözüm
 public double func(double t)
 {return 1.0-10.0/9.0*Math.exp(-t)+1.0/9.0*Math.exp(-10.0*t);}
}

class NA94
{

public static void main(String args[]) throws IOException
  {
  fm1 b1=new fm1();
  fm2 b2=new fm2();
  double y[];
  y=new double[2];
  y[0]=0;
  y[1]=0;
  Buthcher bb=new Buthcher("RKV13");
  Runge_Kutta rk=new Runge_Kutta(bb);
  double a[][]=rk.hesapla(b1,0.0,0.2,y,0.01);
  Text.print(Text.T(a));
  Plot pp=new Plot(a);
  pp.addFunction(b2,0.0,0.2,20);
  pp.addSpline(2,100);
  pp.setPlotType(2,33);
  pp.plot();
 }
}
```

Program 8.7-2 Solution of set of differential equation with 6th order Runge Kutta



The same set of equation solved by using Python language:

Program 8.7-3 f_xi abstract method for multidimension differential equation set in Python

```
from abc import ABC, abstractmethod
from math import *

class f_xi(ABC):
  # multifunction multi independent variable
  # vector of dependent variables are returned
  # example f[0]=x[0]+sin(x[1])
  #         f[1]=x[0]*x[1]-x[1]
  # func(x,1) returns the value of f[1]
  # func(x,0) returns the value of f[0]

  @abstractmethod
  def func(self,x,equation_ref):
    pass;
```

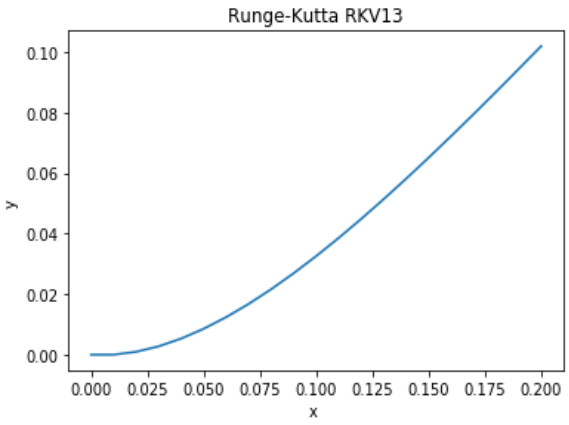
Program 8.7-4 f_ Solution of set of differential equation with Runge Kutta with Python

```
# -*- coding: utf-8 -*-
"""
Created on Fri Aug 31 13:10:28 2018
@author: Mustafa Turhan Çoban
"""
from Runge_Kutta import *;
from  Butcher import *;
from f_xi import *;
import numpy as np;
import matplotlib.pyplot as plt;

class f1(f_xi):
    def func(self,x,x_ref):
        if x_ref==0: F=x[2];
        elif x_ref==1: F=10.0-10.0*x[1]-11.0*x[2];
        else: F=0.0;
        return F;

rk=Runge_Kutta("RKV13");
f=f1();
```

```
y=[0 for i in range(2)];
y[0]=0.0;
y[1]=0.0;
XY=rk.calc_n(f,0.0,0.2,y,0.01,0);
print(XY);
plt.title('Runge-Kutta RKV13')
plt.xlabel('x ')
plt.ylabel('y ');
plt.plot(XY[0],XY[1]);
```



Few more example will be given in this chapter to understand nature of the real differential equation solutions. Fluid mechanics is a topic that mechanical engineering, chemical engineering and meterology is closely interested. Turbulent flow is one of the important sub topic. In the classical approach to the turbulance random movements are assumed. When it is investigated further it is shown that some diferential equations can be given chaotic results. This created a new science of chaotic systems. Edward Lorents, an athmospheric scientist developed Lorents set of diferential equations to investigate atmospheric turbulance effects. The equation set:

$$\frac{dx_1}{dt} = \sigma(x_2 - x_2)$$

$$\frac{dx_2}{dt} = (1 + \lambda - x_3)x_1 - x_2$$

$$\frac{dx_3}{dt} = x_1 x_2 - \gamma x_2$$

In this equations σ, λ, and γ are pozitif constants. Equation behave chaotically when
$\sigma > \gamma + 1$ and

$\lambda > \frac{(\sigma+1)(\sigma+\gamma+1)}{\sigma-\gamma-1}$. For example with $\sigma = 10$, $\lambda = 24$ ve $\gamma = 2$ system behaves chaotically. If we solve the Lorents equation set with $0s \leq t \leq 30s$time range, $x_1(0) = 1.0$; $x_2(0) = 0.0$; $x_3(0) = 20.0$ conditions

Program 8.7-3 Solution of Lorenz equation with RKV13

```
//=====================================================
// Numerical Analysis package in java
// example to show differential equation solution
// Runge-Kutta-Verner 16 constant step method with
// multible differential equations
// Dr. Turhan Coban
// =====================================================
// dy1/dx=f(x,y1,y2,..,yn)
```

```
// dy2/dx=f(x,y1,y2,..,yn)
// ......
// dyn/dx=f(x,y1,y2,..,yn)
// LORENTS equation solution
import java.io.*;

class fm2 extends f_xi
{
//multivariable function
double sigma=10.0;
 double lambda=28.0;
 double gamma=8.0/3.0;



 double func(double x[],int x_ref)
 {
   //x[0] is x, x[1] is y

   if(x_ref==0) return sigma*(x[2]-x[1]);
   else if(x_ref==1) return (1+lambda-x[3])*x[1]-x[2];
   else if(x_ref==2)       return x[1]*x[2]-gamma*x[3];
   else return 0;
 }
}

class NA95
{

public static void main(String args[]) throws IOException
  {

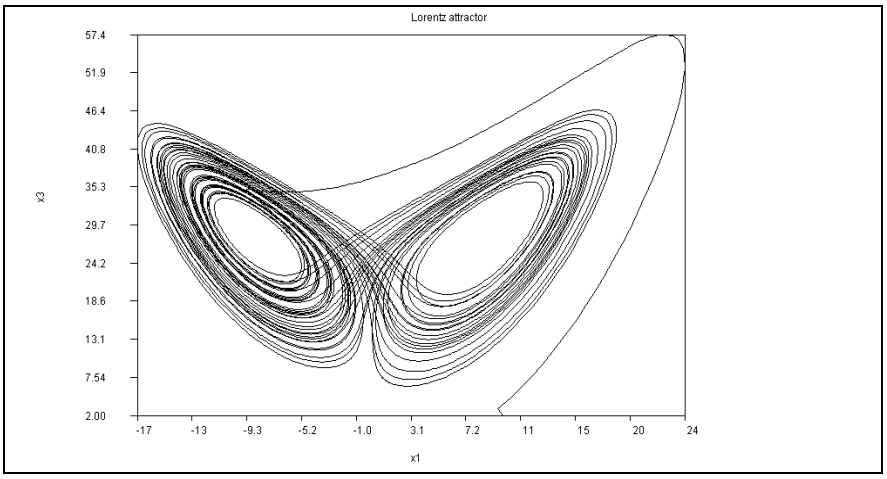
  fm2 b1=new fm2();
  double y[];
  y=new double[3];
  y[0]=2.0;
  y[1]=2.0;
  y[2]=2.0;
   Buthcher bb=new Buthcher("RKV13");
  Runge_Kutta rk=new Runge_Kutta(bb);
  double a[][]=rk.hesapla(b1,0.0,40.0,y,0.001);
  Text.printT(a);
  Plot p1=new Plot(a[1],a[3]);
  p1.setXlabel("x1");
  p1.setYlabel("x3");
  p1.setPlabel("Lorenz attractor");
  p1.plot();

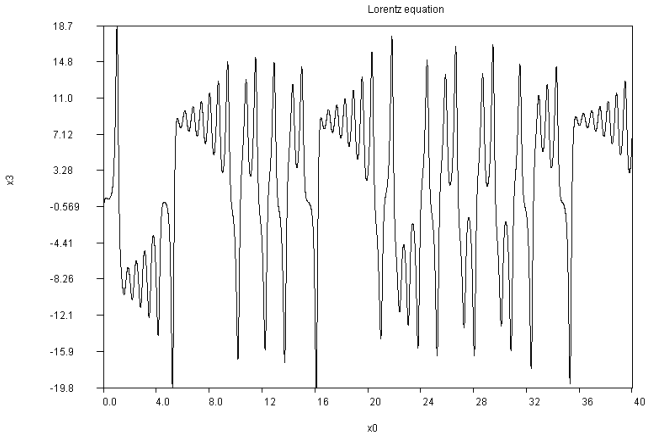
  Plot p2=new Plot(a[0],a[1]);
  p2.setXlabel("x0");
  p2.setYlabel("x3");
  p2.setPlabel("Lorentz equation");
  p2.plot();

 }
}
```

Output 8.7-3 Solution of lorentz equation by using 6th order Runge-Kutta method

$\sigma = 10$, $\lambda=24$ and $\gamma=8/3$





Python equivalent of this solution:

Program 8.7-3A Solution of Lorenz equation with RKV13 python version

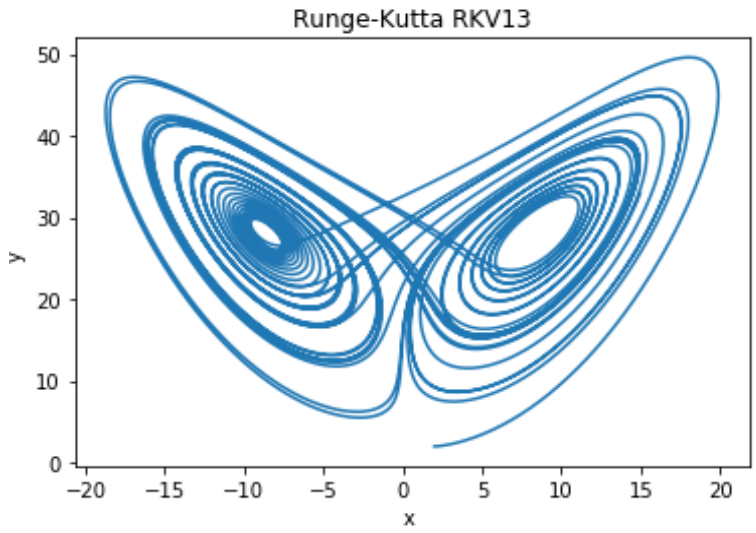
```
# -*- coding: utf-8 -*-
"""
Created on Fri Aug 31 13:10:28 2018

@author: Mustafa Turhan Çoban
"""
from Runge_Kutta import *;
from  Butcher import *;
from f_xi import *;
import numpy as np;
import matplotlib.pyplot as plt;

class f1(f_xi):
    def __init__(self):
      self.name="Lorentz equation";
      self.sigma=10.0;
      self.lambd=28.0;
      self.gamma=8.0/3.0;
    def func(self,x,x_ref):
        if x_ref==0: F=self.sigma*(x[2]-x[1]);
        elif x_ref==1: F=(1.0+self.lambd-x[3])*x[1]-x[2];
        elif x_ref==2: F=x[1]*x[2]-self.gamma*x[3];
        else: F=0.0;
        return F;

rk=Runge_Kutta("RKV13");
f=f1();
y=[0 for i in range(3)];
```

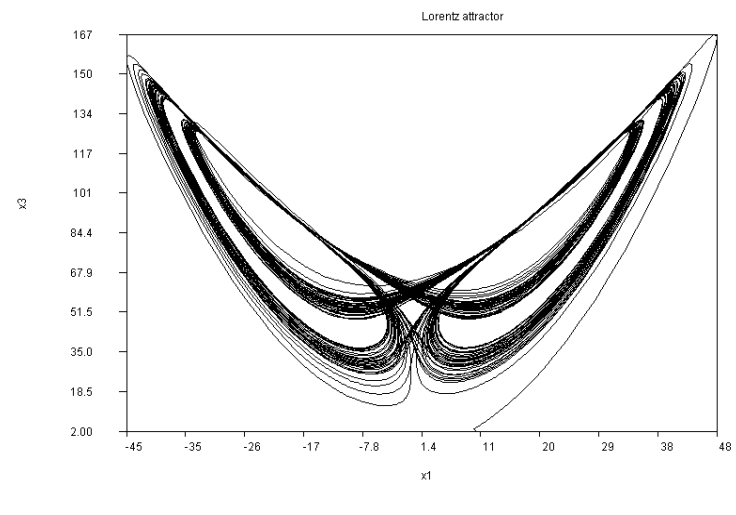
```
y[0]=2.0;
y[1]=2.0;
y[2]=2.0;
XY=rk.calc_n(f,0.0,40.0,y,0.001,0);
print(XY);
plt.title('Runge-Kutta RKV13')
plt.xlabel('x ')
plt.ylabel('y ');
plt.plot(XY[1],XY[3]);
```

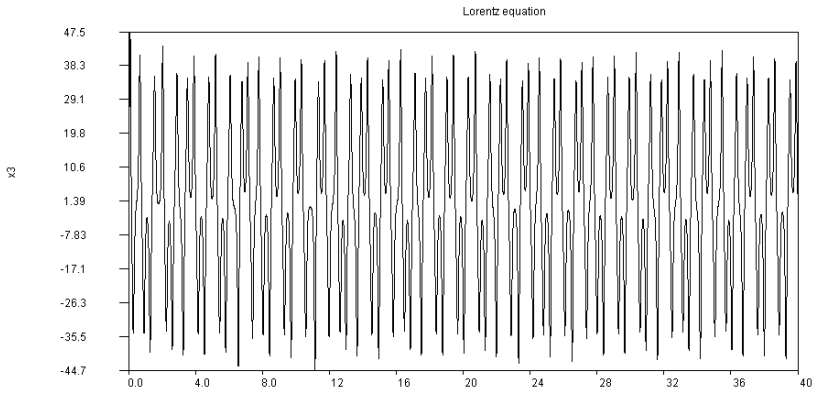


If parameters is changed a completely different kind of behavior can be observed.

Output 8.7-4 Solution of Lorenz equation with 16 degree Runge-Kutta-Verner method

$\sigma = 10$, $\lambda=90$ ve $\gamma=5.5$





Another interesting example in chaotic behavior is Predator-Prey model. When the population of Predators are increased, the population of the Prey animal will decreased and it will caused population decrease of the Predator population as well. This relation is explained and put into a matematical model by italian mathematician Vito Voltera and American Biolog Alfret J. Lotka. Their equation is:

$$\frac{dy_1}{dt} = ay_1 - by_1y_2$$
$$\frac{dy_2}{dt} = -cy_2 + dy_1y_2$$

In this equation y_1 is Prey animal population, y_2 is Predator population, a is Prey animal birth rate, b is Predator dead rate, c is Predator-Prey relation ratio in Predator dead rate and d is Predator-Prey relation ratio in Predator population increase.

Program 8.7-4 Predator-Prey model (Lotka-Volterra equation)

```
//=====================================================
// Numerical Analysis package in java
// example to show differential equation solution
// Runge-Kutta-Verner 16 method with multible differential equations
// Dr. Turhan Coban
// =====================================================
// dy1/dx=f(x,y1,y2,..,yn)
// dy2/dx=f(x,y1,y2,..,yn)
// ......
// dyn/dx=f(x,y1,y2,..,yn)
// Lotka - Volterra equation
// Preditor-Prey model
import java.io.*;

class fm1 extends f_xi
{
//Preditor-Prey model
// Lotka-Volterra equation
 double a=1.2;  //prey birth rate
 double b=0.6;  //preditor death rate
 double c=0.8;  //effect of preditor-prey relation on prey death rate
 double d=0.3;  //effect of preditor-prey relations on preditor birth rate
 double func(double x[],int x_ref)
 {
   //x[0] is x, x[1] is y

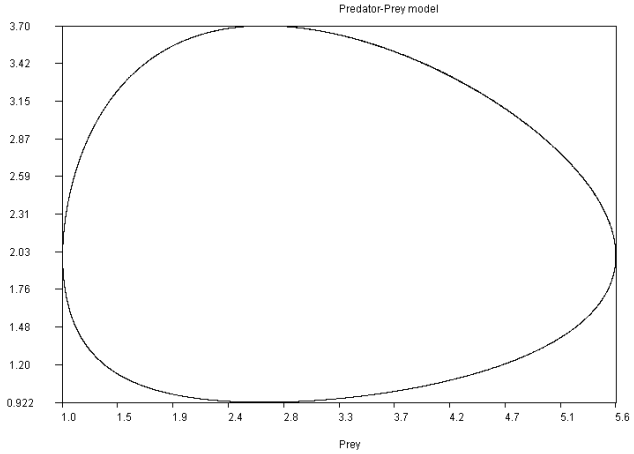
   if(x_ref==0) return a*x[1]-b*x[1]*x[2];
   else return -c*x[2]+d*x[1]*x[2];
 }
}

class NA96
{
```

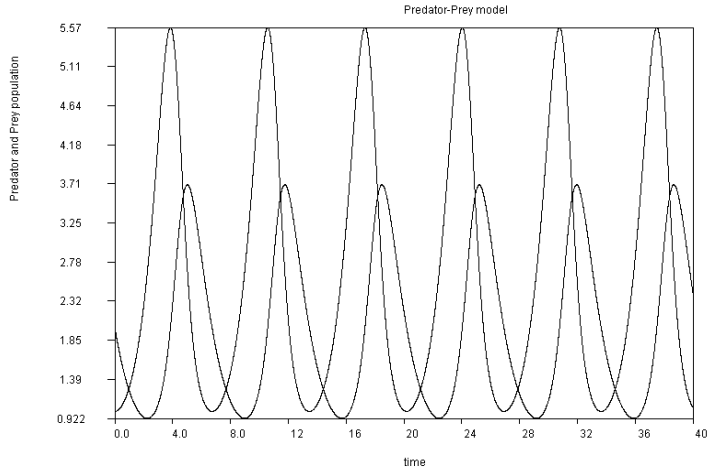
```
public static void main(String args[]) throws IOException
{
  fm1 b1=new fm1();
  double y[];
  y=new double[2];
  y[0]=1.0;
  y[1]=2.0;
  double a[][]=NA92F.RKV16(b1,0.0,40.0,y,10000);
  Plot p1=new Plot(a[0],a[1]);
  p1.addData(a[0],a[2]);
  p1.setXlabel("time");
  p1.setYlabel("Predator and Prey population");
  p1.setPlabel("Predator-Prey model");
  p1.plot();

  Plot p2=new Plot(a[1],a[2]);
  p2.setXlabel("Prey");
  p2.setYlabel("Predator");
  p2.setPlabel("Predator-Prey model");
  p2.plot();
 }
}
```

Output 8.7-5 Predator-Prey model (Lotka-Volterra equation) a=1.2 b= 0.6 c=0.8 d=0.3, Predator-pray population change with respect to each other



Output 8.7-6 Predator-Prey model (Lotka-Volterra equation) a=1.2 b= 0.6 c=0.8 d=0.3, Predator-pray population change with time



While investigating hard beats, Van der Pol developed the set of differential equation to represent hard beats. This equations are

$\frac{d^2y}{dt^2} + \mu(y^2 - 1)\frac{dy}{dt} + y = 0$ If this equation are converted into a system of equations with $y=y_1$ and

$$\frac{dy}{dt} = y_2$$

$$\frac{dy_1}{dt} = y_2$$

$$\frac{dy_2}{dt} = -y_1 - \mu(y_1^2 - 1)y_2 = 0$$

If this equation is solved for $\mu=1$ and $y_1(0)=0$, $y_2(0)=0.5$:

Program 8.7-5 Van der Pol differential equation

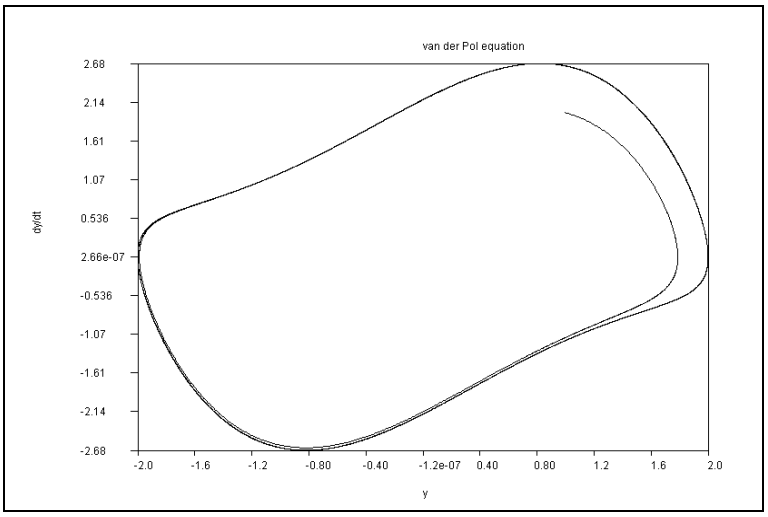
```
//=======================================================
// Numerical Analysis package in java
// example to show differential equation solution
// Runge-Kutta_verner 16 method with multible differential equations
// Dr. Turhan Coban
// =======================================================
// dy1/dx=f(x,y1,y2,..,yn)
// dy2/dx=f(x,y1,y2,..,yn)
// .....
// dyn/dx=f(x,y1,y2,..,yn)
// Van der Pol equation
// hard bit differential equatin model

import java.io.*;

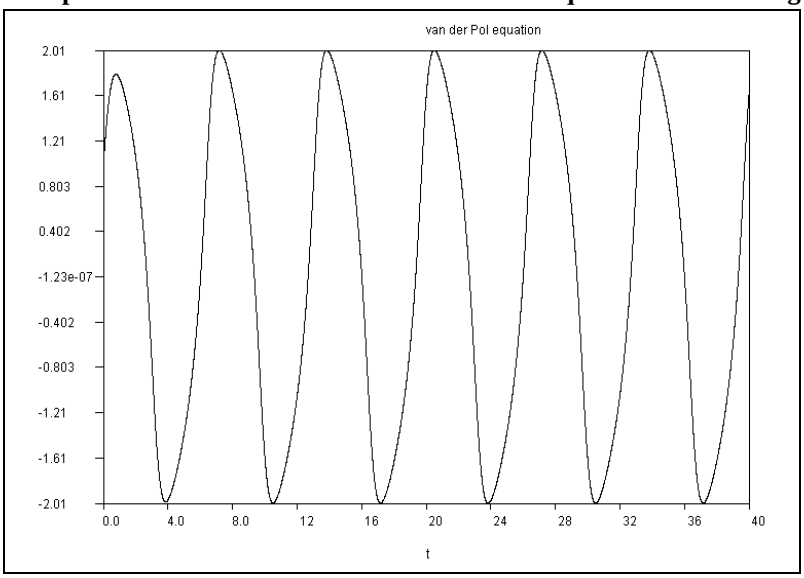

class fm1 extends f_xi
{
//Van der Pol equation
//simulates hard beats
 double mu=1.0;
 // d^2y/dt^2+mu(y^2-1)dy/dt=0
 double func(double x[],int x_ref)
 {
   //x[0] is x, x[1] is y
   if(x_ref==0) return x[2];
   else  return -x[1]-mu*(x[1]*x[1]-1.0)*x[2]; }}
class NA97
{
public static void main(String args[]) throws IOException
  {
  fm1 b1=new fm1();
  double y[];
  y=new double[3];
  y[0]=1;
  y[1]=2.0;
  double a[][]=NA92F.RKV16(b1,0.0,40.0,y,10000);
  Plot p1=new Plot(a[1],a[2]);
  p1.setXlabel("y");
  p1.setYlabel("dy/dt");
  p1.setPlabel("van der Pol equation");
  p1.plot();
  Plot p2=new Plot(a[0],a[1]);
  p2.setXlabel("t");
  p2.setYlabel("y");
```

```
 p2.setPlabel("van der Pol equation");
 p2.plot();
}}
```

Output 8.7-7 Van der Pol denklem setinin 6 derece Runge-Kutta metodu ile çözülmesi



Output 8.7-8 Solution of Van der Pol denklem equation with 6th degree Runge-Kutta method



The next example is the harmonic motion with damping. The diferential equation is:

$$\frac{d^2y}{dt^2} + b\frac{dy}{dt} + \omega^2 y = 0$$

The equation will be solved with initial conditions y(0)=1 $\frac{dy(0)}{dt} = 1$and ω =1 ve b=0.1. When b=0, it is called simple harmonic motion. a periodic motion that is neither driven nor damped. When b=0.1, it is a damped motion. Oscilations will get smaller and stop after a certain time. Converted form of the equation:

$$\frac{dy_1}{dt} = y_2$$

$$\frac{dy_2}{dt} = -by_2 - \omega^2 y$$

$y_1(0)=0$

$y_2(0)=1$

Program 8.7-6 Harmonic motion with damping

```
//=====================================================
// Numerical Analysis package in java
// example to show differential equation solution
// and differentiation (derivative) functions
// Dr. Turhan Coban
// =====================================================
import java.io.*;

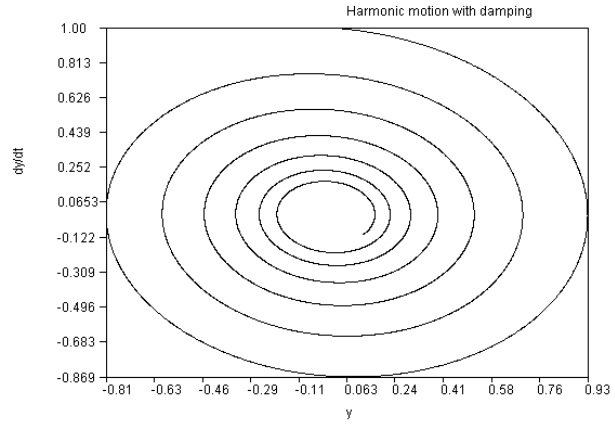
class fm1 extends f_xi
{
//Harmonic motion
 double b=0.1;
 double w=1.0;
 // d^2y/dt^2+b*dy/dt+w^2y=0
 double func(double x[],int x_ref)
 {
   //x[0] is x, x[1] is y
   if(x_ref==0) return x[2];
   else  return -b*x[2]-w*w*x[1];
 }
}

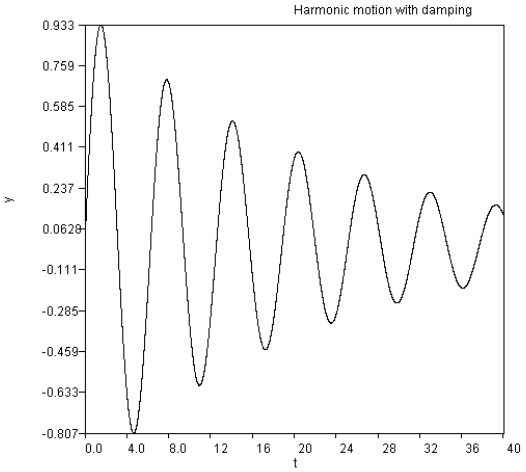
class NA98A
{

public static void main(String args[]) throws IOException
  {

  fm1 b1=new fm1();
  double y[];
  y=new double[3];
  y[0]=0.0;
  y[1]=1.0;
  double a[][]=NA92F.RKV16(b1,0.0,40.0,y,10000);
  Plot p1=new Plot(a[1],a[2]);
  p1.setXlabel("y");
  p1.setYlabel("dy/dt");
  p1.setPlabel("Harmonic motion with damping");
  p1.plot();
  Plot p2=new Plot(a[0],a[1]);
  p2.setXlabel("t");
  p2.setYlabel("y");
  p2.setPlabel("Harmonic motion with damping");
  p2.plot();
 }
}
```

Output 8.7-9 Harmonic motion with damping





Python code of the Harmonic motion:

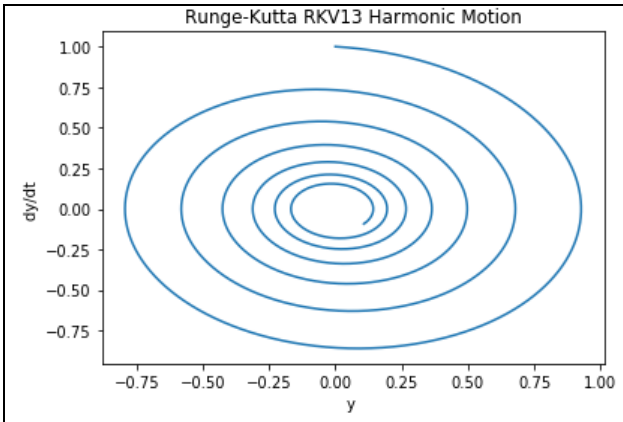
Program 8.7-6A Harmonic motion with damping python code

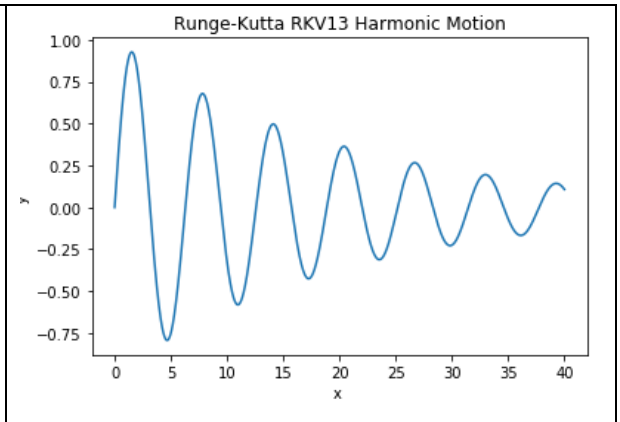
```
# -*- coding: utf-8 -*-
"""
Created on Fri Aug 31 13:10:28 2018

@author: Mustafa Turhan Çoban
"""
from Runge_Kutta import *;
from  Butcher import *;
from f_xi import *;
import numpy as np;
import matplotlib.pyplot as plt;

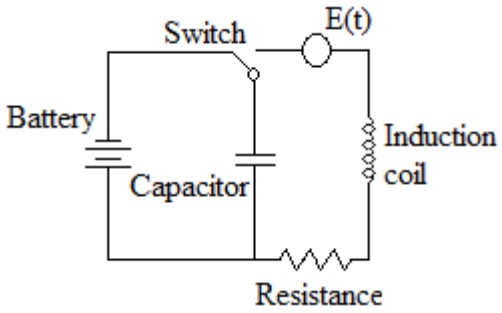
class f1(f_xi):
    #Harmonic motion
    #d^2y/dt^2+c*dy/dt+w^2y=0
     def __init__(self):
       self.name="Harmonic motion";
       self.b=0.1;
       self.w=1.0;
     def func(self,x,x_ref):
         if x_ref==0: F=x[2];
         elif x_ref==1: F=-self.b*x[2]-self.w*self.w*x[1];
         else: F=0.0;
         return F;
```

```
rk=Runge_Kutta("RKV13");
f=f1();
y=[0 for i in range(2)];
y[0]=0.0;
y[1]=1.0;
XY=rk.calc_n(f,0.0,40.0,y,0.001,0);
print(XY);
plt.title('Runge-Kutta RKV13 Harmonic Motion')
plt.xlabel('x ')
plt.ylabel('y ');
plt.plot(XY[0],XY[1]);
```





The next example is the harmonic motion with damping in an electrical circuit. This kind of behavior can be seen frequently in circuits when current is not constant and changed as a function of time. Such a circuit is shown in the figure below.



The differential equation of the system:

$L\frac{d^2q}{dt^2} + R\frac{dq}{dt} + \frac{q}{C} - E(t) = 0$ In this equation q is capacitor energy storage capacity (C, Coulomb). L enductance (H, Henry), R res,stance (Ω, Ohm), C capacitance (F), and $E(t)=E_0\sin(wt)$ is time dependent voltage, w frequency and t is time (s) . Current density is the derivative of capacitor energy storage capacity.

$i = \frac{dq}{dt}$ (Ampere). An analytical solution of this equation for R=0 ve $E(t)=E_0\sin(wt)$:

$q(t) = \frac{-E_0}{L(p^2-w^2)}\frac{w}{p}sin(pt) + \frac{E_0}{L(p^2-w^2)}sin(wt)$. And the current density

$i = \frac{dq(t)}{dt} = \frac{-E_0 w}{L(p^2-w^2)}sin(pt) + \frac{E_0 w}{L(p^2-w^2)}sin(wt)$. In this equation

$p = \frac{1}{\sqrt{LC}}$dir.

Now this equation will be solved numerically for initial conditions q(0)=0 ve $i(0) = \frac{dq(0)}{dt} = 0$ and physical conditions w^2=3.5 1/s^2 p=2, L=1 H, E_0=1 V, C=0.25 F. Since analytical solution is known, two solutions can be compared. It is a good exampel to observe the numerical solution error generation due to very complex behaviour of the function.

Converted equation :

$$\frac{dy_1}{dt} = y_2$$

$$\frac{dy_2}{dt} = \frac{\left[-Ry_2 - \frac{y_1}{C} + E_0\, sin(wt)\right]}{L}$$

$y_1(0)=0$
$y_2(0)=0$

Program 8.7-7 Harmonic motion electrical circuit (Changing voltage problem)

```
//=====================================================
// Numerical Analysis package in java
// example to show differential equation solution
// Runge-Kutta 6 method with multible differential equations
// Dr. Turhan Coban
// =====================================================
// dy1/dx=f(x,y1,y2,..,yn)
// dy2/dx=f(x,y1,y2,..,yn)
// ......
// dyn/dx=f(x,y1,y2,..,yn)
// Changing voltage problem

import java.io.*;

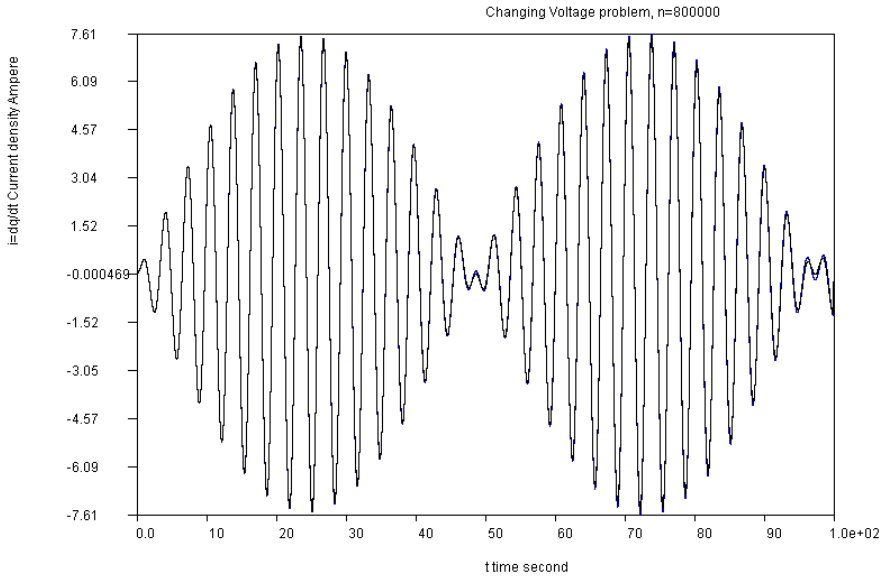

class fm1 extends f_xi
{
 double E0=1.0;  //Volt
 double L=1.0;   // Henry
 double C=0.25;  // Coulomb
 double w2=3.5;  // s^2
 double R=0.0;     // Ohm
 double w=Math.sqrt(w2);
 double p=1.0/Math.sqrt(L*C);
 // Ld^2q/dt^2+R*dq/dt+q/C-E(t)=0
 double func(double x[],int x_ref)
 {
   //x[0] is t, x[1] is q x[2] is i
   if(x_ref==0) return x[2];
   else  return (-R*x[2]-x[1]/C+E0*Math.sin(w*x[0]))/L;
 }
 double i(double t)
 {
 return -E0/L/(p*p-w2)*w*Math.cos(p*t)+E0/L/(p*p-w2)*w*Math.cos(w*t);
 }
 double q(double t)
 {
 return -E0/L/(p*p-w2)*w/p*Math.sin(p*t)+E0/L/(p*p-w2)*Math.sin(w*t);
 }
}

class NA98
{

public static void main(String args[]) throws IOException
```

```
 {

 fm1 b1=new fm1();
 double y[];
 y=new double[3];
 y[0]=0.0;
 y[1]=0.0;
 int n=800000;
 double a[][]=NA93.RK6(b1,0.0,100.0,y,n);
 Plot p1=new Plot(a[0],a[2]);
 double t=0;
 double dt=100.0/n;
 for(int i=0;i<(n-1);i++) {a[1][i]=b1.i(t);t+=dt;}
 p1.addData(a[0],a[1]);
 p1.setXlabel("t time second");
 p1.setYlabel("i=dq/dt Current density Ampere");
 p1.setPlabel("Changing Voltage problem");
 p1.plot();
 }
}
```



Python version of the code is given below, analyical solution is omitted:

Program 8.7-7A Harmonic motion electrical circuit (Changing voltage problem) Python version

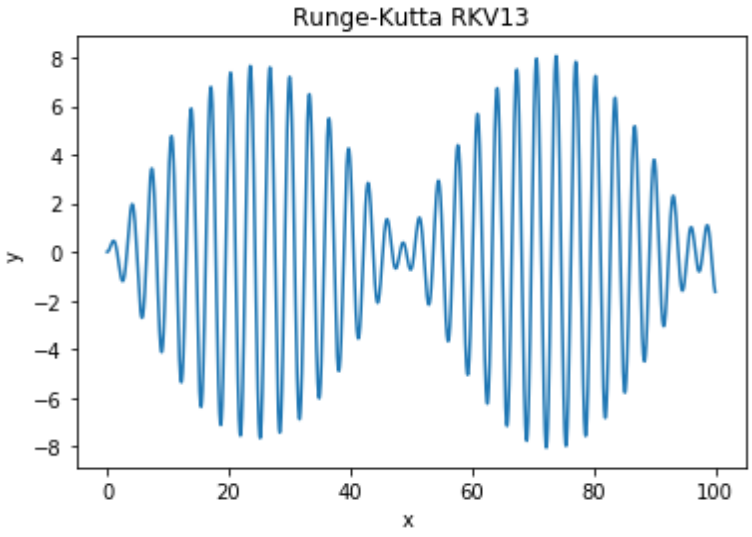
```
# -*- coding: utf-8 -*-
"""
Created on Fri Aug 31 13:10:28 2018

@author: Mustafa Turhan Çoban
"""
from Runge_Kutta import *;
from  Butcher import *;
from f_xi import *;
from math import *
import numpy as np;
import matplotlib.pyplot as plt;

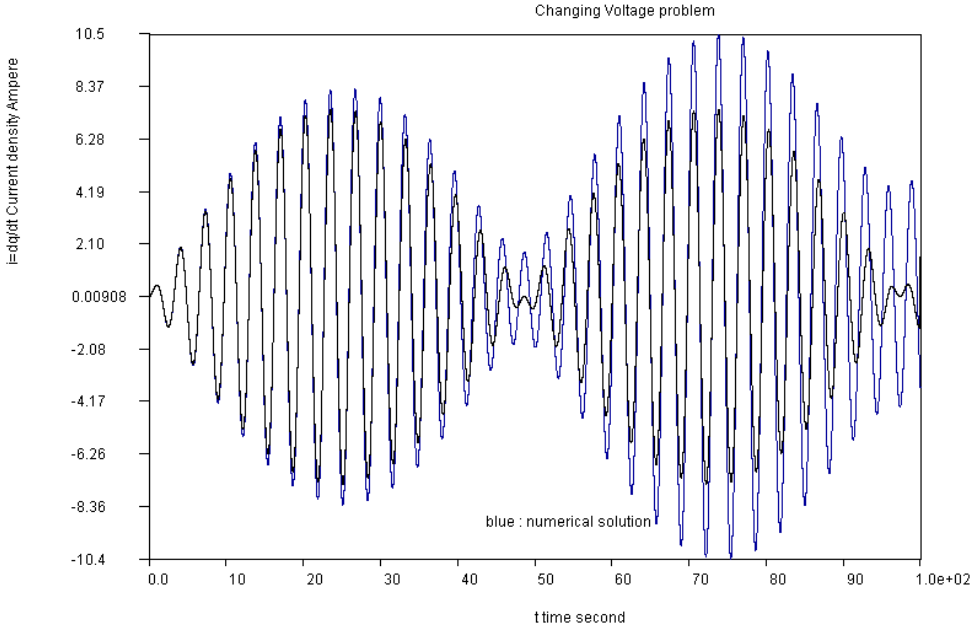
class f1(f_xi):
    def __init__(self):
      self.name="Harmonic electronic circuit";
      self.E0=1.0;#Volt
      self.L=1.0;#Henry
      self.C=0.25;#Coulomb
```

```
        self.w2=3.5;#s^2
        self.R=0.0;#Ohm
        self.w=sqrt(self.w2);
        self.p=1.0/sqrt(self.L*self.C);
    def func(self,x,x_ref):
        if x_ref==0: F=x[2];
        elif x_ref==1: F=(-self.R*x[2]-x[1]/self.C+self.E0*sin(self.w*x[0]))/self.L;
        else: F=0.0;
        return F;

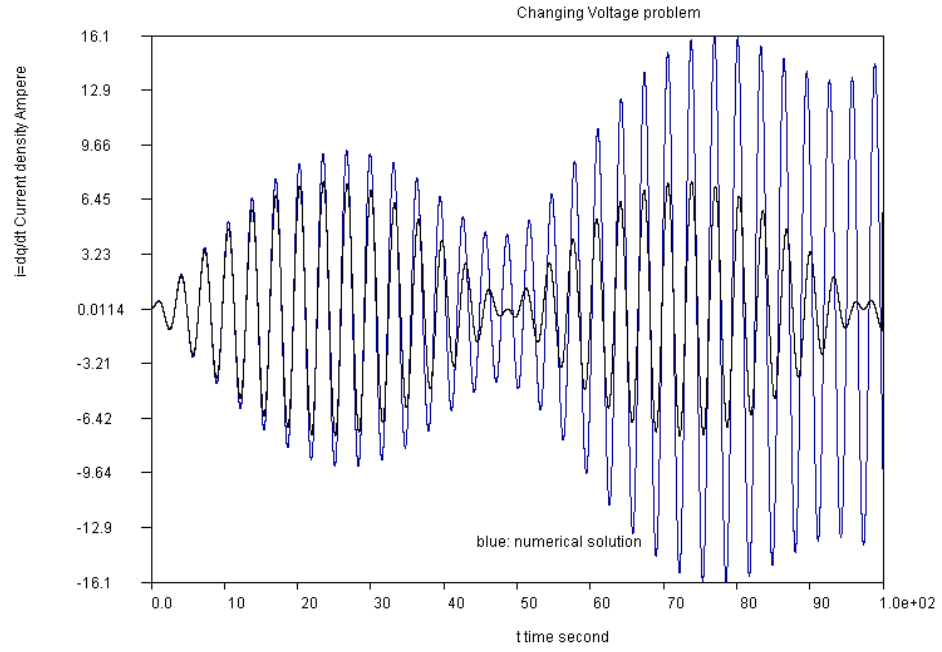
rk=Runge_Kutta("RKV13");
f=f1();
y=[0 for i in range(2)];
y[0]=0.0;
y[1]=0.0;
XY=rk.calc_n(f,0.0,100.0,y,0.001,0);
print(XY);
plt.title('Runge-Kutta RKV13')
plt.xlabel('x ')
plt.ylabel('y ');
plt.plot(XY[0],XY[2]);
```



To calculate current both numerical and analytical solutions are calculated and plotted. As number of points N=800000 points are used for the numerical calculations. If N=50000, error generation in numerical solution can be observed.



For N=25000 level of numerical error even gets bigger by time



If differential equation is changed to Runge-Kutta-Verner 16th degree constant step size with N=4000, Result is very accurate

Program 8.7-7C Harmonic motion electrical circuit RKV16 N=4000

```
//=====================================================
// Numerical Analysis package in java
// example to show differential equation solution
// Runge-Kutta 6 method with multible differential equations
// Dr. Turhan Coban
// =====================================================
// dy1/dx=f(x,y1,y2,..,yn)
```

```
// dy2/dx=f(x,y1,y2,..,yn)
// .....
// dyn/dx=f(x,y1,y2,..,yn)
// Changing voltage problem

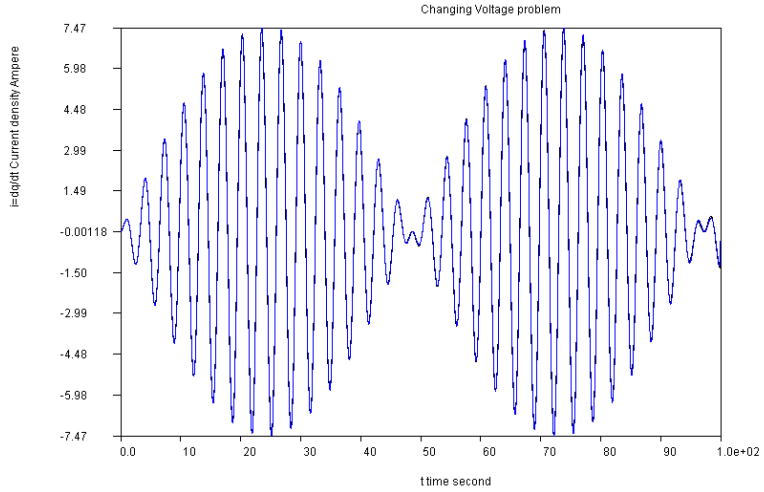
import java.io.*;


class fm1 extends f_xi
{
 double E0=1.0;  //Volt
 double L=1.0;   // Henry
 double C=0.25;  // Coulomb
 double w2=3.5;  // s^2
 double R=0.0;     // Ohm
 double w=Math.sqrt(w2);
 double p=1.0/Math.sqrt(L*C);
 // Ld^2q/dt^2+R*dq/dt+q/C-E(t)=0
 double func(double x[],int x_ref)
 {
   //x[0] is t, x[1] is q x[2] is i
   if(x_ref==0) return x[2];
   else  return (-R*x[2]-x[1]/C+E0*Math.sin(w*x[0]))/L;
 }
 double i(double t)
 {
 return -E0/L/(p*p-w2)*w*Math.cos(p*t)+E0/L/(p*p-w2)*w*Math.cos(w*t);
 }
 double q(double t)
 {
 return -E0/L/(p*p-w2)*w/p*Math.sin(p*t)+E0/L/(p*p-w2)*Math.sin(w*t);
 }
}

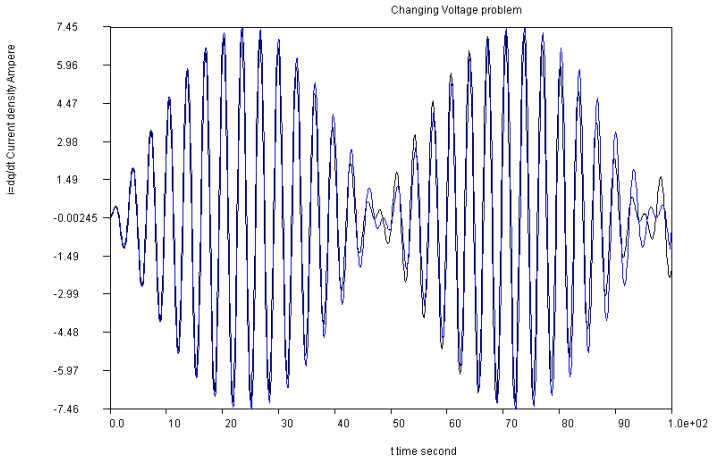
class NA98C
{

public static void main(String args[]) throws IOException
  {

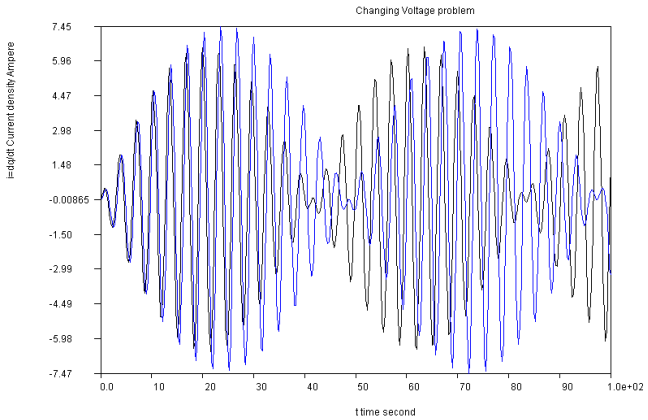
  fm1 b1=new fm1();
  double y[];
  y=new double[3];
  y[0]=0.0;
  y[1]=0.0;
  int n=4000;
  double a[][]=NA92F.RKV16(b1,0.0,100.0,y,n);
  Plot p1=new Plot(a[0],a[2]);
  double t=0;
  double dt=100.0/n;
  for(int i=0;i<(n-1);i++) {a[1][i]=b1.i(t);t+=dt;}
  p1.addData(a[0],a[1]);
  p1.setXlabel("t time second");
  p1.setYlabel("i=dq/dt Current density Ampere");
  p1.setPlabel("Changing Voltage problem");
  p1.plot();
 }
}
```



With RKV16 N=1000 system starts to diverge from exact solution



With RKV16 and N=400 divergence is quite clear.



Another example. Assume that a bullet is thrown from the gun into an atmosphere. Due to high velocity of the bullet, friction will effect the movement of the bullet. The diferential equation of the motion in cartesian coordinate system can be given as:

$$\frac{d^2x}{dt^2} = -\gamma(\sqrt{v_x^2 + v_y^2})v_x$$

$$\frac{d^2y}{dt^2} = -g - \gamma(\sqrt{v_x^2 + v_y^2})v_y$$

In this equation $\gamma = \frac{k}{m}$,where k friction coefficient between the bullet and air, v_x is x direction component of the velocity, v_y is y direction component of the velocity, m mass, g gravitational acceleration, and t is time. With t=x[0] , x=x[1], y=x[2], v_x=x[3], v_y=x[4]

variable convertion the diferential equation becomes

$$\frac{dx[1]}{dx[0]} = x[3]$$

$$\frac{dx[2]}{dx[0]} = x[4]$$

$$\frac{dx[3]}{dx[0]} = -\gamma(\sqrt{x^2[3] + x^2[4]})x[3]$$

$\frac{dx[4]}{dx[0]} = -g - \gamma(\sqrt{x^2[3] + x^2[4]})x[4]$ z

Initial conditions x[0]=0 da x[1]=0 x[2]=0,

V_0=250 m/s x[3]= V_0 sin(θ) m/s, x[4]= V_0 cos(θ) m/s, g=9.806 m/s^2, γ=0.01 , θ =30,45,60 degree

Program 8.7-8 gun bullet trajectory with air friction

```
//=====================================================
// Numerical Analysis package in java
// example to show differential equation solution
// and differentiation (derivative) functions
// Dr. Turhan Coban
// =====================================================
import java.io.*;

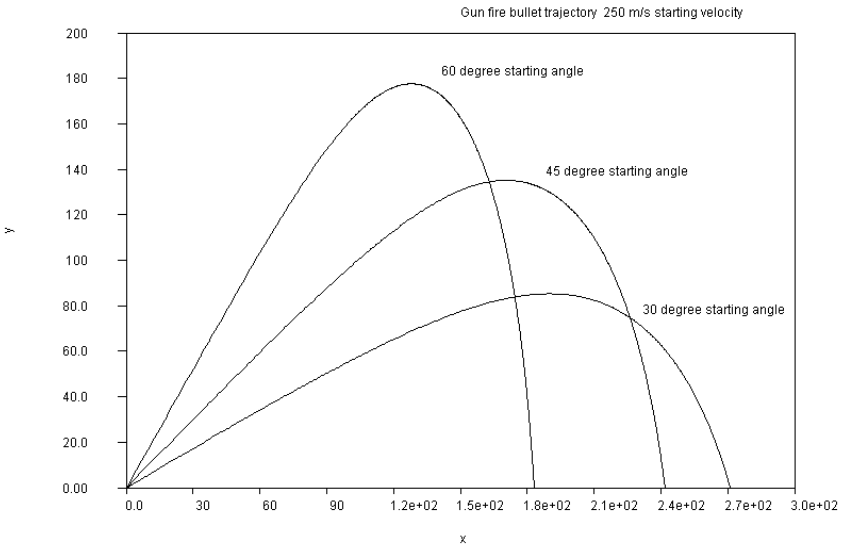

class fm1 extends f_xi
{
 double gamma;
              public fm1(double gammai)
   {gamma=gammai;}
 // sürtünmeli eğik atiş problemi
 double func(double x[],int x_ref)
 {
  //x[0] is x, x[1] is y
  double g=9.806;
  if(x_ref==0) return x[3];
  else if(x_ref==1) return x[4];
  else if(x_ref==2) return -gamma*Math.sqrt(x[3]*x[3]+x[4]*x[4])*x[3];
  else            return -g - gamma*Math.sqrt(x[3]*x[3]+x[4]*x[4])*x[4];
 }
}

class diferansiyel3F
{

public static void main(String args[]) throws IOException
  {
  fm1 b1=new fm1(0.01);
  double y[];
  y=new double[4];
  double V0=250.0;//  m/s
  double alfa=Math.PI/3; //60 derece
  y[0]=0.0;
  y[1]=0.0;
  y[2]=V0*Math.sin(alfa);
  y[3]=V0*Math.cos(alfa);
  double a[][]=diferansiyel3.RK6(b1,0.0,20.0,y,1000);
```

```
 Plot p1=new Plot(a[1],a[2]);
 alfa=Math.PI/4;  //45 derece
 y[2]=V0*Math.sin(alfa);
 y[3]=V0*Math.cos(alfa);
 double b[][]=diferansiyel3.RK6(b1,0.0,20.0,y,1000);
 p1.addData(b[1],b[2]);
 alfa=Math.PI/6;  //30 derece
 y[2]=V0*Math.sin(alfa);
 y[3]=V0*Math.cos(alfa);
 double c[][]=diferansiyel3.RK6(b1,0.0,20.0,y,1000);
 p1.addData(c[1],c[2]);
 double xa[]=new double[2];
 double ya[]=new double[2];
 xa[0]=0;xa[1]=12.0;
 ya[0]=0;ya[1]=0.0;
 p1.addData(xa,ya);
 p1.setXlabel("x");
 p1.setYlabel("y");
 p1.setPlabel("sürtünmeli top atişi 60,45 ve 30 derece açi ve 250 m/s mermi çikiş hizi");
 p1.setMinMax(0,300,0,200);
 p1.plot();
 }
}
```

Output 8.7-10 gun bullet trajectory with air friction
θ=30, 45, 60 derece, bullet gun exit speed 250 m/s, gravitational accaleration 8.806 m/s^2, Friction coefficient γ = (k / m)= 0.01



Not that a bigger y scale is used in order to see trajectory ore clearly. Python version of the problem is given below:

Program 8.7-8A gun bullet trajectory with air friction , Python version

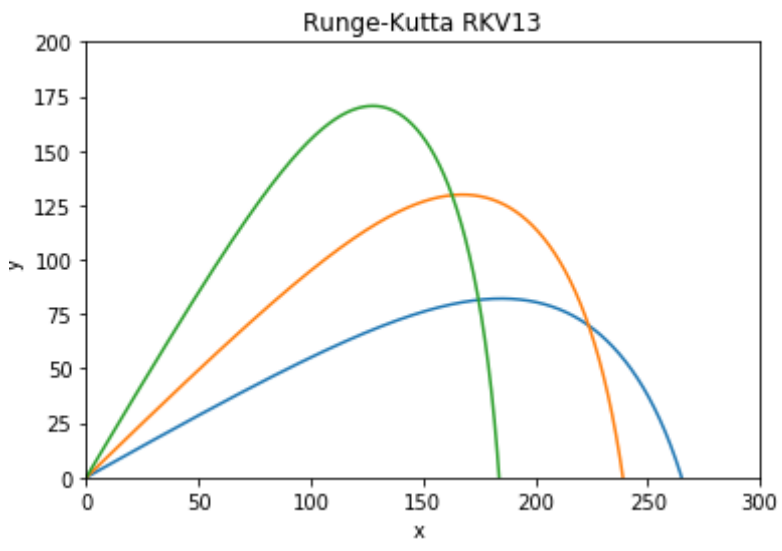
```
# -*- coding: utf-8 -*-
"""
Created on Fri Aug 31 13:10:28 2018

@author: Mustafa Turhan Çoban
"""
from Runge_Kutta import *;
from  Butcher import *;
from f_xi import *;
from math import *
import numpy as np;
```

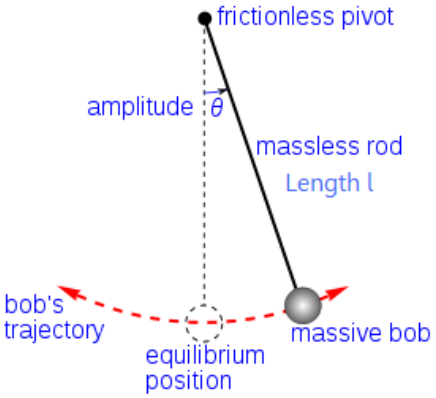
```
import matplotlib.pyplot as plt;

class f1(f_xi):
    def __init__(self,gammai):
      self.name="Bullet trajectory";
      self.gamma=gammai;
      self.g=9.806;#m/s^2
    def func(self,x,x_ref):
        if x_ref==0: F=x[3];
        elif x_ref==1: F=x[4];
        elif x_ref==2: F=-self.gamma*sqrt(x[3]*x[3]+x[4]*x[4])*x[3];
        elif x_ref==3: F=-self.g - self.gamma*sqrt(x[3]*x[3]+x[4]*x[4])*x[4];
        else: F=0.0;
        return F;

rk=Runge_Kutta("RKV13");
f=f1(0.01);
y=[0 for i in range(4)];
V0=250.0;#m/s
alpha=pi/3.0;
y[0]=0.0;
y[1]=0.0;
y[2]=V0*sin(alpha);
y[3]=V0*cos(alpha);
a=rk.calc_n(f,0.0,20.0,y,0.001,0);
alpha=pi/4.0;
y[2]=V0*sin(alpha);
y[3]=V0*cos(alpha);
b=rk.calc_n(f,0.0,20.0,y,0.001,0);
alpha=pi/6.0;
y[2]=V0*sin(alpha);
y[3]=V0*cos(alpha);
c=rk.calc_n(f,0.0,20.0,y,0.001,0);
plt.title('Runge-Kutta RKV13')
plt.xlabel('x ')
plt.ylabel('y ');
plt.ylim(ymin=0);
plt.ylim(ymax=200);
plt.xlim(xmin=0);
plt.xlim(xmax=300);
plt.plot(a[1],a[2],b[1],b[2],c[1],c[2]);
```



Pendulum is another interesting problem. Pendulum is an old mechanical problem. A **pendulum** is a weight suspended from a pivot, length l, so it can swing freely. When a pendulum is displaced from its resting equilibrium position with a displacement angle θ, it is subject to a restoring force due to gravity that will accelerate it back toward the equilibrium position (θ=0). When released, the restoring force combined with the pendulum's mass causes it to oscillate about the equilibrium position, swinging back and forth. The time for one complete cycle, a left swing and a right swing, is called the period. From its discovery around 1602 by Galileo Galilei the regular motion of pendulums was used for timekeeping, and was the world's most accurate timekeeping technology until the 1930s.



The diferential equation of the pendulum motion is

$$\frac{d^2\theta}{dt^2} = -\frac{g}{l} sin(\theta)$$

In this equation , θ swing angle, t time, and g is gravitational acceleration coefficient(m/s^2) . This is a non-linear differential equation. For very small swing angles, from sin(θ) Taylor series opening

$$sin(\theta) = \theta - \frac{\theta^3}{3!} + \frac{\theta^5}{5!} - \frac{\theta^7}{7!} + \frac{\theta^9}{9!} + \ldots.$$

Diferential equation can be approximated as

$$\frac{d^2\theta}{dt^2} = -\frac{g}{l}\theta$$

This is a simple linear equation that can be solved by analytical methods easily. The solution is

$$\theta(t) = \theta_0 \, cos(\sqrt{\frac{g}{l}}t)$$

An analytical solution of the non-linear equation is not existed, but it can be solved by using numerical methods. Both solution is calculated and plotted by using the following program. As it is seen from the plot outputs, for small angles (π/16 radian) both solutions are quite similar, but when the angle grows to bigger values(π/2 radian), solutions differs.

Program 8.7-8 Pendulum problem

```
//=======================================================
// Numerical Analysis package in java
// example to show differential equation solution
// Runge-Kutta 6 method with multible differential equations
// Dr. Turhan Coban
// =======================================================
// dy1/dx=f(x,y1,y2,..,yn)
// dy2/dx=f(x,y1,y2,..,yn)
// ......
```

```
// dyn/dx=f(x,y1,y2,..,yn)
// Pendulum problem

import java.io.*;

class fm1 extends f_xi
{
// Pendulum equation
 double g=9.806;  //gravitational acceleration m/s2
 double l=0.6;   //pendulum length m

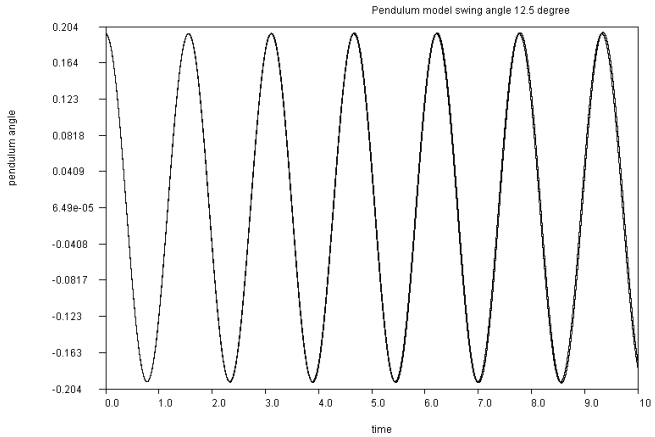
 double func(double x[],int x_ref)
 {
   //x[0] is t, x[1] is teta(pendulum swing angle) x[2] v velocity
   if(x_ref==0) return x[2];
   else return -g/l*Math.sin(x[1]);
 }
}

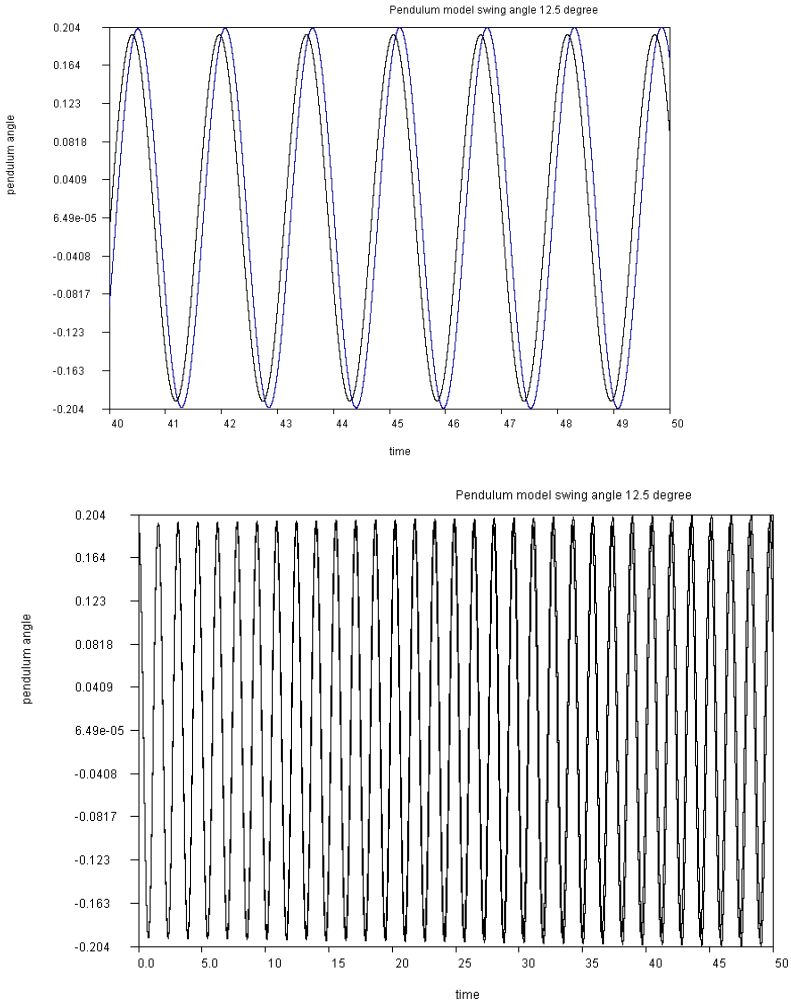
class NA101
{

public static void main(String args[]) throws IOException
{
  fm1 b1=new fm1();
  double y[];
  y=new double[2];
  y[0]=Math.PI/16.0; //11.25 degree swing angle
  y[1]=0.0;
  double a[][]=NA93.RK6(b1,0.0,10.0,y,100000);
  int n=a[0].length;
  double teta[]=new double[n];
  for(int i=0;i<n;i++) {teta[i]=y[0]*Math.cos(Math.sqrt(b1.g/b1.l)*a[0][i]);}
  Plot p1=new Plot(a[0],a[1]);
  p1.addData(a[0],teta);
  p1.setXlabel("time");
  p1.setYlabel("pendulum angle");
  p1.setPlabel("Pendulum model");
  p1.plot();
 }
}
```

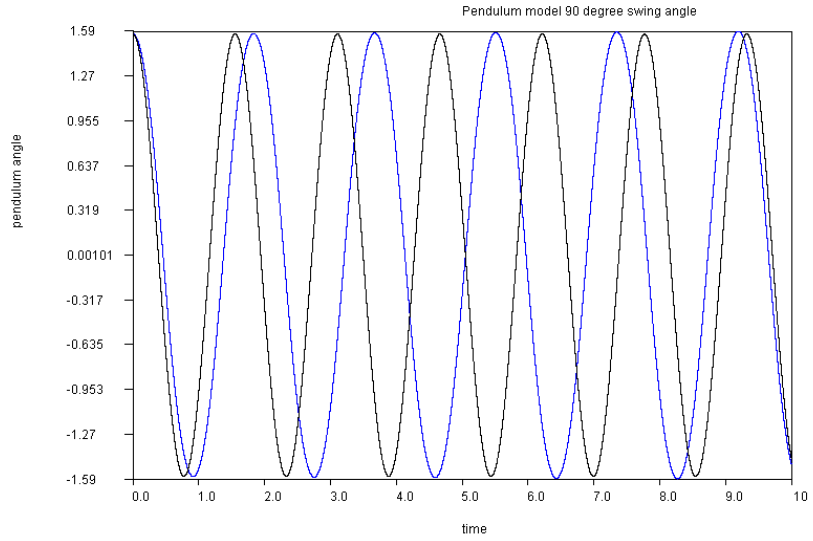
Output 8.7-11 Pendulum problem

$\theta=\pi/16$ radian, gravitational accaleration g=9.806 m/s^2, Pendulum length l= 0.6 m





Output 8.7-12 Pendulum problem $\theta=\pi/16$ radian, gravitational accaleration g=9.806 m/s^2, Pendulum length l= 0.6 m



Even with small angle two solutions differs by long period of time, but the wrong one could be the numerical solution here due to added error.

9.7 BOUNDARY PROBLEM AND SHOOTING METHODS IN DIFFERENTIAL EQUATIONS

In all differential equations solutions so far, it is asumed that initial condition (value of dependent value at satrting independent value). In the real system given boundary condition is not necassarily initial condition. Numerical differential equation solution methods requires an initial condition to be given. When a different type of boundary value is defined, following steps can be taken to solve the problem

Step 1 : Guess a value for the initial condition and solve the diferential equation for this initial condition

Step 2 : Repeat step 1 3 to 5 times. Find the value correspond to the given boundary condition (for example at the other end)

Step 3 : Make a curve fitting or interpolation or root finding between initial conditions and corresponding end result

Step 4 : By using the the curve fitting find initial condition corresponds to the given boundary condition

As an example let us investigate the following problem:

$$\frac{1}{r^2}\frac{\partial}{\partial r}\left(r^2\frac{\partial V}{\partial r}\right) \rightarrow \frac{\partial^2 V}{\partial r^2}+\frac{2}{r}\frac{\partial V}{\partial r}=0$$

Boundary conditions:

r=1 V=100

r=2 V=0

Program 8.8-1 Boundary value problem solution

```
 //=====================================================
// Numerical Analysis package in java
// example to show differential equation solution
// Runge-Kutta methods with root finding function
// Dr. Turhan Coban
// =====================================================
// dy/dx=f(x,y)
// Multiequation method with error correction
import java.io.*;
import java.util.*;
import javax.swing.*;

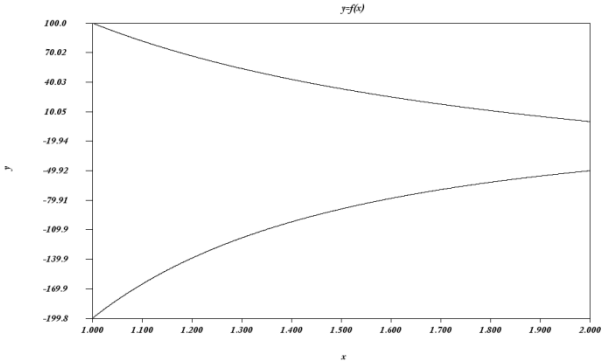
class fm1 extends f_xi
{
//multivariable function
 double func(double x[],int x_ref)
 { //x[0] is x, x[1] is y
   if(x_ref==0) return x[2];
   if(x_ref==1) return -2/x[0]*x[2];
   else return 0;
 }
}

class boundary
{   Runge_Kutta rk;
    int index_iy; //index of y value 0..number_of_equations for given b.c
    int index_ix; //index of x value which boundary condition is given
    int index_initial_y; //index of initial_y value to be calculated
    double y_value;
    double h;
    int n;
    double x0,xn;
    f_xi f;
              public boundary(f_xi fi,String eqn,int index_iyi,int index_ixi,int index_initial_yi,double y_valuei,double hi,double x0i,double xni)
    {   f=fi;
                  rk=new Runge_Kutta(eqn);
                  h=hi;
                  n=(int)((xn-x0)/h+1e-30);
```

```
    index_iy=index_iyi;
    index_ix=index_ixi;
    index_initial_y=index_initial_yi;
    y_value=y_valuei;
    x0=x0i;
    xn=xni;
   }
    public static double[] doublearray(String s)
    {
    StringTokenizer token=new StringTokenizer(s);
    int n1=token.countTokens()-1;
    int m=n1+1;
    double a[]=new double[m];
    int j=0;
    while(token.hasMoreTokens())
    {
    Double ax=new Double(token.nextToken());
    a[j++]=ax.doubleValue();
    }
    return a;
    }
  public double func(double y[],double yi)
  {   y[index_initial_y]=yi;
                double a[][]=rk.hesapla(f,x0,xn,y,h);
     double yy=a[index_iy+1][index_ix]-y_value;
     return yy;
  }
  public double[][] func(double y[])
  {   double a[][]=rk.hesapla(f,x0,xn,y,h);
     return a;
  }
  public double[][] solve_bisection(double y[],double xl,double xu)
  {     y[index_initial_y]=bisection(y,xl,xu);
       double a[][]=func(y);
       return a;
  }

public  double bisection(double y[],double xl,double xu)
{ // bisection root finding method
double test;
double xr=0;
double es,ea;
double fxl,fxr;
int maxit=200,iter=0;
es=0.001;
ea=1.1*es;
while((ea>es)&&(iter<maxit))
{   xr=(xl+xu)/2.0;
   iter++;
   if((xl+xu)!=0)
      { ea=Math.abs((xu-xl)/(xu+xl))*100;}
   fxl= func(y,xl);
   fxr= func(y,xr);
   test= fxl*fxr;
   if(test==0.0)      ea=0;
   else if(test<0.0)  xu=xr;
   else               {xl=xr;}
}
if(iter>=maxit) JOptionPane.showMessageDialog(null,"Maximum number of iteration is exceeded \n"+
  " result might not be valid","MAKSİMUM NUMBER OF ITERATION WARNING",JOptionPane.WARNING_MESSAGE);
return xr;
}
  public static void main(String args[]) throws IOException
  {
```

```
  fm1 b3=new fm1();
  double y[];
  y=new double[2];
  y[0]=100.0;
  y[1]=10;
  double x0=1.0;
  double xn=2.0;
  int index_iyi=0;
  double h=0.02;
  int index_ixi=(int)((xn-x0)/h+1e-30);
  int index_initial_yi=1;
  double y_value=0.0;
  boundary b=new boundary(b3,"RK6",index_iyi,index_ixi,index_initial_yi,y_value,h,x0,xn);
  double a[][]=b.solve_bisection(y,-300,300.0);
  Text.print(Text.T(a));
  Plot pp=new Plot(a);
  pp.plot();
  }
}
```



r	V1	V2
1	100	-197.0230103
1.02	96.05953979	-189.3723666
1.04	92.27209246	-182.1588482
1.06	88.6289155	-175.3497777
1.08	85.12191994	-168.9154752
1.1	81.74361044	-162.8289341
1.12	78.48703176	-157.0655375
1.14	75.34572101	-151.6028088
1.16	72.31366483	-146.4201919
1.18	69.38526099	-141.4988583
1.2	66.55528383	-136.8215349
1.22	63.81885313	-132.372353
1.24	61.17140607	-128.1367132
1.26	58.60867181	-124.1011654
1.28	56.1266485	-120.2533022
1.3	53.72158245	-116.5816629
1.32	51.3899492	-113.0756487
1.34	49.12843622	-109.7254457
1.36	46.93392731	-106.5219562
1.38	44.80348818	-103.4567372
1.4	42.73435344	-100.521944
1.42	40.72391456	-97.71028082
1.44	38.76970894	-95.01495479
1.46	36.86940985	-92.42963513
1.48	35.02081715	-89.94841593
1.5	33.22184883	-87.56578233
1.52	31.47053318	-85.27657992
1.54	29.76500158	-83.07598678
1.56	28.10348185	-80.9594881
1.58	26.48429208	-78.92285301
1.6	24.90583502	-76.96211338
1.62	23.36659276	-75.07354452
1.64	21.86512187	-73.25364748
1.66	20.40004892	-71.49913277
1.68	18.97006626	-69.80690556
1.7	17.57392815	-68.17405199
1.72	16.21044711	-66.59782661

1.74	14.87849058	-65.07564085
1.76	13.57697776	-63.60505238
1.78	12.30487671	-62.18375529
1.8	11.06120161	-60.80957106
1.82	9.845010187	-59.48044024
1.84	8.655401382	-58.19441465
1.86	7.491513089	-56.94965032
1.88	6.352520082	-55.74440082
1.9	5.237632066	-54.57701115
1.92	4.146091843	-53.44591207
1.94	3.077173602	-52.3496148
1.96	2.030181306	-51.28670613
1.98	1.004447183	-50.25584386
2	-6.70E-04	-49.25575256

Now let us try another one

$$\frac{\partial^2 y}{\partial x^2} = y^3 - y\frac{\partial y}{\partial x}$$

Boundary conditions:

x=1 y=0.5
x=2 y=1/3

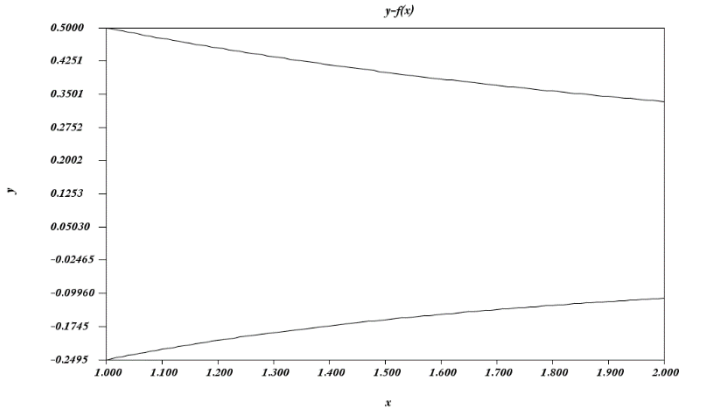
Program 8.8-2 Boundary value problem solution

```
 //=====================================================
// Numerical Analysis package in java
// example to show differential equation solution
// Runge-Kutta methods with root finding function
// Dr. Turhan Coban
// =====================================================
// dy/dx=f(x,y)
// Multiequation method with error correction
import java.io.*;
import java.util.*;
import javax.swing.*;

class fm2 extends f_xi
{
//multivariable function
 double func(double x[],int x_ref)
 { //x[0] is x, x[1] is y
   if(x_ref==0) return x[2];
   if(x_ref==1) return x[1]*x[1]*x[1]-x[1]*x[2];
   else return 0;
 }
}

class boundarytest1
{      public static void main(String args[]) throws IOException
  {
  fm2 b3=new fm2();
  double y[];
  y=new double[2];
  y[0]=0.5;
  y[1]=-1.0;
  double x0=1.0;
  double xn=2.0;
  int index_iyi=0;
  double h=0.01;
  int index_ixi=(int)((xn-x0)/h+1e-30);
  int index_initial_yi=1;
  double y_value=1.0/3.0;
  boundary b=new boundary(b3,"RK6",index_iyi,index_ixi,index_initial_yi,y_value,h,x0,xn);
  double a[][]=b.solve_bisection(y,-4.0,4.0);
  //double a[][]=b.solve_cubic_spline(y,0.0,20.0);
  Text.print(Text.T(a));
```

```
  Plot pp=new Plot(a);
  pp.plot();
  }
}
```



1	0.5	-0.249497592
1.01	0.497505024	-0.247006338
1.02	0.495034961	-0.244552198
1.03	0.492589439	-0.242134437
1.04	0.490168094	-0.23975234
1.05	0.487770571	-0.237405207
1.06	0.485396519	-0.235092355
1.07	0.483045595	-0.232813119
1.08	0.480717464	-0.23056685
1.09	0.478411796	-0.228352914
1.1	0.476128266	-0.226170693
1.11	0.47386656	-0.224019581
1.12	0.471626364	-0.221898991
1.13	0.469407374	-0.219808345
1.14	0.46720929	-0.217747083
1.15	0.46503182	-0.215714654
1.16	0.462874673	-0.213710523
1.17	0.460737568	-0.211734166
1.18	0.458620226	-0.209785071
1.19	0.456522375	-0.207862738
1.2	0.454443748	-0.205966679
1.21	0.452384081	-0.204096416
1.22	0.450343117	-0.202251482
1.23	0.448320602	-0.200431422
1.24	0.446316288	-0.198635789
1.25	0.44432993	-0.196864148
1.26	0.442361289	-0.195116071
1.27	0.440410128	-0.193391143
1.28	0.438476217	-0.191688956
1.29	0.436559327	-0.190009111
1.3	0.434659236	-0.188351217
1.31	0.432775724	-0.186714894
1.32	0.430908575	-0.185099768
1.33	0.429057577	-0.183505474
1.34	0.427222522	-0.181931655
1.35	0.425403206	-0.180377961
1.36	0.423599426	-0.178844049
1.37	0.421810986	-0.177329585
1.38	0.42003769	-0.17583424
1.39	0.418279347	-0.174357693
1.4	0.416535771	-0.172899631
1.41	0.414806774	-0.171459745
1.42	0.413092177	-0.170037734
1.43	0.411391799	-0.168633302
1.44	0.409705466	-0.167246161
1.45	0.408033005	-0.165876028
1.46	0.406374245	-0.164522624
1.47	0.404729018	-0.163185678
1.48	0.403097161	-0.161864924
1.49	0.401478512	-0.160560101
1.5	0.399872911	-0.159270953
1.51	0.398280202	-0.15799723

1.52	0.396700229	-0.156738686
1.53	0.395132843	-0.15549508
1.54	0.393577892	-0.154266177
1.55	0.39203523	-0.153051744
1.56	0.390504713	-0.151851556
1.57	0.388986197	-0.15066539
1.58	0.387479543	-0.149493028
1.59	0.385984613	-0.148334257
1.6	0.38450127	-0.147188865
1.61	0.383029382	-0.146056649
1.62	0.381568815	-0.144937406
1.63	0.380119441	-0.143830938
1.64	0.378681132	-0.142737051
1.65	0.377253761	-0.141655556
1.66	0.375837206	-0.140586264
1.67	0.374431343	-0.139528993
1.68	0.373036053	-0.138483563
1.69	0.371651217	-0.137449797
1.7	0.370276719	-0.136427522
1.71	0.368912444	-0.135416568
1.72	0.367558279	-0.134416768
1.73	0.366214111	-0.133427959
1.74	0.364879831	-0.132449979
1.75	0.363555331	-0.13148267
1.76	0.362240505	-0.130525878
1.77	0.360935246	-0.12957945
1.78	0.359639451	-0.128643237
1.79	0.358353019	-0.127717092
1.8	0.357075848	-0.126800871
1.81	0.355807839	-0.125894432
1.82	0.354548895	-0.124997636
1.83	0.353298919	-0.124110347
1.84	0.352057815	-0.12323243
1.85	0.350825491	-0.122363754
1.86	0.349601853	-0.121504188
1.87	0.348386812	-0.120653606
1.88	0.347180276	-0.119811883
1.89	0.345982157	-0.118978896
1.9	0.344792368	-0.118154524
1.91	0.343610822	-0.117338648
1.92	0.342437436	-0.116531153
1.93	0.341272124	-0.115731922
1.94	0.340114805	-0.114940843
1.95	0.338965397	-0.114157807
1.96	0.337823819	-0.113382703
1.97	0.336689992	-0.112615424
1.98	0.335563837	-0.111855867
1.99	0.334445279	-0.111103926
2	0.33333424	-0.1103595

In the third problem

$$\frac{\partial^2 y}{\partial x^2} = -x^2 + x\frac{\partial y}{\partial x} - y$$

Boundary conditions:

x=0 y=-2

x=1 y=1

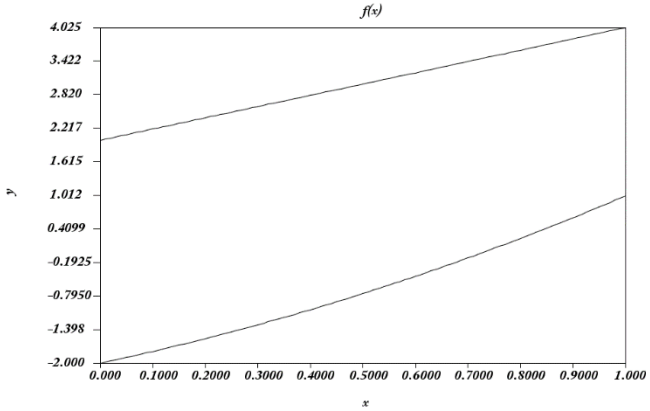
Program 8.8-3 Boundary value problem solution

```
//=====================================================
// Numerical Analysis package in java
// example to show differential equation solution
// Runge-Kutta methods with root finding function
// Dr. Turhan Coban
// =====================================================
// dy/dx=f(x,y)
// Multiequation method with error correction
import java.io.*;
import java.util.*;
import javax.swing.*;

class fm2 extends f_xi
{
//multivariable function
```

```
 double func(double x[],int x_ref)
 { //x[0] is x, x[1] is y
   if(x_ref==0) return x[2];
   if(x_ref==1) return -x[0]*x[0]+x[0]*x[2]-x[1];
   else return 0;
 }
}

class boundarytest2
{      public static void main(String args[]) throws IOException
  {
  fm2 b3=new fm2();
  double y[];
  y=new double[2];
  y[0]=-2.0;
  y[1]=1.0;
  double x0=0.0;
  double xn=1.0;
  int index_iyi=0;
  double h=0.01;
  int index_ixi=(int)((xn-x0)/h+1e-30);
  int index_initial_yi=1;
  double y_value=1.0;
  boundary b=new boundary(b3,"RK6",index_iyi,index_ixi,index_initial_yi,y_value,h,x0,xn);
  double a[][]=b.solve_bisection(y,-4.0,4.0);
  //double a[][]=b.solve_cubic_spline(y,0.0,20.0);
  Text.print(Text.T(a));
  Plot pp=new Plot(a);
  pp.plot();
  }
}
```



0	-2	2.000961
0.01	-1.97999	2.021062
0.02	-1.95978	2.041164
0.03	-1.93937	2.061269
0.04	-1.91876	2.081375
0.05	-1.89794	2.101484
0.06	-1.87693	2.121594
0.07	-1.85571	2.141707
0.08	-1.83429	2.161822
0.09	-1.81268	2.181938
0.1	-1.79086	2.202057
0.11	-1.76884	2.222178
0.12	-1.74661	2.242302
0.13	-1.72419	2.262427
0.14	-1.70157	2.282555
0.15	-1.67874	2.302684
0.16	-1.65571	2.322816
0.17	-1.63249	2.342951
0.18	-1.60906	2.363087
0.19	-1.58543	2.383226
0.2	-1.56159	2.403367

0.21	-1.53756	2.42351
0.22	-1.51332	2.443656
0.23	-1.48889	2.463804
0.24	-1.46425	2.483954
0.25	-1.43941	2.504106
0.26	-1.41437	2.524262
0.27	-1.38913	2.544419
0.28	-1.36368	2.564579
0.29	-1.33804	2.584741
0.3	-1.31219	2.604906
0.31	-1.28614	2.625074
0.32	-1.25989	2.645243
0.33	-1.23344	2.665416
0.34	-1.20678	2.685591
0.35	-1.17993	2.705769
0.36	-1.15287	2.725949
0.37	-1.12561	2.746132
0.38	-1.09815	2.766318
0.39	-1.07049	2.786506
0.4	-1.04262	2.806697
0.41	-1.01455	2.826891
0.42	-0.98628	2.847088
0.43	-0.95781	2.867288
0.44	-0.92914	2.88749
0.45	-0.90027	2.907696
0.46	-0.87119	2.927904
0.47	-0.84191	2.948115
0.48	-0.81243	2.96833
0.49	-0.78275	2.988547
0.5	-0.75286	3.008768
0.51	-0.72277	3.028991
0.52	-0.69248	3.049218
0.53	-0.66199	3.069448
0.54	-0.6313	3.089681
0.55	-0.6004	3.109918
0.56	-0.5693	3.130158
0.57	-0.538	3.150401
0.58	-0.50649	3.170648
0.59	-0.47479	3.190898
0.6	-0.44288	3.211152
0.61	-0.41077	3.231409
0.62	-0.37845	3.25167
0.63	-0.34594	3.271934
0.64	-0.31322	3.292202
0.65	-0.28029	3.312474
0.66	-0.24717	3.33275
0.67	-0.21384	3.353029
0.68	-0.18031	3.373313
0.69	-0.14658	3.3936
0.7	-0.11264	3.413892
0.71	-0.0785	3.434187
0.72	-0.04416	3.454487
0.73	-0.00962	3.474791
0.74	0.02513	3.495099
0.75	0.060081	3.515412
0.76	0.095235	3.535728
0.77	0.130593	3.55605
0.78	0.166153	3.576376
0.79	0.201917	3.596706
0.8	0.237884	3.617041
0.81	0.274054	3.637381
0.82	0.310428	3.657725
0.83	0.347005	3.678075
0.84	0.383786	3.698429
0.85	0.42077	3.718789
0.86	0.457958	3.739153
0.87	0.49535	3.759523
0.88	0.532945	3.779898
0.89	0.570744	3.800279
0.9	0.608747	3.820664
0.91	0.646954	3.841056
0.92	0.685364	3.861453
0.93	0.723979	3.881855
0.94	0.762797	3.902264
0.95	0.80182	3.922678
0.96	0.841047	3.943099
0.97	0.880478	3.963525
0.98	0.920113	3.983958
0.99	0.959952	4.004397
1	0.999996	4.024842

As the fourth problem is the Blassius flat plate problem from fluid mechanics is considered. Blassius flat plate problem is one of the basic problem that shapes heat convection as we know today. Problem based on two differential eqaution the first one is called similarity monentum equation

$$f'''(\eta) + \frac{1}{2} f(\eta) f''(\eta) = 0$$

where

$$\eta = \frac{y}{\sqrt{\frac{\nu x}{u_\infty}}}$$

And

$$f'(\eta) = \frac{u}{u_\infty}$$

Boundary conditions:

$$\eta = 0 f'(\eta) = 0$$
$$\eta \to \infty f'(\eta) = 1$$
$$\eta = 0 f''(\eta) = 0 f''(0) = 0$$

And energy equation

$$\frac{\partial^2 \theta}{\partial \eta^2} + \frac{Pr}{2} \frac{\partial \theta}{\partial \eta}$$
$$\theta(0) = 0$$
$$\theta(\infty) = 1$$

Not that Pr is Prandtl number, a parameter we can define as a constant.Equations can be solved directly as a set of differential equations. Equation set is as follows:

$$f'''(\eta) + \frac{1}{2} f(\eta) f''(\eta) = 0$$
$$\theta'' + \frac{Pr}{2}$$

Conversion of the equation:

$$x[0] = \eta$$
$$x[1] = f$$
$$x[2] = f'$$
$$x[3] = f''$$
$$x[4] = \theta$$
$$x[5] = \theta'$$

Differential equation set becomes:

$$\frac{dx[1]}{dx[0]} = x[2]$$
$$\frac{dx[2]}{dx[0]} = x[3]$$
$$\frac{dx[3]}{dx[0]} = -0.5 * x[1] * x[3]$$
$$\frac{dx[4]}{dx[0]} = x[5]$$
$$\frac{dx[4]}{dx[0]} = -\frac{Pr}{2}$$

Boundary conditions:

x[4]=0 at x[0]=0

x[4]=1 at x[0]= ∞

x[2]=0 at x[0]=0

x[2]=1 at x[0]= ∞

x[1]=0 at x[0]= 0

As it is seen from boundary conditions, two given conditions are not initial conditions. One way of solving boundary conditions is using non-linear system of equation solving methods. In order to solve differential equation set 6^{th} order Runge-Kutta Method is used. In order to solve boundary value nonlinear system of equation Nelder & Mead optimization method is used.

Program 8.8-4 Blassius flat plate problem

```
 import java.util.*;
import java.awt.*;
import java.applet.Applet;
import java.awt.event.*;
import javax.swing.*;

class f2 extends f_xj
{//adaptation function
fi_xi ff1;
public f2(fi_xi ffi) {ff1=ffi;}

public double func(double x[])
{
double ff=0.0;
double fa[]=ff1.func(x);
for(int i=0;i<fa.length;i++)  ff+=fa[i]*fa[i];
return ff;
}
}

class fa extends fi_xi
{ // Nonlinear system of equation
double y[];
fm2 b3;
double a[][];
double b[][];
public fa(double Pr,double yi[])
{ b3=new fm2(Pr);
  y=new double[yi.length];
  for(int i=0;i<y.length;i++) y[i]=yi[i];
}

public double[] func(double x[])
{
// shooting method
y[2]=x[0];
y[4]=x[1];
a=RK6(b3,0.0,6.8,y,10000);
//System.out.println(a[0][10000]+" "+a[1][10000]+" "+a[2][10000]+" "+a[3][10000]+" "+a[4][10000]+" "+a[5][10000]+" ");
double ff[]=new double[2];
ff[0]=a[2][10000]-1.0;
ff[1]=a[4][10000]-1.0;
return ff;
}
 //differenetial equation solution
public static double[][] RK6(f_xi fp,double x0,double xn,double f0[],int N)
{
//6th order Runge Kutta Method
//fp  : given set of derivative functions dfj/dxi(fj,x)
// xo : initial value of the independent variable
// xn : final value of the independent variable
// f0 : initial value of the dependent variable
// N  : number of dependent variable to be calculated
// fi : dependent variable
double h=(xn-x0)/N;
int M=f0.length;
double fi[][];
```

```
fi=new double[M][N+1];
double xi[]=new double[M+1];
double k[]=new double[6];
int i,j;
double x;
for(j=0;j<M;j++)
  {
  fi[j][0]=f0[j];
  xi[j+1]=f0[j];
  }
for(x=x0,i=0;i<N;x+=h,i++)
  {
  for(j=1;j<=M;j++)
    {
    xi[0]=x;
    xi[j]=fi[j-1][i];
    k[0]=h*fp.func(xi,j-1);
    xi[0]=x+h/2.0;
    xi[j]=fi[j-1][i]+k[0]/2;
    k[1]=h*fp.func(xi,j-1);
    xi[0]=x+h/2.0;
    xi[j]=fi[j-1][i]+k[0]/4.0+k[1]/4.0;
    k[2]=h*fp.func(xi,j-1);
    xi[0]=x+h;
    xi[j]=fi[j-1][i]-k[1]+2.0*k[2];
    k[3]=h*fp.func(xi,j-1);
    xi[0]=x+2.0/3.0*h;
    xi[j]=fi[j-1][i]+7.0/27.0*k[0]+10.0/27.0*k[1]+1.0/27.0*k[3];
    k[4]=h*fp.func(xi,j-1);
    xi[0]=x+1.0/5.0*h;
    xi[j]=fi[j-1][i]+28.0/625.0*k[0]-1.0/5.0*k[1]+546.0/625.0*k[2]+54.0/625.0*k[3]-378/625.0*k[4];
    k[5]=h*fp.func(xi,j-1);
    fi[j-1][i+1]=fi[j-1][i]+k[0]/24.0+5.0*k[3]/48.0+27.0*k[4]/56.0+125.0*k[5]/336.0;
    xi[j]=fi[j-1][i];
    }
  }
double a[][]=new double[M+1][N+1];
for(x=x0,i=0;i<=N;x+=h,i++)
{
  a[0][i]=x;
  for(j=1;j<=M;j++)
  {
  a[j][i]=fi[j-1][i];
  }
}

return a;
}
}

//Blassius flat plate differential equation
class fm2 extends f_xi
{ double Pr;
  fm2(double Pri) {Pr=Pri;}
// multivariable function
 double func(double x[],int x_ref)
 {
   //Blasius flat plate diferential equation
   //f'''+0.5*ff'' = 0
   //f'''=-0.5*x[1]*x[3] f''=x[3] f'=x[2] f=x[1]]
   double a=0;
   if(x_ref==4) a= -0.5*Pr*x[1]*x[5];
   if(x_ref==3) a= x[5];
   if(x_ref==2) a= -0.5*x[1]*x[3];
```

```
   if(x_ref==1) a= x[3];
   if(x_ref==0) a= x[2];;
   return a;
  }
}

public class blassius_flat_plate
{
                public static double[] inputdata(String s)
    {
    String s1=JOptionPane.showInputDialog(s);
    StringTokenizer token=new StringTokenizer(s1);
    int n=token.countTokens()-1;
    int m=n+1;
    double a[]=new double[m];
    int j=0;
    while(token.hasMoreTokens())
    {
    Double ax=new Double(token.nextToken());
    a[j++]=ax.doubleValue();
    }
    return a;
    }
public static String toString(double x[])
{ String s="";
int n=x.length;
  for(int k=0;k<n;k++)
  {s+=x[k]+" ";}
  return s;
}
public static double[] nelder(f_xj fnelder,double a[],double da[],int maxiteration,double tolerance,int printlist)
{
                int i,j;
double x[][]=new double[a.length+1][a.length];
double p[][]=new double[a.length+1][a.length+1];
for(i=0;i<x.length;i++)
   {for(j=0;j<x[0].length;j++)
    {if(i==j){x[i][j]=a[j]+da[j];p[i][j]=x[i][j];}
     else    {x[i][j]=a[j];p[i][j]=x[i][j]; }
    }
    p[i][j] = fnelder.func(p[i]);
   }


   // Inlet variable definitions
   // fnelder : abstract multivariable function f(x)
   // x : independent variable set of n+1 simplex elements
   // maxiteration : maximum iteration number
   // tolerance :
        int NDIMS = x.length-1;
        int NPTS = x.length;
        int FUNC = NDIMS;
        int ncalls = 0;
          ////// construct the starting simplex ///////////////////
          //double p[][]=new double[NPTS][NPTS]; // [row][col] = [whichvx][coord,FUNC]
          double z[]=new double[NDIMS];
          double best = 1E99;
          ///////////////// calculate the first function values for the simplex /////////////////

          int iter=0;

          for (iter=1; iter<maxiteration; iter++)
          {
                        ////////////  define lo, nhi, hi (low high next_to_high //////////////
```

```
int  ilo=0, ihi=0, inhi = -1; // -1 means missing
double flo = p[0][FUNC];
double fhi = flo;
double pavg,sterr;
for (i=1; i<NPTS; i++)
{
  if (p[i][FUNC] < flo)
    {flo=p[i][FUNC]; ilo=i;}
  if (p[i][FUNC] > fhi)
    {fhi=p[i][FUNC]; ihi=i;}
}
double fnhi = flo;
inhi = ilo;
for (i=0; i<NPTS; i++)
 if ((i != ihi) && (p[i][FUNC] > fnhi))
   {fnhi=p[i][FUNC]; inhi=i;}
////////// exit criteria //////////////
if ((iter % 4*NDIMS) == 0)
{
   // calculate the avarage (including maximum value)
   pavg=0;
   for(i=0;i<NPTS;i++)
     pavg+=p[i][FUNC];
   pavg/=NPTS;
   double tot=0;
   if(printlist!=0)
   {  System.out.print(iter);
      for (j=0; j<=NDIMS; j++)
        { System.out.print(p[ilo][j]+" ");}
      System.out.println("");
   }
   for(i=0;i<NPTS;i++)
   { tot=(p[i][FUNC]-pavg)*(p[i][FUNC]-pavg);}
   sterr=Math.sqrt(tot/NPTS);
   //if(sterr < tolerance)
     { for (j=0; j<NDIMS; j++)
         { z[j]=p[ilo][j];}
       //break;
     }
   best = p[ilo][FUNC];
}

///// calculate avarage without maximum value //////

double ave[] = new double[NDIMS];
for (j=0; j<NDIMS; j++)
 ave[j] = 0;
for (i=0; i<NPTS; i++)
 if (i != ihi)
   for (j=0; j<NDIMS; j++)
     ave[j] += p[i][j];
for (j=0; j<NDIMS; j++)
  ave[j] /= (NPTS-1);


//////// reflect ////////////////

double r[] = new double[NDIMS];
for (j=0; j<NDIMS; j++)
 r[j] = 2*ave[j] - p[ihi][j];
double fr = fnelder.func(r);

if ((flo <= fr) && (fr < fnhi))  // in zone: accept
{
```

```
    for (j=0; j<NDIMS; j++)
      p[ihi][j] = r[j];
    p[ihi][FUNC] = fr;
    continue;
  }

  if (fr < flo)  //// expand
  {
    double e[] = new double[NDIMS];
    for (j=0; j<NDIMS; j++)
      e[j] = 3*ave[j] - 2*p[ihi][j];
    double fe = fnelder.func(e);
    if (fe < fr)
    {
      for (j=0; j<NDIMS; j++)
        p[ihi][j] = e[j];
      p[ihi][FUNC] = fe;
      continue;
    }
    else
    {
      for (j=0; j<NDIMS; j++)
        p[ihi][j] = r[j];
      p[ihi][FUNC] = fr;
      continue;
    }
  }

  ////////////// shrink:

  if (fr < fhi)
  {
    double c[] = new double[NDIMS];
    for (j=0; j<NDIMS; j++)
      c[j] = 1.5*ave[j] - 0.5*p[ihi][j];
    double fc = fnelder.func(c);
    if (fc <= fr)
    {
      for (j=0; j<NDIMS; j++)
        p[ihi][j] = c[j];
      p[ihi][FUNC] = fc;
      continue;
    }
    else   /////// daralt
    {
      for (i=0; i<NPTS; i++)
        if (i != ilo)
        {
          for (j=0; j<NDIMS; j++)
            p[i][j] = 0.5*p[ilo][j] + 0.5*p[i][j];
          p[i][FUNC] = fnelder.func(p[i]);
        }
      continue;
    }
  }

  if (fr >= fhi)   ///
  {
    double cc[] = new double[NDIMS];
    for (j=0; j<NDIMS; j++)
      cc[j] = 0.5*ave[j] + 0.5*p[ihi][j];
    double fcc = fnelder.func(cc);
    if (fcc < fhi)
    {
```

```
                    for (j=0; j<NDIMS; j++)
                      p[ihi][j] = cc[j];
                    p[ihi][FUNC] = fcc;
                    continue;
                  }
                  else    /////////
                  {
                    for (i=0; i<NPTS; i++)
                      if (i != ilo)
                      {
                        for (j=0; j<NDIMS; j++)
                          p[i][j] = 0.5*p[ilo][j] + 0.5*p[i][j];
                        p[i][FUNC] = fnelder.func(p[i]);
                      }
                  }
                }
              }
        return z;
}
    public static double[] nelder(f_xj fnelder,double a[],double da[],double tolerance)
    {return nelder(fnelder,a,da,500,tolerance,0);}

    public static double[] nelder(f_xj fnelder,double a[],double da[])
    {return nelder(fnelder,a,da,500,1.0e-10,0);}

    public static double[] nelder(f_xj fnelder,double a[])
    {
                    double [] da=new double[a.length];
          for(int i=0;i<a.length;i++) da[i]=0.1*a[i];
          return nelder(fnelder,a,da);
    }
// Polynomial least square
  public static double[] gausswithpartialpivot(double a[][],double b[])
  { //Gauss elimination with partial pivoting
  int n=b.length;
  double x[]=new double[n];
  double carpan=0;
  double toplam=0;
  double buyuk;
  double dummy=0;
  //gauss elimination
  int i,j,k,p,ii,jj;
  for(k=0;k<(n-1);k++)
  { //partial pivoting
              p=k;
              buyuk=Math.abs(a[k][k]);
              for(ii=k+1;ii<n;ii++)
              {  dummy=Math.abs(a[ii][k]);
                if(dummy > buyuk) {buyuk=dummy;p=ii;}
              }
              if(p!=k)
              { for(jj=k;jj<n;jj++)
                { dummy=a[p][jj];
                  a[p][jj]=a[k][jj];
                  a[k][jj]=dummy;
                }
                dummy=b[p];
                b[p]=b[k];
                b[k]=dummy;
              }
              //
   for(i=k+1;i<n;i++)
   {  carpan=a[i][k]/a[k][k];
      a[i][k]=0;
```

```
      for(j=k+1;j<n;j++)
      {   a[i][j]-=carpan*a[k][j]; }
         b[i]   =b[i]   -carpan*b[k];
              }
  }
  //backward substitution
  x[n-1]=b[n-1]/a[n-1][n-1];
  for(i=n-2;i>=0;i--)
  {
    toplam=0;
    for(j=i+1;j<n;j++)
    {  toplam+=a[i][j]*x[j];}
    x[i]=(b[i]-toplam)/a[i][i];
  }
  return x;
}

public static double[] PolynomialLSQ(double xi[],double yi[],int n)
{ //Polynomial least square
int l=xi.length;
int i,j,k;
int np1=n+1;
double A[][];
A=new double[np1][np1];
double B[];
B=new double[np1];
double X[];
X=new double[np1];
for(i=0;i<n+1;i++)
  {  for(j=0;j<n+1;j++)
     {if(i==0 && j==0) A[i][j]=l;
     else  for(k=0;k<l;k++) A[i][j] += Math.pow(xi[k],(i+j));
     }
     for(k=0;k<l;k++) { if(i==0)  B[i]+= yi[k];
                  else      B[i] += Math.pow(xi[k],i)*yi[k];}
  }
X=gausswithpartialpivot(A,B);
double max=0;
for(i=0;i<n+1;i++)
 if(Math.abs(X[i]) > max) max = Math.abs(X[i]);
for(i=0;i<n+1;i++)
 if((Math.abs(X[i]/max) > 0) && (Math.abs(X[i]/max) < 1.0e-100)) X[i]=0;
return X;
}

public static double funcPolynomialLSQ(double e[],double x)
{
// this function calculates the value of
// least square curve fitting function
int n=e.length;
double ff;
if(n!=0.0)
  { ff=e[n-1];
  for(int i=n-2;i>=0;i--)
   { ff=ff*x+e[i]; }
  }
else
  ff=0;
return ff;
}

public static double error(double x[],double y[],double e[])
{
//calculates absolute square root error of a least square approach
```

```
double n=x.length;
 int k;
 double total=0;
 for(k=0;k<n;k++)
 {
 total+=(y[k]-funcPolynomialLSQ(e,x[k]))*(y[k]-funcPolynomialLSQ(e,x[k]));
 }
total=Math.sqrt(total);
return total;
}

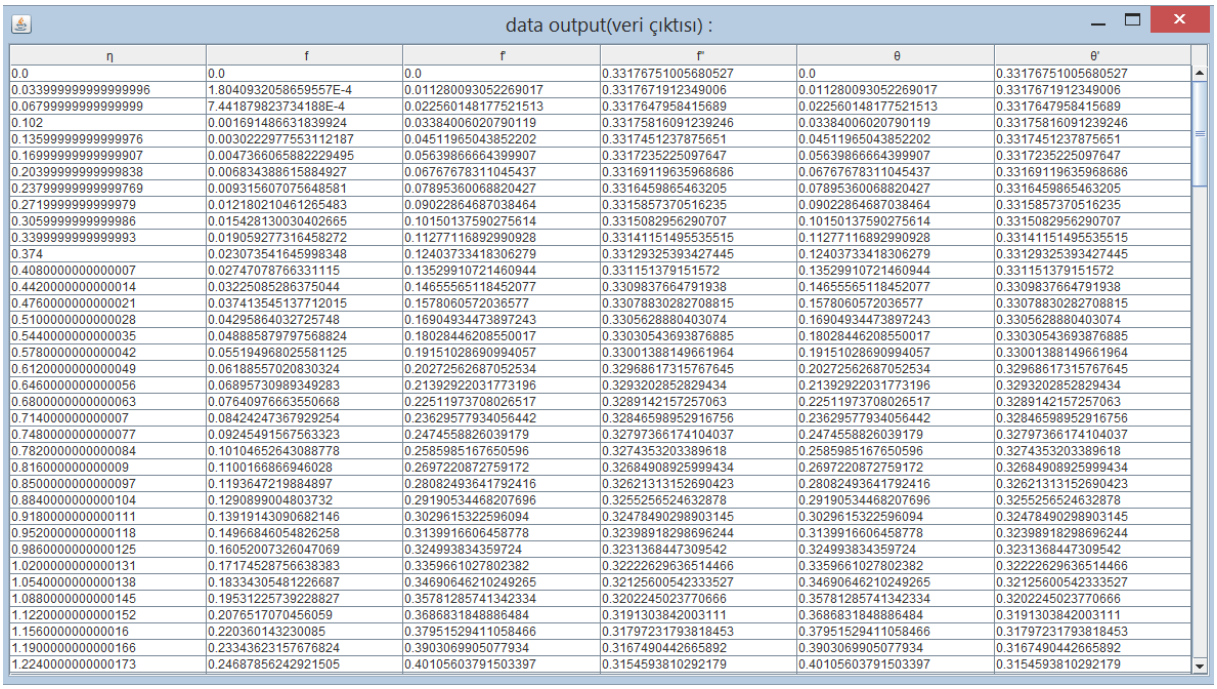
public static double[][]  funcPolynomialLSQ(double xi[],double yi[],int polinomkatsayisi,int aradegersayisi)
 {
              //aradegersayisi:  x--o--o--x--o--o--x zincirinde x deneysel noktalar ise
              // ara değer sayisi 2 dir
               int n=xi.length;
              int nn=(n-1)*(aradegersayisi+1)+1;
              double z[][]=new double[2][nn];
   double E[]=PolynomialLSQ(xi,yi,polinomkatsayisi);
   System.out.println("katsayilar :\n"+Matrix.toStringT(E));
   double dx=0;
   int k=0;
   int i;
   for(i=0;i<(n-1);i++)
      {z[0][k]=xi[i];z[1][k]=funcPolynomialLSQ(E,z[0][k]);k++;
      for(int j=0;j<aradegersayisi;j++)
      {dx=(xi[i+1]-xi[i])/((double)aradegersayisi+1.0);
      z[0][k]=z[0][k-1]+dx;z[1][k]=funcPolynomialLSQ(E,z[0][k]);k++;}
      }
      z[0][k]=xi[i];z[1][k]=funcPolynomialLSQ(E,z[0][k]);
   return z;
 }

public static void main(String arg[])
{
  double y[]={0,0,0.1,0,0.1};
 //String s="leave a space between initial root guess values: xmin0 xmin1.. ";
 double x[]={0.1,0.1};//inputdata(s);
 fa ff;
 f2 f;
 double r1[];
 double Pr[]={1.0};
 double xi[]=new double[Pr.length];
 double yi[]=new double[Pr.length];
   String s[]={""+'\u03B7',"f","f'","f'",""+'\u03B8',""+'\u03B8'+"'"};
 for(int i=0;i<xi.length;i++)
 {ff=new fa(Pr[i],y);
  f=new f2(ff);
  r1=nelder(f,x);
  xi[i]=Math.log(Pr[i]);
  yi[i]=Math.log(r1[1]);
  System.out.println("Pr = "+Pr[i]+" f'="+r1[0]+" teta'="+r1[1]);
   int nn=ff.a.length;
   int mm=ff.a[0].length;
   System.out.println("nn="+nn+"mm="+mm);
   double b[][]=new double[nn][201];
   for(int j=0;j<mm;j+=50)
   { for(int k=0;k<nn;k++)
                b[k][j/50]=ff.a[k][j];
   }
  Text.printT(b,s);
 }

 double[] b=PolynomialLSQ(xi,yi,1);
```

```
b[0]=Math.exp(b[0]);
System.out.println("Nux="+b[0]+"Rex^0.5Pr^"+b[1]);

}
}
```



data output(veri çıktısı) :

η	f	f'	f''	θ	θ'
0.0	0.0	0.0	0.33176751005680527	0.0	0.33176751005680527
0.0339999999999999996	1.8040932058659557E-4	0.011280093052269017	0.3317671912349006	0.011280093052269017	0.3317671912349006
0.06799999999999999	7.441879823734188E-4	0.022560148177521513	0.3317647958415689	0.022560148177521513	0.3317647958415689
0.102	0.001691486631839924	0.03384006020790119	0.3317581609239246	0.03384006020790119	0.3317581609239246
0.13599999999999976	0.0030222977553112187	0.04511965043852202	0.3317451237875651	0.04511965043852202	0.3317451237875651
0.16999999999999907	0.0047366065882229495	0.05639866664399907	0.3317235225097647	0.05639866664399907	0.3317235225097647
0.20399999999999838	0.006834388615884927	0.06767678311045437	0.33169119635968686	0.06767678311045437	0.33169119635968686
0.23799999999999769	0.009315607075648581	0.07895360068820427	0.3316459865463205	0.07895360068820427	0.3316459865463205
0.2719999999999979	0.012180210461265483	0.09022864687038464	0.3315857370516235	0.09022864687038464	0.3315857370516235
0.3059999999999986	0.015428130030402665	0.10150137590275614	0.3315082956290707	0.10150137590275614	0.3315082956290707
0.3399999999999993	0.019059277316458272	0.11277116892990928	0.33141151495535515	0.11277116892990928	0.33141151495535515
0.374	0.023073541645998348	0.12403733418306279	0.33129325393427445	0.12403733418306279	0.33129325393427445
0.4080000000000007	0.027470787663311115	0.13529910721460944	0.331151379151572	0.13529910721460944	0.331151379151572
0.4420000000000014	0.03225085286375044	0.14655565118452077	0.3309837664791938	0.14655565118452077	0.3309837664791938
0.4760000000000021	0.037413545137712015	0.1578060572036577	0.33078830282708815	0.1578060572036577	0.33078830282708815
0.5100000000000028	0.04295864032725748	0.16904934473897243	0.3305628880403074	0.16904934473897243	0.3305628880403074
0.5440000000000035	0.048885879797568824	0.18028446208550017	0.33030543693876885	0.18028446208550017	0.33030543693876885
0.5780000000000042	0.055194968025581125	0.19151028690994057	0.33001388149661964	0.19151028690994057	0.33001388149661964
0.6120000000000049	0.06188557020830324	0.20272562687052534	0.32968617315767645	0.20272562687052534	0.32968617315767645
0.6460000000000056	0.06895730989349283	0.21392922031773196	0.3293202852829434	0.21392922031773196	0.3293202852829434
0.6800000000000063	0.07640976663550668	0.22511973708026517	0.3289142157257063	0.22511973708026517	0.3289142157257063
0.714000000000007	0.08424247367929254	0.23629577934056442	0.32846598952916756	0.23629577934056442	0.32846598952916756
0.7480000000000077	0.09245491567563323	0.2474558826039179	0.32797366174104037	0.2474558826039179	0.32797366174104037
0.7820000000000084	0.10104652643088778	0.2585985167650596	0.3274353203389618	0.2585985167650596	0.3274353203389618
0.816000000000009	0.1100166866946028	0.2697220872759172	0.32684908925999434	0.2697220872759172	0.32684908925999434
0.8500000000000097	0.1193647219884897	0.28082493641792416	0.32621313152690423	0.28082493641792416	0.32621313152690423
0.8840000000000104	0.1290899004803732	0.29190534468207696	0.3255256524632878	0.29190534468207696	0.3255256524632878
0.9180000000000111	0.13919143090682146	0.3029615322596094	0.32478490298903145	0.3029615322596094	0.32478490298903145
0.9520000000000118	0.14966846054826258	0.3139916606458778	0.32398918298696244	0.3139916606458778	0.32398918298696244
0.9860000000000125	0.16052007326047069	0.324993834359724	0.3231368447309542	0.324993834359724	0.3231368447309542
1.0200000000000131	0.17174528756638383	0.3359661027802382	0.32222629636514466	0.3359661027802382	0.32222629636514466
1.0540000000000138	0.18334305481226687	0.34690646210249265	0.32125600542333527	0.34690646210249265	0.32125600542333527
1.0880000000000145	0.19531225739228827	0.35781285741342334	0.3202245023770666	0.35781285741342334	0.3202245023770666
1.1220000000000152	0.2076517070456059	0.3686831848886484	0.3191303842003111	0.3686831848886484	0.3191303842003111
1.156000000000016	0.220360143230085	0.37951529411058466	0.31797231793818453	0.37951529411058466	0.31797231793818453
1.1900000000000166	0.23343623157676824	0.3903069905077934	0.3167490442665892	0.3903069905077934	0.3167490442665892
1.2240000000000173	0.24687856242921505	0.40105603791503397	0.3154593810292179	0.40105603791503397	0.3154593810292179

9.8 PROBLEMS

PROBLEM 1

dy/dx = y*sin(x)* sin(x) diferansiyel equation is given. Solution is desired in the x=0 to x=1 range for h=dx=0.2 with the initial conditions of (x=0) y=2. Use fourth order Runge-Kutte method.

PROBLEM 2

$d^2y/dx^2+6\ dy/dx= \sin(\pi x)$ diferansiyel equation at x=0 to x=1 range with the boundary conditions of x=0 y=2 and
x=1 de y=-1
Solve diferential equation for h=0.02 step size.

PROBLEM 3

dy/dx= sin(πx) diferansiyel equation at x=0 to x=1 range with the boundary conditions of
x=0 y=2 and
Solve diferential equation for h=0.02 step size.

PROBLEM 4

Falkner-Scan flow equation is given as

$f''' + f*f'' + \beta(1-f'^2) = 0$

Boundary conditions of the equation

$f(0)=0$

$f'(0)=1$

$f'(\infty)=1$

$\beta = 0.3$ ve $\beta = 1.0$

Instead of infinity(∞) , take value of 8.

PROBLEM 5

Solve diferential equation
$y'' = x^2 - y^2 - 3y'$
with initial conditions y(0)=0, y'(0)=0

PROBLEM 6
Solve diferential equation
$y''' = y'^2 - 4y$
with initial conditions y(0)=0, y'(0)=1, y''(0)=0 .

PROBLEM 7
Solve diferential equation

$x' = 3x-y$
$y'=x+y$
with initial conditions x(0)=0.2, ve y(0)=0.5 .

PROBLEM 8
Solve diferential equation
$2x''(t)-5x'(t)-3x(t)=45e^{2t}$ with initial conditions x(0)=0 ve x'(0)=1 .

PROBLEM 9
Solve diferential equation
$x'(t)=x(t)-x(t)y(t)$
$y'(t)=-y(t)+x(t)y(t)$ with initial conditions x(0)=4 ve y(0)=1 at t=[0,8] range.

PROBLEM 10
Solve diferential equation
$x'(t) = \frac{2t}{1+t^2}x'(t) - \frac{2}{1+t^2}x(t) + 1$ with initial conditions x(0)=1.25 ve x(4)=-0.95 at range t=[0,4].

PROBLEM 11
Solve diferential equation
$y'' = -(y')^2 - y + \ln x$
with initial conditions: y(1)=0 y(2)=ln2 at the range 1<=x<=2

PROBLEM 12
For diferential equation
$y' - 2(1 + y^2) = 0$ with , $y(0)=0$ starting value and, h=0,25 step size find $y(1)$ değerini with Heun method

PROBLEM 13
Solve diferential equation
$\frac{dy}{dx} = yx^2 - y$ at the range x=0 ve x=2 , with initial condition y(0)=1 .

c) solve analytically from x=0 to x=2.
d) solve numerically by hand with h=0.5 step size with 4th order Runge_Kutte and calculate y(2)
e) solve numerically by computer with h=0.1 step size with 4th order Runge-Kutta
f) Plot to results and compare with analytical solution

PROBLEM 14
Solve diferential equation
$\frac{dy}{dx} = -5y + e^{-2x}$ with initial conditions : x_0=0 y_0=1 . $y(1)$, and with step size h=0.2 by using Simpson 1/3 rule.

PROBLEM 15
Solve diferential equation

$\frac{dy}{dx} = \frac{1}{x+y}$ with initial conditions x_0=0 , y_0=2 , at the interval 0-0.6 with step size h=0.2,

a-) with 4th order Runge-Kutta method

b-) with Euler method

PROBLEM 16

Cooling rate of a substance are given with the following equation

$$\frac{dT}{dt} = -k(T - T_a)$$

In the equation T indicates the temperature of the substance in (°C), T_a is environmental temperature (°C) ,k is a constant (minute^{-1}) dir. Therefore cooling rate is function of temperature difference between body and environment. If a metal ball initially at 90 °C drop into a water bath at T_a=20 °C and if k=0.1 minute^{-1}, Find the time period required ball to reach 30 °C temperature

a) By analytical methods

b) By a numerical method

PROBLEM 17

Solution of diferential equation

$\frac{dy}{dx} = yx^2 - y$ are investigated with range x=0 ve x=2. Initial condition is given as y(0)=1

a) Solve it anaytically Plot the solution.

b) Solve it with Heun method with step size h=0.5 adim

c) Plot the numerical results also and compare it with analytical results

PROBLEM 18

In a chemical reaction a chemical A and a chemical B reacts to give chemical C. Concentration of chemical C as a function of time, y(t) is given with the following equation

$y' = k(a - y)(b - y)$ $y(0)=0$

In this equation k is a positive constant, a and b is the initial concentration of chemicals A and B.

Assume that

k=0.01 , a= 70 milimole/liter , b= 50 milimole/liter .

By using 4th order Runge-Kutta method with step size h=0.5 estimate concentration at t=1 second.

PROBLEM 19

Solve differential equation

$\frac{dy}{dx} = yx^2 - y$

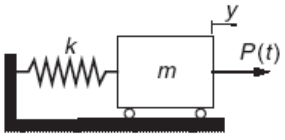
With range of x=0 ve x=2. Initial value is given as y(0)=1 .

a) Solve the problem analytically

b) Solve it with Heun method, step size h=0.5

c) Plot both results and compare relative error.

PROBLEM 20



In the spring-mass system shown in the figure force P(t) is equal to

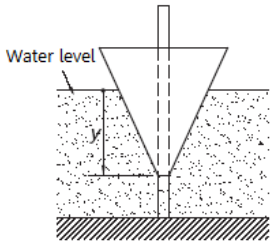
$$\boldsymbol{P(t) = \begin{cases} 10tNt < 2s \\ 20Nt \geq 2s \end{cases}}$$

Diferential equation of the movement

$\frac{d^2y}{dt^2} = \frac{P(t)}{m} - \frac{k}{m}y$ with m=2.5 kg, k=75 N/m. Determine amplitude (maximum amount of movement) of the movement.

PROBLEM 21 A cone buoy can slide freely on a bar is floating on the surface. If cone is disturbed by any effect, it will oscilate. The diferential equation of the oscilation is given as:

$\frac{d^2y}{dt^2} = g(1 - ay^3)$ If a=16 m^{-3}, g=9.80665 m/s^2 and cone position initially moved upward to y=0.1 m and released, find the period(s) and amplitude(m) of the oscilation.



PROBLEM 23

Gun problem with air friction was given in the examples. Solve the same problem with 6th order Runge-Kutta method:

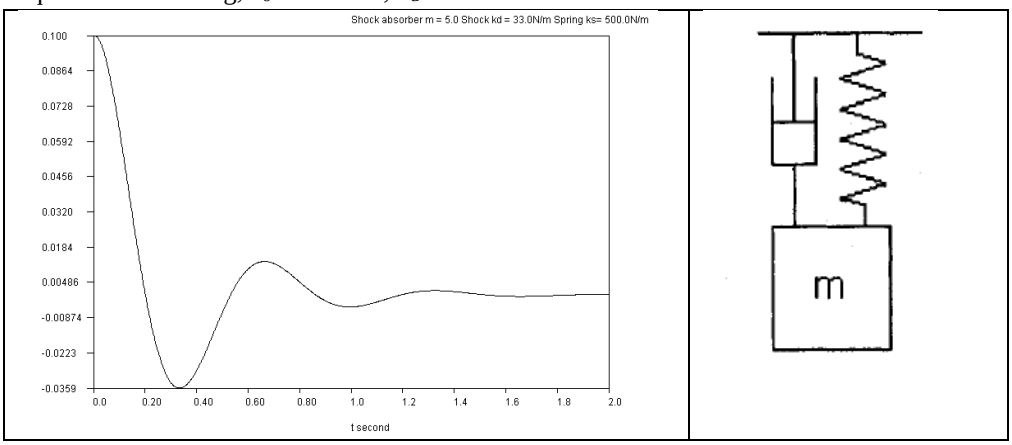
Nozle exit velocity 250 m/s, gravitational acceleration 9.806 m/s^2, friction coefficient γ = (k / m)= 0.01. A target at 750 m distance will be hit. Determine the angle θ.

PROBLEM 24

In cars and other machines for vibration control mechanical system with a spring and a shock absorber is used. The differential equation of such a system is given as:

$$m\frac{d^2x}{dt^2} + k_d\frac{dx}{dt} + k_s x = 0$$

In this equation m is mass, k_d shock absorber coefficient, k_s is spring coefficient. Solve the diferential equation for m=5 kg, k_s =500 N/m, k_d =33 N/m values.

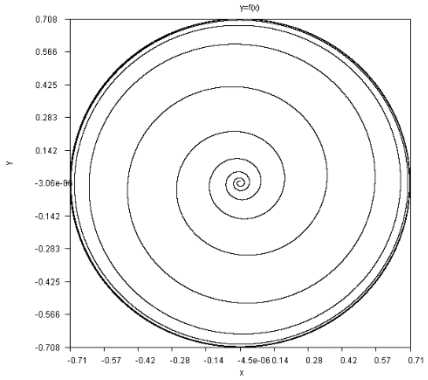


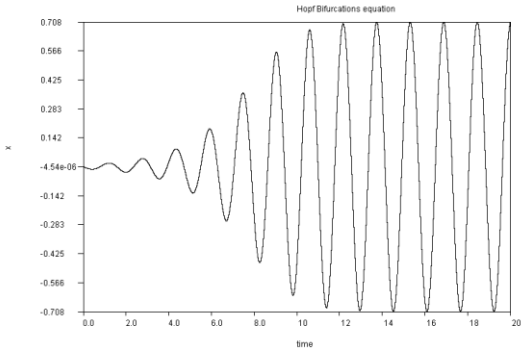
PROBLEM 25

Hopf Bifurcations equation is given by

$$\frac{dx}{dt} = -(g + x^2 + y^2)x - wy$$
$$\frac{dy}{dt} = -(g + x^2 + y^2)y + wx$$

Solve the set of differential equations with g=-0.5,w=4, with the initial conditions x=0,y=0.01 at t=0. T_{final}=20 s





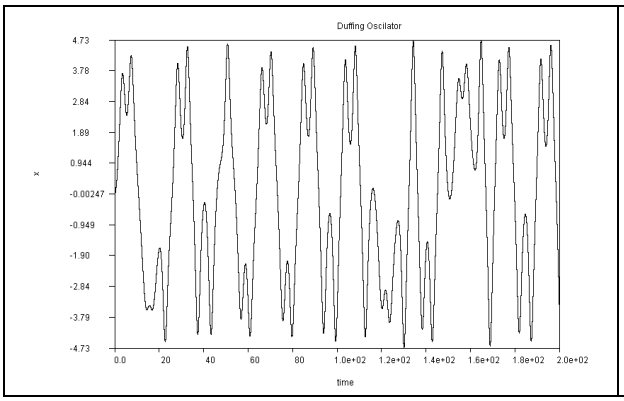
PROBLEM 26

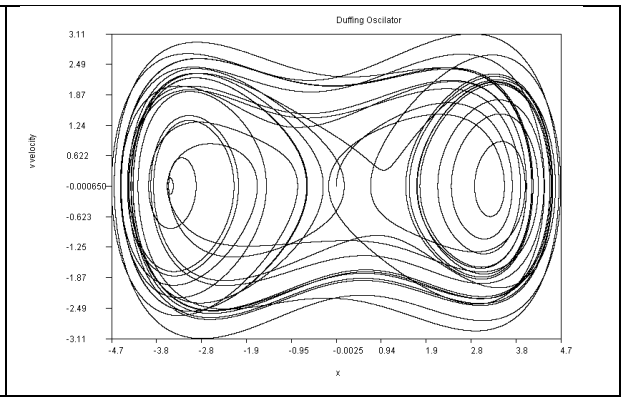
The Duffing Oscilator differential equations are given as

$$\frac{dx}{dt} = v$$

$$\frac{d^2x}{dt^2} = \frac{dv}{dx} = -0.2v + x - 0.1x^3 + cos(t)$$

Solve the differential equation set with the initial conditions of x=v=0 at t=0 for 0 to t=200 s period.



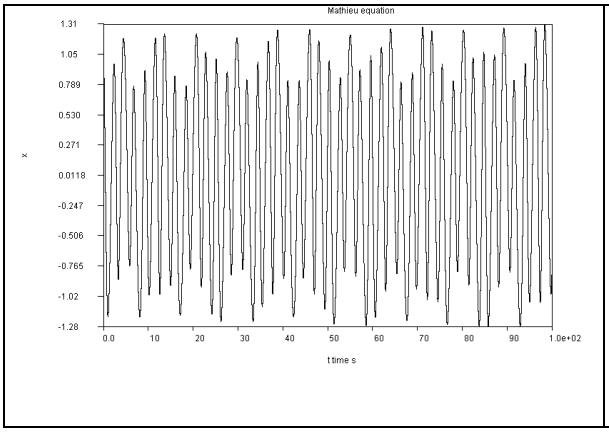


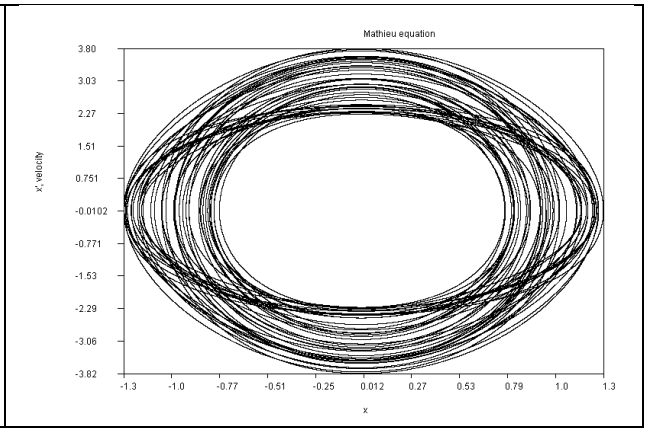
PROBLEM 27

The Mathieu equation is given as

$$\frac{d^2x}{dt^2} + (\delta + 2\varepsilon\, cos(2t))x = 0$$

Solve the differential equation with initial conditions δ=8, ε=2.5, x=0,x'=0 for the time period of 0<=t<=100 s.



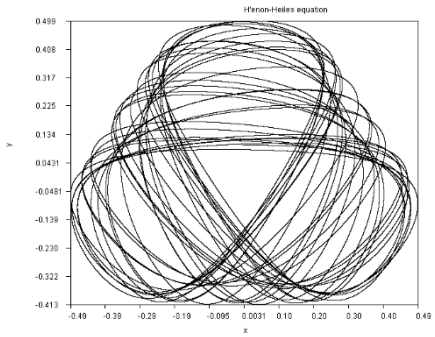


PROBLEM 28

H'enon-Heiles equation is given as

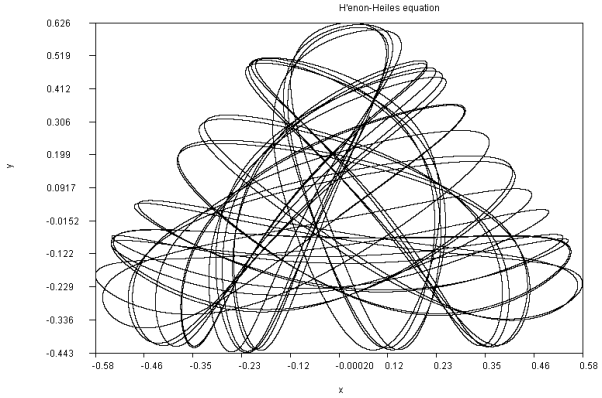
$$\frac{dx}{dt} = P_x$$
$$\frac{dy}{dt} = P_y$$
$$\frac{dP_x}{dt} = -x - 2xy$$
$$\frac{dP_y}{dt} = -y - x^2 + y^2$$

Solve the differential equation with initial conditions x=0,y=0.1 and P_x=0.49057, P_y=0 for the time period of 0<=t<=200 s.



PROBLEM 29

H'enon-Heiles equation is described in Problem 28. Solve the equation with the initial conditions of x=0,y=0 and P_x= 0.35355, P_y= 0.35355 for the time period of 0<=t<=200 s.



PROBLEM 30

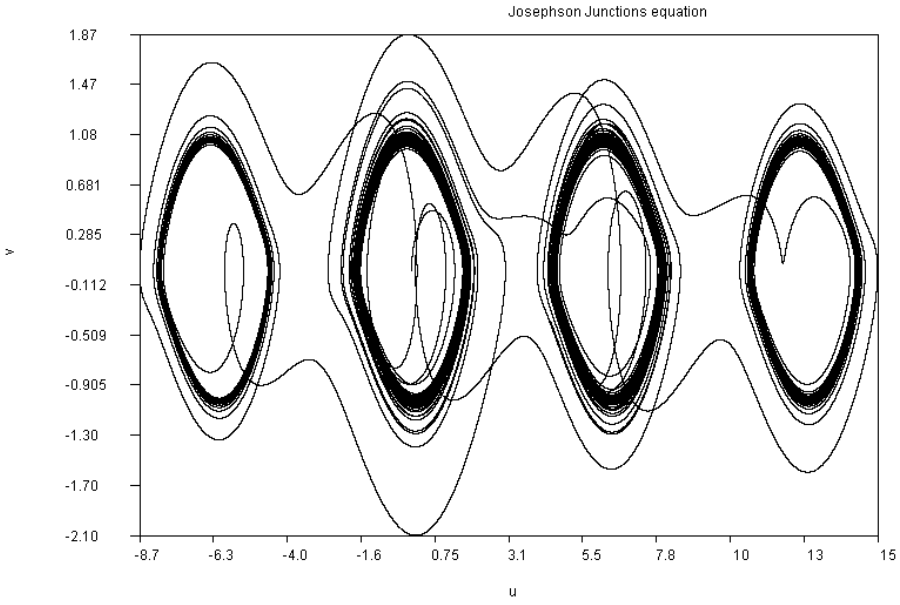
Intermittency in Josephson Junctions: The study of the chaotic behavior of (quantum) Josephson junctions is of much fundamental and even practical interest. Written in dimensionless form, the differential equation for the quantum phase difference, φ, across the junction is given by

$$\frac{d^2\varphi}{dt^2} + \frac{1}{\sqrt{\beta_c}}\frac{d\varphi}{dt} + \sin(\varphi) = A\,sin(\Omega\, t)$$

where β_c is the so-called McCumber parameter and Ω is the (normalized) angular frequency of the driving current. $u = v = \varphi$ and $v = \frac{d\varphi}{dt}$, this can be written as

$$\frac{du}{dt} = v$$
$$\frac{dv}{dt} = \text{Asin}(\Omega t) \text{ -r}v\text{- sin}(u)$$

For differential equation solution parametric values will be taken as A=0.9045, r=0.5, Ω=0.47. Solve the differential equation for the initial conditions u=v=0 for 0<=t<=1200 s



PROBLEM 31

Planar motion of a three body celestial system can be expressed by the following equations:

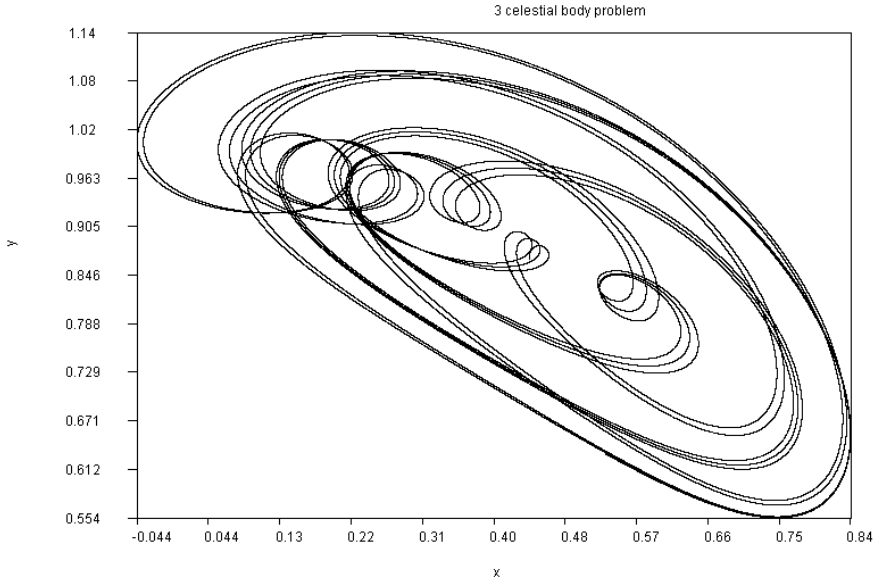
$$\frac{dx}{dt} = p + y$$

$$\frac{dy}{dt} = q - x$$

$$\frac{dp}{dt} = q - \frac{m(x+n)}{[(x+n)^2 + y^2]^{3/2}} - \frac{n(x-m)}{[(x+n)^2 + y^2]^{3/2}}$$

$$\frac{dq}{dt} = -p - \frac{my}{[(x+n)^2 + y^2]^{3/2}} - \frac{ny}{[(x+n)^2 + y^2]^{3/2}}$$

where the p and q are the canonical momenta of the coordinates x and y,
respectively. There are five equilibrium positions, called Lagrangian positions. One of the Lagrangian position is , L4, located at $(x_{average}, y_{average}) = (0.48, 0.866)$. In order to solve the diferential equation of the motion initial starting point will be taken as: (x,y,p,q)={0.44,0.866,-0.866,0.48} as initial conditions. Solve the differential equations for 0<t<200. Take m=0.98 n=0.02



PROBLEM 32 For Falkner–Skan solution transformation for fluid laminar free convection, we get the following variables for describing expressions for the stream function ψ and dimensionless coordinate variable η:

$\eta = \left(\frac{1}{4}Gr\right)^{1/4}\frac{y}{x}$ $\psi = 4\nu\left(\frac{1}{4}Gr\right)^{1/4} f(\eta)$ *where*

$$Gr = \frac{g(\rho_\infty/\rho_w - 1)x^3}{\nu^2}$$

Velocity profile for the free convection boundary layer:

$w_x = \frac{\partial\psi}{dy} = \frac{4\nu}{x}\left(\frac{1}{4}Gr\right)^{1/2} f'(\eta)$ $w_y = -\frac{\partial\psi}{dx} = \frac{\nu}{x}\left(\frac{1}{4}Gr\right)^{1/4}[\eta f'(\eta) - 3f(\eta)]$

Where x is vertical dimension, y is horizontal dimension, Gr is Grashoff number, w_x and w_y are x and y are dimension velocities. ψ is the stream function.

$\theta(\eta) = \frac{t-t_\infty}{t_w-t_\infty}$is the dimensionless temperature**.** Under these condition falkner-skan solution is converted to the following differential equation set:

$$f'''(\eta) + 3f(\eta)f''(\eta) - 2(f'(\eta))^2 + \theta(\eta) = 0$$
$$\theta''(\eta) + 3\,Pr\,f\,(\eta)\theta'(\eta) = 0$$

Where Pr is the Prandtl number. *The boundary conditions:*

$$\eta = 0: f(\eta) = f'(\eta) = 0\theta(\eta) = 1$$
$$\eta \to \infty: f'(\eta) = 0\theta(\eta) = 0$$

Solve the differential equation set.

PROBLEM 33 A compound pendulum consisting of a shaft with a center of mass at point G is shown in the

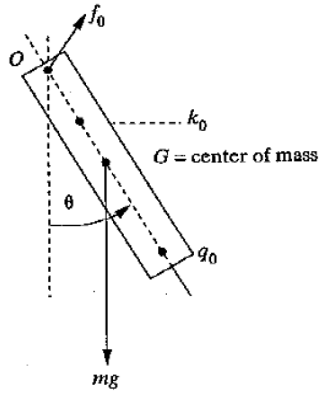


figure. For such a system diferential equation can be given as $J\frac{d^2\theta}{dt^2} + mgr\,sin(\theta) = 0$. In this equation J is mass momentum of inertia and for the rod given $J = \frac{1}{3}ml^2$and r is the center of gravity so $r = \frac{l}{2}$Differential equation becomes $\frac{d^2\theta}{dt^2} + \frac{3mg}{2l}sin(\theta) = 0$. Note that for small values of initial θvalue $sin(\theta)$ approximates to θso differential equation can be solved analytically. Approximate analytical solution is: $\theta(t) = \theta_0\,cos(\left[\sqrt{\frac{3mg}{2l}}\right]t)$ Solve differential equation numerically and compare the results for big and small initial value of θ

PROBLEM 34 : Solve boundary value equation :

$\frac{d^2y}{dt^2} = -\left(\frac{dy}{dt}\right)^2 - y + ln(x)$ x=1 y=0 and x=2 y=ln(2)

EIGENVALUE AND EIGENVECTOR CALCULATIONS

9.1 INTRODUCTION TO EIGENVALUES

Non-zero solution of **[A]{X} = 0** equation is existed, but not necessarily a single solution. A set of {X} vectors can be existed for any given matrix [A]. The equation can also be written as follows for the unit matrix I

[A]{X} = [I]{λ} {X} = 0

[A- λI]{X}= 0

In this equation **λ** is called eigen-value and **{X}** is called eigen-vector. If the equation is written in the open format:

$$\begin{bmatrix} a_{11}-\lambda & a_{12} & a_{13} & \dots & a_{1n} \\ a_{11} & a_{22}-\lambda & a_{23} & \dots & a_{2n} \\ a_{11} & a_{11} & a_{33}-\lambda & \dots & a_{3n} \\ \dots & \dots & \dots & \dots. & \dots. \\ a_{1n} & a_{2n} & a_{3n} & \dots & a_{nn}-\lambda \end{bmatrix} \begin{bmatrix} X_1 \\ X_2 \\ X_3 \\ \dots \\ X_5 \end{bmatrix} = 0$$

In this chapter, solution methods of eigenvalues and eigenvectors will be investigated. It should be note that eigenvalues could be complex numbers as well.

9.2 DOMINANT EIGENVALUE CALCULATION (POWER METHOD)

Assume that A is n x n matrix with eigenvalues $\lambda_1, \lambda_2, \lambda_3,.., \lambda_n$ with an associated collection of linearly independent eigenvectors $x^1, x^2, x^3,\dots, x^n$. It is also assumed that A has one eigenvalue λ_1 which is largest in magitude so that

$|\lambda_1| > |\lambda_2| > |\lambda_3| > |\lambda_4| \dots > |\lambda_{n-1}| > |\lambda_n|$

It shoud be noted that a matrix A does not have to have linealy independent eigenvectors. When it does not, convergence of the power method may fail. Eigen value with the maximum absolute value is called dominant eigen value. If an eigenvalue is dominant it should be either a real value or it should have an eigen vector consist of real values. If x^k is the eigenvector corresponds to eigenvalue λ_k , The following equations can be written.

$A\,x^k = \lambda_k x^k$, $1 <= k < n$.

Where $\{ x^1 , x^2 , x^3 , x^4 ,\dots, x^n \}$ is eigenvector with n components. If A is a symmetric matrix, with n linearly independent eigenvalues. If y^0 is any vector the fact that $\{ x^1 , x^2 , x^3 , x^4 ,\dots, x^n \}$ is linearly independent implies that constants c_k exist with

$$y^0 = \sum_{k=1}^{n} c_k x^k$$

If a series of y^k vectors ae constructed

$$y^{k+1} = Ay^k k \geq 0$$

$$y^1 = A\left(\sum_{k=1}^{n} c_k x^k\right) = \sum_{k=1}^{n} c_k A x^k = \sum_{k=1}^{n} c_k \lambda_k x^k$$

Generalizing the equation will yield

$$y^i = \sum_{k=1}^{n} c_k \lambda_k^i x^k \, , i \geq 0$$

remembering that λ_1 is the eigenvalue with the largest magnitude. If λ_1^i is taken out of the paranthesis, the equation becomes

$$y^i = \lambda_1^i \left[c_1 x^1 + \sum_{k=2}^{n} c_k \left(\frac{\lambda_k}{\lambda_1}\right)^i x^k \right], i \geq 0$$

It is known tht $|\lambda_k/\lambda_1| < 1$. Therefore when $i \to \propto |\lambda_k/\lambda_1|^i \to 0$.

$$y^i \approx \lambda_1^i c_1 x^1, i >> 1$$

If vector y is normalised :

$$y^{k+1} = \frac{Ay^k}{\|Ay^k\|}$$

Vector y became the eigenvector of the dominant eigenvalue. When the eigenvector is known, eigenvalue can be calculated easily.

$$Ax = \lambda x$$

By multilying both side of te equation with the transpose matrix x^T

$x^T Ax = \lambda x^T x$ and

$$\lambda = \frac{x^T Ax}{x^T x}$$

When k is is large, it is known that y^k is approximating to the dominant eigenvalue . In this case $\mu = \frac{(y^k)^T A y^k}{(y^k)^T y^k}, k \geq 0$

It can be written that with interpolation process, μ approximate to the dominant eigenvalue. Deviation of the μ from the eigen value is

$$r^k \cong (\mu_k I - A) y^k$$

In this equation when $r^k = 0$ $\mu_k$becomes eigenvalue and y^k becomes eigenvector. In other words if $\|r_k\|$is small iteration process can be finalize. In the following program computer code from the power method is given

Program 9.2-1 Power method to calculate the dominant eigenvalue

```
import java.io.*;
import javax.swing.*;

class NA131
{
  //===========Eigen value calculations ==============
//*************norm methods definition*****************
public static double norm(double v[])
{
// vector norm
double total=0;;
for(int i=0;i<v.length;i++)
  {total+=v[i]*v[i];}
return Math.sqrt(total);
}

  public static double norm(double v[],int p)
  {
   //p vector norm
  if(p==0) p=1;
  double total=0;
  double x,y;
  for(int i=0;i<v.length;i++)
    {
    x=Math.abs(v[i]);
    total+=Math.pow(x,p);
    }
  return Math.pow(total,(1.0/p));
  }
  public static double norminf(double v[])
    {
     //infinite vector norm
    double x;
```

```
 double max=0;
 for(int i=0;i<v.length;i++)
  {
  x=Math.abs(v[i]);
  if(x>max) max=x;
  }
 return max;
 }
public static double norminf(double a[][])
    {
     //infinite matrix norm
     double x;
     double max = 0;
     double total;
    int i,j;

    int n=a.length;
    for(i=0;i<n;i++)
     {
     total=0;
     for(j=0;j<n;j++)
     {total+=Math.abs(a[i][j]);}
     x=total;
     if(x>max) max=x;
     }
    return max;
    }
public static double norm(double a[][])
   {
   //matrix norm
          double x;
          double max = 0;
          double total;
          int i,j;
          int n=a.length;
          for(j=0;j<n;i++)
            {
            total=0;
            for(i=0;i<n;j++)
            {total+=Math.abs(a[i][j]);}
            x=total;
            if(x>max) max=x;
            }
          return max;
  }
public static double normE(double a[][])
        {
        //Euclidian matrix norm
        double x;
        double total;
        int i,j;
        total=0;
        int n=a.length;
        for(j=0;j<n;i++)
        {for(i=0;i<n;j++) {total+=a[i][j]*a[i][j];}}
        return Math.sqrt(total);
       }
public static double[] multiply(double[][] left,double[] right)
{
//multiplication of one matrix with one vector
int ii,jj,i,j,k;
int m1=left[0].length;
int n1=left.length;
int m2=right.length;
```

```
 double[] b;
 b=new double[m2];
 if(n1 != m2) {
 System.out.println("inner matrix dimensions must agree");
 for(ii=0;ii<n1;ii++)
  {b[ii]=0;}
  return b;   }
for(i=0;i<m1;i++)
  { b[i]=0;
    for(k=0;k<n1;k++)
              b[i]+=left[i][k]*right[k];}
return b;
//end of multiply of a matrix and a vector
}

 public static double  YT_X(double [] left,double [] right)
 {
 //multiplys a vector transpose with a vector
 int n=left.length;
 double tot=0;
 for(int i=0;i<n;i++)
 {
 tot+=left[i]*right[i];
 }
 return tot;
 }

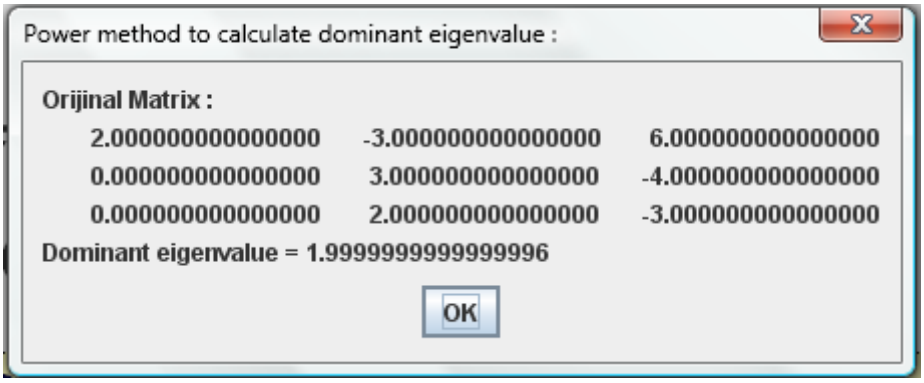
 public static double eigen_power(double A[][])
 {
  double eps=1e-15;
  double err=1;
  int m=100;
  int n=A.length;
  double y[]=new double[n];
  double r[]=new double[n];
  double xnorm;
  double mu=1.0;
  for(int i=0;i<n;i++) y[i]=1.0;
  int k=0;
  double mueski;
  double x[]=new double[n];
  do{
                mueski=mu;
                x=multiply(A,y);
                xnorm=norminf(x);
                for(int i=0;i<n;i++) {y[i]=x[i]/xnorm;}
                mu=YT_X(y,x)/YT_X(y,y);
                for(int i=0;i<n;i++) {r[i]=mu*y[i]-x[i];}
                k++;
                err=Math.abs((mu-mueski)/mu);
    }while(err>eps);
  return mu;
 }



 public static void main(String args[]) throws IOException
 {
 //Eigen Value calculations using  method
 //double a[][]={{3.556,-1.778,0},{-1.778,3.556,-1.778},{0,-1.778,3.556}};
 //double a[][]={{-3.9,-0.6,-3.9,0.4},{1,0,0,0},{0,1,0,0},{0,0,1,0}};
 double a[][]={{2,-3,6},{0,3,-4},{0,2,-3}};
 String s="Orijinal Matrix : \n";
 s+=Matrix.toString(a);
```

```
  s+="Dominant eigenvalue = "+eigen_power(a);
  JOptionPane.showMessageDialog(null,s,
    "Power method to calculate dominant eigenvalue : ",JOptionPane.PLAIN_MESSAGE);
  }
}
```

Output 9.2-1 Power method to calculate dominant eigenvalue



Example : Find the biggest eigenvalue of the following matrix by using power method

$$\boldsymbol{A} = \begin{bmatrix} \mathbf{1} & \mathbf{2} \\ \mathbf{2} & \mathbf{5} \end{bmatrix}$$

Steps of process is shown in spreadheet solution. Starting vector is taken as y={1,1}

1	2	1	3	3
2	5	1	7	7
				7
1	2	0.428571	2.428571	2.428571
2	5	1	5.857143	5.857143
				5.857143
1	2	0.414634	2.414634	2.414634
2	5	1	5.829268	5.829268
				5.829268
1	2	0.414226	2.414226	2.414226
2	5	1	5.828452	5.828452
				5.828452
1	2	0.414214	2.414214	2.414214
2	5	1	5.828428	5.828428
				5.828428
1	2	0.414214	2.414214	2.414214
2	5	1	5.828427	5.828427
				5.828427

λ= 5.828427

Example : Here is a second example:

$$\boldsymbol{A} = \begin{bmatrix} \mathbf{10} & \mathbf{4} & \mathbf{-6} \\ \mathbf{4} & \mathbf{-6} & \mathbf{8} \\ \mathbf{-6} & \mathbf{8} & \mathbf{14} \end{bmatrix}$$

Find the Biggest eigenvalue by power method. Starting vector is again taken as y={1,1,1}

3.556	-1.778	0	1	1.778	1.778

-1.778	3.556	-1.778	1	0	0	
0	-1.778	3.556	1	1.778	1.778	
					1.778	
3.556	-1.778	0	1	3.556	3.556	
-1.778	3.556	-1.778	0	-3.556	3.556	
0	-1.778	3.556	1	3.556	3.556	
				3.556	3.556	1
3.556	-1.778	0	1	5.334	5.334	
-1.778	3.556	-1.778	-1	-7.112	7.112	
0	-1.778	3.556	1	5.334	5.334	
				5.334	7.112	1
3.556	-1.778	0	0.75	4.445	4.445	
-1.778	3.556	-1.778	-1	-6.223	6.223	
0	-1.778	3.556	0.75	4.445	4.445	
				4.445	6.223	-0.125
3.556	-1.778	0	0.714286	4.318	4.318	
-1.778	3.556	-1.778	-1	-6.096	6.096	
0	-1.778	3.556	0.714286	4.318	4.318	
				4.318	6.096	-0.020408163
3.556	-1.778	0	0.708333	4.296833	4.296833	
-1.778	3.556	-1.778	-1	-6.07483	6.074833	
0	-1.778	3.556	0.708333	4.296833	4.296833	
				4.296833	6.074833	-0.003472222
3.556	-1.778	0	0.707317	4.29322	4.29322	
-1.778	3.556	-1.778	-1	-6.07122	6.07122	
0	-1.778	3.556	0.707317	4.29322	4.29322	
				4.29322	6.07122	-0.000594884
3.556	-1.778	0	0.707143	4.2926	4.2926	
-1.778	3.556	-1.778	-1	-6.0706	6.0706	
0	-1.778	3.556	0.707143	4.2926	4.2926	
				4.2926	6.0706	-0.000102041
3.556	-1.778	0	0.707113	4.292494	4.292494	
-1.778	3.556	-1.778	-1	-6.07049	6.070494	
0	-1.778	3.556	0.707113	4.292494	4.292494	
				4.292494	6.070494	-1.75067E-05
3.556	-1.778	0	0.707108	4.292475	4.292475	
-1.778	3.556	-1.778	-1	-6.07048	6.070475	
0	-1.778	3.556	0.707108	4.292475	4.292475	
				4.292475	6.070475	-3.00365E-06
3.556	-1.778	0	0.707107	4.292472	4.292472	
-1.778	3.556	-1.778	-1	-6.07047	6.070472	
0	-1.778	3.556	0.707107	4.292472	4.292472	
				4.292472	6.070472	-5.15345E-07
3.556	-1.778	0	0.707107	4.292472	4.292472	
-1.778	3.556	-1.778	-1	-6.07047	6.070472	
0	-1.778	3.556	0.707107	4.292472	4.292472	
				4.292472	6.070472	-8.84191E-08

λ= 6.070472

9.3 SMALLEST MAGNITUDE EIGENVALUE(INVERSE POWER METHOD)

If Power method for dominant eigenvalue problem applied in inverse way, smallest magnitude eigenvalue can be obtained.

$$y^k = \frac{A^{-1}y^{k-1}}{\|A^{-1}y^{k-1}\|}$$

$$\mu_k = \frac{(y^k)^T A^{-1} y^k}{(y^k)^T y^k}$$

When k is large $1/\mu_k$ becomes eigenvalue.

Program 9.3-1 Power method to calculate smallest magnitude eigenvalue

```
import java.io.*;
import javax.swing.*;


class NA132
{
  //===========Eigen value calculations ==============
public static double norm(double v[])
{
// vector norm
double total=0;;
for(int i=0;i<v.length;i++)
  {total+=v[i]*v[i];}
return Math.sqrt(total);
}

  public static double norm(double v[],int p)
  {
   //p vector norm
  if(p==0) p=1;
  double total=0;
  double x,y;
  for(int i=0;i<v.length;i++)
    {
    x=Math.abs(v[i]);
    total+=Math.pow(x,p);
    }
  return Math.pow(total,(1.0/p));
  }
  public static double norminf(double v[])
    {
     //infinite vector norm
    double x;
    double max=0;
    for(int i=0;i<v.length;i++)
      {
      x=Math.abs(v[i]);
      if(x>max) max=x;
      }
    return max;
    }
  public static double norminf(double a[][])
       {
         //infinite matrix norm
         double x;
         double max = 0;
         double total;
       int i,j;

       int n=a.length;
       for(i=0;i<n;i++)
         {
         total=0;
         for(j=0;j<n;j++)
         {total+=Math.abs(a[i][j]);}
         x=total;
         if(x>max) max=x;
         }
```

```
      return max;
     }
 public static double norm(double a[][])
    {
    //matrix norm
            double x;
            double max = 0;
            double total;
            int i,j;
            int n=a.length;
            for(j=0;j<n;i++)
              {
              total=0;
              for(i=0;i<n;j++)
              {total+=Math.abs(a[i][j]);}
              x=total;
              if(x>max) max=x;
              }
            return max;
   }
 public static double normE(double a[][])
          {
          //Euclidian matrix norm
          double x;
          double total;
          int i,j;
          total=0;
          int n=a.length;
          for(j=0;j<n;i++)
          {for(i=0;i<n;j++) {total+=a[i][j]*a[i][j];}}
          return Math.sqrt(total);
         }
 public static double[] multiply(double[][] left,double[] right)
 {
 //multiplication of one matrix with one vector
 int ii,jj,i,j,k;
 int m1=left[0].length;
 int n1=left.length;
 int m2=right.length;
 double[] b;
 b=new double[m2];
 if(n1 != m2) {
 System.out.println("inner matrix dimensions must agree");
 for(ii=0;ii<n1;ii++)
  {b[ii]=0;}
  return b;   }
for(i=0;i<m1;i++)
  { b[i]=0;
    for(k=0;k<n1;k++)
              b[i]+=left[i][k]*right[k];}
return b;
//end of multiply of a matrix and a vector
}

 public static double  YT_X(double [] left,double [] right)
 {
 //multiplys a vector transpose with a vector
 int n=left.length;
 double tot=0;
 for(int i=0;i<n;i++)
 {
 tot+=left[i]*right[i];
 }
 return tot;
```

```
 }

public static double eigen_power(double A[][])
{
 double eps=1e-15;
 double err=1;
 int m=100;
 int n=A.length;
 double y[]=new double[n];
 double r[]=new double[n];
 double xnorm;
 double mu=1.0;
 for(int i=0;i<n;i++) y[i]=1.0;
 int k=0;
 double mueski;
 double x[]=new double[n];
 do{
                mueski=mu;
                x=multiply(A,y);
                xnorm=norminf(x);
                for(int i=0;i<n;i++) {y[i]=x[i]/xnorm;}
                mu=YT_X(y,x)/YT_X(y,y);
                for(int i=0;i<n;i++) {r[i]=mu*y[i]-x[i];}
                k++;
                err=Math.abs((mu-mueski)/mu);
   }while(err>eps);
 return mu;
}
  public static void pivot(double a[][],int o[],double s[],int k)
{// GaussLU için pivot metodu
  int n=a.length;
  int dummy1=0;
  double buyuk;
  double dummy=0;
  //pivotlama
            int p=k;
            buyuk=Math.abs(a[o[k]][k]);
            for(int ii=k+1;ii<n;ii++)
            {  dummy=Math.abs(a[o[ii]][k]/s[o[ii]]);
              if(dummy > buyuk) {buyuk=dummy;p=ii;}
            }
            dummy1=o[p];
            o[p]=o[k];
            o[k]=dummy1;
}

public static int[] sirali_indeks(int n)
{   // sirali rakamlari taşiyan bir vektör oluşturur
                int o[]=new int[n];
                for(int i=0;i<n;i++) {o[i]=i;}
                return o;
}

public static double[][] gaussLU(double ai[][],int o[])
{ //kismi pivotlu gauss eleme yöntemi
int n=ai.length;
double a[][]=new double[n][n];
double s[]=new double[n];
double carpan=0;
double toplam=0;
double absaij;

//gauss eleme
int i,j,k,p,ii,jj;
```

```
//System.out.println("Orijinal Matris :\n"+Matrix.toString(ai));
for(i=0;i<n;i++)
{for(j=0;j<n;j++){a[i][j]=ai[i][j];}}
for(i=0;i<n;i++)
{ o[i]=i;s[i]=Math.abs(a[i][0]);
  for(j=1;j<n;j++)
    { absaij=Math.abs(a[i][j]);
      if(absaij>s[i]) {s[i]=absaij;}
    }
}
for(k=0;k<(n-1);k++)
{ pivot(a,o,s,k);
            //pivot referansini değiştir
            //System.out.println("pivotlu Matris : k="+k+"\n"+Matrix.toString(a));
            //gauss elemeyi çözme
  for(i=k+1;i<n;i++)
  {  carpan=a[o[i]][k]/a[o[k]][k];
     a[o[i]][k]=carpan;
     for(j=k+1;j<n;j++)
     {   a[o[i]][j]-=carpan*a[o[k]][j]; }
            }
}
return a;
}

public static double[] yerine_koyma(double ai[][],double bi[])
{   int n=ai.length;
            return yerine_koyma(ai,bi,sirali_indeks(n));
}

public static double[][] yerine_koyma(double ai[][],double bi[][])
{   int n=ai.length;
            return yerine_koyma(ai,bi,sirali_indeks(n));
}

public static double[] yerine_koyma(double ai[][],double bi[],int o[])
{ //gauss LU çözümüyle LU indirgenmesi yapilmiş Matris kullanarak linear
  //denklem sistemi kökleri
  int n=ai.length;
  double toplam;
  double x[]=new double[n];
   double a[][]=new double[n][n];
  double b[]=new double[n];
  int i,j;
for(i=0;i<n;i++)
   {for(j=0;j<n;j++){a[i][j]=ai[i][j];};b[i]=bi[i];}
//System.out.println("a=\n "+Matrix.toString(a));
//System.out.println("b=\n "+Matrix.toStringT(b));
for(i=1;i<n;i++)
   {toplam=b[o[i]];for(j=0;j<i;j++){toplam-=a[o[i]][j]*b[o[j]];};b[o[i]]=toplam;}
 //System.out.println("d=\n "+Matrix.toStringT(b));
x[n-1]=b[o[n-1]]/a[o[n-1]][n-1];
for(i=(n-2);i>=0;i--)
   {toplam=0;
    for(j=i+1;j<n;j++)
     {toplam+=a[o[i]][j]*x[j];};
     x[i]=(b[o[i]]-toplam)/a[o[i]][i];
   }
//System.out.println("d=\n "+Matrix.toStringT(x));
return x;
}

public static double[][] yerine_koyma(double ai[][],double bi[][],int o[])
{
```

```
  int n=ai.length;
  int m=bi[0].length;
  double toplam;
  double x[][]=new double[n][m];
  double a[][]=new double[n][n];
  double b[][]=new double[n][m];
  int i,j,k;
for(i=0;i<n;i++)
    {for(j=0;j<n;j++){a[i][j]=ai[i][j];};for(k=0;k<m;k++) {b[i][k]=bi[i][k];}}
//System.out.println("a=\n "+Matrix.toString(a));
//System.out.println("b=\n "+Matrix.toStringT(b));
for(k=0;k<m;k++)
{
for(i=1;i<n;i++)
    {toplam=b[o[i]][k];for(j=0;j<i;j++){toplam-=a[o[i]][j]*b[o[j]][k];};b[o[i]][k]=toplam;}
 //System.out.println("d=\n "+Matrix.toStringT(b));
x[n-1][k]=b[o[n-1]][k]/a[o[n-1]][n-1];
for(i=(n-2);i>=0;i--)
    {toplam=0;
     for(j=i+1;j<n;j++)
      {toplam+=a[o[i]][j]*x[j][k];};
      x[i][k]=(b[o[i]][k]-toplam)/a[o[i]][i];
    }
}//kin sonu
//System.out.println("d=\n "+Matrix.toStringT(x));
return x;
}

public static double [] AXB(double a[][],double b[])
{
//denklem sistemi çözümü b vektörü için
int n=a.length;
int q[]=new int[n];
double c[][]=gaussLU(a,q);
double s[]=yerine_koyma(c,b,q);
return s;
}

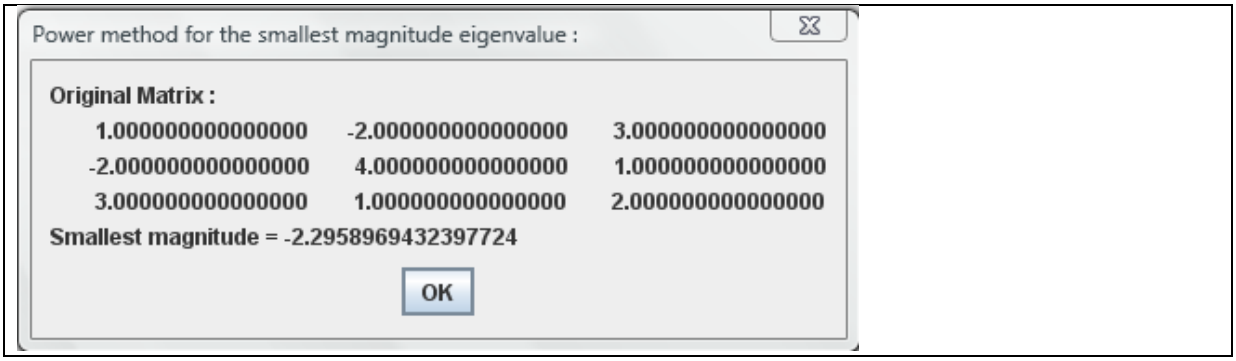
public static double [][] AXB(double a[][],double b[][])
{//denklem sistemi çözümü b matrisi için
int n=a.length;
int q[]=new int[n];
double c[][]=gaussLU(a,q);
double s[][]=yerine_koyma(c,b,q);
return s;
}

public static double eigen_inversepower(double A[][])
{
 double eps=1e-15;
 double err=1;
 int m=100;
 int n=A.length;
 double y[]=new double[n];
 double r[]=new double[n];
 int o[]=new int[n];
 double B[][]=new double[n][n];
 double xnorm;
 double mu=1.0;
 for(int i=0;i<n;i++) y[i]=1.0;
 int k=0;
 double mueski;
 double x[]=new double[n];
 do{
```

```
                mueski=mu;
                B=gaussLU(A,o);
                x=yerine_koyma(B,y,o);
                xnorm=norminf(x);
                for(int i=0;i<n;i++) {y[i]=x[i]/xnorm;}
                x=yerine_koyma(B,y,o);
                mu=YT_X(y,x)/YT_X(y,y);
                for(int i=0;i<n;i++) {r[i]=mu*y[i]-x[i];}
                k++;
                err=Math.abs((mu-mueski)/mu);
   }while(err>eps);
 return 1.0/mu;
 }

 public static void main(String args[]) throws IOException
 {
 //Eigen Value calculations using  method
 //double a[][]={{3.556,-1.778,0},{-1.778,3.556,-1.778},{0,-1.778,3.556}};
 //double a[][]={{-3.9,-0.6,-3.9,0.4},{1,0,0,0},{0,1,0,0},{0,0,1,0}};
 //double a[][]={{-1,2},{2,2}};
 double a[][]={{1,-2,3},{-2,4,1},{3,1,2}};
 String s="Original Matrix : \n";
 s+=Matrix.toString(a);
 s+="Smallest magnitude = "+eigen_inversepower(a);
 JOptionPane.showMessageDialog(null,s,
   "Power method for the smallest magnitude eigenvalue : ",JOptionPane.PLAIN_MESSAGE);
 }
}
```

Output 9.3-1 Power method for the smallest magnitude eigevalue



Example : Find the smalest eigenvalue of $\boldsymbol{A} = \begin{bmatrix} \mathbf{1} & \mathbf{2} \\ \mathbf{2} & \mathbf{5} \end{bmatrix}$ by using inverse power method.

Inverse of the A

$$\boldsymbol{A^{-1}} = \begin{bmatrix} \mathbf{5} & \mathbf{-2} \\ \mathbf{-2} & \mathbf{1} \end{bmatrix}$$

5	-2	1	3	3
-2	1	1	-1	1
				3

5	-2	1	5.666667	5.666667
-2	1	-0.33333	-2.33333	2.333333
				5.666667

5	-2	1	5.823529	5.823529
-2	1	-0.41176	-2.41176	2.411765
				5.823529

5	-2	1	5.828283	5.828283

-2	1	-0.41414	-2.41414	2.414141
				5.828283

5	-2	1	5.828423	5.828423
-2	1	-0.41421	-2.41421	2.414211
				5.828423

5	-2	1	5.828427	5.828427
-2	1	-0.41421	-2.41421	2.414213
				5.828427

λ=1/5.828427= 0.171573

Example : Find the smalest eigenvalue of $\boldsymbol{A} = \begin{bmatrix} \mathbf{3.556} & \mathbf{-1.7780} \\ \mathbf{-1.7783.556} & \mathbf{-1.778} \\ \mathbf{0} & \mathbf{-1.7783.556} \end{bmatrix}$

by using inverse power method.

Inverse of the matrix A

$$\boldsymbol{A^{-1}} = \begin{bmatrix} \mathbf{0.421822270.2812150.14060} \\ \mathbf{0.281214840.562430.28121} \\ \mathbf{0.140607420.2812150.42182} \end{bmatrix}$$

0.421822272	0.281215	0.140607	1	0.843645	0.843645	
0.281214848	0.56243	0.281215	1	1.124859	1.124859	
0.140607424	0.281215	0.421822	1	0.843645	0.843645	
					1.124859	
0.421822272	0.281215	0.140607	0.75	0.703037	0.703037	
0.281214848	0.56243	0.281215	1	0.984252	0.984252	
0.140607424	0.281215	0.421822	0.75	0.703037	0.703037	
				0.984252	0.984252	-0.125
0.421822272	0.281215	0.140607	0.714286	0.68295	0.68295	
0.281214848	0.56243	0.281215	1	0.964165	0.964165	
0.140607424	0.281215	0.421822	0.714286	0.68295	0.68295	
				0.964165	0.964165	-0.02041
0.421822272	0.281215	0.140607	0.708333	0.679603	0.679603	
0.281214848	0.56243	0.281215	1	0.960817	0.960817	
0.140607424	0.281215	0.421822	0.708333	0.679603	0.679603	
				0.960817	0.960817	-0.00347
0.421822272	0.281215	0.140607	0.707317	0.679031	0.679031	
0.281214848	0.56243	0.281215	1	0.960246	0.960246	
0.140607424	0.281215	0.421822	0.707317	0.679031	0.679031	
				0.960246	0.960246	-0.00059
0.421822272	0.281215	0.140607	0.707143	0.678933	0.678933	
0.281214848	0.56243	0.281215	1	0.960148	0.960148	
0.140607424	0.281215	0.421822	0.707143	0.678933	0.678933	
				0.960148	0.960148	-0.0001
0.421822272	0.281215	0.140607	0.707113	0.678916	0.678916	
0.281214848	0.56243	0.281215	1	0.960131	0.960131	
0.140607424	0.281215	0.421822	0.707113	0.678916	0.678916	
				0.960131	0.960131	-1.8E-05
0.421822272	0.281215	0.140607	0.707108	0.678913	0.678913	
0.281214848	0.56243	0.281215	1	0.960128	0.960128	
0.140607424	0.281215	0.421822	0.707108	0.678913	0.678913	
				0.960128	0.960128	-3E-06
0.421822272	0.281215	0.140607	0.707107	0.678913	0.678913	
0.281214848	0.56243	0.281215	1	0.960128	0.960128	
0.140607424	0.281215	0.421822	0.707107	0.678913	0.678913	
				0.960128	0.960128	-5.2E-07
0.421822272	0.281215	0.140607	0.707107	0.678913	0.678913	
0.281214848	0.56243	0.281215	1	0.960128	0.960128	

0.140607424	0.281215	0.421822	0.707107	0.678913	0.678913	
				0.960128	0.960128	-8.8E-08

0.421822272	0.281215	0.140607	0.707107	0.678913	0.678913	
0.281214848	0.56243	0.281215	1	0.960128	0.960128	
0.140607424	0.281215	0.421822	0.707107	0.678913	0.678913	
				0.960128	0.960128	-1.5E-08

0.421822272	0.281215	0.140607	0.707107	0.678913	0.678913	
0.281214848	0.56243	0.281215	1	0.960128	0.960128	
0.140607424	0.281215	0.421822	0.707107	0.678913	0.678913	
				0.960128	0.960128	-2.6E-09

λ=1/0.960128= 1.041528

9.4 COMPUTATION OF INTERMEDIATE EIGENVALUES BY POWER METHOD

Using power method already given largest and smallest eigenvalue. The remaining eigenvalues can also be found by using power method. The procedure is called deflation. **Hotelling deflation** for symmetric matrices will be described in here. According to this procedure deflated matrix A_i to be used in finding the eigenvector X_i and this eigenvector will be used to define new defleted matrix A_{i+1} acording to relation

$[\boldsymbol{A_{i+1}}] = [\boldsymbol{A_i}] - \boldsymbol{\lambda_i}\{\boldsymbol{X^i}\}\{\boldsymbol{X^i}\}^{\boldsymbol{T}}$ i=1,2,…,n-1

Eigenvector $\{\boldsymbol{X^i}\}$ has been normalized as. $\{\boldsymbol{X^i}\}\{\boldsymbol{X^i}\}^{\boldsymbol{T}} = \boldsymbol{1}$ This process can be defined as $k = \frac{1}{|X^i|}$

$$\{\boldsymbol{X^i}\} = \boldsymbol{k}\{\boldsymbol{X^i}\}$$

An example is given by using spreadsheet:

$$\boldsymbol{A} = \rightleftarrows\rightleftarrows \begin{bmatrix} \boldsymbol{12} & \boldsymbol{-6} & \boldsymbol{-6} \\ \boldsymbol{6} & \boldsymbol{16} & \boldsymbol{2} \\ \boldsymbol{-6} & \boldsymbol{2} & \boldsymbol{16} \end{bmatrix}$$

POWER METHOD

A			y	X	X/Xmax
12	6	-6	1	18	1
6	16	2	0.5	13	0.722222222
-6	2	16	-0.5	-13	-0.722222222
				18	

A			y	X	X/Xmax
12	6	-6	1	20.66666667	1
6	16	2	0.7222222	16.11111111	0.779569892
-6	2	16	-0.722222	-16.1111111	-0.779569892
			TOTAL	20.66666667	RESİDUE 2.666666667

A			y	X	X/Xmax
12	6	-6	1	21.35483871	1
6	16	2	0.7795699	16.91397849	0.79204431
-6	2	16	-0.77957	-16.9139785	-0.79204431
				21.35483871	0.688172043

A			y	X	X/Xmax
12	6	-6	1	21.50453172	1
6	16	2	0.7920443	17.08862034	0.794652056
-6	2	16	-0.792044	-17.0886203	-0.794652056
				21.50453172	0.149693012

A			y	X	X/Xmax
12	6	-6	1	21.53582467	1
6	16	2	0.7946521	17.12512878	0.795192617
-6	2	16	-0.794652	-17.1251288	-0.795192617
				21.53582467	0.031292948

A			y	X	X/Xmax
12	6	-6	1	21.54231141	1

6	16	2	0.7951926	17.13269664	0.795304474
-6	2	16	-0.795193	-17.1326966	-0.795304474
				21.54231141	0.006486736

A

			y	X	X/Xmax
12	6	-6	1	21.54365369	1
6	16	2	0.7953045	17.13426264	0.795327612
-6	2	16	-0.795304	-17.1342626	-0.795327612
				21.54365369	0.001342282

A

12	6	-6	1	21.54393134	1
6	16	2	0.7953276	17.13458656	0.795332397
-6	2	16	-0.795328	-17.1345866	-0.795332397
				21.54393134	0.000277654

λ1=		**21.543654**

	1	0.7953324	-0.795	1			
K^2=				0.7953324	2.265107245		0.4414802
				-0.795332			
K=							0.664439764

	1				0.664439764
X1=	0.7953324	*	0.6644398		0.528450471
	-0.795332				-0.528450471

0.6644398	0.6644398	0.5284505	-0.528	0.4414802	0.35112351	-0.351123506
0.5284505				0.351123506	0.2792599	-0.2792599
-0.52845				-0.35112351	-0.2792599	0.2792599

0.4414802	0.3511235	-0.351124			9.511096543	7.56448322	-7.564483216
0.3511235	0.2792599	-0.27926	*	21.543654	=	7.564483216	6.01627857	-6.016278572
-0.351124	-0.27926	0.2792599				-7.56448322	-6.01627857	6.016278572

12	6	-6		9.5110965	7.56448322	-7.56448322		2.488903457	-1.564483	1.5644832
6	16	2	-	7.5644832	6.01627857	-6.01627857	=	-1.564483216	9.9837214	8.0162786
-6	2	16		-7.564483	-6.01627857	6.016278572		1.564483216	8.0162786	9.9837214

A

2.4889035	-1.564483	1.5644832	1	2.488903457		0.127215395
-1.564483	9.9837214	8.0162786	1	16.43551678		0.840069048
1.5644832	8.0162786	9.9837214	1	19.56448322		1
				19.56448322		

2.4889035	-1.564483	1.5644832	0.1272154	0.566836127		0.033506943
-1.564483	9.9837214	8.0162786	0.840069	16.20426757		0.957870264
1.5644832	8.0162786	9.9837214	1	16.91697528		1
				16.91697528		

2.4889035	-1.564483	1.5644832	0.0335069	0.149306812		0.008428414
-1.564483	9.9837214	8.0162786	0.9578703	17.5269674		0.989402588
1.5644832	8.0162786	9.9837214	1	17.71469735		1
				17.71469735		

2.4889035	-1.564483	1.5644832	0.0084284	0.037556983		0.002094851
-1.564483	9.9837214	8.0162786	0.9894026	17.88101228		0.997366052

1.5644832	8.0162786	9.9837214	1	17.92823431	1
				17.92823431	

2.4889035	-1.564483	1.5644832	0.0020949	0.009334649	0.000519106
-1.564483	9.9837214	8.0162786	0.9973661	17.97042604	0.999347305
1.5644832	8.0162786	9.9837214	1	17.9821629	1
				17.9821629	

2.4889035	-1.564483	1.5644832	0.0005191	0.002313135	0.000128539
-1.564483	9.9837214	8.0162786	0.9993473	17.99267155	0.999838382
1.5644832	8.0162786	9.9837214	1	17.99557995	1
				17.99557995	

2.4889035	-1.564483	1.5644832	0.0001285	0.000572769	3.18225E-05
-1.564483	9.9837214	8.0162786	0.9998384	17.99818536	0.999959988
1.5644832	8.0162786	9.9837214	1	17.99890552	1
				17.99890552	

2.4889035	-1.564483	1.5644832	3.182E-05	0.000141801	7.87794E-06
-1.564483	9.9837214	8.0162786	0.99996	17.99955075	0.999990095
1.5644832	8.0162786	9.9837214	1	17.99972904	1
				17.99972904	

2.4889035	-1.564483	1.5644832	7.878E-06	3.51041E-05	1.95023E-06
-1.564483	9.9837214	8.0162786	0.9999901	17.99988878	0.999997548
1.5644832	8.0162786	9.9837214	1	17.99993292	1
				17.99993292	

λ2=		**17.999933**

		12	6	-6	1	0	0
		6	16	2	0	1	0
		-6	2	16	0	0	1
C=		12	6	-6	1	0	0
	0.5	0	13	5	-0.5	1	0
	-0.5	0	5	13	0.5	0	1
		12	6	-6	1	0	0
		0	13	5	-0.5	1	0
	0.38462	0	0	11.076923	0.6923077	-0.38461538	1

inverse A

0.14583	-0.0625	0.090278
-0.0625	0.090278	-0.090278
0.0625	-0.03472	0.090278

			y	X	X/Xmax
0.14583	-0.0625	0.09028	1	0.173611111	1
-0.0625	0.09028	-0.0903	1	-0.0625	-0.36
0.0625	-0.0347	0.09028	1	0.118055556	0.68

0.173611111

0.14583	-0.0625	0.09028	1	0.229722222	1
-0.0625	0.09028	-0.0903	-0.36	-0.15638889	-0.680773881
0.0625	-0.0347	0.09028	0.68	0.136388889	0.593712213

0.229722222

0.14583	-0.0625	0.09028	1	0.24198072	1
-0.0625	0.09028	-0.0903	-0.680774	-0.17755777	-0.733768262
0.0625	-0.0347	0.09028	0.5937122	0.139737001	0.577471631

0.24198072

0.14583	-0.0625	0.09028	1	0.243826705	1
-0.0625	0.09028	-0.0903	-0.733768	-0.18087582	-0.741821219
0.0625	-0.0347	0.09028	0.5774716	0.14011092	0.574633201

0.243826705

0.14583	-0.0625	0.09028	1	0.244073768	1
-0.0625	0.09028	-0.0903	-0.741821	-0.18134658	-0.742999058
0.0625	-0.0347	0.09028	0.5746332	0.14013429	0.574147279

0.244073768

0.14583	-0.0625	0.09028	1	0.244103515	1
-0.0625	0.09028	-0.0903	-0.742999	-0.18140904	-0.743164409
0.0625	-0.0347	0.09028	0.5741473	0.140131319	0.574065142

0.244103515

0.14583	-0.0625	0.09028	1	0.244106434	1
-0.0625	0.09028	-0.0903	-0.743164	-0.18141656	-0.743186296
0.0625	-0.0347	0.09028	0.5740651	0.140129645	0.574051419

0.244106434

0.14583	-0.0625	0.09028	1	0.244106563	1
-0.0625	0.09028	-0.0903	-0.743186	-0.18141729	-0.743188922
0.0625	-0.0347	0.09028	0.5740514	0.140129166	0.574049154

0.244106563

λ3=		**4.096571**

A program developed for power eigenvalues, and the same problem as above spreadsheet is run

Program 9.4-1 Power method to calculate all eigenvalues

```
import java.io.*;
import javax.swing.*;

class eigen_power
{   double A[][];
                double X[][];
                double lambda[];
                int jj;
                public eigen_power(double Ai[][])
                {jj=0;
                 A=Ai;
                 int n=A.length;
    X=new double[n][n];
    lambda=new double[n];
                }
  //===========Eigen value calculations ==============
//*************norm methods definition*****************
public static double norm(double v[])
{
// vector norm
double total=0;;
for(int i=0;i<v.length;i++)
  {total+=v[i]*v[i];}
return Math.sqrt(total);
}

  public static double norm(double v[],int p)
  {
   //p vector norm
  if(p==0) p=1;
  double total=0;
  double x,y;
  for(int i=0;i<v.length;i++)
    {
   x=Math.abs(v[i]);
   total+=Math.pow(x,p);
    }
  return Math.pow(total,(1.0/p));
  }
  public static double norminf(double v[])
    {
     //infinite vector norm
    double x;
    double max=0;
    for(int i=0;i<v.length;i++)
      {
     x=Math.abs(v[i]);
     if(x>max) max=x;
      }
    return max;
    }
  public static double norminf(double a[][])
       {
         //infinite matrix norm
         double x;
         double max = 0;
         double total;
       int i,j;

       int n=a.length;
       for(i=0;i<n;i++)
         {
         total=0;
         for(j=0;j<n;j++)
         {total+=Math.abs(a[i][j]);}
         x=total;
         if(x>max) max=x;
         }
       return max;
       }
  public static double norm(double a[][])
      {
      //matrix norm
              double x;
```

```
            double max = 0;
            double total;
            int i,j;
            int n=a.length;
            for(j=0;j<n;i++)
              {
              total=0;
              for(i=0;i<n;j++)
              {total+=Math.abs(a[i][j]);}
              x=total;
              if(x>max) max=x;
              }
            return max;
    }
  public static double normE(double a[][])
          {
          //Euclidian matrix norm
          double x;
          double total;
          int i,j;
          total=0;
          int n=a.length;
          for(j=0;j<n;i++)
          {for(i=0;i<n;j++) {total+=a[i][j]*a[i][j];}}
          return Math.sqrt(total);
         }

  public static double[] multiply(double[][] left,double[] right)
  {
  //multiplication of one matrix with one vector
  int ii,jj,i,j,k;
  int m1=left[0].length;
  int n1=left.length;
  int m2=right.length;
  double[] b;
  b=new double[m2];
  if(n1 != m2) {
  System.out.println("inner matrix dimensions must agree");
  for(ii=0;ii<n1;ii++)
   {b[ii]=0;}
   return b;   }
 for(i=0;i<m1;i++)
   { b[i]=0;
     for(k=0;k<n1;k++)
               b[i]+=left[i][k]*right[k];}
               return b;

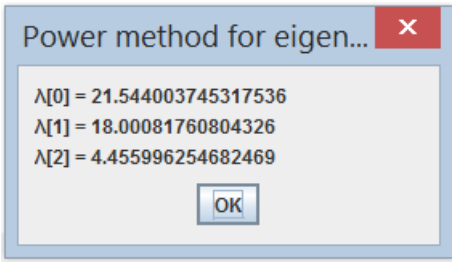
 //end of multiply of a matrix and a vector
 }
 public static double[] multiply(double left,double[] right)
{
//multiplying a vector with a constant
int i;
int n=right.length;
double b[];
b=new double[n];
for(i=0;i<n;i++)
  {
  b[i]=left*right[i];
  }
return b;
}
  public static double  YT_X(double [] left,double [] right)
  {
  //multiplys a vector transpose with a vector
  int n=left.length;
  double tot=0;
  for(int i=0;i<n;i++)
  {
  tot+=left[i]*right[i];
  }
  return tot;
  }
public static double[][] V_VT(double [] left)
```

```
{
//multiplys a vector transpose with a vector
int n=left.length;
double aa[][]=new double[n][n];
for(int i=0;i<n;i++)
  {
  for(int j=0;j<n;j++)
    {
                aa[i][j]=left[i]*left[j];
                }
  }
return aa;
}
  public static double[][] multiply(double left,double[][] right)
{
//multiplying a matrix with a constant
int i,j;
int n=right.length;
int m=right[0].length;
double b[][];
b=new double[n][m];
for(i=0;i<n;i++)
  {
  for(j=0;j<m;j++)
    b[i][j]=right[i][j]*left;
  }
return b;
//end of multiplying a matrix with a constant double
}
public static double[][] substract(double[][] left,double[][] right)
{
//addition of two matrices
int n1=left.length;
int m1=left[0].length;
int n2=right.length;
int m2=right[0].length;
int nMax,mMax;
int i,j;
if(m1>=m2) mMax=m1;
else       mMax=m2;
if(n1>=n2) nMax=n1;
else       nMax=n2;
double b[][];
b=new double[nMax][mMax];
for(i=0;i<n1;i++)
  {
  for(j=0;j<m1;j++)
    {
    b[i][j]=b[i][j]+left[i][j];
    }
  }
for(i=0;i<n2;i++)
  {
  for(j=0;j<m2;j++)
    {
    b[i][j]=b[i][j]-right[i][j];
    }
  }
return b;
}
  public double[] power()
  {double eps=1e-25;
   double err=1;
   int m=300;
   int n=A.length;
   double y[]=new double[n];
   double r[]=new double[n];
   double xnorm;
   double mu=1.0;
   double A1[][]=A;
   for(jj=0;jj<n-1;jj++)
   {
   for(int i=0;i<n;i++) y[i]=1.0;
   int k=0;
```

```
   double mueski;
   double x[]=new double[n];
   do{
                    mueski=mu;
                    x=multiply(A,y);
                    xnorm=norminf(x);
                    for(int i=0;i<n;i++) {y[i]=x[i]/xnorm;}

                    mu=YT_X(y,x)/YT_X(y,y);
                    for(int i=0;i<n;i++) {r[i]=mu*y[i]-x[i];}
                    k++;
                    err=Math.abs((mu-mueski)/mu);
     }while(err>eps);
     //for(int i=0;i<n;i++) {r[i]=r[i]/mu;}
     X[jj]=y;
     lambda[jj]=mu;
     double cc=YT_X(X[jj],X[jj]);
     double kk=1.0/norm(X[jj]);
     X[jj+1]=multiply(kk,X[jj]);
               A=substract(A,multiply(lambda[jj], V_VT(X[jj+1])));
    }//end for(jj
    lambda[jj]=NA132.eigen_inversepower(A1);
   return lambda;
  }
  public double[] lambda()
  {return lambda;}
  public String toString()
  {String s="";
   for(int i=0;i<lambda.length;i++)
   {s+="\u03BB'+"["+i+"] = "+lambda[i]+"\n";}
   return s;
  }

  public static void main(String args[]) throws IOException
  {
  double a[][]={{12,6,-6},{6,16,2},{-6,2,16}};
  eigen_power w=new eigen_power(a);
  double c[]=w.power();
  System.out.println(Matrix.toString(c));
  JOptionPane.showMessageDialog(null,w,
  "Power method for eigenvalues : ",JOptionPane.PLAIN_MESSAGE);
  }
}
```



9.5 JACOBİ METHOD AND HOUSEHOLDER TRANSFORMATION

Jacobi method is used to calculate eigen value problems for the symmetric matrices. For a symmetric matrix, transpose of the matrix is the same as the matrix itself.

$\{A\}^T=\{A\}$

By using this property of the symmetric matrices a transformation called similarity tranformation can be applied. This transformation converts the matrix into diagonal form. Eigenvalues can be calculated by using this form. Assume matrix P be a nonsingular matrix similarity transformation :

$\{B\}=\{P\}^{-1}\{A\}\{P\}$

After applying similarity transformation eigen values reamain the same, but eigenvectors change. Therefore eigenvalues of matrix B is the same as eigenvalues of the matrix A. Now assume that vector $[y]^k$ is the eigenvector of matrix $\{B\}$ corresponding eigenvalue λ_k . In this case :

$\{B\}[y]^k = \lambda_k [y]^k$

$\{B\}=\{P\}^{-1}\{A\}\{P\}$

If both side of the equation is multiplied with $\{P\}$:

$\{P\}\{B\}=\{P\}\{P\}^{-1}\{A\}\{P\}=\{A\}\{P\}$ and

$\{A\}\{P\}[y]^k = \{P\}\{B\}[y]^k= \lambda_k\{P\}[y]^k$. If $[y]^k$ is eigenvector of $\{B\}$, eigenvector of $\{A\}$ took a velue like $[x]^k$. The relation between $[y]^k$ and $[x]^k$

$[x]^k=\{P\}[y]^k$.

Eigenvector of $\{A\}$ cb be obtained from multiplication of eigenvector $\{B\}$ by $\{P\}$

Symmetric matrices can be converted to the diagonal form by applying orthogonal symmetry .Ortagonal ($\{P\}^{-1}=\{P\}^T$) In order to obtained this convertion angular convertion matrix, $R(\phi)$, can be applied Matrix $R(\phi)$ is a unit matrix except row p and column q.

$R_{pp}=R_{qq} = \cos(\phi)=c$

$R_{pq}= - R_{qp}= - \sin(\phi)=-s$

For example for n=8 p=3 q=5

$$R(\varphi)=\begin{bmatrix}1&0&0&0&0&0&0&0\\0&1&0&0&0&0&0&0\\0&0&R_{pp}&0&R_{pq}&0&0&0\\0&0&0&1&0&0&0&0\\0&0&R_{qp}&0&R_{qq}&0&0&0\\0&0&0&0&0&1&0&0\\0&0&0&0&0&0&1&0\\0&0&0&0&0&0&0&1\end{bmatrix}=\begin{bmatrix}1&0&0&0&0&0&0&0\\0&1&0&0&0&0&0&0\\0&0&cos(\varphi)&0&-sin(\varphi)&0&0&0\\0&0&0&1&0&0&0&0\\0&0&sin(\varphi)&0&cos(\varphi)&0&0&0\\0&0&0&0&0&1&0&0\\0&0&0&0&0&0&1&0\\0&0&0&0&0&0&0&1\end{bmatrix}$$

In this equation ϕ is the rotation angle. Rotation matrix R has interesting properties, one of these is multiplication of rotation matrix and transpose value will be given a unit matrix.

$\{R(\phi)\}^T\{R(\phi)\}=I$

Inversion of the rotation matrix is also very simple.

$\{R(\phi)\}^{-1}=\{R(\phi)\}^T$

For the rotation matrix following property is also valid.

$\{R(\phi)\}^T=\{R(-\phi)\}$

If similarity transformation is applied by using R matrix

$\{B\}=\{R(\phi)\}^T\{A\}\{R(\phi)\}$

In this transform angle ϕ can be selected such that $B_{qp}=0$.

$B_{qp}=(A_{qq} - A_{pp})\sin(\phi)\cos(\phi)+A_{pq}[\cos^2(\phi) - \sin^2(\phi)] = 0$

If $\sin(2\phi)=2\sin(\phi)\cos(\phi)$

And remembering that

$\cos(2\phi)=\cos^2(\phi) - \sin^2(\phi)$

$B_{qp}=(A_{qq} - A_{pp})\sin(2\phi)/2+A_{pq}\cos(2\phi) = 0$

ϕ becomes

$$\varphi=\frac{1}{2}arctan\left\{\frac{2A_{pq}}{A_{pp}-A_{qq}}\right\}$$

By using this angle and R matrix similarity transformation is carried out. Aplication of Jacobi method is as follows: P=I is selected. In upper trianglar part of the matrix the value with the biggest magnitude A_{pq} value is found. By using p and q values (indices of the element) angle ϕ is calculated. $\{P\}=\{P\}\{R\}$ tranformation and ve $\{A\}=\{R\}^T\{A\}\{R\}$ tranformation is applied. And biggest magnitude A_{pq} value is found in the upper triangular part of the matrix again and iteration continued until absolute value A_{pq} is close to zero (equal to ε) . Eigenvalues is taken as $\lambda_k=A_{kk}$, and eigen vector becomes $P=[x^1,x^2,\ldots,x^n]$.

One of the problem of Jacobi iteration is the excessive number of iteration steps. If A matrix is pre-transformed before the process above by using Householder transformation, total number of iteration reduces significantly. Householder transformation converts the matrix into upper Hessenberg matrix.

$$B = \begin{bmatrix} B_{11} & B_{12} & B_{13} & \dots & B_{1,n-1} & B_{1,n} \\ B_{21} & B_{22} & B_{23} & \dots & B_{2,n-1} & B_{2,n} \\ 0 & B_{32} & B_{33} & \dots & B_{3,n-1} & B_{3,n} \\ 0 & 0 & B_{43} & \dots & B_{4,n-1} & B_{4,n} \\ \dots & \dots & \dots & \dots & \dots & \dots \\ 0 & 0 & 0 & \dots & B_{n,n-1} & B_{n,n} \end{bmatrix}$$

As it is seen from the matrix itself, upper triangular part plus one below the diagonal is existed in this form, rest of the matrix elements are zero. Householder transformations is as follows:

1. Q=I is taken

2. $\alpha = sign(A_{k+1,k})\sqrt{\sum_{j=k+1}^{n} A_{j,k}^2}$ In this equation $sign(A_{k+1,k})$, is the sign of $A_{k+1,k}$. sign(x)= x / |x|

2. $u^k=[0,\dots,0,A_{k+1,k} + \alpha, A_{k+2,k},\dots.., A_{n,k}]^T$.

3. $P = I - \frac{2uu^T}{u^T u}$

4. Q = Q P is applied.

5. $A=PAP^T$.

6. B=A

B is the upper Hessenberg matrix.

An example problem: Upper Hessenberg transformation of Matrix

$$[A] = \begin{bmatrix} 4 & -2 & 6 & 4 \\ -2 & 2 & -1 & 3 \\ 6 & -1 & 22 & 13 \\ 4 & 3 & 13 & 46 \end{bmatrix}$$

And then find the eigenvalues:

Program 9.5-1 Jacobi method to calculate eigenvalues of the symmetric matrices.

```
import java.io.*;
import javax.swing.*;

class NA133
{  // Jacobi eigenvalue calculation method
public static double[][] I(int n)
{
//unit matrix
double b[][];
b=new double[n][n];
for(int i=0;i<n;i++)
   b[i][i]=1.0;
return b;
}

public static double[][] multiply(double[][] left,double right)
{
//multiplying a matrix with a constant
int i,j;
int n=left.length;
int m=left[0].length;
double b[][];
b=new double[n][m];
for(i=0;i<n;i++)
  {
  for(j=0;j<m;j++)
    b[i][j]=left[i][j]*right;
  }
return b;
//end of multiplying a matrix with a constant double
}
```

```
public static double[][] multiply(double[][] left,double[][] right)
{
//multiplication of two matrices
int ii,jj,i,j,k;
int m1=left[0].length;
int n1=left.length;
int m2=right[0].length;
int n2=right.length;
double[][] b;
b=new double[m1][n2];
 if(n1 != m2)
 {
 System.out.println("inner matrix dimensions must agree");
 for(ii=0;ii<n1;ii++)
   {
   for(jj=0;jj<m2;jj++)
     b[ii][jj]=0;
   }
   return b;
 }
 for(i=0;i<m1;i++)
   {
   for(j=0;j<n2;j++)
     {
     for(k=0;k<n1;k++)
                 b[i][j]+=left[i][k]*right[k][j];
     }
   }
return b;
//end of multiply of two matrices
}

public static double[][] substract(double[][] left,double[][] right)
{
//addition of two matrices
int n1=left.length;
int m1=left[0].length;
int n2=right.length;
int m2=right[0].length;
int nMax,mMax;
int i,j;
if(m1>=m2) mMax=m1;
else       mMax=m2;
if(n1>=n2) nMax=n1;
else       nMax=n2;
double b[][];
b=new double[nMax][mMax];
for(i=0;i<n1;i++)
  {
  for(j=0;j<m1;j++)
    {
    b[i][j]=b[i][j]+left[i][j];
    }
  }
for(i=0;i<n2;i++)
  {
  for(j=0;j<m2;j++)
    {
    b[i][j]=b[i][j]-right[i][j];
    }
  }
return b;
//end of matrix substraction method
```

```
}

public static double[][] T(double [][] left)
{
//transpose matrix (if A=a(i,j) T(A)=a(j,i)
int i,j;
int n=left.length;
int m=left[0].length;
double b[][];
b=new double[m][n];
for(i=0;i<n;i++)
  {
  for(j=0;j<m;j++)
    {
    b[j][i]=left[i][j];
    }
  }
return b;
}

public static double VT_V(double [] left)
{
//multiply a vector transpose with a vector
int n=left.length;
double tot=0;
for(int i=0;i<n;i++)
  {
  tot+=left[i]*left[i];
  }
return tot;
}

public static double[][] V_VT(double [] left)
{
//multiplys a vector transpose with a vector
int n=left.length;
double aa[][]=new double[n][n];
for(int i=0;i<n;i++)
  {
  for(int j=0;j<n;j++)
    {
                aa[i][j]=left[i]*left[j];
                }
  }
return aa;
}

  public static double[] eigen_jacobi(double A[][])
  {
   A=Householder(A);
   double eps=1e-15;
   double err=1;
   int m=100;
   int n=A.length;
   double P[][]=new double[n][n];
   double R[][]=new double[n][n];
   int p=n,q=n;
   double max=-9.99e100;
   double fi;
   P=I(n);
   for(int i=0;i<n;i++)
   {for(int j=i+1;j<n;j++)
      {if(Math.abs(A[i][j]) > max) {max=Math.abs(A[i][j]);p=i;q=j;}}
   }
```

```
 while(Math.abs(A[p][q]) > eps )
 {
  fi=0.5*Math.atan(2.0*A[p][q]/(A[p][p]-A[q][q]));
  R=I(n);
  R[p][p]=Math.cos(fi);
  R[q][q]=Math.cos(fi);
  R[p][q]=-Math.sin(fi);
  R[q][p]=Math.sin(fi);
  P=multiply(P,R);
  A=multiply(T(R),multiply(A,R));
  max=-9.99e100;
  for(int i=0;i<n;i++)
 {for(int j=i+1;j<n;j++){if(Math.abs(A[i][j])>max) {max=Math.abs(A[i][j]);p=i;q=j;}}}
  }
 double lambda[]=new double[n];
 for(int i=0;i<n;i++) {lambda[i]=A[i][i];}
 return lambda;
}

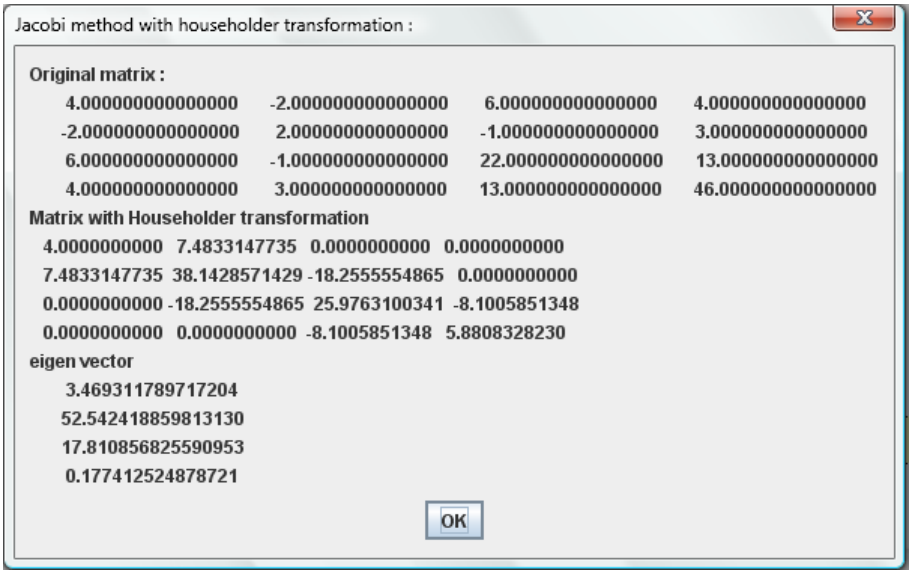
public static double sgn(double x)
{return x/Math.abs(x);}

public static double[][] Householder(double A[][])
{ //Householder transformation
 int n=A.length;
 double Q[][]=I(n);
 double alpha;
 double toplam;
 double u[]=new double[n];
 double C[][]=new double[n][n];
 double P[][]=new double[n][n];
 double B[][]=new double[n][n];
 double cc;
 for(int k=0;k<(n-2);k++)
 {
   toplam=0;
   for(int j=k+1;j<n;j++) {toplam+=A[j][k]*A[j][k];}
   toplam=Math.sqrt(toplam);
   alpha=sgn(A[k+1][k])*toplam;
   for(int j=0;j<n;j++)
   {   if(j<=k) u[j]=0.0;
       else if(j==k+1) u[j]=A[j][k]+alpha;
       else u[j]=A[j][k];
   }
   C=V_VT(u);
   cc=2.0/VT_V(u);
   C=multiply(C,cc);
   P=substract(I(n),C);
   Q=multiply(Q,P);
   A=multiply(P,multiply(A,P));
 }
 return A;
 }


public static void main(String args[]) throws IOException
{
//Sinir değer hesaplari
double c[];
c=new double[2];
//double a[][]={{12,6,-6},{6,16,2},{-6,2,16}};
double a[][]={{4,-2,6,4},{-2,2,-1,3},{6,-1,22,13},{4,3,13,46}};
String s="Original matrix : \n";
s+=Matrix.toString(a);
```

```
 //c=eigen_jacobi(a);
 s+="Matrix with Householder transformation  \n";
 s+=Matrix.toStringT(Householder(a));
 c=eigen_jacobi(a);
 s+="eigen vector \n";
 s+=Matrix.toStringT(c);
 JOptionPane.showMessageDialog(null,s,
   "Jacobi method with householder transformation : ",JOptionPane.PLAIN_MESSAGE);
 }
}
```

Output 9.5-1 Jacobi method with Householder transformation to solve eigenvalues of the symmetric matrices.



Program 9.5-2 Jacobi method to calculate eigenvalues of the symmetric matrices, Python version

```
# -*- coding: utf-8 -*-
"""
Created on Sat Aug 25 17:26:02 2018

@author: Mustafa Turhan Çoban
"""
from math import *;

def I(n):
   #unit matrix of dimension n
   B = [[0 for x in range(n)] for y in range(n)];
   for i in range(0,n):
      for j in range(0,n):
         if i==j: B[i][j]=1.0;
   return B;

def AT(A):
   #transpose matrix
   n=len(A);
   m=len(A[0]);
   B = [[0 for x in range(m)] for y in range(n)];
   for i in range(0,n):
      for j in range(0,m):
         B[i][j]=A[j][i];
   return B;
def multiply_m_m(left,right):
   #multiplying two matrisis
```

```
    n1=len(left);
    m1=len(left[0]);
    n2=len(right);
    m2=len(right[0]);
    b = [[0 for x in range(n1)] for y in range(m2)];
    if m1 != n2:
       print("warning : matrix sizes should be same");
       return b;
    for i in range(0,n1):
       for j in range(0,m2):
           for k in range(0,m1):
              b[i][j]+=left[i][k]*right[k][j];
    return b;

def eigen_jacobi(A):
    # eigen value of a symmetric matrix
    # Jacobi method
    eps=1e-15;
    err=1;
    m=100;
    max=-9.99e100;
    n=len(A);
    P= [[0.0 for x in range(n)] for y in range(n)];
    R= [[0.0 for x in range(n)] for y in range(n)];
    p=n;q=n;
    lambda1=[0.0 for x in range(n)];
    P=I(n);
    for i in range(n):
       for j in range(i+1,n):
          if abs(A[i][j]) > max: max=abs(A[i][j]);p=i;q=j;
    while abs(A[p][q]) > eps:
       fi=0.5*atan(2.0*A[p][q]/(A[p][p]-A[q][q]+1e-40));
       R=I(n);
       R[p][p]=cos(fi);
       R[q][q]=cos(fi);
       R[p][q]=-sin(fi);
       R[q][p]=sin(fi);
       P=multiply_m_m(P,R);
       A= multiply_m_m(AT(R), multiply_m_m(A,R));
       max=-9.99e100;
       for i in range(0,n):
          for j in range(i+1,n):
             if abs(A[i][j]) > max: max=abs(A[i][j]);p=i;q=j;
    for i in range(0,n):lambda1[i]=A[i][i];
    PP= [[0.0 for x in range(n)] for y in range(n+1)];
    PP[0]=lambda1;
    for i in range(1,n+1):
       for j in range(0,n):
          PP[i][j]=P[i-1][j];
    return PP;

a=[[4,-2,6,4],[-2,2,-1,3],[6,-1,22,13],[4,3,13,46]];
c=eigen_jacobi(a);
print("eigen values=\n",c[0]);
```

```
runfile('D:/okul/SCO1/eigen_jacobi.py', wdir='D:/okul/SCO1')
eigen values=
 [3.4693117897172057, 0.1774125248787215, 17.810856825590943, 52.54241885981311]
```

9.6 CHARACTERISTIC POLYNOMIAL AND LEVERRIER METHOD

Characteristic matrix obtained by substracting of the matrix from eigenvalue matrix(unit matrix and eigenvalue multiplication). Characteristic polynomial obtained when the determinant of this matrix is taken. Roots of characteristic polynomial is given eigenvalues.

$$\Delta(\lambda) = det\left\{ I\lambda - \begin{bmatrix} a_{11} & a_{12} & a_{13} & \dots & a_{1n} \\ a_{11} & a_{22} & a_{23} & \dots & a_{2n} \\ a_{11} & a_{11} & a_{33} & \dots & a_{3n} \\ \dots & \dots & \dots & \dots. & \dots. \\ a_{1n} & a_{2n} & a_{3n} & \dots & a_{nn} \end{bmatrix} \right\} \qquad \Delta(\lambda) =$$

$$det\left\{ \begin{bmatrix} \lambda - a_{11} & a_{12} & a_{13} & \dots & a_{1n} \\ a_{11} & \lambda - a_{22} & a_{23} & \dots & a_{2n} \\ a_{11} & a_{11} & \lambda - a_{33} & \dots & a_{3n} \\ \dots & \dots & \dots & \dots. & \dots. \\ a_{1n} & a_{2n} & a_{3n} & \dots & \lambda - a_{nn} \end{bmatrix} \right\}$$

Process can be explained by an example problem. If matrix A is given as: $A = \begin{bmatrix} 1 & 1 \\ 6 & 2 \end{bmatrix}$, characteristic polynomial of this matrix will be:

$$\Delta(\lambda) = det(\lambda I - A) = det\left\{ \begin{bmatrix} \lambda - 1 & 1 \\ 6 & \lambda - 2 \end{bmatrix} \right\} = (\lambda - 1)((\lambda - 2) - 6 = \lambda^2 - 3\lambda - 4$$

Roots of this polynomial can be obtained by using quadratic equation solution:

$$\lambda_{1,2} = \frac{3 \pm \sqrt{9 + 16}}{2} = \{4, -1\}$$

So the roots of the characteristic polynomial will give us eigenvalues. Method will be valid for non-symmetric matrices as well. In case of non symmetric matrices complex root finding methods such as Bairstow method should be utilised. In this section Leverrier method to obtain characteristic polynomial will be investigated. Algorithm of the method can be given as:

B_n=A and a_n=-trace(B_n) (trace process add the diagonal elements of the matrix) $trace(B_n) = \sum_{i=0}^{n-1} b_{ii}$
for k=n-2 to 0
{ C_k=(B_{k+1}+a_{k+1}I)
  B_k=AC_k
  $a_k = -\frac{trace(B_n)}{n-k}$
}

As it is seen from the algorithm, applying of the method is quite simple. In our example program, characteristic polynomial will be obtained by using Leverrier method and then smallest and biggest eigenvalues are obtained by using power method, by applying a search method in this region, region of each eigenvalue is obtained and then solved by using Brent root finding algorithm. Another interesting result of the Leverrier algorithm is that inverse matrix can be obtained from simple relation

$$A^{-1} = \frac{C_0}{a_0}$$

As an example problem let us calculate eigenvalues of :

$$[A] = \begin{bmatrix} 12 & 6 & -6 \\ 6 & 16 & 2 \\ -6 & 2 & 16 \end{bmatrix}$$

The following first program solves root of the charecteristic polynomial by using brent method.

Program 9.6.1 Leverrier method with brent root finding method (real eigen values, symmetric matrix)

```
 public class Leverrier
{
double D[][]; //inverse of matri
public static double trace(double B[][])
{
double sum=0;
int n=B.length;
for(int i=0;i<n;i++)
{sum+=B[i][i];
}
return sum;
}
```

```
public static double[][] topla(double A[][],double a)
{
int n=A.length;
int m=A[0].length;
double B[][]=new double[n][m];
for(int i=0;i<n;i++)
{for(int j=0;j<m;j++)
  if(i!=j) B[i][j]=A[i][j];
  else     B[i][j]=A[i][j]+a;
}
return B;
}
public static double[][] carp(double left,double[][] right)
{
//multiplying a matrix with a constant
int i,j;
int n=right.length;
int m=right[0].length;
double b[][];
b=new double[n][m];
for(i=0;i<n;i++)
  {
  for(j=0;j<m;j++)
    b[i][j]=right[i][j]*left;
  }
return b;
//end of multiplying a matrix with a constant double
}
public static double[][] carp(double[][] left,double[][] right)
{
//iki matrisin çarpımı
int ii,jj,i,j,k;
int m1=left[0].length;
int n1=left.length;
int m2=right[0].length;
int n2=right.length;
double[][] b;
b=new double[n1][m2];
 if(m1 != n2)
 {
 System.out.println("uyarı: matris boyutları aynı olmalıdır");
 for(ii=0;ii<n1;ii++)
   {
   for(jj=0;jj<m2;jj++)
     b[ii][jj]=0;
   }
   return b;
 }
 for(i=0;i<n1;i++)
   {
   for(j=0;j<m2;j++)
     {
     for(k=0;k<m1;k++){
       //System.out.println(i+"  "+j+"   "+k);
                b[i][j]+=left[i][k]*right[k][j];
                //System.out.println(b[i][j]);
```

```
    }
     }
   }
return b;
}
public  double[] polynomial_coefficients(double A[][])
{
int n=A.length;
int m=A[0].length;
double B[][]=new double[n][m];
for(int i=0;i<n;i++)
{for(int j=0;j<m;j++)
  B[i][j]=A[i][j];
}
System.out.println("i="+(n-1)+"\n"+Matrix.toString(B));
double a[]=new double[n+1];
double C[][]=new double[n][m];
a[n]=1.0;
a[n-1]=-trace(B);
for(int k=n-2;k>=0;k--)
{
C=topla(B,a[k+1]);
B=carp(A,C);
System.out.println("i="+k+"\n"+Matrix.toString(C));
a[k]=-trace(B)/(double)(n-k);
}
D=carp((1.0/a[0]),C);
return a;
}
//polynom fonksiyonu
public static double func(double e[],double x)
{
int n=e.length;
double ff;
if(n!=0.0)
  { ff=e[n-1];
  for(int i=n-2;i>=0;i--)
   { ff=ff*x+e[i]; }
  }
else
  ff=0;
return ff;
}
public static double[][] findrootlimits(double e[],double x1,double x2,int n,int nbb)
{
            // find root limits
            // n : number of sub regions
            // x1,x2 : limit values
            // nbb = number of roots in the given region
            //
            int nb;
            int i;
            double x,fp,fc,dx;
            double xb[][]=new double[2][nbb];
            nb=0;
            dx=(x2-x1)/n;
            x=x1;
```

```
		fp=func(e,x1);
		for (i=1;i<=n;i++)
		{
			x+=dx;
			fc=func(e,x);
			// if the region with a root is found.....
			if (fc*fp < 0.0 || fp==0) {
				xb[0][nb]=x-dx;
				xb[1][nb]=x;
				nb++;
			}
			fp=fc;
			if (nbb == nb) return xb;
		}
if( nb == 0)
		System.out.println("search is completed no roots is found");
else if(nb<nbb)
  {		System.out.println("search is completed only "+nb+" roots are found \n"+
    "you search for"+nbb+" roots");
   double xc[][]=new double[2][nb];
   for (i=0;i<nb;i++) {xc[0][i]=xb[0][i];xc[1][i]=xb[1][i];}
   return xc;
  }
return xb;
}
public static double brent(double e1[], double x1,double x2)
{
		//referans : Numerical Recipes in C, second edition, William H. Press,
    //Saul A. Teukolsky, William T. Vetterling, Brian P. Flannery
    //cambridge university press
		double ITMAX=200;
    double EPS=3.0e-10;
		int iter;
		double a=x1,b=x2,c,d,e,min1,min2;
		double tol=1.0e-5;
		c=0;
		d=0;
		e=0;
		double fa=func(e1,a),fb=func(e1,b),fc,p,q,r,s,tol1,xm;
		if (fb*fa > 0.0) System.out.println("Root limits is not selected correctly \n results may not be correct");
		fc=fb;
		for (iter=1;iter<=ITMAX;iter++) {
			if (fb*fc > 0.0) {
				c=a;
				fc=fa;
				e=d=b-a;
			}
			if (Math.abs(fc) < Math.abs(fb)) {
				a=b;
				b=c;
				c=a;
				fa=fb;
				fb=fc;
				fc=fa;
			}
			tol1=2.0*EPS*Math.abs(b)+0.5*tol;
			xm=0.5*(c-b);
```

```
                    if (Math.abs(xm) <= tol1 || fb == 0.0) return b;
                    if (Math.abs(e) >= tol1 && Math.abs(fa) > Math.abs(fb)) {
                            s=fb/fa;
                            if (a == c) {
                                    p=2.0*xm*s;
                                    q=1.0-s;
                            } else {
                                    q=fa/fc;
                                    r=fb/fc;
                                    p=s*(2.0*xm*q*(q-r)-(b-a)*(r-1.0));
                                    q=(q-1.0)*(r-1.0)*(s-1.0);
                            }
                            if (p > 0.0)  q = -q;
                            p=Math.abs(p);
                            min1=3.0*xm*q-Math.abs(tol1*q);
                            min2=Math.abs(e*q);
                            if (2.0*p < (min1 < min2 ? min1 : min2)) {
                                    e=d;
                                    d=p/q;
                            } else {
                                    d=xm;
                                    e=d;
                            }
                    } else {
                            d=xm;
                            e=d;
                    }
                    a=b;
                    fa=fb;
                    if (Math.abs(d) > tol1)
                            b += d;
                    else
                            b += (xm > 0.0 ? Math.abs(tol1) : -Math.abs(tol1));
                    fb=func(e1,b);
            }
            System.out.println("Warning : Maximum number of ,teration is exceeded \n"+
   " result might not be valid");
return b;
}
public  double[] Leverrier(double A[][])
{
int n=A.length;
double a[]=polynomial_coefficients(A);
System.out.println("coefficient of polynomial a[0]..a[n] \n"+Matrix.toStringT(a));
double lambda[]=new double[n];
lambda[n-1]=SCO13C.eigen_power(A);
lambda[0]=SCO13D.eigen_inversepower(A);
double xx[][]=findrootlimits(a,lambda[0]-0.1,lambda[n-1]+0.1,40,n);
System.out.println("root values : \n"+Matrix.toString(xx));
for(int k=1;k<(n-1);k++)
{lambda[k]=brent(a, xx[0][k],xx[1][k]);}
return lambda;
}

public static void main(String arg[])
{
//double A[][]={{1,2},{2,5}};
```

```
double A[][]={{12,6,-6},{6,16,2},{-6,2,16}};
Leverrier lv=new Leverrier();
double a[]=lv.Leverrier(A);
System.out.println("A-1 = \n"+Matrix.toString(lv.D));
System.out.println("lambda = \n"+Matrix.toStringT(a));
}
}
```

```
 ---------- Capture Output ----------
> "C:\java\bin\java.exe" Leverrier
i=2
      12.000000000000000       6.000000000000000      -6.000000000000000
       6.000000000000000      16.000000000000000       2.000000000000000
      -6.000000000000000       2.000000000000000      16.000000000000000

i=1
     -32.000000000000000       6.000000000000000      -6.000000000000000
       6.000000000000000     -28.000000000000000       2.000000000000000
      -6.000000000000000       2.000000000000000     -28.000000000000000

i=0
     252.000000000000000    -108.000000000000000     108.000000000000000
    -108.000000000000000     156.000000000000000     -60.000000000000000
     108.000000000000000     -60.000000000000000     156.000000000000000

coefficient of polynomial a[0]..a[n]
   -1728.000000000000000000000000
     564.000000000000000000000000
     -44.000000000000000000000000
       1.000000000000000000000000

mu=24.0k=0
       0.500000000000000000000000       1.000000000000000000000000       0.500000000000000000000000

mu=19.999999999999996k=1
       0.450000000000000000000000       1.000000000000000000000000       0.350000000000000000000000

mu=19.4k=2
       0.479381443298969150000000       1.000000000000000000000000       0.252577319587628860000000

mu=19.38144329896907k=3
       0.528191489361702100000000       1.000000000000000000000000       0.163297872340425480000000

mu=19.495744680851065k=4
       0.582614864127469200000000       1.000000000000000000000000       0.074047800938557190000000

mu=19.643784786641923k=5
       0.638730860685793200000000       1.000000000000000000000000      -0.015828129513989270000000

mu=19.800728905086782k=6
       0.694910736431455100000000       1.000000000000000000000000      -0.105331235346734660000000

mu=19.95880194789526k=7
       0.750090926717002400000000       1.000000000000000000000000      -0.193136050660745500000000

mu=20.114273458980524k=8
       0.803404977938862800000000       1.000000000000000000000000      -0.277948014492058300000000
```

mu=20.26453383864906k=9
0.8541300757290246000000000 1.000000000000000000000000 -0.358636332687068850000000

mu=20.407507789000007k=10
0.9016965273334043000000000 1.000000000000000000000000 -0.434299076055947250000000

mu=20.541581011888532k=11
0.9456990079338860000000000 1.000000000000000000000000 -0.504292458058621900000000

mu=20.66560913148607k=12
0.9858960708066604000000000 1.000000000000000000000000 -0.568232627542047700000000

mu=21.240148614932221k=13
1.000000000000000000000000 0.978284641339463800000000 -0.612382648602020600000000

mu=21.544003739648911k=14
1.000000000000000000000000 0.948189074374913700000000 -0.642478216315932200000000

mu=21.544003744145073k=15
1.000000000000000000000000 0.923044249041736700000000 -0.667623041804101500000000

mu=21.544003745075031k=16
1.000000000000000000000000 0.902035765079277700000000 -0.688631525798617600000000

mu=21.544003745267375k=17
1.000000000000000000000000 0.884483191470719000000000 -0.706184099413807000000000

mu=21.544003745307161k=18
1.000000000000000000000000 0.869818028544754900000000 -0.720849262341142400000000

mu=21.544003745315386k=19
1.000000000000000000000000 0.857565295218218600000000 -0.733101995667962500000000

mu=21.544003745317091k=20
1.000000000000000000000000 0.847328145128248700000000 -0.743339145757991000000000

mu=21.544003745317441k=21
1.000000000000000000000000 0.838775013417067800000000 -0.751892277469184000000000

mu=21.544003745317511k=22
1.000000000000000000000000 0.831628877878783900000000 -0.759038413007470700000000

mu=21.544003745317521k=23
1.000000000000000000000000 0.825658286654899400000000 -0.765009004231355900000000

mu=0.19004597963216294k=0
1.000000000000000000000000 -0.047619047619047620000000 0.809523809523809500000000

mu=0.22192536045782874k=1
1.000000000000000000000000 -0.475953565505804370000000 0.688225538971807700000000

mu=0.22426665037604132k=2
1.000000000000000000000000 -0.591802328644433900000000 0.645751014172066800000000

mu=0.2244077059039705k=3
1.000000000000000000000000 -0.619895378935789700000000 0.633324644965418600000000

```
mu=0.22441616283344146k=4
      1.000000000000000000000000      -0.626576893576217500000000      0.629905193526249400000000

mu=0.22441667278567667k=5
      1.000000000000000000000000      -0.628166793713999700000000      0.628990927686914100000000

mu=0.22441670368044042k=6
      1.000000000000000000000000      -0.628546737822805100000000      0.628750766608927100000000

mu=0.22441670555850934k=7
      1.000000000000000000000000      -0.628637954431483200000000      0.628688463361304100000000

mu=0.2244167056729508k=8
      1.000000000000000000000000      -0.628659947150822700000000      0.628672450932751400000000

mu=0.22441670567993627k=9
      1.000000000000000000000000      -0.628665269812347700000000      0.628668365191772400000000

mu=0.22441670568036312k=10
      1.000000000000000000000000      -0.628666562280800600000000      0.628667328558598500000000

mu=0.22441670568038927k=11
      1.000000000000000000000000      -0.628666877029119500000000      0.628667066725289600000000

mu=0.22441670568039082k=12
      1.000000000000000000000000      -0.628666953869320900000000      0.628667000829622500000000

mu=0.22441670568039093k=13
      1.000000000000000000000000      -0.628666972668676400000000      0.628666984293950300000000

root values :
      4.355996254682469     17.754202059924630     21.211803558051650
      4.788196441948346     18.186402247190507     21.644003745317526

A-1 =
     -0.145833333333333      0.062500000000000     -0.062500000000000
      0.062500000000000     -0.090277777777778      0.034722222222222
     -0.062500000000000      0.034722222222222     -0.090277777777778

lambda =
      4.455996254682469000000000
     18.000000892251040000000000
     21.544003745317520000000000


> Terminated with exit code 0.
```

A Second example :

$$[A] = \begin{bmatrix} 4 & -2 & 6 & 4 \\ -2 & 2 & -1 & 3 \\ 6 & -1 & 22 & 13 \\ 4 & 3 & 13 & 46 \end{bmatrix}$$

```
 ---------- Capture Output ----------
> "C:\java\bin\java.exe" Leverrier
i=3
```

```
      4.000000000000000      -2.000000000000000       6.000000000000000       4.000000000000000
     -2.000000000000000       2.000000000000000      -1.000000000000000       3.000000000000000
      6.000000000000000      -1.000000000000000      22.000000000000000      13.000000000000000
      4.000000000000000       3.000000000000000      13.000000000000000      46.000000000000000

i=2
     -70.000000000000000      -2.000000000000000       6.000000000000000       4.000000000000000
     -2.000000000000000     -72.000000000000000      -1.000000000000000       3.000000000000000
      6.000000000000000      -1.000000000000000     -52.000000000000000      13.000000000000000
      4.000000000000000       3.000000000000000      13.000000000000000     -28.000000000000000

i=1
     969.000000000000000     142.000000000000000     -234.000000000000000     -24.000000000000000
     142.000000000000000    1063.000000000000000       77.000000000000000     -99.000000000000000
    -234.000000000000000      77.000000000000000     255.000000000000000     -57.000000000000000
     -24.000000000000000     -99.000000000000000     -57.000000000000000      99.000000000000000

i=0
   -1364.000000000000000   -1492.000000000000000     212.000000000000000     156.000000000000000
   -1492.000000000000000   -1988.000000000000000     196.000000000000000     204.000000000000000
     212.000000000000000     196.000000000000000     -68.000000000000000     -12.000000000000000
     156.000000000000000     204.000000000000000     -12.000000000000000     -36.000000000000000

coefficient of polynomial a[0]..a[n]
     576.000000000000000000000000
   -3456.000000000000000000000000
    1193.000000000000000000000000
     -74.000000000000000000000000
       1.000000000000000000000000

root values :
      0.077412524878722       2.705662841625443       17.161039583732414      51.328293701439755
      1.391537683252082       4.019787999998804       18.475164742105775      52.642418859813110

A-1 =
     -2.368055555555555      -2.590277777777778       0.368055555555556       0.270833333333333
     -2.590277777777778      -3.451388888888889       0.340277777777778       0.354166666666667
      0.368055555555556       0.340277777777778      -0.118055555555556      -0.020833333333333
      0.270833333333333       0.354166666666667      -0.020833333333333      -0.062500000000000

lambda =
      0.17741252487872160000000000
      3.46931167646750450000000000
     17.81085541180162200000000000
     52.54241885981315000000000000

> Terminated with exit code 0.
```

The following program will be used Leverier method to find charecteristic matrix then use Horner method to solve it in python code

Program 9.6.2 Leverrier method with horner root finding method with sysnthetic division process(complex eigen values) python code

```
 from math import *

def f_poly(c1,x):
    # this function calculates the value of
    # c least square curve fitting function
    n=len(c1);
    if n!=0.0:
       ff=c1[n-1];
       for i in range(n-2,-1,-1):
          ff=ff*x+c1[i];
```

```
      else:
        ff=0,0;
      return ff;

def horner(a):
   # roots of a polynomial by using synthetic division process
      n=len(a);
      x1=complex(1.56435,0.1);
      n1=n;
      nmax=150;
      tolerance=1.0e-15;
      n1=n+1;
      ref=3.1;
      y=[complex(0.0,0.0) for i in range(n1-2)];
      n2=n1-2;
      for k in range(0,n2):
            n=len(a);
            x=x1;
            xold=3.4878;
            ref=abs(xold-x);
            b=[complex(0.0,0) for i in range(n)];
            c=[complex(0.0,0.0) for i in range(n-1)];
            b[n-1]=a[n-1];
            ii=0;
            while ref>tolerance and ii<nmax:
              for i in range(n-2,-1,-1):
                 b[i]=a[i]+b[i+1]*x;
              for i in range(n-2,-1,-1):  c[i]=b[i+1];
              Q=f_poly(c,x);
              P=b[0];
              x=x-P/Q;
              ref=abs(xold-x);
              ii=ii+1;
              xold=x;
            a=c;
            n=len(a);
            y[k]=x;
      return y;

def trace(B):
   sum=0;
   n=len(B);
   for i in range(0,n):sum+=B[i][i];
   return sum;

def add(A,a):
   n=len(A);
   m=len(A[0]);
   w, h = 8, 5;
   B = [[0 for x in range(m)] for y in range(n)];
   for i in range(0,n):
      for j in range(0,m):
        if i!=j: B[i][j]=A[i][j];
        else:    B[i][j]=A[i][j]+a;
   return B;

def multiply_c_m(left,right):
   #multiplying a matrix with a constant
    n=len(right);
    m=len(right[0]);
    b = [[0 for x in range(m)] for y in range(n)];
    for i in range(0,n):
       for j in range(0,m):
         b[i][j]=right[i][j]*left;
```

```
    return b;
#end of multiplying a matrix with a constant double

def multiply_m_m(left,right):
  #multiplying two matrisis
  n1=len(left);
  m1=len(left[0]);
  n2=len(right);
  m2=len(right[0]);
  b = [[0 for x in range(n1)] for y in range(m2)];
  if m1 != n2:
    print("uyarı: matris boyutları aynı olmalıdır");
    return b;
  for i in range(0,n1):
     for j in range(0,m2):
         for k in range(0,m1):
            b[i][j]+=left[i][k]*right[k][j];
  return b;

def polynomial_coefficients(A):
  n=len(A);
  m=len(A[0]);
  B = [[0 for x in range(m)] for y in range(n)];
  for i in range(0,n):
     for j in range(0,m):
        B[i][j]=A[i][j];
  a = [0.0 for y in range(n+1)];
  C = [[0.0 for x in range(m)] for y in range(n)];
  a[n]=1.0;
  a[n-1]=-trace(B);
  for k in range(n-2,-1,-1):
    C=add(B,a[k+1]);
    B=multiply_m_m(A,C);
    a[k]=-trace(B)/(n-k);
  return a;

def eigen_horner(A):
  b=polynomial_coefficients(A);
  return horner(b);

A=[[4,1,1],[0,2,2],[-2,0,9]];
print(eigen_horner(A));
```

```
runfile('D:/okul/SCO1/eigen_horner.py', wdir='D:/okul/SCO1')
[(1.7783225824834876+4.930380657631324e-32j), (4.817630421052084-1.1799137694746532e-16j),
(8.40404699646443+5.4124212604319174e-17j)]
```

```
A=[[4,-2,6,4],[-2,2,-1,3],[6,-1,22,13],[4,3,13,46]];
print(eigen_horner(A));
```

```
runfile('D:/okul/SCO1/eigen_horner.py', wdir='D:/okul/SCO1')
[(0.1774125248787216+0j), (3.469311789717206-6.393786517153874e-23j), (17.810856825590946+1.6791691784404488e-23j),
(52.542418859813125-1.6382924242895428e-24j)]
```

Program 9.6.3 Leverrier method with LeGuerre root finding method with sysnthetic division process(complex eigen values) python code

```
 # -*- coding: utf-8 -*-
"""
Created on Sun Sep  2 20:11:52 2018

@author: Mustafa Turhan Çoban
```

```
"""
from math import *
from cmath import *
def f_poly(c1,x):
    # this function calculates the value of
    # c least square curve fitting function
    n=len(c1);
    if n!=0.0:
      ff=c1[n-1];
      for i in range(n-2,-1,-1):
        ff=ff*x+c1[i];
    else:
      ff=0,0;
    return ff;

def df_poly(c1,x):
    h=0.001
    return (-f_poly(c1,(x+2.0*h))+8.0*f_poly(c1,(x+h))-8.0*f_poly(c1,(x-h))+f_poly(c1,(x-2.0*h)))/(12.0*h);

def d2f_poly(c1,x):
    dx=0.001
    y1=-1.0/12.0*f_poly(c1,(x-2.0*dx))+4.0/3.0*f_poly(c1,(x-dx));
    y2=-5.0/2.0*f_poly(c1,x)+4.0/3.0*f_poly(c1,(x+dx))-1.0/12.0*f_poly(c1,(x+2.0*dx));
    return (y1+y2)/dx;

def trace(B):
     sum=0;
     n=len(B);
     for i in range(0,n):sum+=B[i][i];
     return sum;

def topla(A,a):
     n=len(A);
     m=len(A[0]);
     B = [[0 for x in range(m)] for y in range(n)];
     for i in range(0,n):
       for j in range(0,m):
         if i!=j: B[i][j]=A[i][j];
         else:    B[i][j]=A[i][j]+a;
     return B;

def carp_m_m(left,right):
     #multiplying two matrisis
     n1=len(left);
     m1=len(left[0]);
     n2=len(right);
     m2=len(right[0]);
     b = [[0 for x in range(n1)] for y in range(m2)];
     if m1 != n2:
       print("warning : matrix sizes should be same");
       return b;
     for i in range(0,n1):
       for j in range(0,m2):
         for k in range(0,m1):
           b[i][j]+=left[i][k]*right[k][j];
     return b;
def eigen_LeGuerre(A):
    #eigen value result is complex use Horner method with synthetic division
    b=polynomial_coefficients(A);
    return LeGuerre_SD(b);

def polynomial_coefficients(A):
     n=len(A);
     m=len(A[0]);
     B = [[0 for x in range(m)] for y in range(n)];
     for i in range(0,n):
       for j in range(0,m):
         B[i][j]=A[i][j];
     a = [0.0 for y in range(n+1)];
     C = [[0.0 for x in range(m)] for y in range(n)];
     a[n]=1.0;
     a[n-1]=-trace(B);
     for k in range(n-2,-1,-1):
       C=topla(B,a[k+1]);
```

```
          B=carp_m_m(A,C);
          a[k]=-trace(B)/(n-k);
      return a;
def LeGuerre_SD(a):
  # roots of a polynomial by using synthetic division process
  # complex roots
    n=len(a);
    x1=complex(1.56435,1.76353);
    x=complex(1.56435,1.87363);
    n1=n;
    nmax=500;
    tolerance=1.0e-15;
    n1=n+1;
    ref=3.1;
    y=[complex(0.0,0.0) for i in range(n1-2)];
    n2=n1-2;
    for k in range(0,n2):
          n=len(a);
          m=n-1;
          x=x1;
          xold=1.4878;
          ref=abs(xold-x);
          b=[complex(0.0,0) for i in range(n)];
          c=[complex(0.0,0.0) for i in range(n-1)];
          b[n-1]=a[n-1];
          ii=0;
          while ref>tolerance and ii<nmax:
            for i in range(n-2,-1,-1):
               b[i]=a[i]+b[i+1]*x;
            for i in range(n-2,-1,-1):  c[i]=b[i+1];
            dfx=df_poly(a,x);
            fx=f_poly(a,x);
            d2fx=d2f_poly(a,x);
            G=dfx/fx;
            H=G*G-d2fx/fx;
            y1=sqrt((m*H-G*G)*(m-1.0));
            z1=G+y1;
            z2=G-y1;
            zbool=z1.real*z1.real+z1.imag*z1.imag> z2.real*z2.real+z2.imag*z2.imag;
            if zbool: z=z1;
            else: z=z2;
            aa=n/z;
            x=x-aa;
            ref=abs(xold-x);
            ii=ii+1;
            xold=x;
          if n==2: y[k]=-a[0]/a[1];
          else:    y[k]=x;
          a=c;
          n=len(a);
    return y;

A=[[4,1,1],[0,2,2],[-2,0,9]];
b= eigen_LeGuerre(A);
print("b=",b);
```

```
runfile('D:/okul/SCO1/LeGuerre.py', wdir='D:/okul/SCO1')
b= [(1.7783225824834867+1.0465662590966393e-240j), (4.817630421052081-1.0052689051413437e-152j), (8.404046996464423-
2.010537810284278e-152j)]
```

In root finding of the charecteristic polynomial, Bairstow method can be a better option comparing brent method due to direct approach of polynamial root finding of Bairstow method. Next version of Leverrier method is given here utilising Bairstow method. It should be note that eigenvalues of non-symmetric matrices can also be found by this method.

Program 9.6.3 Leverrier method with several different root finding method s(imaginary eigen values, non-symmetric matrix)

```
 public class Leverrier7
{  //variable root finding version
double D[][]; //inverse of matrix
public static double trace(double B[][])
```

```
{
double sum=0;
int n=B.length;
for(int i=0;i<n;i++)
{sum+=B[i][i];
}
return sum;
}

public static double[][] add(double A[][],double a)
{
int n=A.length;
int m=A[0].length;
double B[][]=new double[n][m];
for(int i=0;i<n;i++)
{for(int j=0;j<m;j++)
  if(i!=j) B[i][j]=A[i][j];
  else     B[i][j]=A[i][j]+a;
}
return B;
}
public static double[][] multiply(double left,double[][] right)
{
//multiplying a matrix with a constant
int i,j;
int n=right.length;
int m=right[0].length;
double b[][];
b=new double[n][m];
for(i=0;i<n;i++)
  {
  for(j=0;j<m;j++)
    b[i][j]=right[i][j]*left;
  }
return b;
//end of multiplying a matrix with a constant double
}
public static double[][] multiply(double[][] left,double[][] right)
{
//iki matrisin çarpımı
int ii,jj,i,j,k;
int m1=left[0].length;
int n1=left.length;
int m2=right[0].length;
int n2=right.length;
double[][] b;
b=new double[n1][m2];
 if(m1 != n2)
 {
 System.out.println("uyarı: matris boyutları aynı olmalıdır");
 for(ii=0;ii<n1;ii++)
   {
   for(jj=0;jj<m2;jj++)
     b[ii][jj]=0;
   }
   return b;
 }
 for(i=0;i<n1;i++)
   {
   for(j=0;j<m2;j++)
     {
     for(k=0;k<m1;k++){
                    b[i][j]+=left[i][k]*right[k][j];
    }
     }
   }
return b;
}
public  double[] polynomial_coefficients(double A[][])
{
int n=A.length;
int m=A[0].length;
double B[][]=new double[n][m];
for(int i=0;i<n;i++)
```

```
{for(int j=0;j<m;j++)
  B[i][j]=A[i][j];
}
double a[]=new double[n+1];
double C[][]=new double[n][m];
a[n]=1.0;
a[n-1]=-trace(B);
for(int k=n-2;k>=0;k--)
{
C=add(B,a[k+1]);
B=multiply(A,C);
a[k]=-trace(B)/(double)(n-k);
}
D=multiply((1.0/a[0]),C);
return a;
}

public  c[] Leverrier(String name,double A[][])
{
int n=A.length;
double a[]=polynomial_coefficients(A);
c x[];
if(name.equals("Hasan"))
{c x1[]=Hasan.root(a);x=x1;}
else if(name.equals("Halley"))
{c x1[]=Halley.root(a);x=x1;}
else if(name.equals("Lagrange"))
{c x1[]=inverse_quadratic.root(a);x=x1;}
else if(name.equals("LeGuerre"))
{c x1[]=Leguerre.root(a);x=x1;}
else if(name.equals("Muller"))
{c x1[]=muller.root(a);x=x1;}
else if(name.equals("Horner"))
{horner h=new horner();c x1[]=h.root(a);x=x1;}
else // if(name.equals("Bairstow")
{bairstow h=new bairstow();c x1[]=h.root(a);x=x1;}
return x;
}

public static void main(String arg[])
{
double A[][]={{8,-1,-5},{-4,4,-2},{18,-5,-7}};
Leverrier7 lv=new Leverrier7();
double c[]=lv.polynomial_coefficients(A);
System.out.println("Charecteristic polynomial : \n"+Matrix.toString(c));
c a[]=lv.Leverrier("Hasan",A);
System.out.println("lambda = \n"+Matrix.toStringT(a));
}
}
```

An example for the nonsymmetric matrix eigenvalue problem:

$$[A] = \begin{bmatrix} 8 & -1 & -5 \\ -4 & 4 & -2 \\ 18 & -5 & -7 \end{bmatrix}$$

```
public static void main(String arg[])
{
double A[][]={{8,-1,-5},{-4,4,-2},{18,-5,-7}};
Leverrier1 lv=new Leverrier1();
complex a[]=lv.Leverrier("Bairstow",A);
System.out.println("lambda = \n"+Matrix.toStringT(a));
}
```

```
 ---------- Capture Output ----------
> "C:\java\bin\java.exe" Leverrier1
Charecteristic polynomial :
     -20.000000000000000               24.000000000000000               -5.000000000000000                1.000000000000000

lambda =
(   1.0000000000)
```

```
( 2.0000000000 - 4.0000000000i )
( 2.0000000000 + 4.0000000000i )
```

```
A=[[8,-1,-5],[-4,4,-2],[18,-5,-7]];
print(eigen_horner(A));
```

```
runfile('D:/okul/SCO1/eigen_horner.py', wdir='D:/okul/SCO1')
[(1+0j), (1.9999999999999998+4j), (2-4j)]
```

Another non-symmetric example

$$[A] = \begin{bmatrix} 4 & 1 & 1 \\ 0 & 2 & 2 \\ -2 & 0 & 9 \end{bmatrix}$$

```
 public static void main(String arg[])
{
double A[][]={{4,1,1},{0,2,2},{-2,0,9}};
Leverrier7 lv=new Leverrier7();
double c[]=lv.polynomial_coefficients(A);
System.out.println("Charecteristic polynomial : \n"+Matrix.toString(c));
c a[]=lv.Leverrier("Hasan",A);
System.out.println("lambda = \n"+Matrix.toStringT(a));
}
}
```

```
 ---------- Capture Output ----------
> "C:\java\bin\java.exe" Leverrier7
Charecteristic polynomial :
    -72.000000000000000          64.000000000000000          -15.000000000000000          1.000000000000000

lambda =
( 4.8176304211)
( 8.4040469965)
( 1.7783225825)


> Terminated with exit code 0.
```

```
A=[[4,1,1],[0,2,2],[-2,0,9]];
print(eigen_horner(A));
```

```
runfile('D:/okul/SCO1/eigen_horner.py', wdir='D:/okul/SCO1')
[(1.7783225824834876+4.930380657631324e-32j), (4.817630421052084-1.1799137694746532e-16j),
(8.40404699646443+5.4124212604319174e-17j)]
```

```
a=[[8,-1,-5],[-4,4,-2],[18,-5,-7]];
b= eigen_LeGuerre(a);
print("b=",b);
```

```
runfile('D:/okul/SCO1/LeGuerre.py', wdir='D:/okul/SCO1')
b= [(1.0000000000000002-7.974909351313641e-17j), (2+3.9999999999999996j), (2-4j)]
```

9.7 QL ALGORITHM

If a symmetric matrix A is considered, This matrix can be devided into two submatrix as Q orthogonal matrix and L lower triangular matrix.

A=Q L . Since Q is an orthogonal matrix $Q^T = Q^{-1}$. If both side of the eqaution is multiplied by Q^T , it becomes

$Q^T A = L$. A new B matrix can be created as:

$B=L\ Q = Q^T A\ Q$. Matrix Bis a similarity transformed form of matrix A, therefore eigenvalues of both matrices are the same.

QL algorithm is based on a series of othogonal transformations. The recurrence relations used is

$\mathbf{A_m=Q_m\ L_m}$ and $\mathbf{A_{m+1}= L_m\ Q_m = Q_m^T\ A_m\ Q_m}$

Because A is a symmetric matrix, eigenvalue will be the real numbers

Program 9.7-1 QL algorithm to calculate eigenvalues of the symmetric matrices.

```
import java.io.*;
import java.io.*;
import javax.swing.*;

class NA134
{
public static double[][] tridiagonal(double b[][], double d[], double e[])
{
//reduces matrix to tridiaonal form by using HOUSEHOLDER TRANSFORMATION
//this method is used by QL method to calculate eigen values
//and eigen  vectors of a symmetric matrix
        int l,k,j,i;
   int n=b.length;
        double scale,hh,h,g,f;
   double a[][]=new double[n+1][n+1];
   double c[][]=new double[n][n];
   for(i=0;i<n;i++)
    for(j=0;j<n;j++)
      a[i][j]=b[i][j];
        for (i=n;i>=2;i--) {
                l=i-1;
                h=scale=0.0;
                if (l > 1) {
                        for (k=1;k<=l;k++)
                                scale += Math.abs(a[i-1][k-1]);
                        if (scale == 0.0)
                                e[i-1]=a[i-1][l-1];
                        else {
                                for (k=1;k<=l;k++) {
                                        a[i-1][k-1] /= scale;
                                        h += a[i-1][k-1]*a[i-1][k-1];
                                }
                                f=a[i-1][l-1];
                                g=(f >= 0.0 ? -Math.sqrt(h) : Math.sqrt(h));
                                e[i-1]=scale*g;
                                h -= f*g;
                                a[i-1][l-1]=f-g;
                                f=0.0;
                                for (j=1;j<=l;j++) {
                                        a[j-1][i-1]=a[i-1][j-1]/h;
                                        g=0.0;
                                        for (k=1;k<=j;k++)
                                                g += a[j-1][k-1]*a[i-1][k-1];
                                        for (k=j+1;k<=l;k++)
                                                g += a[k-1][j-1]*a[i-1][k-1];
                                        e[j-1]=g/h;
                                        f += e[j-1]*a[i-1][j-1];
                                }
                                hh=f/(h+h);
                                for (j=1;j<=l;j++) {
                                        f=a[i-1][j-1];
                                        e[j-1]=g=e[j-1]-hh*f;
                                        for (k=1;k<=j;k++)
                                                a[j-1][k-1] -= (f*e[k-1]+g*a[i-1][k-1]);
                                }
                        }
                } else
                        e[i-1]=a[i-1][l-1];
                d[i-1]=h;
        }
```

```
		d[1-1]=0.0;
		e[1-1]=0.0;
		/* Contents of this loop can be omitted if eigenvectors not
				wanted except for statement d[i-1]=a[i-1][i-1]; */
		for (i=1;i<=n;i++) {
			l=i-1;
			if (d[i-1] != 0) {
				for (j=1;j<=l;j++) {
					g=0.0;
					for (k=1;k<=l;k++)
						g += a[i-1][k-1]*a[k-1][j-1];
					for (k=1;k<=l;k++)
						a[k-1][j-1] -= g*a[k-1][i-1];
				}
			}
			d[i-1]=a[i-1][i-1];
			a[i-1][i-1]=1.0;
			for (j=1;j<=l;j++) a[j-1][i-1]=a[i-1][j-1]=0.0;
		}
return  a;
}

public static double pythag(double a, double b)
{
//this method is used by QL method
		double absa,absb;
		absa=Math.abs(a);
		absb=Math.abs(b);
		if (absa > absb) return absa*Math.sqrt(1.0+(absb/absa)*(absb/absa));
		else return (absb==0.0 ? 0.0 : absb*Math.sqrt(1.0+(absa/absb)*(absa/absb)));
}

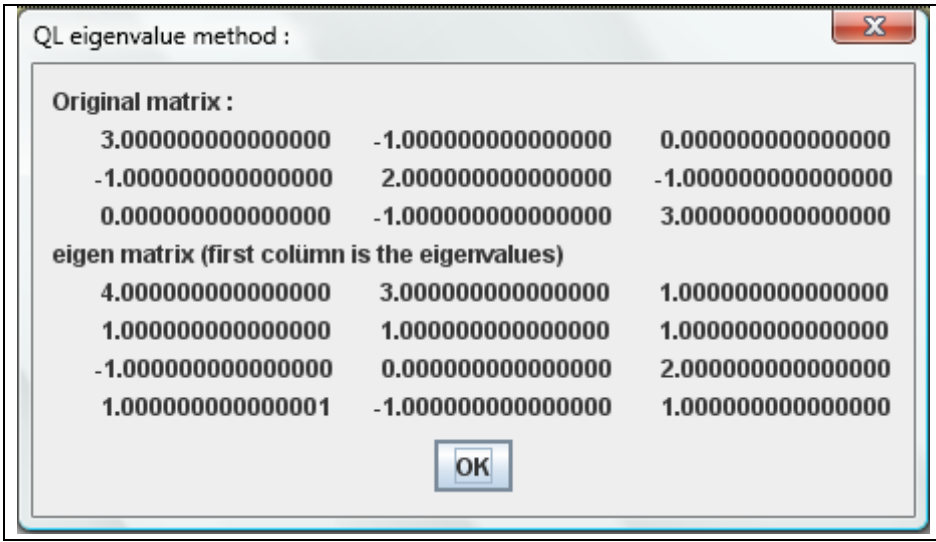
public static double[][] QL(double d[], double e[], double a[][])
{
// QL algorithm : eigenvalues of a symmetric matrix reduced to tridiagonal
// form by using method tridiagonal
   int n=d.length;
		int m,l,iter,i,j,k;
		double s,r,p,g,f,dd,c,b;
		for (i=2;i<=n;i++) e[i-2]=e[i-1];
		e[n-1]=0.0;
   double z[][]=new double[n][n];
   for(i=0;i<n;i++)
    for(j=0;j<n;j++)
      z[i][j]=a[i][j];
		for (l=1;l<=n;l++) {
			iter=0;
			do {
				for (m=l;m<=n-1;m++) {
					dd=Math.abs(d[m-1])+Math.abs(d[m]);
					if ((double)(Math.abs(e[m-1])+dd) == dd) break;
				}
				if (m != l) {
          if (iter++ == 30) System.out.println("Too many iterations in QL");
          g=(d[l]-d[l-1])/(2.0*e[l-1]);
          r=Matrix.pythag(g,1.0);
          g=d[m-1]-d[l-1]+e[l-1]/(g+Matrix.SIGN(r,g));
          s=c=1.0;
          p=0.0;
          for (i=m-1;i>=l;i--) {
                                                f=s*e[i-1];
                                                b=c*e[i-1];
                                                e[i]=(r=Matrix.pythag(f,g));
                                                if (r == 0.0) {
```

```
                                                d[i] -= p;
                                                e[m-1]=0.0;
                                                break;
                                        }
                                        s=f/r;
                                        c=g/r;
                                        g=d[i ]-p;
                                        r=(d[i-1]-g)*s+2.0*c*b;
                                        d[i ]=g+(p=s*r);
                                        g=c*r-b;
                                        for (k=1;k<=n;k++) {
                                                f=z[k-1][i ];
                                                z[k-1][i ]=s*z[k-1][i-1]+c*f;
                                                z[k-1][i-1]=c*z[k-1][i-1]-s*f;
                                        }
                                }
                                if (r == 0.0 && i >= l) continue;
                                d[l-1] -= p;
                                e[l-1]=g;
                                e[m-1]=0.0;
                        }
                } while (m != l);
        }
   return z;
}

public static double[][] eigenQL(double a[][])
{
 // QL algoritm to solve eigen value problems
 // symmetric matrices only (real eigen values)
 // first column of the matrix returns eigen values
 // second..n+1 column returns eigen vectors.
 // Note : If matrix is not symmetric DO NOT use
int i,j;
 int n=a.length;
 double sum[]=new double[n];;
 double d[]=new double[n];
 double b[][]=new double[n][n];
 double e[]=new double[n];
 double z[][]=new double[n+1][n];
 b=tridiagonal(a,d,e);
 b=QL(d,e,b);
 for(j=0;j<n;j++)
   {
   z[0][j]=d[j];
   for(i=0;i<n;i++)
     {
     z[i+1][j]=b[i][j]/b[0][j];
     if(z[i+1][j]<1e-13) z[i+1][i]=0;
     }
   }
 // result: first row is eigenvalues, the rest is eigenvector
 return z;
}

  public static void main(String args[]) throws IOException
  {
  //Eigen Value calculations
  double c[][];
  c=new double[3][3];
  //double a[][]={{-19,-4.7,6.45},{-4.7,4.6,11.8},  {6.45,11.8,-8.3}};
  double a[][]={{3,-1,0},{-1,2,-1},{0,-1,3}};
  String s="Original matrix : \n";
  s+=Matrix.toString(a);
```

```
  c=new double[3][3];
  c=eigenQL(a);
  s+="eigen matrix (first column is the eigenvalues) \n";
  s+=Matrix.toString(c);
  JOptionPane.showMessageDialog(null,s,
    "QL eigenvalue method : ",JOptionPane.PLAIN_MESSAGE);
  }
}
```



9.8 QR ALGORİTHM

By using a similarity transformation matrix A can be subdivided into a orthogonal Q and an upper triangular matrix R

A=Q R

Because Q is an orthogonal matrix $Q^T = Q^{-1}$ can be written. If equation is multiplied with Q^T

Q^T A = R, a new matrix B can be created as

B=R Q = Q^T A Q

B and A is realated to each other by similarity transformation, i.e. they have the same eigenvalues. Similarity transforms applied to B several times. When B is converted into an upper triangular form, eigenvalues can be obtained from this upper triangular form. Converting matrix to upper triangular form could be a very long process. In order to shorten the process a householder tranformation can be applied. Matrix does not have to be symmetric in QR algorithm, therfore it can be applied into non-symmetric matrices. Please note that eigen values of the non-symmetric matrices can be complex number, therefore the proces steps should be calculated with complex number algorithms.

Program 9.8-1 QR algorithm

```
 import java.io.*;
import javax.swing.*;


class NA135
{
  //===========Eigen value calculations ==============
public static double[][] balance(double b[][])
{
// balance of a matrix for more accurate eigenvalue
// calculations
double radix=2.0;
double sqrdx=radix*radix;
```

```
double c,r,f,s,g;
int m,j,i,last;
int n=b.length;
last=0;
double a[][];
a=new double[n][n];
f=1;
s=1;
for(i=1;i<=n;i++)
  for(j=1;j<=n;j++)
    a[i-1][j-1]=b[i-1][j-1];
while(last==0)
  {
  last=1;
  for(i=1;i<=n;i++)
    {
    c=0;r=0;
    for(j=1;j<=n;j++)
      {
      if(j != i)
              {
              c+=Math.abs(a[j-1][i-1]);
              r+=Math.abs(a[i-1][j-1]);
              } //end of if(j!=..
      } //end of for(j=1...
    if(c != 0 &&  r != 0 )
      {
      g=r/radix;
      f=1.0;
      s=c+r;
      while(c<g)
              {
              f*=radix;
              c*=sqrdx;
              }
      g=r*radix;
      while(c>g)
              {
              f/=radix;
              c/=sqrdx;
              }
      } //end of if(c != 0 && ....
    if( (c+r)/f < 0.95*s )
      {
      last=0;
      g=1.0/f;
      for(j=1;j<=n;j++)  { a[i-1][j-1]*=g; }
      for(j=1;j<=n;j++)  { a[j-1][i-1]*=f; }
      } //end of if( ((c+r..
    }//end of for(i=1;i<=n....
  } //end of while last==0
return a;
}

public static double[][] Hessenberg(double b[][])
{
// Calculates the hessenberg matrix
```

```
// it is used in QR method to calculate eigenvalues
// of a matrix(symmetric or non-symmetric)
int m,j,i;
int n=b.length;
double a[][];
a=new double[n][n];
for(i=0;i<n;i++)
  for(j=0;j<n;j++)
    a[i][j]=b[i][j];
double x,y;
if(n>2)
  {
  for(m=2;m<=(n-1);m++)
     {
     x=0.0;
     i=m;
     for(j=m;j<=n;j++)
       {
       if(Math.abs(a[j-1][m-2]) > Math.abs(x))
                {
                x=a[j-1][m-2];
                i=j;
                } //end of if(Math.abs(..
       }//end of for(j=m,j<=n...
     if(i!=m)
       {
       for(j=(m-1);j<=n;j++)
                {
                y=a[i-1][j-1];
                a[i-1][j-1]=a[m-1][j-1];
                a[m-1][j-1]=y;
                }//end of for(j=(m-1)..
       for(j=1;j<=n;j++)
                {
                y=a[j-1][i-1];
                a[j-1][i-1]=a[j-1][m-1];
                a[j-1][m-1]=y;
                }//end of for(j=1;j<=n....
       } //end of if(i!=m)
     if(x != 0.0)
       {
       for(i=(m+1);i<=n;i++)
                {
                y=a[i-1][m-2];
                if(y!=0.0)
                  {
                  y=y/x;
                  a[i-1][m-2]=y;
                  for(j=m;j<=n;j++)
                    {
                    a[i-1][j-1]-=y*a[m-1][j-1];
                    }
                  for(j=1;j<=n;j++)
                    {
                    a[j-1][m-1]+=y*a[j-1][i-1];
                    }
                  }//end of if(y!=0..
```

```
            }//end of for(i=(m+1)...
        } //end of if(x != 0.0...
      }//end of for(m=2;m<=(n-1)..
   }//end of Hessenberg
for(i=1;i<=n;i++)
  for(j=1;j<=n;j++)
  {
  if(i>(j+1)) a[i-1][j-1]=0;
  }
return a;
}

public static double[][] QR(double b[][])
{
//calculates eigenvalues of a Hessenberg matrix
int n=b.length;
double rm[][]=new double[2][n];
double a[][]=new double[n+1][n+1];
double wr[]=new double[n+1];
double wi[]=new double[n+1];
int nn,m,l,k,j,its,i,mmin;
double z,y,x,w,v,u,t,s,r=0,q=0,p=0,anorm;
for(i=0;i<n;i++)
   for(j=0;j<n;j++)
      a[i+1][j+1]=b[i][j];
anorm=Math.abs(a[1][1]);
for (i=2;i<=n;i++)
   for (j=(i-1);j<=n;j++)
       anorm += Math.abs(a[i][j]);
nn=n;
t=0.0;
while (nn >= 1) {
   its=0;
   do {
      for (l=nn;l>=2;l--) {
      s=Math.abs(a[l-1][l-1])+Math.abs(a[l][l]);
      if (s == 0.0) s=anorm;
      if ((Math.abs(a[l][l-1]) + s) == s) break;
                        }
      x=a[nn][nn];
      if (l == nn) {
         wr[nn]=x+t;
               wi[nn--]=0.0;
                           }
      else {
          y=a[nn-1][nn-1];
                w=a[nn][nn-1]*a[nn-1][nn];
                if (l == (nn-1)) {
                   p=0.5*(y-x);
                   q=p*p+w;
                   z=Math.sqrt(Math.abs(q));
                   x += t;
                   if (q >= 0.0) {
                      z=p+Matrix.SIGN(z,p);
                            wr[nn-1]=wr[nn]=x+z;
                            if (z!=0) wr[nn]=x-w/z;
                            wi[nn-1]=wi[nn]=0.0;
```

```
                        }
        else {
          wr[nn-1]=wr[nn]=x+p;
                wi[nn-1]= -(wi[nn]=z);
                  }
        nn -= 2;
                        }
      else {
        if (its == 30) System.out.println("Matrix.java, module QR, Too many iterations in hqr");
        if (its == 10 || its == 20) {
          t += x;
                for (i=1;i<=nn;i++) a[i][i] -= x;
                s=Math.abs(a[nn][nn-1])+Math.abs(a[nn-1][nn-2]);
                y=x=0.75*s;
                w = -0.4375*s*s;
                                        }
        ++its;
        for (m=(nn-2);m>=l;m--) {
        z=a[m][m];
        r=x-z;
        s=y-z;
        p=(r*s-w)/a[m+1][m]+a[m][m+1];
        q=a[m+1][m+1]-z-r-s;
        r=a[m+2][m+1];
        s=Math.abs(p)+Math.abs(q)+Math.abs(r);
        p /= s;
        q /= s;
        r /= s;
        if (m == l) break;
        u=Math.abs(a[m][m-1])*(Math.abs(q)+Math.abs(r));
        v=Math.abs(p)*(Math.abs(a[m-1][m-1])+
Math.abs(z)+Math.abs(a[m+1][m+1]));
        if ((u+v) == v) break;
                                        }
        for (i=m+2;i<=nn;i++) {
          a[i][i-2]=0.0;
          if (i != (m+2)) a[i][i-3]=0.0;
                        }
        for (k=m;k<=nn-1;k++) {
        if (k != m) {
          p=a[k][k-1];
                q=a[k+1][k-1];
                r=0.0;
                if (k != (nn-1)) r=a[k+2][k-1];
                if ((x=Math.abs(p)+Math.abs(q)+Math.abs(r)) != 0.0) {
                  p /= x;
                  q /= x;
                  r /= x;
                                        }                       }
                if ((s=Matrix.SIGN(Math.sqrt(p*p+q*q+r*r),p)) != 0.0) {
                  if (k == m) {
                    if (l != m)
                    a[k][k-1] = -a[k][k-1];
                                }
                  else
                    a[k][k-1] = -s*x;
                  p += s;
```

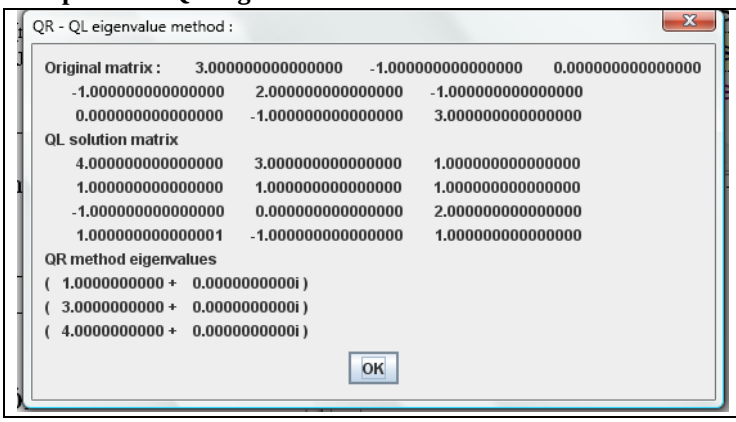
```
                        x=p/s;
                        y=q/s;
                        z=r/s;
                        q /= p;
                        r /= p;
                        for (j=k;j<=nn;j++) {
                        p=a[k][j]+q*a[k+1][j];
                        if (k != (nn-1)) {
                          p += r*a[k+2][j];
                          a[k+2][j] -= p*z;
                                  }
                        a[k+1][j] -= p*y;
                        a[k][j] -= p*x;
                                              }
                        mmin = nn<k+3 ? nn : k+3;
                        for (i=l;i<=mmin;i++) {
                          p=x*a[i][k]+y*a[i][k+1];
                          if (k != (nn-1)) {
                            p += z*a[i][k+2];
                                  a[i][k+2] -= p*r;
                                          }
                          a[i][k+1] -= p*q;
                          a[i][k] -= p;
                       }
                  }
               }
      }
    }
  } while (l < nn-1);
}
for(i=0;i<n;i++)
{
rm[0][i]=wr[i+1];
rm[1][i]=wi[i+1];
}
return rm;
} //end of QR

public static double[][] eigenValue(double b[][])
{
// this routine input a matrix (non symetric or symmetric)
// and calculate eigen values
// method balance can be used prior to this method to balance
// the input matrix
int n=b.length;
double d[][]=new double[2][n];
d=QR(Hessenberg(b));
return d;
}

public static c[] eigenValueC(double b[][])
{
// this routine input a matrix (non symetric or symmetric)
// and calculate eigen values
// method balance can be used prior to this method to balance
// the input matrix
//output eigenvalues will be in a vector of c form
```

```
int n=b.length;
double d[][]=new double[2][n];
d=QR(Hessenberg(b));
c c[]=new c[n];
for(int i=0;i<n;i++)
  {
  c[i]=new c(d[0][i],d[1][i]);
  }
return c;
}
  public static void main(String args[]) throws IOException
  {
  //Eigen Value calculations using  method
  //double a[][]={{3,-1,0},{-1,2,-1},{0,-1,3}};
  //double a[][]={{1,-2,3},{-2,4,1},{3,1,2}};
  double a[][]={{3.556,-1.778,0},{-1.778,3.556,-1.778},{0,-1.778,3.556}};
  String s="Original matrix : ";
  s+=Matrix.toString(a);
  s+="QL solution matrix \n";
  s+=Matrix.toString(SCO13A.eigenQL(a));
  //Eigenvlue calculation
  c c1[];
  int n=a.length;
  c1=new c[n];
  c1=eigenValueC(a);
  s+="QR method eigenvalues \n";
  s+=Matrix.toStringT(c1);
  JOptionPane.showMessageDialog(null,s,
     "QR - QL eigenvalue method : ",JOptionPane.PLAIN_MESSAGE);
  }
}
```

Output 9.8-1 QR algorithm

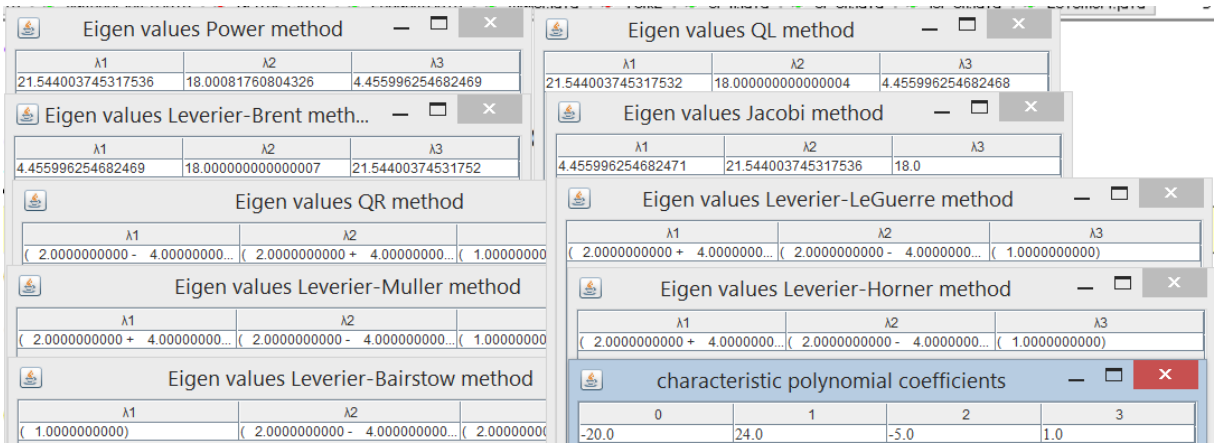


A general interface program for all different eigenvalues are given for an easy application utilisation. Two eigenvalue problem, one symmetric and one non-symmetric matrix eigenvalue problem is solved and the results are given together as output.

```
 public class eigen
{
public static double[][] Householder(double A[][])
{return NA133.Householder(A);}
public  static double[] characteristic_polynomial(double A[][])
{ Leverrier2 lv=new Leverrier2();
```

```
  return lv.polynomial_coefficients(A);
}
//real eigenvalues by using power method for limit values
//and brent method for the remaining eigenvalues
public  static double[] eigen_LBrent(double A[][])
{Leverrier lv=new Leverrier();
return lv.Leverrier(A);}
//real eigenvalues by using jacobi method
public  static double[] eigen_Jacobi(double A[][])
{return NA133.eigen_jacobi(A);}
//real eigenvalues by using QL method
public  static double[] eigen_QL(double A[][])
{               double w[][]=NA134.eigenQL(A);
                int n=w[0].length;
                double x[]=new double[n];
                for(int i=0;i<n;i++) x[i]=w[0][i];
                return x;
}
public static double[] eigen_power(double A[][])
{eigen_power w=new eigen_power(A);
return w.power();}
//complex eigenvalues using Leverier Bairstow method
public static complex[] eigen_LBairstow(double A[][])
{ Leverrier1 lv=new Leverrier1();
return lv.Leverrier(A);}
public static complex[] eigen_LHorner(double A[][])
{ Leverrier2 lv=new Leverrier2();
return lv.Leverrier(A);}
public static complex[] eigen_LMuller(double A[][])
{ Leverrier3 lv=new Leverrier3();
return lv.Leverrier(A);}
public static complex[] eigen_LLeGuerre(double A[][])
{ Leverrier4 lv=new Leverrier4();
return lv.Leverrier(A);}
//complex eigenvalues using QR method
public static complex[] eigen_QR(double A[][])
{ return NA135.eigenValueC(A);}
public static String[] heading()
{String baslik[]={""+'\u03BB'+"1",""+'\u03BB'+"2",""+'\u03BB'+"3",""+'\u03BB'+"4",""+'\u03BB'+"5",""+'\u03BB'+"6",
""+'\u03BB'+"7",""+'\u03BB'+"8",""+'\u03BB'+"9",""+'\u03BB'+"10",""+'\u03BB'+"11",""+'\u03BB'+"12"};
return baslik;}
public static void print(complex c[],String s)
{Text.print(c,heading(),s);}
public static void print(double c[],String s)
{Text.print(c,heading(),s);}
public static void main(String arg[])
{
double A[][]={{8,-1,-5},{-4,4,-2},{18,-5,-7}};
double b[]=characteristic_polynomial(A);
Text.print(b,"characteristic polynomial coefficients");
complex c[]=eigen_LBairstow(A);
print(c,"Eigen values Leverier-Bairstow method");
complex c1[]=eigen_LHorner(A);
print(c1,"Eigen values Leverier-Horner method");
complex c2[]=eigen_LMuller(A);
print(c2,"Eigen values Leverier-Muller method");
complex c3[]=eigen_LLeGuerre(A);
print(c3,"Eigen values Leverier-LeGuerre method");
complex c4[]=eigen_QR(A);
print(c4,"Eigen values QR method");
double A1[][]={{12,6,-6},{6,16,2},{-6,2,16}};
double d[]=eigen_LBrent(A1);
print(d,"Eigen values Leverier-Brent method");
double d1[]=eigen_Jacobi(A1);
```

```
print(d1,"Eigen values Jacobi method");
double d2[]=eigen_QL(A1);
print(d2,"Eigen values QL method");
double d3[]=eigen_power(A1);
print(d3,"Eigen values Power method");
}}
```



9.9 ROOTS OF THE POLYNOMIALS AND COMPANION MATRIX

Consider a nth degree polynomial

$f(x)= a_n x^n + \ldots + a_4 x^4 + a_3 x^3 + a_2 x^2 + a_1 x + a_0$

If the coefficients of the polynomial are divided by a_n , it becomes

$f(x)= x^n + b_{n-1} x^4 + \ldots + b_4 x^4 + b_3 x^3 + b_2 x^2 + b_1 x + b_0$

The roots of this polynomial is the same as the eigenvalues of the Matrix

$$A = \begin{bmatrix} -b_{n-1} & -b_{n-2} & \ldots & -b_1 & -b_0 \\ 1 & 0 & \ldots & 0 & 0 \\ 0 & 1 & \ldots & 0 & 0 \\ \ldots & \ldots & \ldots & \ldots & \ldots \\ & & \ldots & 1 & 0 \end{bmatrix}$$

This matrix is called a companion matrix. Since the matrix is not symmetric, eigen values or root of the polynomials can be complex values. QR algorithm can be used to calculate this complex eigenvalues.

Program 9.9-1 QR method to solve companion matrix (root of the nth degree polynomials)

```
 import java.util.*;
import java.awt.*;
import javax.swing.*;
import java.awt.event.*;


class NA136 extends JFrame implements ActionListener
{ private static final long serialVersionUID = 875678563L;
public static double[][] balancedEigenValue(double b[][])
{
// this routine input a matrix (non symetric or symmetric)
// and calculates eigen values
// method balance is used to balance the matrix previous to
// actual calculations
int n=b.length;
double d[][]=new double[2][n];
d=NA135.QR(NA135.Hessenberg(NA135.balance(b)));
return d;
}
public static c[] poly_rootsC(double c[])
```

```
{
// roots of a degree n polynomial
// P(x)=c[n]*x^n+c[n-1]*x^(n-1)+....+c[1]*x+c[0]=0;
// roots are returned as c variables
int n=c.length-1;
double a[][]=new double[n][n];
a=characteristic_matrix(c);
double d[][]=new double[2][n];
d=balancedEigenValue(a);
c e[]=new c[n];
for(int i=0;i<n;i++)
  e[i]=new c(d[0][i],d[1][i]);
return e;
}

public static double SIGN(double a,double b)
{
//returns the value of double a with sign of double b;
//if a=-2, b= 3 SIGN(a,b) returns  2
//if a=-2, b=-3 SIGN(a,b) returns -2
//if a= 2, b=-3 SIGN(a,b) returns -2
//if a= 2, b= 3 SIGN(a,b) returns  2
if(b!=0)
  return Math.abs(a)*b/Math.abs(b);
else
  return Math.abs(a);
}
public static double[][] characteristic_matrix(double c[])
{
//this routine converts polynomial coefficients to a matrix
//with the same eigenvalues (roots)
int n=c.length-1;
int i;
double a[][]=new double[n][n];
for(i=0;i<n;i++)
{
  a[0][i]=-c[i+1]/c[0];
}
for(i=0;i<n-1;i++)
{
  a[i+1][i]=1;
}
return a;
}

public static double[][] tridiagonal(double b[][], double d[], double e[])
{
//reduces matrix to tridiaonal form by using HOUSEHOLDER TRANSFORMATION
//this method is used by QL method to calculate eigen values
//and eigen  vectors of a symmetric matrix
            int l,k,j,i;
    int n=b.length;
            double scale,hh,h,g,f;
    double a[][]=new double[n+1][n+1];
    double c[][]=new double[n][n];
    for(i=0;i<n;i++)
      for(j=0;j<n;j++)
        a[i][j]=b[i][j];
            for (i=n;i>=2;i--) {
                        l=i-1;
                        h=scale=0.0;
                        if (l > 1) {
                                    for (k=1;k<=l;k++)
                                                scale += Math.abs(a[i-1][k-1]);
```

```
                                if (scale == 0.0)
                                        e[i-1]=a[i-1][l-1];
                                else {
                                        for (k=1;k<=l;k++) {
                                                a[i-1][k-1] /= scale;
                                                h += a[i-1][k-1]*a[i-1][k-1];
                                        }
                                        f=a[i-1][l-1];
                                        g=(f >= 0.0 ? -Math.sqrt(h) : Math.sqrt(h));
                                        e[i-1]=scale*g;
                                        h -= f*g;
                                        a[i-1][l-1]=f-g;
                                        f=0.0;
                                        for (j=1;j<=l;j++) {
                                                a[j-1][i-1]=a[i-1][j-1]/h;
                                                g=0.0;
                                                for (k=1;k<=j;k++)
                                                        g += a[j-1][k-1]*a[i-1][k-1];
                                                for (k=j+1;k<=l;k++)
                                                        g += a[k-1][j-1]*a[i-1][k-1];
                                                e[j-1]=g/h;
                                                f += e[j-1]*a[i-1][j-1];
                                        }
                                        hh=f/(h+h);
                                        for (j=1;j<=l;j++) {
                                                f=a[i-1][j-1];
                                                e[j-1]=g=e[j-1]-hh*f;
                                                for (k=1;k<=j;k++)
                                                        a[j-1][k-1] -= (f*e[k-1]+g*a[i-1][k-1]);
                                        }
                                }
                        } else
                                e[i-1]=a[i-1][l-1];
                        d[i-1]=h;
                }
                d[1-1]=0.0;
                e[1-1]=0.0;
                /* Contents of this loop can be omitted if eigenvectors not
                                wanted except for statement d[i-1]=a[i-1][i-1]; */
                for (i=1;i<=n;i++) {
                        l=i-1;
                        if (d[i-1] != 0) {
                                for (j=1;j<=l;j++) {
                                        g=0.0;
                                        for (k=1;k<=l;k++)
                                                g += a[i-1][k-1]*a[k-1][j-1];
                                        for (k=1;k<=l;k++)
                                                a[k-1][j-1] -= g*a[k-1][i-1];
                                }
                        }
                        d[i-1]=a[i-1][i-1];
                        a[i-1][i-1]=1.0;
                        for (j=1;j<=l;j++) a[j-1][i-1]=a[i-1][j-1]=0.0;
                }
return  a;
}

public static double pythag(double a, double b)
{
//this method is used by QL method
                double absa,absb;
                absa=Math.abs(a);
                absb=Math.abs(b);
                if (absa > absb) return absa*Math.sqrt(1.0+(absb/absa)*(absb/absa));
```

```
            else return (absb==0.0 ? 0.0 : absb*Math.sqrt(1.0+(absa/absb)*(absa/absb)));
}

public static double[][] QL(double d[], double e[], double a[][])
{
// QL algorithm : eigenvalues of a symmetric matrix reduced to tridiagonal
// form by using method tridiagonal
   int n=d.length;
            int m,l,iter,i,j,k;
            double s,r,p,g,f,dd,c,b;
            for (i=2;i<=n;i++) e[i-2]=e[i-1];
            e[n-1]=0.0;
   double z[][]=new double[n][n];
   for(i=0;i<n;i++)
     for(j=0;j<n;j++)
       z[i][j]=a[i][j];
            for (l=1;l<=n;l++) {
                        iter=0;
                        do {
                                    for (m=l;m<=n-1;m++) {
                                                dd=Math.abs(d[m-1])+Math.abs(d[m]);
                                                if ((Math.abs(e[m-1])+dd) == dd) break;
                                    }
                                    if (m != l) {
          if (iter++ == 30) System.out.println("Too many iterations in QL");
          g=(d[l]-d[l-1])/(2.0*e[l-1]);
          r=pythag(g,1.0);
          g=d[m-1]-d[l-1]+e[l-1]/(g+SIGN(r,g));
          s=c=1.0;
          p=0.0;
          for (i=m-1;i>=l;i--) {
                                                            f=s*e[i-1];
                                                            b=c*e[i-1];
                                                            e[i]=(r=pythag(f,g));
                                                            if (r == 0.0) {
                                                                        d[i] -= p;
                                                                        e[m-1]=0.0;
                                                                        break;
                                                            }
                                                            s=f/r;
                                                            c=g/r;
                                                            g=d[i ]-p;
                                                            r=(d[i-1]-g)*s+2.0*c*b;
                                                            d[i ]=g+(p=s*r);
                                                            g=c*r-b;
                                                            for (k=1;k<=n;k++) {
                                                                        f=z[k-1][i ];
                                                                        z[k-1][i ]=s*z[k-1][i-1]+c*f;
                                                                        z[k-1][i-1]=c*z[k-1][i-1]-s*f;
                                                            }
                                                }
                                                if (r == 0.0 && i >= l) continue;
                                                d[l-1] -= p;
                                                e[l-1]=g;
                                                e[m-1]=0.0;
                                    }
                        } while (m != l);
            }
   return z;
}

public static double[][] eigenQL(double a[][])
{
 // QL algoritm to solve eigen value problems
```

```
  // symmetric matrices only (real eigen values)
  // first column of the matrix returns eigen values
  // second..n+1 column returns eigen vectors.
  // Note : If matrix is not symmetric DO NOT use
  // this method use eigenValue method (a QR algorithm)
  int i,j;
  int n=a.length;
  double sum[]=new double[n];;
  double d[]=new double[n];
  double b[][]=new double[n][n];
  double e[]=new double[n];
  double z[][]=new double[n+1][n];
  b=tridiagonal(a,d,e);
  b=QL(d,e,b);
  for(j=0;j<n;j++)
    {
    z[0][j]=d[j];
    for(i=0;i<n;i++)
      {
      z[i+1][j]=b[i][j]/b[0][j];
      if(z[i+1][j]<1e-13) z[i+1][i]=0;
      }
    }
  // result: first row is eigenvalues, the rest is eigenvector
  return z;
 }

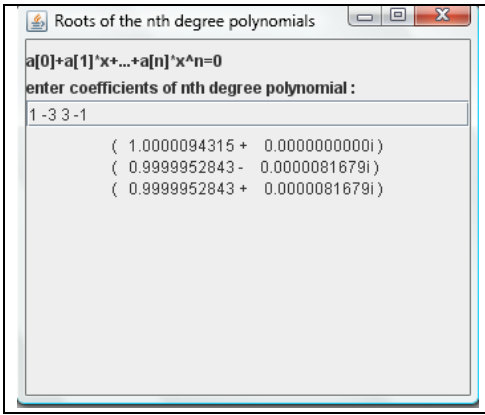
    private JLabel prompt1,prompt2;
    private JTextField input;
    JTextArea t;
    JPanel YaziPaneli;
    int n;
    String s;
    Container c;
    public NA136()
    {
       super("Roots of the nth degree polynomials");
       c=getContentPane();
       c.setLayout(new FlowLayout());
       YaziPaneli=new JPanel();
       YaziPaneli.setFont(new Font("Serif",Font.BOLD,12));
       YaziPaneli.setLayout( new GridLayout(3,1) );
       t=new JTextArea();
       t.setBackground(c.getBackground());
       prompt1= new JLabel("a[0]+a[1]*x+...+a[n]*x^n=0");
       prompt2= new JLabel("enter coefficients of nth degree polynomial : ");
       input = new JTextField(30);
       input.setBackground(c.getBackground());
       YaziPaneli.add(prompt1);
       YaziPaneli.add(prompt2);
       YaziPaneli.add(input);
       c.add(YaziPaneli);
       c.add(t);
       input.addActionListener(this);
    }

    public void  actionPerformed(ActionEvent e)
    {
          s=input.getText();
          StringTokenizer token=new StringTokenizer(s);
          t.setText("");
          n=token.countTokens()-1;
          int m=n+1;
          double a[]=new double[m];
```

```
        c z[]=new c[n];
        for(int i=0;i<n;i++)
        {
        z[i]=new c();
        }
        int j=0;
            while(token.hasMoreTokens())
            {
            Double ax=new Double(token.nextToken());
            a[j++]=ax.doubleValue();
            }
            z=poly_rootsC(a);
            int i=0;
            t.setText(Matrix.toStringT(z));
            input.setText("");
   }

    public static void main(String[] args)
    {
    NA136 pencere= new NA136();
    pencere.addWindowListener(new BasicWindowMonitor());
    pencere.setSize(350,300);
    pencere.setVisible(true);
    }
}
```

Output 9.7-1 Roots of the nth degree polynomial by using QR algorithm



9.10 SINGULAR VALUE DECOMPOSITION FOR SOLVING SYSTEM OF EQUATIONS

Let A be an m x n matrix such that the number of rows m is greater than or equal to the number of columns n. Then there exists:

1) an m x n column orthogonal matrix **U** (a matrix Q is orthogonal if its transpose is equal to its inverse)

2) an n x n diagonal matrix **S,** with positive or zero elements, and

3) an n x n orthogonal matrix V such that:

$$\boldsymbol{A=USV^T}$$

This is the **Singular Value Decomposition (SVD) of matrix A.** If a matrix can be decomposed this way, its inverse value will be existed as:

$$\boldsymbol{A^{-1}=V\ [diag\ (s_1^{-1},s_2^{-1},\ldots,s_n^{-1})]\ (U^T)}$$

It should ve note that because of the orhogonal properties of U and V inverses are equal to transpose values. Then a system of equation can be simply solve as $x=A^{-1}b$

Now let us look at these processes in more details with an example

$$A=\begin{bmatrix}4 & 0\\ 3 & -5\end{bmatrix}$$

The singular values of A are the square roots of the eigenvalues of A^TA

$$A^T=\begin{bmatrix}4 & 3\\ 0 & -5\end{bmatrix}$$

$$A^TA = \begin{bmatrix}4 & 3\\ 0 & -5\end{bmatrix}\begin{bmatrix}4 & 0\\ 3 & -5\end{bmatrix} = \begin{bmatrix}25 & -15\\ -15 & 25\end{bmatrix}$$

Eigenvalues of A^TA is

$\lambda = \begin{bmatrix}40\\ 10\end{bmatrix}$Root of the eigenvalues constructs the matrix S, by using eigenvalues as the diagonal of the matrix. And then the inverse can be easily computed by inverting the eigenvalues.

$$S = \begin{bmatrix}\sqrt{40} & 0\\ 0 & \sqrt{10}\end{bmatrix} = \begin{bmatrix}6.3245 & 0\\ 0 & 3.1622\end{bmatrix}$$

$$S^{-1} = \begin{bmatrix}1/6.3245 & 0\\ 0 & 1/3.1622\end{bmatrix} = \begin{bmatrix}0.1581 & 0\\ 0 & 0.3162\end{bmatrix}$$

The eigenvectors corresponding to these eigenvalues are:

$$[A^TA - \lambda I][X^1] = \begin{bmatrix}25-40 & -15\\ -15 & 25-40\end{bmatrix}\begin{bmatrix}x_1^1\\ x_2^1\end{bmatrix} = 0$$

$$[A^TA - \lambda I][X^2] = \begin{bmatrix}25-10 & -15\\ -15 & 25-10\end{bmatrix}\begin{bmatrix}x_1^2\\ x_2^2\end{bmatrix} = 0$$

$$[X] = \begin{bmatrix}0.707106781186548 & 0.707106781186548\\ -0.707106781186548 & 0.707106781186548\end{bmatrix}$$

The eigenvector values will be normalized by using Length factor

$L = \sqrt{(x_1)^2 + (x_2)^2}$Then the normalized Value will be V vector

$$V = \begin{bmatrix}0.707106781186548 & 0.707106781186548\\ -0.707106781186548 & 0.707106781186548\end{bmatrix}$$

The U vector will be calculated as

$$U = AVS^{-1} = \begin{bmatrix}0.4472 & 0.8944\\ 0.8944 & -0.4472\end{bmatrix}$$

A program is developed to calculate singular value decomposition by using jacobi method. The example problem solved in this program as follows:

$$\begin{bmatrix}3 & 2 & -1\\ 2 & -2 & 4\\ -1 & 0.5 & -1\end{bmatrix}\begin{bmatrix}x_0\\ x_1\\ x_2\end{bmatrix} = \begin{bmatrix}1\\ -2\\ 0\end{bmatrix}$$

Program 9.8.1 Singular value decomposition linear system of equation solving method by using Jacobi eigenvalue method

```
import java.io.*;
import javax.swing.*;

class SVD
{
double U[][];
double V[][];
double S[][];
double invS[][];

public static double[][] Transpose(double [][] left)
{
//transpose matrix (if A=a(i,j) Transpose(A)=a(j,i)
int i,j;
int n=left.length;
int m=left[0].length;
double b[][];
b=new double[m][n];
for(i=0;i<n;i++)
{
for(j=0;j<m;j++)
{
b[j][i]=left[i][j];
}
}
return b;
}

public static double[][] T(double [][] left)
{return Transpose(left);}

public static double[][] I(int n)
{
//unit matrix
double b[][];
```

```
b=new double[n][n];
for(int i=0;i<n;i++)
b[i][i]=1.0;
return b;
}
//Converts vector to diagonal values of matrix
public static double[][] S(double[] left)
{
int n=left.length;
double b[][];
b=new double[n][n];
for(int i=0;i<n;i++)
b[i][i]=left[i];
return b;
}
//Converts double to diagonal values of matrix
public static double[][] S(double left,int n)
{
double b[][];
b=new double[n][n];
for(int i=0;i<n;i++)
b[i][i]=left;
return b;
}
//takes invert value of a diagonal matrix
public static double[][] invS(double[][] S)
{
int n=S.length;
double b[][];
b=new double[n][n];
for(int i=0;i<n;i++)
{  if(S[i][i]==0) b[i][i]=0;
                else b[i][i]=1.0/S[i][i];
}
return b;
}
//square root of a diagonal matrix
public static double[][] sqrtS(double[] left)
{
int n=left.length;
double b[][];
b=new double[n][n];
for(int i=0;i<n;i++)
b[i][i]=Math.sqrt(left[i]);
return b;
}
//multiply two matrices
public static double[][] multiply(double[][] left,double[][] right)
{
//multiplication of two matrices
int ii,jj,i,j,k;
int m1=left[0].length;
int n1=left.length;
int m2=right[0].length;
int n2=right.length;
double[][] b;
b=new double[m1][n2];
if(n1 != m2)
{
System.out.println("m1="+m1+"n1="+n1+"m2="+m2+"n2="+n2+"inner matrix dimensions must agree");
for(ii=0;ii<n1;ii++)
{
for(jj=0;jj<m2;jj++)
b[ii][jj]=0;
}
return b;
}
for(i=0;i<m1;i++)
{
for(j=0;j<n2;j++)
{
for(k=0;k<n1;k++)
                b[i][j]+=left[i][k]*right[k][j];
}
```

```
}
return b;
//end of multiply of two matrices
}
//multiply a matrix with a double value
public static double[][] multiply(double[][] left,double right)
{
//multiplying a matrix with a constant
int i,j;
int n=left.length;
int m=left[0].length;
double b[][];
b=new double[n][m];
for(i=0;i<n;i++)
{
for(j=0;j<m;j++)
b[i][j]=left[i][j]*right;
}
return b;
//end of multiplying a matrix with a constant double
}
//multiply a matrix with a vector
public static double[] multiply(double[][] left,double[] right)
{
//multiplication of one matrix with one vector
int ii,jj,i,j,k;
int m1=left[0].length;
int n1=left.length;
int m2=right.length;
double[] b;
b=new double[m2];
if(n1 != m2)
{
System.out.println("inner matrix dimensions must agree");
for(ii=0;ii<n1;ii++)
{
b[ii]=0;
}
return b;
}
for(i=0;i<m1;i++)
{
b[i]=0;
for(k=0;k<n1;k++)
b[i]+=left[i][k]*right[k];
}
return b;
//end of multiply of a matrix and a vector
}

//substract two matrices
public static double[][] substract(double[][] left,double[][] right)
{
//addition of two matrices
int n1=left.length;
int m1=left[0].length;
int n2=right.length;
int m2=right[0].length;
int nMax,mMax;
int i,j;
if(m1>=m2) mMax=m1;
else       mMax=m2;
if(n1>=n2) nMax=n1;
else       nMax=n2;
double b[][];
b=new double[nMax][mMax];
for(i=0;i<n1;i++)
{
for(j=0;j<m1;j++)
{
b[i][j]=b[i][j]+left[i][j];
}
}
for(i=0;i<n2;i++)
```

```
{
for(j=0;j<m2;j++)
{
b[i][j]=b[i][j]-right[i][j];
}
}
return b;
//end of matrix substraction method
}

public SVD(double A[][])
{initilize(A);}

public static double[] solve(double a[][],double b[])
{ SVD q=new SVD(a);
double x[]=multiply(multiply(q.V,q.invS),multiply(q.T(q.U),b));
return x;
}

public  double[] solve(double b[])
{
double x[]=multiply(multiply(V,invS),multiply(T(U),b));
return x;
}

public void initilize(double A[][])
{
double ATA[][]=multiply(Transpose(A),A);
//double aa[][]=Householder(ATA);
double c[][];
c=eigen_jacobi(ATA);//eigenQL(aa);
int n=c[0].length;
int n1=c.length;
double dummy[]=new double[n1];
for(int i=0;i<n;i++)
{for(int k=i+1;k<n;k++)
{if(c[0][i]<c[0][k])
{ for(int j=0;j<n1;j++)
{ dummy[j]=c[j][k];
c[j][k]=c[j][i];
c[j][i]=dummy[j];
}
}
}
}
double x[]=new double[n];
for(int i=0;i<n;i++)
{x[i]=c[0][i];}
S=sqrtS(x);
invS=invS(S);
double xx[][]=new double[n][n];
V=new double[n][n];
U=new double[n][n];
double L[]=new double[n];
for(int i=0;i<n;i++)
{ for(int j=0;j<n;j++)
{xx[i][j]=c[i+1][j];}
}

double bb[]=new double[n];
for(int j=0;j<n;j++)
{ for(int i=0;i<n;i++)
{L[j]+=xx[i][j]*xx[i][j];}
L[j]=Math.sqrt(L[j]);
}
for(int i=0;i<n;i++)
{ for(int j=0;j<n;j++)
{V[i][j]=xx[i][j]/L[i];}
}
U=multiply(A,multiply(V,invS));
}

public static void print(String s)
{JOptionPane.showMessageDialog(null,s,
```

```
"Singular value decomposition : ",JOptionPane.PLAIN_MESSAGE);}

public static double[][] eigen_jacobi(double A[][])
{
double eps=1e-15;
double err=1;
int m=100;
int n=A.length;
double P[][]=new double[n][n];
double R[][]=new double[n][n];
int p=n,q=n;
double max=-9.99e100;
double fi;
P=I(n);
for(int i=0;i<n;i++)
{for(int j=i+1;j<n;j++)
{if(Math.abs(A[i][j]) > max) {max=Math.abs(A[i][j]);p=i;q=j;}}
}
while(Math.abs(A[p][q]) > eps )
{
fi=0.5*Math.atan(2.0*A[p][q]/(A[p][p]-A[q][q]));
R=I(n);
R[p][p]=Math.cos(fi);
R[q][q]=Math.cos(fi);
R[p][q]=-Math.sin(fi);
R[q][p]=Math.sin(fi);
P=multiply(P,R);
A=multiply(T(R),multiply(A,R));
max=-9.99e100;
for(int i=0;i<n;i++)
{for(int j=i+1;j<n;j++){if(Math.abs(A[i][j])>max) {max=Math.abs(A[i][j]);p=i;q=j;}}}
}
double lambda[]=new double[n];
for(int i=0;i<n;i++) {lambda[i]=A[i][i];}
double PP[][]=new double[n+1][n];
PP[0]=lambda;
for(int i=0;i<n;i++)
{for(int j=0;j<n;j++)
{PP[i+1][j]=P[i][j];}
}
return PP;
}

public static void main(String args[]) throws IOException
{
double a[][]={{3,2,-1},{2,-2,4},{-1,0.5,-1}};
double b[]={1,-2,0};
//Dynamic call to SVD
SVD q=new SVD(a);
double x[]=q.solve(b);
System.out.println("S:\n"+Matrix.toString(q.S));
System.out.println("V:\n"+Matrix.toString(q.V));
System.out.println("U:\n"+Matrix.toString(q.U));
//Static call to SVD
//double x2[]=solve(a,b);
System.out.println("SINGULAR VALUE DECOMPOSITION : \n"+Matrix.toString(x));
//System.out.println("STATIC SINGULAR VALUE DECOMPOSITION : \n"+Matrix.toString(x2));
}
}
```

Output 9.8-1 Singular value decomposition linear system of equation solving method by using Jacobi eigenvalue method results

```
---------- Capture Output ----------
                        "D:\java\bin\javaw.exe" SVD
S:
5.126196943738645       0.000000000000000       0.000000000000000
0.000000000000000       3.734641762863597       0.000000000000000
0.000000000000000       0.000000000000000       0.156702894292504

V:
0.347022679391807       0.881301927095452      -0.320752510957550
-0.450285808796899      0.456574244972559       0.767321737750118
0.822689461694906      -0.121847741569467       0.555279368868110
```

```
U:
-0.133079756513558       0.985074940622975       0.109165652853519
0.953021289812956        0.096947557746556       0.286969671237626
-0.272103288421038       -0.142227045260808      0.951699147854431

SINGULAR VALUE DECOMPOSITION :
1.000000000000170000000000
-2.000000000000042000000000
-2.000000000000300000000000

> Terminated with exit code 0.
```

Program 9.8.2 Singular value decomposition linear system of equation solving method by using Jacobi eigenvalue method (Python code)

```
# -*- coding: utf-8 -*-
"""
Created on Sun Aug 26 07:28:01 2018

@author: Mustafa Turhan Çoban
"""
from math import *;
from eigen_jacobi import *;
def SVD(A):
    #singular value decomposition
    B=AT(A);
    ATA=multiply_m_m(B,A);
    c=eigen_jacobi(ATA);
    n=len(c[0]);
    n1=len(c);
    dummy=[0.0 for x in range(n1)];
    V = [[0.0 for x in range(n)] for y in range(n)];
    U = [[0.0 for x in range(n)] for y in range(n)];
    xx = [[0.0 for x in range(n)] for y in range(n)];
    x=[0.0 for x in range(n)];
    for i in range(n):
        for k in range(i+1,n):
            if c[0][i]<c[0][k]:
                for j in range(n1):
                    dummy[j]=c[j][k];
                    c[j][k]=c[j][i]
                    c[j][i]=dummy[j];
    S = [[0.0 for x in range(n)] for y in range(n)];
    SI = [[0.0 for x in range(n)] for y in range(n)];
    for i in range(0,n):
        x[i]=c[0][i];
        S[i][i]=sqrt(x[i]);
        SI[i][i]=1.0/S[i][i];
    L=[0.0 for x in range(n)];
    for i in range(n):
        for j in range(n):
            xx[i][j]=c[i+1][j];
    for j in range(0,n):
        for i in range(0,n):
          L[j]+=xx[i][j]*xx[i][j];

    for i in range(0,n):
        for j in range(0,n):
          V[i][j]=xx[i][j]/L[j];
    VT=AT(V);
    U=multiply_m_m(A,V);
    U=multiply_m_m(U,SI);
    UT=AT(U);
    #AA=multiply_m_m(U,multiply_m_m(S,VT));
    #BB=multiply_m_m(V,multiply_m_m(SI,UT));
    #CC=multiply_m_m(AA,BB);
    x=[S,V,U];
    return x;

def invB(B):
   #inverse matrix (n!=m)
   x=SVD(B);
   S=x[0];
```

```
    V=x[1];
    U=x[2];
    SI = invS(S);
    UT=AT(U);
    return multiply_m_m(V,multiply_m_m(SI,UT));

def multiply_m_v(left,right):
    #multiplication of one matrix with one vector
    m1=len(left[0]);
    n1=len(left);
    m2=len(right);
    b = [0.0 for i in range(m2)];
    if n1 != m2:
        print("inner matrix dimensions must agree");
        for ii in range(n1):b[ii]=0;
        return b;
    for i in range(m1):
        b[i]=0;
        for k in range(n1):
            b[i]+=left[i][k]*right[k];

def multiply_v_m(left,right):
    #multiplication of one vector with one matrix
    m2=len(right[0]);
    n2=len(right);
    m1=len(left);
    b = [0.0 for i in range(m1)];
    if n2 != m1:print("inner matrix dimensions must agree");
    for ii in range(n2): b[ii]=0;
    for i in range(m2):
        b[ii]=0;
        for k in range(m1):
            b[i]+=right[i][k]*left[k];
    return b;

def invS(S):
    n=len(S);
    SI = [[0.0 for x in range(n)] for y in range(n)];
    for i in range(0,n):
        SI[i][i]=1.0/S[i][i];
    return SI;

def solve(A,B):
    #solution of the system of equation by using SVD
    AI=invB(A);
    X=multiply_v_m(B,AI);
    return X;

A=[[3,2,-1],[2,-2,4],[-1,0.5,-1]];
C=SVD(A);
S=C[0];
V=C[1];
U=C[2];
print("S=\n",S);
print("V=\n",V);
print("U=\n",U);
B=[1,-2,0];
X=solve(A,B);
print("X=",X);
```

```
runfile('D:/okul/SCO1/SVD.py', wdir='D:/okul/SCO1')
Reloaded modules: eigen_jacobi
S=
 [[5.126196943738645, 0.0, 0.0], [0.0, 3.734641762863597, 0.0], [0.0, 0.0, 0.1567028942925037]]
V=
 [[0.3470226793918069, 0.8813019270954525, -0.3207525109575496], [-0.4502858087968988, 0.45657424497255894,
0.7673217377501177], [0.8226894616949058, -0.12184774156946693, 0.5552793688681098]]
U=
 [[-0.13307975651355772, 0.9850749406229755, 0.10916565285351665], [0.9530212898129558, 0.0969475577465563,
0.28696967123762784], [-0.2721032884210384, -0.1422270452608084, 0.9516991478544303]]

X= [1.000000000000028, -2.000000000000068, -2.0000000000000497]
```

9.11 SINGULAR VALUE DECOMPOSITION FOR SOLVING POLYNOMIAL LEAST SQUARE CURVE FITTING PROBLEM

The singular value decomposition can be applied in many different areas. One of which is solving the polynomial least square problem. Let A be an $m \times n$ matrix, with $m > n$, and **b** is a vector. Note that degree of polynomial is (n-1). For a given data set $[x_i,y_i]$ i=0..(m-1) A and b can be represented as:

$$A = \begin{bmatrix} 1 & x_0 & x_0^2 & \dots & x_0^{n-1} \\ 1 & x_1 & x_1^2 & \dots & x_1^{n-1} \\ \dots & \dots & .. & \dots & \dots \\ 1 & x_{m-2} & x_{m-2}^2 & \dots & x_{m-2}^{n-1} \\ 1 & x_{m-1} & x_{m-1}^2 & \dots & x_{m-1}^{n-1} \end{bmatrix}$$

$$b = \begin{bmatrix} y_0 \\ y_1 \\ \dots \\ y_{m-2} \\ y_{m-1} \end{bmatrix}$$

The least squares objective is to find a vector **x** that will minimize $\|A\mathbf{x} - \mathbf{b}\|_2$. Suppose that the singular value decomposition of A is known, $A = U SV^T$, Then

$$\|Ax - b\|_2 = \|USV^T x - UU^T b\|_2 = \|SV^T x - U^T b\|_2$$

Can be written. If $z = V^T x$and $c = U^T b$is defined, Equation becomes:

$$\|Ax - b\|_2 = \|Sz - c\|_2 = \left\{ \sum_{i=1}^{k} (s_i z_i - c_i)^2 + \sum_{i=k+1}^{m} (c_i)^2 \right\}$$

The norm is minimized when the derivatives set equal to zero

$$2(s_i z_i - c_i)\frac{dz_i}{dx_i} = 2(s_i z_i - c_i)V^T = 0 i = 1..k$$

Therefore z_i can be chosen as

$$z_i = \begin{Bmatrix} \left(\frac{c_i}{s_i}\right) i <= k \\ arbitrarynumber i > k \end{Bmatrix}$$

From these least square coefficients can be calculated as x=Vz. A program developed to solve SVD version of polynomial least square approach:

Program 9.11-1 Polynomial least square solution by using singular value decomposition

```
import java.io.*;
import java.util.*;
import javax.swing.*;

class SVDLeastsquare
{

public static double[][] Transpose(double [][] left)
{
//transpose matrix (if A=a(i,j) Transpose(A)=a(j,i)
int i,j;
int n=left.length;
int m=left[0].length;
double b[][];
b=new double[m][n];
for(i=0;i<n;i++)
  {
  for(j=0;j<m;j++)
    {
    b[j][i]=left[i][j];
    }
  }
return b;
}

public static double[][] T(double [][] left)
{return Transpose(left);}
```

```
public static double[][] I(int n)
{
//unit matrix
double b[][];
b=new double[n][n];
for(int i=0;i<n;i++)
   b[i][i]=1.0;
return b;
}
//Converts vector to diagonal values of matrix
public static double[][] S(double[] left,int m)
{
int n=left.length;
double b[][];
b=new double[m][n];
for(int i=0;i<n;i++)
   b[i][i]=left[i];
return b;
}

//Converts double to diagonal values of matrix
public static double[][] S(double left,int n)
{
double b[][];
b=new double[n][n];
for(int i=0;i<n;i++)
   b[i][i]=left;
return b;
}

//takes invert value of a diagonal matrix
public static double[][] invS(double[][] S)
{
int m=S.length;
int n=S[0].length;
double b[][];
b=new double[m][n];
for(int i=0;i<n;i++)
   {  if(S[i][i]==0) b[i][i]=0;
                else b[i][i]=1.0/S[i][i];
              }
return b;
}

//square root of a diagonal matrix
public static double[][] sqrtS(double[] left,int m)
{
int n=left.length;
double b[][];
b=new double[m][n];
for(int i=0;i<n;i++)
   b[i][i]=Math.sqrt(left[i]);
return b;
}
//multiply two matrices
public static double[][] multiply(double[][] left,double[][] right)
{
//multiplication of two matrices
int ii,jj,i,j,k;
int m1=left[0].length;
int n1=left.length;
int m2=right[0].length;
int n2=right.length;
double[][] b;
b=new double[n1][m2];
 if(m1 !=n2 )
 {
 System.out.println("inner matrix dimensions must agree"+"m1="+m1+"n2="+n2);
 return b;
 }
 for(i=0;i<n1;i++)
   {
   for(j=0;j<m2;j++)
```

```
    {
    for(k=0;k<m1;k++)
                {b[i][j]+=left[i][k]*right[k][j];
                }
    }
  }
return b;
//end of multiply of two matrices
}

//multiply a matrix with a double value
public static double[][] multiply(double[][] left,double right)
{
//multiplying a matrix with a constant
int i,j;
int n=left.length;
int m=left[0].length;
double b[][];
b=new double[n][m];
for(i=0;i<n;i++)
  {
  for(j=0;j<m;j++)
    b[i][j]=left[i][j]*right;
  }
return b;
//end of multiplying a matrix with a constant double
}

public static double[] multiply(double[][] left,double[] right)
{
//multiplication of one matrix with one vector
int ii,jj,i,j,k;
int n1=left.length;
int m1=left[0].length;

int m2=right.length;
double[] b;
b=new double[m2];
 for(i=0;i<n1;i++)
   {
    b[i]=0;
    for(k=0;k<m1;k++)
                {b[i]+=left[i][k]*right[k];
                }
   }
return b;
//end of multiply of a matrix and a vector
}

//substract two matrices
public static double[][] substract(double[][] left,double[][] right)
{
//addition of two matrices
int n1=left.length;
int m1=left[0].length;
int n2=right.length;
int m2=right[0].length;
int nMax,mMax;
int i,j;
if(m1>=m2) mMax=m1;
else       mMax=m2;
if(n1>=n2) nMax=n1;
else       nMax=n2;
double b[][];
b=new double[nMax][mMax];
for(i=0;i<n1;i++)
  {
  for(j=0;j<m1;j++)
    {
    b[i][j]=b[i][j]+left[i][j];
    }
  }
for(i=0;i<n2;i++)
  {
```

```
   for(j=0;j<m2;j++)
     {
     b[i][j]=b[i][j]-right[i][j];
     }
   }
return b;
//end of matrix substraction method
}

public static double[] SVD_LSQ(double xi[],double yi[],int n_leastsquare)
{ // Singular value decomposition least square approximation
  double U[][];
  double V[][];
  double S[][];
  double invS[][];
  double A[][]=A(xi,n_leastsquare);
  int m=A.length;
  int m1=A[0].length;
  System.out.println("A="+Matrix.toString(A));
  double ATA[][]=multiply(Transpose(A),A);
  System.out.println("ATA="+Matrix.toString(ATA));
  //double aa[][]=Householder(ATA);
  double cc[][];
  cc=eigen_jacobi(ATA);//eigenQL(aa);
   System.out.println("c="+Matrix.toString(cc));
  int n=cc[0].length;
  int n1=cc.length;
  double dummy[]=new double[n1];
  for(int i=0;i<n;i++)
  {for(int k=i+1;k<n;k++)
    {if(cc[0][i]<cc[0][k])
      { for(int j=0;j<n1;j++)
        { dummy[j]=cc[j][k];
          cc[j][k]=cc[j][i];
          cc[j][i]=dummy[j];
        }
      }
    }
  }
  double x[]=new double[n];
  for(int i=0;i<n;i++)
  {x[i]=cc[0][i];}
  S=sqrtS(x,m1);
  System.out.println("S="+Matrix.toString(S));
  invS=invS(S);
  System.out.println("invS="+Matrix.toString(invS));
  double xx[][]=new double[n][n];
  V=new double[n][n];
  U=new double[n][n];
  double L[]=new double[n];

  for(int i=0;i<n;i++)
  { for(int j=0;j<n;j++)
    {xx[i][j]=cc[i+1][j];}
  }

  double bb[]=new double[n];
  for(int j=0;j<n;j++)
  { for(int i=0;i<n;i++)
    {L[j]+=xx[i][j]*xx[i][j];}
    L[j]=Math.sqrt(L[j]);
  }
  for(int i=0;i<n;i++)
  { for(int j=0;j<n;j++)
    {V[i][j]=xx[i][j]/L[i];}
  }
  System.out.println("VT="+Matrix.toString(T(V)));
  U=multiply(A,multiply(V,invS));
   System.out.println("U="+Matrix.toString(U));
  double c[]=multiply(T(U),yi);
   System.out.println("c="+Matrix.toString(c));
   double z[]=new double[n];
   for(int i=0;i<n;i++) z[i]=c[i]/S[i][i];
   System.out.println("z="+Matrix.toString(z));
```

```
   double a[]=multiply(V,z);
   System.out.println("a="+Matrix.toString(a));
   return a;
}
public static double func(double e[],double x)
{
// this function calculates the value of
// least square curve fitting function
int n=e.length;
double ff;
if(n!=0.0)
  { ff=e[n-1];
  for(int i=n-2;i>=0;i--)
   { ff=ff*x+e[i]; }
  }
else
  ff=0;
return ff;
}

public static double error(double x[],double y[],double e[])
{
//calculates absolute square root error of a least square approach
double n=x.length;
 int k;
 double total=0;
 for(k=0;k<n;k++)
 {
 total+=(y[k]-func(e,x[k]))*(y[k]-func(e,x[k]));
 }
total=Math.sqrt(total);
return total;
}

public static double[][]  funcSVD_LSQ(double xi[],double yi[],int polinomkatsayisi,int aradegersayisi)
 {
                //aradegersayisi:  x--o--o--x--o--o--x zincirinde x deneysel noktalar ise
                // ara değer sayısı 2 dir
                 int n=xi.length;
                int nn=(n-1)*(aradegersayisi+1)+1;
                double z[][]=new double[2][nn];
   double E[]=SVD_LSQ(xi,yi,polinomkatsayisi);
   System.out.println("coefficients :\n"+Matrix.toStringT(E));
   double dx=0;
   int k=0;
   int i;
   for(i=0;i<(n-1);i++)
      {z[0][k]=xi[i];z[1][k]=func(E,z[0][k]);k++;
      for(int j=0;j<aradegersayisi;j++)
      {dx=(xi[i+1]-xi[i])/((double)aradegersayisi+1.0);
      z[0][k]=z[0][k-1]+dx;z[1][k]=func(E,z[0][k]);k++;}
      }
      z[0][k]=xi[i];z[1][k]=func(E,z[0][k]);
   return z;
 }
  public static void print(String s)
  {JOptionPane.showMessageDialog(null,s,
    "Singular value decomposition : ",JOptionPane.PLAIN_MESSAGE);}

  public static double[][] eigen_jacobi(double A[][])
  {             //Eigen value calculation
   double eps=1e-15;
   double err=1;
   int m=100;
   int n=A.length;
   double P[][]=new double[n][n];
   double R[][]=new double[n][n];
   int p=n,q=n;
   double max=-9.99e100;
   double fi;
   P=I(n);
   for(int i=0;i<n;i++)
   {for(int j=i+1;j<n;j++)
      {if(Math.abs(A[i][j]) > max) {max=Math.abs(A[i][j]);p=i;q=j;}}
```

```
   }
   while(Math.abs(A[p][q]) > eps )
   {
    fi=0.5*Math.atan2(2.0*A[p][q],(A[p][p]-A[q][q]));
    R=I(n);
    R[p][p]=Math.cos(fi);
    R[q][q]=Math.cos(fi);
    R[p][q]=-Math.sin(fi);
    R[q][p]=Math.sin(fi);
    P=multiply(P,R);
    A=multiply(T(R),multiply(A,R));
    max=-9.99e100;
    for(int i=0;i<n;i++)
   {for(int j=i+1;j<n;j++){if(Math.abs(A[i][j])>max) {max=Math.abs(A[i][j]);p=i;q=j;}}}
    }
   double lambda[]=new double[n];
   for(int i=0;i<n;i++) {lambda[i]=A[i][i];}
   double PP[][]=new double[n+1][n];
   PP[0]=lambda;
   for(int i=0;i<n;i++)
   {for(int j=0;j<n;j++)
     {PP[i+1][j]=P[i][j];}
   }
   return PP;
  }

  public static double[] gausswithpartialpivot(double a[][],double b[])
  { // Gauss elimination with partial pivoting
  int n=b.length;
  double x[]=new double[n];
  double carpan=0;
  double toplam=0;
  double buyuk;
  double dummy=0;
  //gauss elimination
  int i,j,k,p,ii,jj;
  for(k=0;k<(n-1);k++)
  { //partial pivoting
                p=k;
                buyuk=Math.abs(a[k][k]);
                for(ii=k+1;ii<n;ii++)
                {  dummy=Math.abs(a[ii][k]);
                   if(dummy > buyuk) {buyuk=dummy;p=ii;}
                }
                if(p!=k)
                { for(jj=k;jj<n;jj++)
                  { dummy=a[p][jj];
                    a[p][jj]=a[k][jj];
                    a[k][jj]=dummy;
                  }
                  dummy=b[p];
                  b[p]=b[k];
                  b[k]=dummy;
                }
                //
    for(i=k+1;i<n;i++)
    {  carpan=a[i][k]/a[k][k];
       a[i][k]=0;
       for(j=k+1;j<n;j++)
       {   a[i][j]-=carpan*a[k][j]; }
           b[i]   =b[i]   -carpan*b[k];
                }
  }
  // backward substitution
  x[n-1]=b[n-1]/a[n-1][n-1];
  for(i=n-2;i>=0;i--)
  {
    toplam=0;
    for(j=i+1;j<n;j++)
    {  toplam+=a[i][j]*x[j];}
    x[i]=(b[i]-toplam)/a[i][i];
  }
  return x;
 }
```

```
public static double[][] A(double xi[],int n_leastsquare)
{
int n=xi.length;
int m=n_leastsquare+1;
int l=xi.length;
int i,j,k;
double A[][];
A=new double[n][m];
double B[];
B=new double[n];
double X[];
X=new double[n];
double cc=1.0;
for(i=0;i<n;i++)
  {
    cc=1.0;
              for(j=0;j<m;j++)
    {
              A[i][j]=cc;
              cc*=xi[i];
    }
  }
return A;
}

  public static void main(String args[]) throws IOException
  {
  double x[];
  double y[];
  String s1=JOptionPane.showInputDialog("Data file name (a.txt)");
  // JFileChooser fc=new JFileChooser();
  // if (fc.showOpenDialog(null) == JFileChooser.APPROVE_OPTION) {File file = fc.getSelectedFile();s1=file.getName(); }
  double a[][]=Text.readDoubleT(s1);
  x=a[0];
  y=a[1];
  int n=Integer.parseInt(JOptionPane.showInputDialog("polinom derecesi (n)"));
  double z[][]=funcSVD_LSQ(x,y,n,4);
  System.out.println("Polynomial Least Square Curve fitting\n"+Matrix.toStringT(z));
  Plot pp=new Plot(a[0],a[1]);
  pp.setPlabel("Polynomial Singular Value Decomposition Least Square Curve fitting");
  pp.setXlabel("x");
  pp.setYlabel("y=f(x)");
  pp.setPlotType(0,22);
  pp.addData(z[0],z[1]);
  pp.setPlotType(1,0);
  pp.setGrid(1,1);
  pp.plot();
  }
}
```

Girdi verisi:

```
0 1
.25 1.2840
0.5 1.6487
0.75 2.1170
1 2.7183
```

Output 9.11-1 Polynomial least square application

```
---------- Capture Output ----------
> "C:\turhan\java\bin\javaw.exe" SVDLeastsquare
A=          1.000000000000000       0.000000000000000       0.000000000000000
            1.000000000000000       0.250000000000000       0.062500000000000
            1.000000000000000       0.500000000000000       0.250000000000000
            1.000000000000000       0.750000000000000       0.562500000000000
            1.000000000000000       1.000000000000000       1.000000000000000

ATA=        5.000000000000000       2.500000000000000       1.875000000000000
            2.500000000000000       1.875000000000000       1.562500000000000
            1.875000000000000       1.562500000000000       1.382812500000000
```

```
c=      7.353236163159007    0.878108943148838    0.026467393692153
        0.798724502630908   -0.592863568473215    0.102722724234778
        0.471216204284530    0.510173089573159   -0.719498928070819
        0.374158332416491    0.623086035674515    0.686854667619525

S=      2.711685115045441    0.000000000000000    0.000000000000000
        0.000000000000000    0.937074673197840    0.000000000000000
        0.000000000000000    0.000000000000000    0.162688025656939

invS=   0.368774381085631    0.000000000000000    0.000000000000000
        0.000000000000000    1.067150813699214    0.000000000000000
        0.000000000000000    0.000000000000000    6.146733885066024

VT=     0.798724502630908    0.471216204284530    0.374158332416491
       -0.592863568473215    0.510173089573159    0.623086035674515
        0.102722724234778   -0.719498928070819    0.686854667619525

U=      0.294549134115641   -0.632674839508811    0.631409249820203
        0.346616000605336   -0.455009009469034   -0.210363306816900
        0.415930368028146   -0.194227333182423   -0.524396756007415
        0.606301605673112    0.576683558131304    0.430753667951314

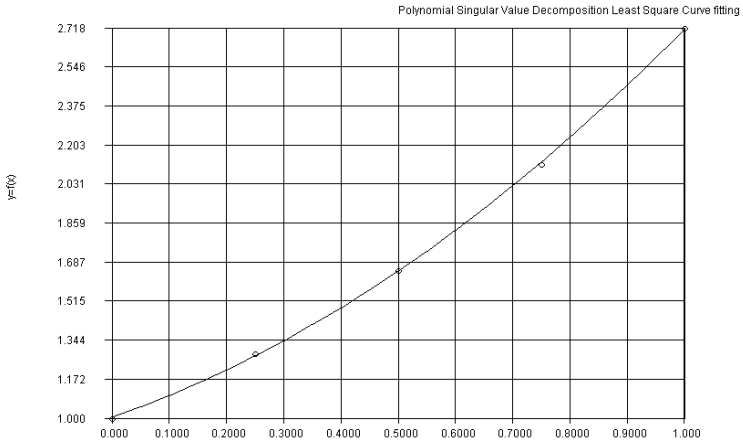
c=      4.137234195787196000000000    0.347321695039527500000000    0.009914473890339837000000000
0.000000000000000000000000    0.000000000000000000000000

z=      1.525705979957730400000000    0.370644629476822100000000    0.060941632614354244000000

a=      1.005137142857140000000000    0.864182857142866600000000    0.843657142857138200000000

coefficients :
      1.005137142857140000000000
      0.864182857142866600000000
      0.843657142857138200000000

Polynomial Least Square Curve fitting
  0.0000000000   1.0051371429
  0.0500000000   1.0504554286
  0.1000000000   1.0999920000
  0.1500000000   1.1537468571
  0.2000000000   1.2117200000
  0.2500000000   1.2739114286
  0.3000000000   1.3403211429
  0.3500000000   1.4109491429
  0.4000000000   1.4857954286
  0.4500000000   1.5648600000
  0.5000000000   1.6481428571
  0.5500000000   1.7356440000
  0.6000000000   1.8273634286
  0.6500000000   1.9233011429
  0.7000000000   2.0234571429
  0.7500000000   2.1278314286
  0.8000000000   2.2364240000
  0.8500000000   2.3492348571
  0.9000000000   2.4662640000
  0.9500000000   2.5875114286
  1.0000000000   2.7129771429
```



The same least square concept can be apply to a more general linear least square formula as well. If a general least square equation is defined as

$$f(x) = \sum_{i=0}^{n} a_i \varphi_i(x)$$

Then A matrix becomes:

$$A = \begin{bmatrix} \varphi_0(x_0) & \varphi_1(x_0) & \varphi_2(x_0) & \dots & \varphi_n(x_0) \\ \varphi_0(x_1) & \varphi_1(x_1) & \varphi_2(x_1) & \dots & \varphi_n(x_1) \\ \dots & \dots & \dots & \dots & \dots \\ \varphi_0(x_{m-1}) & \varphi_1(x_{m-1}) & \varphi_2(x_{m-1}) & \dots & \varphi_n(x_{m-1}) \\ \varphi_0(x_m) & \varphi_1(x_m) & \varphi_2(x_m) & \dots & \varphi_n(x_m) \end{bmatrix}$$

The remaining of the problem will be similar to the previous case of polynomial least square, since this definition is more general, it will also cover the polynomial least square as well ($\varphi_0(x) = 1, \varphi_1(x) = x, \varphi_2(x) = x^2, \dots$). A modified version of the program is given for this case. As an example the same polynomial least square problem is solved.

Program 9.11-2 General linear least square solution by using singular value decomposition

```
import java.io.*;
import java.util.*;
import javax.swing.*;

abstract class f_xr
{  abstract double func(double x,int equation_ref);}

class fa extends f_xr
{
            double func(double x,int i)
            {
    double xx=1.0;
     if(i==0) xx=1.0;
     else if(i==1) xx=x;
     else if(i==2) xx=x*x;
     else if(i==3) xx=x*x*x;
     else if(i==4) xx=x*x*x*x;
     else if(i==5) xx=x*x*x*x*x;
     else if(i==6) xx=x*x*x*x*x*x;
     else if(i==7) xx=x*x*x*x*x*x*x;
     else if(i==8) xx=x*x*x*x*x*x*x*x;
     else if(i==9) xx=x*x*x*x*x*x*x*x*x;

  return xx;
            }
}

class SVDGeneralleastsquare
{

public static double[][] Transpose(double [][] left)
```

```
{
//transpose matrix (if A=a(i,j) Transpose(A)=a(j,i)
int i,j;
int n=left.length;
int m=left[0].length;
double b[][];
b=new double[m][n];
for(i=0;i<n;i++)
  {
  for(j=0;j<m;j++)
    {
    b[j][i]=left[i][j];
    }
  }
return b;
}

public static double[][] T(double [][] left)
{return Transpose(left);}

public static double[][] I(int n)
{
//unit matrix
double b[][];
b=new double[n][n];
for(int i=0;i<n;i++)
    b[i][i]=1.0;
return b;
}
//Converts vector to diagonal values of matrix
public static double[][] S(double[] left,int m)
{
int n=left.length;
double b[][];
b=new double[m][n];
for(int i=0;i<n;i++)
    b[i][i]=left[i];
return b;
}

//Converts double to diagonal values of matrix
public static double[][] S(double left,int n)
{
double b[][];
b=new double[n][n];
for(int i=0;i<n;i++)
    b[i][i]=left;
return b;
}

//takes invert value of a diagonal matrix
public static double[][] invS(double[][] S)
{
int m=S.length;
int n=S[0].length;
double b[][];
b=new double[m][n];
for(int i=0;i<n;i++)
    {  if(S[i][i]==0) b[i][i]=0;
                else b[i][i]=1.0/S[i][i];
              }
return b;
}

//square root of a diagonal matrix
public static double[][] sqrtS(double[] left,int m)
```

```
{
int n=left.length;
double b[][];
b=new double[m][n];
for(int i=0;i<n;i++)
    b[i][i]=Math.sqrt(left[i]);
return b;
}
//multiply two matrices
public static double[][] multiply(double[][] left,double[][] right)
{
//multiplication of two matrices
int ii,jj,i,j,k;
int m1=left[0].length;
int n1=left.length;
int m2=right[0].length;
int n2=right.length;
double[][] b;
b=new double[n1][m2];
 if(m1 !=n2 )
 {
 System.out.println("inner matrix dimensions must agree"+"m1="+m1+"n2="+n2);
 return b;
 }
 for(i=0;i<n1;i++)
   {
   for(j=0;j<m2;j++)
     {
     for(k=0;k<m1;k++)
                 {b[i][j]+=left[i][k]*right[k][j];
                 }
     }
   }
return b;
//end of multiply of two matrices
}

//multiply a matrix with a double value
public static double[][] multiply(double[][] left,double right)
{
//multiplying a matrix with a constant
int i,j;
int n=left.length;
int m=left[0].length;
double b[][];
b=new double[n][m];
for(i=0;i<n;i++)
  {
  for(j=0;j<m;j++)
    b[i][j]=left[i][j]*right;
  }
return b;
//end of multiplying a matrix with a constant double
}

public static double[] multiply(double[][] left,double[] right)
{
//multiplication of one matrix with one vector
int ii,jj,i,j,k;
int n1=left.length;
int m1=left[0].length;

int m2=right.length;
double[] b;
b=new double[m2];
 for(i=0;i<n1;i++)
```

```
   {
     b[i]=0;
     for(k=0;k<m1;k++)
                {b[i]+=left[i][k]*right[k];
                }
   }
return b;
//end of multiply of a matrix and a vector
}

//substract two matrices
public static double[][] substract(double[][] left,double[][] right)
{
//addition of two matrices
int n1=left.length;
int m1=left[0].length;
int n2=right.length;
int m2=right[0].length;
int nMax,mMax;
int i,j;
if(m1>=m2) mMax=m1;
else       mMax=m2;
if(n1>=n2) nMax=n1;
else       nMax=n2;
double b[][];
b=new double[nMax][mMax];
for(i=0;i<n1;i++)
  {
  for(j=0;j<m1;j++)
    {
    b[i][j]=b[i][j]+left[i][j];
    }
  }
for(i=0;i<n2;i++)
  {
  for(j=0;j<m2;j++)
    {
    b[i][j]=b[i][j]-right[i][j];
    }
  }
return b;
//end of matrix substraction method
}

public static double[] SVD_LSQ(f_xr f,double xi[],double yi[],int n_leastsquare)
{ // Singular value decomposition least square approximation
  double U[][];
  double V[][];
  double S[][];
  double invS[][];
  double A[][]=A(f,xi,n_leastsquare);
  int m=A.length;
  int m1=A[0].length;
  System.out.println("A="+Matrix.toString(A));
  double ATA[][]=multiply(Transpose(A),A);
  System.out.println("ATA="+Matrix.toString(ATA));
  //double aa[][]=Householder(ATA);
  double cc[][];
  cc=eigen_jacobi(ATA);//eigenQL(aa);
   System.out.println("c="+Matrix.toString(cc));
  int n=cc[0].length;
  int n1=cc.length;
  double dummy[]=new double[n1];
  for(int i=0;i<n;i++)
  {for(int k=i+1;k<n;k++)
    {if(cc[0][i]<cc[0][k])
```

```
      { for(int j=0;j<n1;j++)
        { dummy[j]=cc[j][k];
          cc[j][k]=cc[j][i];
          cc[j][i]=dummy[j];
        }
      }
    }
  }
  double x[]=new double[n];
  for(int i=0;i<n;i++)
  {x[i]=cc[0][i];}
  S=sqrtS(x,m1);
  System.out.println("S="+Matrix.toString(S));
  invS=invS(S);
  System.out.println("invS="+Matrix.toString(invS));
  double xx[][]=new double[n][n];
  V=new double[n][n];
  U=new double[n][n];
  double L[]=new double[n];

  for(int i=0;i<n;i++)
  { for(int j=0;j<n;j++)
    {xx[i][j]=cc[i+1][j];}
  }

  double bb[]=new double[n];
  for(int j=0;j<n;j++)
  { for(int i=0;i<n;i++)
    {L[j]+=xx[i][j]*xx[i][j];}
    L[j]=Math.sqrt(L[j]);
  }
  for(int i=0;i<n;i++)
  { for(int j=0;j<n;j++)
    {V[i][j]=xx[i][j]/L[i];}
  }
  System.out.println("VT="+Matrix.toString(T(V)));
  U=multiply(A,multiply(V,invS));
   System.out.println("U="+Matrix.toString(U));
  double c[]=multiply(T(U),yi);
   System.out.println("c="+Matrix.toString(c));
   double z[]=new double[n];
   for(int i=0;i<n;i++) z[i]=c[i]/S[i][i];
   System.out.println("z="+Matrix.toString(z));
   double a[]=multiply(V,z);
   System.out.println("a="+Matrix.toString(a));
   return a;
}
public static double func(f_xr f,double e[],double x)
{
// this function calculates the value of
// least square curve fitting function
int n=e.length;
double ff;
if(n!=0.0)
  { ff=0;
  for(int i=0;i<e.length;i++)
   { ff+=e[i]*f.func(x,i); }
  }
else
  ff=0;
return ff;
}

public static double error(f_xr f,double x[],double y[],double e[])
{
//calculates absolute square root error of a least square approach
```

```
double n=x.length;
 int k;
 double total=0;
 for(k=0;k<n;k++)
 {
 total+=(y[k]-func(f,e,x[k]))*(y[k]-func(f,e,x[k]));
 }
total=Math.sqrt(total);
return total;
}

public static double[][]  funcSVD_LSQ(f_xr f,double xi[],double yi[],int polinomkatsayisi,int aradegersayisi)
 {
              //aradegersayisi:  x--o--o--x--o--o--x zincirinde x deneysel noktalar ise
              // ara değer sayısı 2 dir
               int n=xi.length;
              int nn=(n-1)*(aradegersayisi+1)+1;
              double z[][]=new double[2][nn];
   double E[]=SVD_LSQ(f,xi,yi,polinomkatsayisi);
   System.out.println("coefficients :\n"+Matrix.toStringT(E));
   double dx=0;
   int k=0;
   int i;
   for(i=0;i<(n-1);i++)
      {z[0][k]=xi[i];z[1][k]=func(f,E,z[0][k]);k++;
      for(int j=0;j<aradegersayisi;j++)
      {dx=(xi[i+1]-xi[i])/((double)aradegersayisi+1.0);
      z[0][k]=z[0][k-1]+dx;z[1][k]=func(f,E,z[0][k]);k++;}
      }
      z[0][k]=xi[i];z[1][k]=func(f,E,z[0][k]);
   return z;
 }

 public static void print(String s)
 {JOptionPane.showMessageDialog(null,s,
    "Singular value decomposition : ",JOptionPane.PLAIN_MESSAGE);}

 public static double[][] eigen_jacobi(double A[][])
 {             //Eigen value calculation
  double eps=1e-15;
  double err=1;
  int m=100;
  int n=A.length;
  double P[][]=new double[n][n];
  double R[][]=new double[n][n];
  int p=n,q=n;
  double max=-9.99e100;
  double fi;
  P=I(n);
  for(int i=0;i<n;i++)
  {for(int j=i+1;j<n;j++)
     {if(Math.abs(A[i][j]) > max) {max=Math.abs(A[i][j]);p=i;q=j;}}
  }
  while(Math.abs(A[p][q]) > eps )
  {
   fi=0.5*Math.atan2(2.0*A[p][q],(A[p][p]-A[q][q]));
   R=I(n);
   R[p][p]=Math.cos(fi);
   R[q][q]=Math.cos(fi);
   R[p][q]=-Math.sin(fi);
   R[q][p]=Math.sin(fi);
   P=multiply(P,R);
   A=multiply(T(R),multiply(A,R));
   max=-9.99e100;
   for(int i=0;i<n;i++)
  {for(int j=i+1;j<n;j++){if(Math.abs(A[i][j])>max) {max=Math.abs(A[i][j]);p=i;q=j;}}}
```

```
   }
  double lambda[]=new double[n];
  for(int i=0;i<n;i++) {lambda[i]=A[i][i];}
  double PP[][]=new double[n+1][n];
  PP[0]=lambda;
  for(int i=0;i<n;i++)
  {for(int j=0;j<n;j++)
    {PP[i+1][j]=P[i][j];}
  }
  return PP;
 }

 public static double[] gausswithpartialpivot(double a[][],double b[])
 { // Gauss elimination with partial pivoting
 int n=b.length;
 double x[]=new double[n];
 double carpan=0;
 double toplam=0;
 double buyuk;
 double dummy=0;
 //gauss elimination
 int i,j,k,p,ii,jj;
 for(k=0;k<(n-1);k++)
 { //partial pivoting
            p=k;
            buyuk=Math.abs(a[k][k]);
            for(ii=k+1;ii<n;ii++)
            {  dummy=Math.abs(a[ii][k]);
              if(dummy > buyuk) {buyuk=dummy;p=ii;}
            }
            if(p!=k)
            { for(jj=k;jj<n;jj++)
              { dummy=a[p][jj];
                a[p][jj]=a[k][jj];
                a[k][jj]=dummy;
              }
              dummy=b[p];
              b[p]=b[k];
              b[k]=dummy;
            }
            //
   for(i=k+1;i<n;i++)
   {  carpan=a[i][k]/a[k][k];
      a[i][k]=0;
      for(j=k+1;j<n;j++)
      {   a[i][j]-=carpan*a[k][j]; }
         b[i]   =b[i]   -carpan*b[k];
            }
 }
 // backward substitution
 x[n-1]=b[n-1]/a[n-1][n-1];
 for(i=n-2;i>=0;i--)
 {
   toplam=0;
   for(j=i+1;j<n;j++)
   {  toplam+=a[i][j]*x[j];}
   x[i]=(b[i]-toplam)/a[i][i];
 }
 return x;
}

public static double[][] A(f_xr f,double xi[],int n_leastsquare)
{
int n=xi.length;
int m=n_leastsquare+1;
int l=xi.length;
```

```
int i,j,k;
double A[][];
A=new double[n][m];
double B[];
B=new double[n];
double X[];
X=new double[n];
double cc=1.0;
for(i=0;i<n;i++)
  {  for(j=0;j<m;j++) { A[i][j]=f.func(xi[i],j);}}
return A;
}

  public static void main(String args[]) throws IOException
  {
  double x[];
  double y[];
  String s1=JOptionPane.showInputDialog("Data file name (a.txt)");
  // JFileChooser fc=new JFileChooser();
  // if (fc.showOpenDialog(null) == JFileChooser.APPROVE_OPTION) {File file = fc.getSelectedFile();s1=file.getName();
}
  double a[][]=Text.readDoubleT(s1);
  x=a[0];
  y=a[1];
  int n=Integer.parseInt(JOptionPane.showInputDialog("polinom derecesi (n)"));
  fa f1=new fa();
  double z[][]=funcSVD_LSQ(f1,x,y,n,4);
  System.out.println("Polynomial Least Square Curve fitting\n"+Matrix.toStringT(z));
  Plot pp=new Plot(a[0],a[1]);
  pp.setPlabel("Polynomial Singular Value Decomposition Least Square Curve fitting");
  pp.setXlabel("x");
  pp.setYlabel("y=f(x)");
  pp.setPlotType(0,22);
  pp.addData(z[0],z[1]);
  pp.setPlotType(1,0);
  pp.setGrid(1,1);
  pp.plot();
  }
}
```

```
---------- Capture Output ----------
> "D:\java\bin\javaw.exe" SVDGeneralleastsquare
A=       1.000000000000000       0.000000000000000       0.000000000000000
       1.000000000000000       0.250000000000000       0.062500000000000
       1.000000000000000       0.500000000000000       0.250000000000000
       1.000000000000000       0.750000000000000       0.562500000000000
       1.000000000000000       1.000000000000000       1.000000000000000

ATA=       5.000000000000000       2.500000000000000       1.875000000000000
       2.500000000000000       1.875000000000000       1.562500000000000
       1.875000000000000       1.562500000000000       1.382812500000000

c=       7.353236163159007       0.878108943148838       0.026467393692153
       0.798724502630908      -0.592863568473215       0.102722724234778
       0.471216204284530       0.510173089573159      -0.719498928070819
       0.374158332416491       0.623086035674515       0.686854667619525

S=       2.711685115045441       0.000000000000000       0.000000000000000
       0.000000000000000       0.937074673197840       0.000000000000000
       0.000000000000000       0.000000000000000       0.162688025656939

invS=       0.368774381085631       0.000000000000000       0.000000000000000
       0.000000000000000       1.067150813699214       0.000000000000000
       0.000000000000000       0.000000000000000       6.146733885066024

VT=       0.798724502630908       0.471216204284530       0.374158332416491
      -0.592863568473215       0.510173089573159       0.623086035674515
       0.102722724234778      -0.719498928070819       0.686854667619525
```

```
U=      0.294549134115641     -0.632674839508811      0.631409249820203
      0.346616000605336     -0.455009009469034     -0.210363306816900
      0.415930368028146     -0.194227333182423     -0.524396756007415
      0.502492236384071      0.149670189351023     -0.310691097751344
      0.606301605673112      0.576683558131304      0.430753667951314

c=      4.13723419578719600000000000          0.34732169503952750000000000          0.00991447389033983700000000
0.00000000000000000000000000      0.00000000000000000000000000

z=      1.52570597995773040000000000          0.37064462947682210000000000          0.06094163261435424400000000

a=      1.00513714285714000000000000          0.86418285714286660000000000          0.84365714285713820000000000

coefficients :
      1.00513714285714000000000000
      0.86418285714286660000000000
      0.84365714285713820000000000

Polynomial Least Square Curve fitting
  0.0000000000  1.0051371429
  0.0500000000  1.0504554286
  0.1000000000  1.0999920000
  0.1500000000  1.1537468571
  0.2000000000  1.2117200000
  0.2500000000  1.2739114286
  0.3000000000  1.3403211429
  0.3500000000  1.4109491429
  0.4000000000  1.4857954286
  0.4500000000  1.5648600000
  0.5000000000  1.6481428571
  0.5500000000  1.7356440000
  0.6000000000  1.8273634286
  0.6500000000  1.9233011429
  0.7000000000  2.0234571429
  0.7500000000  2.1278314286
  0.8000000000  2.2364240000
  0.8500000000  2.3492348571
  0.9000000000  2.4662640000
  0.9500000000  2.5875114286
  1.0000000000  2.7129771429
```

As the last example let us look at SVD polynomial curve fitting for two dimensional case (can be applied to multidimensions) Curve fitting data is created by using the function

$f(x, y) = 1 + x + y + xy + x^2 + y^2$

0	0	1	0	0.3	1.39	0	0.6	1.96	0	1	3
0.1	0	1.11	0.1	0.3	1.53	0.1	0.6	2.13	0.1	1	3.21
0.2	0	1.24	0.2	0.3	1.69	0.2	0.6	2.32	0.2	1	3.44
0.3	0	1.39	0.3	0.3	1.87	0.3	0.6	2.53	0.3	1	3.69
0.4	0	1.56	0.4	0.3	2.07	0.4	0.6	2.76	0.4	1	3.96
0.5	0	1.75	0.5	0.3	2.29	0.5	0.6	3.01	0.5	1	4.25
0.6	0	1.96	0.6	0.3	2.53	0.6	0.6	3.28	0.6	1	4.56
0.7	0	2.19	0.7	0.3	2.79	0.7	0.6	3.57	0.7	1	4.89
0.8	0	2.44	0.8	0.3	3.07	0.8	0.6	3.88	0.8	1	5.24
0.9	0	2.71	0.9	0.3	3.37	0.9	0.6	4.21	0.9	1	5.61
1	0	3	1	0.3	3.69	1	0.6	4.56	1	1	6
0	0.1	1.11	0	0.4	1.56	0	0.7	2.19			
0.1	0.1	1.23	0.1	0.4	1.71	0.1	0.8	2.63			
0.2	0.1	1.37	0.2	0.4	1.88	0.2	0.8	2.84			
0.3	0.1	1.53	0.3	0.4	2.07	0.3	0.8	3.07			
0.4	0.1	1.71	0.4	0.4	2.28	0.4	0.8	3.32			
0.5	0.1	1.91	0.5	0.4	2.51	0.5	0.8	3.59			
0.6	0.1	2.13	0.6	0.4	2.76	0.6	0.8	3.88			
0.7	0.1	2.37	0.7	0.4	3.03	0.7	0.8	4.19			
0.8	0.1	2.63	0.8	0.4	3.32	0.8	0.8	4.52			
0.9	0.1	2.91	0.9	0.4	3.63	0.9	0.8	4.87			
1	0.1	3.21	1	0.4	3.96	.0	0.9	2.71			
0	0.2	1.24	0	0.5	1.75	0.1	0.9	2.91			
0.1	0.2	1.37	0.1	0.5	1.91	0.2	0.9	3.13			
0.2	0.2	1.52	0.2	0.5	2.09	0.3	0.9	3.37			
0.3	0.2	1.69	0.3	0.5	2.29	0.4	0.9	3.63			
0.4	0.2	1.88	0.4	0.5	2.51	0.5	0.9	3.91			
0.5	0.2	2.09	0.5	0.5	2.75	0.6	0.9	4.21			
0.6	0.2	2.32	0.6	0.5	3.01	0.7	0.9	4.53			

0.7	0.2	2.57	0.7	0.5	3.29	0.8	0.9	4.87	
0.8	0.2	2.84	0.8	0.5	3.59	0.9	0.9	5.23	
0.9	0.2	3.13	0.9	0.5	3.91	1	0.9	5.61	
1	0.2	3.44	1	0.5	4.25		0.8	5.24	

Program 9.11-3 General multidimensional linear least square solution by using singular value decomposition

```
import java.io.*;
import java.util.*;
import javax.swing.*;

// surface curve fitting
//f(x,y)=a0+a1*x+a2*y+a3*x^2+a4*y^2+a5*xy+a6*x^3+a7*y^3+a8*x^2y
//      +a9*xy^2+a10*x^4+a11*y^4+a12*x^2y^2+a13x^3y+a14*xy^3
abstract class f_xir
{
  abstract double func(double x[],int equation_ref);
}

class fa extends f_xir
{
              double func(double x[],int i)
              { // a general surface curve fitting model
   double xx=0.0;
   //density function
   if(i==0)      {xx=1.0;          }
   else if (i==1) {xx=x[0];         }
   else if (i==2) {xx=x[1];         }
   else if (i==3) {xx=x[0]*x[0];    }
   else if (i==4) {xx=x[1]*x[1];    }
   else if (i==5) {xx=x[0]*x[1];    }
   else if (i==6) {xx=x[0]*x[0]*x[0];}
   else if (i==7) {xx=x[1]*x[1]*x[1];}
   else if (i==8) {xx=x[0]*x[0]*x[1];}
   else if (i==9) {xx=x[0]*x[1]*x[1];}
   else if (i==10){xx=x[0]*x[0]*x[0]*x[0];}
   else if (i==11){xx=x[1]*x[1]*x[1]*x[1];}
   else if (i==12){xx=x[0]*x[1]*x[1]*x[1];}
   else if (i==13){xx=x[0]*x[0]*x[1]*x[1];}
   else if (i==14){xx=x[0]*x[0]*x[0]*x[1];}
   else if (i==15){xx=x[0]*x[0]*x[0]*x[1];}
   return xx;
              }
}


class SVDGeneralleastsquare1
{

public static double[][] Transpose(double [][] left)
{
//transpose matrix (if A=a(i,j) Transpose(A)=a(j,i)
int i,j;
int n=left.length;
int m=left[0].length;
double b[][];
b=new double[m][n];
for(i=0;i<n;i++)
  {
  for(j=0;j<m;j++)
    {
    b[j][i]=left[i][j];
    }
  }
return b;
}

public static double[][] T(double [][] left)
{return Transpose(left);}

public static double[][] I(int n)
{
//unit matrix
```

```
double b[][];
b=new double[n][n];
for(int i=0;i<n;i++)
   b[i][i]=1.0;
return b;
}
//Converts vector to diagonal values of matrix
public static double[][] S(double[] left,int m)
{
int n=left.length;
double b[][];
b=new double[m][n];
for(int i=0;i<n;i++)
   b[i][i]=left[i];
return b;
}

//Converts double to diagonal values of matrix
public static double[][] S(double left,int n)
{
double b[][];
b=new double[n][n];
for(int i=0;i<n;i++)
   b[i][i]=left;
return b;
}

//takes invert value of a diagonal matrix
public static double[][] invS(double[][] S)
{
int m=S.length;
int n=S[0].length;
double b[][];
b=new double[m][n];
for(int i=0;i<n;i++)
   {  if(S[i][i]==0) b[i][i]=0;
                else b[i][i]=1.0/S[i][i];
              }
return b;
}

//square root of a diagonal matrix
public static double[][] sqrtS(double[] left,int m)
{
int n=left.length;
double b[][];
b=new double[m][n];
for(int i=0;i<n;i++)
   b[i][i]=Math.sqrt(left[i]);
return b;
}
//multiply two matrices
public static double[][] multiply(double[][] left,double[][] right)
{
//multiplication of two matrices
int ii,jj,i,j,k;
int m1=left[0].length;
int n1=left.length;
int m2=right[0].length;
int n2=right.length;
double[][] b;
b=new double[n1][m2];
 if(m1 !=n2 )
 {
 System.out.println("inner matrix dimensions must agree"+"m1="+m1+"n2="+n2);
 return b;
 }
 for(i=0;i<n1;i++)
   {
   for(j=0;j<m2;j++)
    {
    for(k=0;k<m1;k++)
                {b[i][j]+=left[i][k]*right[k][j];
                }
```

```
    }
   }
return b;
//end of multiply of two matrices
}

//multiply a matrix with a double value
public static double[][] multiply(double[][] left,double right)
{
//multiplying a matrix with a constant
int i,j;
int n=left.length;
int m=left[0].length;
double b[][];
b=new double[n][m];
for(i=0;i<n;i++)
  {
  for(j=0;j<m;j++)
    b[i][j]=left[i][j]*right;
  }
return b;
//end of multiplying a matrix with a constant double
}

public static double[] multiply(double[][] left,double[] right)
{
//multiplication of one matrix with one vector
int ii,jj,i,j,k;
int n1=left.length;
int m1=left[0].length;

int m2=right.length;
double[] b;
b=new double[m2];
 for(i=0;i<n1;i++)
   {
     b[i]=0;
     for(k=0;k<m1;k++)
                {b[i]+=left[i][k]*right[k];
                }
   }
return b;
//end of multiply of a matrix and a vector
}

//substract two matrices
public static double[][] substract(double[][] left,double[][] right)
{
//addition of two matrices
int n1=left.length;
int m1=left[0].length;
int n2=right.length;
int m2=right[0].length;
int nMax,mMax;
int i,j;
if(m1>=m2) mMax=m1;
else       mMax=m2;
if(n1>=n2) nMax=n1;
else       nMax=n2;
double b[][];
b=new double[nMax][mMax];
for(i=0;i<n1;i++)
  {
  for(j=0;j<m1;j++)
    {
    b[i][j]=b[i][j]+left[i][j];
    }
  }
for(i=0;i<n2;i++)
  {
  for(j=0;j<m2;j++)
    {
    b[i][j]=b[i][j]-right[i][j];
    }
```

```
   }
 return b;
 //end of matrix substraction method
 }
 public static double[] SVD_LSQ(f_xir f,double xi[][],double yi[],int n_leastsquare)
 { // Singular value decomposition least square approximation
  double U[][];
  double V[][];
  double S[][];
  double invS[][];
  double A[][]=A(f,xi,n_leastsquare);
  int m=A.length;
  int m1=A[0].length;
  System.out.println("A="+Matrix.toString(A));
  double ATA[][]=multiply(Transpose(A),A);
  System.out.println("ATA="+Matrix.toString(ATA));
  //double aa[][]=Householder(ATA);
  double cc[][];
  cc=eigen_jacobi(ATA);//eigenQL(aa);
   System.out.println("c="+Matrix.toString(cc));
  int n=cc[0].length;
  int n1=cc.length;
  double dummy[]=new double[n1];
  for(int i=0;i<n;i++)
  {for(int k=i+1;k<n;k++)
    {if(cc[0][i]<cc[0][k])
      { for(int j=0;j<n1;j++)
        { dummy[j]=cc[j][k];
          cc[j][k]=cc[j][i];
          cc[j][i]=dummy[j];
        }
      }
    }
  }
  double x[]=new double[n];
  for(int i=0;i<n;i++)
  {x[i]=cc[0][i];}
  S=sqrtS(x,m1);
  System.out.println("S="+Matrix.toString(S));
  invS=invS(S);
  System.out.println("invS="+Matrix.toString(invS));
  double xx[][]=new double[n][n];
  V=new double[n][n];
  U=new double[n][n];
  double L[]=new double[n];

  for(int i=0;i<n;i++)
  { for(int j=0;j<n;j++)
    {xx[i][j]=cc[i+1][j];}
  }

  double bb[]=new double[n];
  for(int j=0;j<n;j++)
  { for(int i=0;i<n;i++)
    {L[j]+=xx[i][j]*xx[i][j];}
    L[j]=Math.sqrt(L[j]);
  }
  for(int i=0;i<n;i++)
  { for(int j=0;j<n;j++)
    {V[i][j]=xx[i][j]/L[i];}
  }
  System.out.println("VT="+Matrix.toString(T(V)));
  U=multiply(A,multiply(V,invS));
   System.out.println("U="+Matrix.toString(U));
  double c[]=multiply(T(U),yi);
   System.out.println("c="+Matrix.toString(c));
   double z[]=new double[n];
   for(int i=0;i<n;i++) z[i]=c[i]/S[i][i];
   System.out.println("z="+Matrix.toString(z));
   double a[]=multiply(V,z);
   System.out.println("a="+Matrix.toString(a));
   return a;
 }
 public static double func(f_xir f,double e[],double x[])
```

```
{
// this function calculates the value of
// least square curve fitting function
int n=e.length;
double ff;
if(n!=0.0)
  { ff=0;
  for(int i=0;i<e.length;i++)
   { ff+=e[i]*f.func(x,i); }
  }
else
  ff=0;
return ff;
}

public static double error(f_xir f,double x[][],double z[],double e[])
{
//calculates absolute square root error of a least square approach
double n=x.length;
 int k;
 double xt[][]=T(x);
 double total=0;
 for(k=0;k<n;k++)
 {
 total+=(z[k]-func(f,e,xt[k]))*(z[k]-func(f,e,xt[k]));
 }
total=Math.sqrt(total);
return total;
}
  public static void print(String s)
  {JOptionPane.showMessageDialog(null,s,
     "Singular value decomposition : ",JOptionPane.PLAIN_MESSAGE);}
  public static double[][] eigen_jacobi(double A[][])
  {               //Eigen value calculation
   double eps=1e-15;
   double err=1;
   int m=100;
   int n=A.length;
   double P[][]=new double[n][n];
   double R[][]=new double[n][n];
   int p=n,q=n;
   double max=-9.99e100;
   double fi;
   P=I(n);
   for(int i=0;i<n;i++)
   {for(int j=i+1;j<n;j++)
      {if(Math.abs(A[i][j]) > max) {max=Math.abs(A[i][j]);p=i;q=j;}}
   }
   while(Math.abs(A[p][q]) > eps )
   {
    fi=0.5*Math.atan2(2.0*A[p][q],(A[p][p]-A[q][q]));
    R=I(n);
    R[p][p]=Math.cos(fi);
    R[q][q]=Math.cos(fi);
    R[p][q]=-Math.sin(fi);
    R[q][p]=Math.sin(fi);
    P=multiply(P,R);
    A=multiply(T(R),multiply(A,R));
    max=-9.99e100;
    for(int i=0;i<n;i++)
   {for(int j=i+1;j<n;j++){if(Math.abs(A[i][j])>max) {max=Math.abs(A[i][j]);p=i;q=j;}}}
    }
   double lambda[]=new double[n];
   for(int i=0;i<n;i++) {lambda[i]=A[i][i];}
   double PP[][]=new double[n+1][n];
   PP[0]=lambda;
   for(int i=0;i<n;i++)
   {for(int j=0;j<n;j++)
     {PP[i+1][j]=P[i][j];}
   }
   return PP;
  }

  public static double[] gausswithpartialpivot(double a[][],double b[])
```

```
  { // Gauss elimination with partial pivoting
  int n=b.length;
  double x[]=new double[n];
  double carpan=0;
  double toplam=0;
  double buyuk;
  double dummy=0;
  //gauss elimination
  int i,j,k,p,ii,jj;
  for(k=0;k<(n-1);k++)
  { //partial pivoting
                p=k;
                buyuk=Math.abs(a[k][k]);
                for(ii=k+1;ii<n;ii++)
                {  dummy=Math.abs(a[ii][k]);
                   if(dummy > buyuk) {buyuk=dummy;p=ii;}
                }
                if(p!=k)
                { for(jj=k;jj<n;jj++)
                  { dummy=a[p][jj];
                    a[p][jj]=a[k][jj];
                    a[k][jj]=dummy;
                  }
                  dummy=b[p];
                  b[p]=b[k];
                  b[k]=dummy;
                }
                //
    for(i=k+1;i<n;i++)
    {  carpan=a[i][k]/a[k][k];
       a[i][k]=0;
       for(j=k+1;j<n;j++)
       {   a[i][j]-=carpan*a[k][j]; }
           b[i]   =b[i]   -carpan*b[k];
                }
  }
  // backward substitution
  x[n-1]=b[n-1]/a[n-1][n-1];
  for(i=n-2;i>=0;i--)
  {
    toplam=0;
    for(j=i+1;j<n;j++)
    {  toplam+=a[i][j]*x[j];}
    x[i]=(b[i]-toplam)/a[i][i];
  }
  return x;
}

public static double[][] A(f_xir f,double xi[][],int n_leastsquare)
{
int n=xi[0].length;
double txi[][]=T(xi);
int m=n_leastsquare+1;
int l=xi.length;
int i,j,k;
double A[][];
A=new double[n][m];
double cc=1.0;
for(i=0;i<n;i++)
  {  for(j=0;j<m;j++) { A[i][j]=f.func(txi[i],j);}}
return A;
}
public static double[][] MT(double [][] left)
{
//transpose matrix (if A=a(i,j) T(A)=a(j,i)
int i,j;
int n=left.length;
int m=left[0].length;
double b[][];
b=new double[m][n];
for(i=0;i<n;i++)
  {
  for(j=0;j<m;j++)
    {
```

```
    b[j][i]=left[i][j];
    }
  }
return b;
}

  public static void main(String args[]) throws IOException
  {
  double z[];
  String s1=JOptionPane.showInputDialog("Data file name (a.txt)");
  // JFileChooser fc=new JFileChooser();
  // if (fc.showOpenDialog(null) == JFileChooser.APPROVE_OPTION) {File file = fc.getSelectedFile();s1=file.getName(); }
  double a[][]=Text.readDoubleT(s1);
  double x[][]={a[0],a[1]};
  z=a[2];
  int n=Integer.parseInt(JOptionPane.showInputDialog("polinom derecesi (n)"));
  fa f1=new fa();
  double A[][]=A(f1,x,n);
  double E[]=SVD_LSQ(f1,x,z,n);
  double x1[]={0.5,0.5};
  double z1=func(f1,E,x1);
  System.out.println("z="+z1);
  }
}
```

```
---------- Capture Output ----------
> "E:\co\java\bin\java.exe" SVDGeneralleastsquare1
A=      1.000000000000000       0.000000000000000       0.000000000000000       0.000000000000000       0.000000000000000
0.000000000000000       0.000000000000000
       1.000000000000000       0.100000000000000       0.000000000000000       0.010000000000000       0.000000000000000
0.000000000000000       0.001000000000000
       1.000000000000000       0.200000000000000       0.000000000000000       0.040000000000000       0.000000000000000
0.000000000000000       0.008000000000000
       1.000000000000000       0.300000000000000       0.000000000000000       0.090000000000000       0.000000000000000
0.000000000000000       0.027000000000000
       1.000000000000000       0.400000000000000       0.000000000000000       0.160000000000000       0.000000000000000
0.000000000000000       0.064000000000000
       1.000000000000000       0.500000000000000       0.000000000000000       0.250000000000000       0.000000000000000
0.000000000000000       0.125000000000000
       1.000000000000000       0.600000000000000       0.000000000000000       0.360000000000000       0.000000000000000
0.000000000000000       0.216000000000000
       1.000000000000000       0.700000000000000       0.000000000000000       0.490000000000000       0.000000000000000
0.000000000000000       0.343000000000000
       1.000000000000000       0.800000000000000       0.000000000000000       0.640000000000000       0.000000000000000
0.000000000000000       0.512000000000000
       1.000000000000000       0.900000000000000       0.000000000000000       0.810000000000000       0.000000000000000
0.000000000000000       0.729000000000000
       1.000000000000000       1.000000000000000       0.000000000000000       1.000000000000000       0.000000000000000
0.000000000000000       1.000000000000000
       1.000000000000000       0.000000000000000       0.100000000000000       0.000000000000000       0.010000000000000
0.000000000000000       0.000000000000000
       1.000000000000000       0.100000000000000       0.100000000000000       0.010000000000000       0.010000000000000
0.010000000000000       0.001000000000000
       1.000000000000000       0.200000000000000       0.100000000000000       0.040000000000000       0.010000000000000
0.020000000000000       0.008000000000000
       1.000000000000000       0.300000000000000       0.100000000000000       0.090000000000000       0.010000000000000
0.030000000000000       0.027000000000000
       1.000000000000000       0.400000000000000       0.100000000000000       0.160000000000000       0.010000000000000
0.040000000000000       0.064000000000000
       1.000000000000000       0.500000000000000       0.100000000000000       0.250000000000000       0.010000000000000
0.050000000000000       0.125000000000000
       1.000000000000000       0.600000000000000       0.100000000000000       0.360000000000000       0.010000000000000
0.060000000000000       0.216000000000000
       1.000000000000000       0.700000000000000       0.100000000000000       0.490000000000000       0.010000000000000
0.070000000000000       0.343000000000000
       1.000000000000000       0.800000000000000       0.100000000000000       0.640000000000000       0.010000000000000
0.080000000000000       0.512000000000000
       1.000000000000000       0.900000000000000       0.100000000000000       0.810000000000000       0.010000000000000
0.090000000000000       0.729000000000000
       1.000000000000000       1.000000000000000       0.100000000000000       1.000000000000000       0.010000000000000
0.100000000000000       1.000000000000000
       1.000000000000000       0.000000000000000       0.200000000000000       0.000000000000000       0.040000000000000
0.000000000000000       0.000000000000000
```

1.000000000000000 0.100000000000000 0.200000000000000 0.010000000000000 0.040000000000000 0.020000000000000 0.001000000000000
1.000000000000000 0.200000000000000 0.200000000000000 0.040000000000000 0.040000000000000 0.040000000000000 0.008000000000000
1.000000000000000 0.300000000000000 0.200000000000000 0.090000000000000 0.040000000000000 0.060000000000000 0.027000000000000
1.000000000000000 0.400000000000000 0.200000000000000 0.160000000000000 0.040000000000000 0.080000000000000 0.064000000000000
1.000000000000000 0.500000000000000 0.200000000000000 0.250000000000000 0.040000000000000 0.100000000000000 0.125000000000000
1.000000000000000 0.600000000000000 0.200000000000000 0.360000000000000 0.040000000000000 0.120000000000000 0.216000000000000
1.000000000000000 0.700000000000000 0.200000000000000 0.490000000000000 0.040000000000000 0.140000000000000 0.343000000000000
1.000000000000000 0.800000000000000 0.200000000000000 0.640000000000000 0.040000000000000 0.160000000000000 0.512000000000000
1.000000000000000 0.900000000000000 0.200000000000000 0.810000000000000 0.040000000000000 0.180000000000000 0.729000000000000
1.000000000000000 1.000000000000000 0.200000000000000 1.000000000000000 0.040000000000000 0.200000000000000 1.000000000000000
1.000000000000000 0.000000000000000 0.300000000000000 0.000000000000000 0.090000000000000 0.000000000000000 0.000000000000000
1.000000000000000 0.100000000000000 0.300000000000000 0.010000000000000 0.090000000000000 0.030000000000000 0.001000000000000
1.000000000000000 0.200000000000000 0.300000000000000 0.040000000000000 0.090000000000000 0.060000000000000 0.008000000000000
1.000000000000000 0.300000000000000 0.300000000000000 0.090000000000000 0.090000000000000 0.090000000000000 0.027000000000000
1.000000000000000 0.400000000000000 0.300000000000000 0.160000000000000 0.090000000000000 0.120000000000000 0.064000000000000
1.000000000000000 0.500000000000000 0.300000000000000 0.250000000000000 0.090000000000000 0.150000000000000 0.125000000000000
1.000000000000000 0.600000000000000 0.300000000000000 0.360000000000000 0.090000000000000 0.180000000000000 0.216000000000000
1.000000000000000 0.700000000000000 0.300000000000000 0.490000000000000 0.090000000000000 0.210000000000000 0.343000000000000
1.000000000000000 0.800000000000000 0.300000000000000 0.640000000000000 0.090000000000000 0.240000000000000 0.512000000000000
1.000000000000000 0.900000000000000 0.300000000000000 0.810000000000000 0.090000000000000 0.270000000000000 0.729000000000000
1.000000000000000 1.000000000000000 0.300000000000000 1.000000000000000 0.090000000000000 0.300000000000000 1.000000000000000
1.000000000000000 0.000000000000000 0.400000000000000 0.000000000000000 0.160000000000000 0.000000000000000 0.000000000000000
1.000000000000000 0.100000000000000 0.400000000000000 0.010000000000000 0.160000000000000 0.040000000000000 0.001000000000000
1.000000000000000 0.200000000000000 0.400000000000000 0.040000000000000 0.160000000000000 0.080000000000000 0.008000000000000
1.000000000000000 0.300000000000000 0.400000000000000 0.090000000000000 0.160000000000000 0.120000000000000 0.027000000000000
1.000000000000000 0.400000000000000 0.400000000000000 0.160000000000000 0.160000000000000 0.160000000000000 0.064000000000000
1.000000000000000 0.500000000000000 0.400000000000000 0.250000000000000 0.160000000000000 0.200000000000000 0.125000000000000
1.000000000000000 0.600000000000000 0.400000000000000 0.360000000000000 0.160000000000000 0.240000000000000 0.216000000000000
1.000000000000000 0.700000000000000 0.400000000000000 0.490000000000000 0.160000000000000 0.280000000000000 0.343000000000000
1.000000000000000 0.800000000000000 0.400000000000000 0.640000000000000 0.160000000000000 0.320000000000000 0.512000000000000
1.000000000000000 0.900000000000000 0.400000000000000 0.810000000000000 0.160000000000000 0.360000000000000 0.729000000000000
1.000000000000000 1.000000000000000 0.400000000000000 1.000000000000000 0.160000000000000 0.400000000000000 1.000000000000000
1.000000000000000 0.000000000000000 0.500000000000000 0.000000000000000 0.250000000000000 0.000000000000000 0.000000000000000
1.000000000000000 0.100000000000000 0.500000000000000 0.010000000000000 0.250000000000000 0.050000000000000 0.001000000000000
1.000000000000000 0.200000000000000 0.500000000000000 0.040000000000000 0.250000000000000 0.100000000000000 0.008000000000000
1.000000000000000 0.300000000000000 0.500000000000000 0.090000000000000 0.250000000000000 0.150000000000000 0.027000000000000
1.000000000000000 0.400000000000000 0.500000000000000 0.160000000000000 0.250000000000000 0.200000000000000 0.064000000000000

1.000000000000000 0.500000000000000 0.500000000000000 0.250000000000000 0.250000000000000 0.250000000000000 0.125000000000000
1.000000000000000 0.600000000000000 0.500000000000000 0.360000000000000 0.250000000000000 0.300000000000000 0.216000000000000
1.000000000000000 0.700000000000000 0.500000000000000 0.490000000000000 0.250000000000000 0.350000000000000 0.343000000000000
1.000000000000000 0.800000000000000 0.500000000000000 0.640000000000000 0.250000000000000 0.400000000000000 0.512000000000000
1.000000000000000 0.900000000000000 0.500000000000000 0.810000000000000 0.250000000000000 0.450000000000000 0.729000000000000
1.000000000000000 1.000000000000000 0.500000000000000 1.000000000000000 0.250000000000000 0.500000000000000 1.000000000000000
1.000000000000000 0.000000000000000 0.600000000000000 0.000000000000000 0.360000000000000 0.000000000000000 0.000000000000000
1.000000000000000 0.100000000000000 0.600000000000000 0.010000000000000 0.360000000000000 0.060000000000000 0.001000000000000
1.000000000000000 0.200000000000000 0.600000000000000 0.040000000000000 0.360000000000000 0.120000000000000 0.008000000000000
1.000000000000000 0.300000000000000 0.600000000000000 0.090000000000000 0.360000000000000 0.180000000000000 0.027000000000000
1.000000000000000 0.400000000000000 0.600000000000000 0.160000000000000 0.360000000000000 0.240000000000000 0.064000000000000
1.000000000000000 0.500000000000000 0.600000000000000 0.250000000000000 0.360000000000000 0.300000000000000 0.125000000000000
1.000000000000000 0.600000000000000 0.600000000000000 0.360000000000000 0.360000000000000 0.360000000000000 0.216000000000000
1.000000000000000 0.700000000000000 0.600000000000000 0.490000000000000 0.360000000000000 0.420000000000000 0.343000000000000
1.000000000000000 0.800000000000000 0.600000000000000 0.640000000000000 0.360000000000000 0.480000000000000 0.512000000000000
1.000000000000000 0.900000000000000 0.600000000000000 0.810000000000000 0.360000000000000 0.540000000000000 0.729000000000000
1.000000000000000 1.000000000000000 0.600000000000000 1.000000000000000 0.360000000000000 0.600000000000000 1.000000000000000
1.000000000000000 0.000000000000000 0.700000000000000 0.000000000000000 0.490000000000000 0.000000000000000 0.000000000000000
1.000000000000000 0.100000000000000 0.800000000000000 0.010000000000000 0.640000000000000 0.080000000000000 0.001000000000000
1.000000000000000 0.200000000000000 0.800000000000000 0.040000000000000 0.640000000000000 0.160000000000000 0.008000000000000
1.000000000000000 0.300000000000000 0.800000000000000 0.090000000000000 0.640000000000000 0.240000000000000 0.027000000000000
1.000000000000000 0.400000000000000 0.800000000000000 0.160000000000000 0.640000000000000 0.320000000000000 0.064000000000000
1.000000000000000 0.500000000000000 0.800000000000000 0.250000000000000 0.640000000000000 0.400000000000000 0.125000000000000
1.000000000000000 0.600000000000000 0.800000000000000 0.360000000000000 0.640000000000000 0.480000000000000 0.216000000000000
1.000000000000000 0.700000000000000 0.800000000000000 0.490000000000000 0.640000000000000 0.560000000000000 0.343000000000000
1.000000000000000 0.800000000000000 0.800000000000000 0.640000000000000 0.640000000000000 0.640000000000000 0.512000000000000
1.000000000000000 0.900000000000000 0.800000000000000 0.810000000000000 0.640000000000000 0.720000000000000 0.729000000000000
1.000000000000000 1.000000000000000 0.800000000000000 1.000000000000000 0.640000000000000 0.800000000000000 1.000000000000000
1.000000000000000 0.000000000000000 0.900000000000000 0.000000000000000 0.810000000000000 0.000000000000000 0.000000000000000
1.000000000000000 0.100000000000000 0.900000000000000 0.010000000000000 0.810000000000000 0.090000000000000 0.001000000000000
1.000000000000000 0.200000000000000 0.900000000000000 0.040000000000000 0.810000000000000 0.180000000000000 0.008000000000000
1.000000000000000 0.300000000000000 0.900000000000000 0.090000000000000 0.810000000000000 0.270000000000000 0.027000000000000
1.000000000000000 0.400000000000000 0.900000000000000 0.160000000000000 0.810000000000000 0.360000000000000 0.064000000000000
1.000000000000000 0.500000000000000 0.900000000000000 0.250000000000000 0.810000000000000 0.450000000000000 0.125000000000000
1.000000000000000 0.600000000000000 0.900000000000000 0.360000000000000 0.810000000000000 0.540000000000000 0.216000000000000
1.000000000000000 0.700000000000000 0.900000000000000 0.490000000000000 0.810000000000000 0.630000000000000 0.343000000000000
1.000000000000000 0.800000000000000 0.900000000000000 0.640000000000000 0.810000000000000 0.720000000000000 0.512000000000000

1.000000000000000 0.900000000000000 0.900000000000000 0.810000000000000 0.810000000000000
0.810000000000000 0.729000000000000
1.000000000000000 1.000000000000000 0.900000000000000 1.000000000000000 0.810000000000000
0.900000000000000 1.000000000000000
1.000000000000000 0.000000000000000 1.000000000000000 0.000000000000000 1.000000000000000
0.000000000000000 0.000000000000000
1.000000000000000 0.100000000000000 1.000000000000000 0.010000000000000 1.000000000000000
0.100000000000000 0.001000000000000
1.000000000000000 0.200000000000000 1.000000000000000 0.040000000000000 1.000000000000000
0.200000000000000 0.008000000000000
1.000000000000000 0.300000000000000 1.000000000000000 0.090000000000000 1.000000000000000
0.300000000000000 0.027000000000000
1.000000000000000 0.400000000000000 1.000000000000000 0.160000000000000 1.000000000000000
0.400000000000000 0.064000000000000
1.000000000000000 0.500000000000000 1.000000000000000 0.250000000000000 1.000000000000000
0.500000000000000 0.125000000000000
1.000000000000000 0.600000000000000 1.000000000000000 0.360000000000000 1.000000000000000
0.600000000000000 0.216000000000000
1.000000000000000 0.700000000000000 1.000000000000000 0.490000000000000 1.000000000000000
0.700000000000000 0.343000000000000
1.000000000000000 0.800000000000000 1.000000000000000 0.640000000000000 1.000000000000000
0.800000000000000 0.512000000000000
1.000000000000000 0.900000000000000 1.000000000000000 0.810000000000000 1.000000000000000
0.900000000000000 0.729000000000000
1.000000000000000 1.000000000000000 1.000000000000000 1.000000000000000 1.000000000000000
1.000000000000000 1.000000000000000

ATA= 110.000000000000000 55.000000000000000 52.700000000000010 38.499999999999990
36.809999999999990 26.400000000000000 30.250000000000004
55.000000000000000 38.499999999999990 26.400000000000000 30.250000000000004 18.480000000000004
18.479999999999997 25.332999999999990
52.700000000000010 26.400000000000000 36.809999999999990 18.479999999999997 29.333000000000000
18.480000000000004 14.520000000000000
38.499999999999990 30.250000000000004 18.479999999999997 25.332999999999990 12.936000000000000
14.520000000000000 22.082500000000000
36.809999999999990 18.480000000000004 29.333000000000000 12.936000000000000 25.055700000000005
14.751000000000001 10.164000000000001
26.400000000000000 18.479999999999997 18.480000000000004 14.520000000000000 14.751000000000001
12.936000000000000 12.159840000000003
30.250000000000004 25.332999999999990 14.520000000000000 22.082500000000000 10.164000000000001
12.159840000000003 19.784050000000000

c= 219.504428259675500 28.702983116123430 17.638208881209824 1.426056364129608 0.742715638872023
0.387516764404438 0.016840975585218
0.678516129735166 -0.194564596683707 -0.666047544380233 0.019293949794631 -0.165264810510046
0.173054503125007 -0.022551560658574
0.389511657261007 0.400765142949076 0.057440463547604 -0.388528295658421 0.637125488424625 -
0.162347225682147 0.318007572217993
0.364210077375480 -0.436450759746670 0.336072964237781 0.101810957227808 -0.104947860890572 -
0.736568923722789 0.001847336416266
0.290205914286845 0.465145820726967 0.181009297977709 0.181994103032159 0.014377901356756 -
0.028421155186339 -0.795310748808766
0.267852110867113 -0.424893037876136 0.464520654361461 0.252798269242814 0.364042774047860
0.579228702094058 -0.001311160489701
0.207623852000690 0.019806586183445 0.380905579084776 -0.686472145299542 -0.529175872741307
0.245230419227492 -0.001380568026120
0.237014050601445 0.462337131538502 0.215646377248121 0.519649123177981 -0.378131764370365
0.068630372002015 0.515593237146670

S= 14.815681835800724 0.000000000000000 0.000000000000000 0.000000000000000 0.000000000000000
0.000000000000000 0.000000000000000
0.000000000000000 5.357516506378999 0.000000000000000 0.000000000000000 0.000000000000000
0.000000000000000 0.000000000000000
0.000000000000000 0.000000000000000 4.199786766159662 0.000000000000000 0.000000000000000
0.000000000000000 0.000000000000000
0.000000000000000 0.000000000000000 0.000000000000000 1.194176018905759 0.000000000000000
0.000000000000000 0.000000000000000
0.000000000000000 0.000000000000000 0.000000000000000 0.000000000000000 0.861809514261721
0.000000000000000 0.000000000000000
0.000000000000000 0.000000000000000 0.000000000000000 0.000000000000000 0.000000000000000
0.622508445247483 0.000000000000000
0.000000000000000 0.000000000000000 0.000000000000000 0.000000000000000 0.000000000000000
0.000000000000000 0.129772784455055

invS= 0.067496049866810 0.000000000000000 0.000000000000000 0.000000000000000 0.000000000000000
0.000000000000000 0.000000000000000
0.000000000000000 0.186653647974642 0.000000000000000 0.000000000000000 0.000000000000000
0.000000000000000 0.000000000000000
0.000000000000000 0.000000000000000 0.238107326795168 0.000000000000000 0.000000000000000
0.000000000000000 0.000000000000000
0.000000000000000 0.000000000000000 0.000000000000000 0.837397489288317 0.000000000000000
0.000000000000000 0.000000000000000
0.000000000000000 0.000000000000000 0.000000000000000 0.000000000000000 1.160349222712703
0.000000000000000 0.000000000000000
0.000000000000000 0.000000000000000 0.000000000000000 0.000000000000000 0.000000000000000
1.606403909271372 0.000000000000000
0.000000000000000 0.000000000000000 0.000000000000000 0.000000000000000 0.000000000000000
0.000000000000000 7.705775939070945

VT= 0.678516129735166 0.389511657261007 0.364210077375480 0.290205914286845 0.267852110867113
0.207623852000690 0.237014050601445
-0.194564596683707 0.400765142949076 -0.436450759746670 0.465145820726967 -0.424893037876136
0.019806586183445 0.462337131538501
-0.666047544380233 0.057440463547604 0.336072964237781 0.181009297977709 0.464520654361461
0.380905579084776 0.215646377248121
0.019293949794631 -0.388528295658421 0.101810957227808 0.181994103032159 0.252798269242814 -
0.686472145299542 0.519649123177981
-0.165264810510046 0.637125488424625 -0.104947860890572 0.014377901356756 0.364042774047861 -
0.529175872741307 -0.378131764370365
0.173054503125007 -0.162347225682147 -0.736568923722789 -0.028421155186339 0.579228702094058
0.245230419227492 0.068630372002015
-0.022551560658574 0.318007572217993 0.001847336416266 -0.795310748808766 -0.001311160489701 -
0.001380568026120 0.515593237146670

U= 0.045797158528040 -0.036316191737729 -0.158590800310863 0.016156705116479 -0.191764894417094
0.277995430337026 -0.173777273511338
0.048638083393061 -0.027881255591492 -0.156740757405259 -0.014419389893200 -0.118108018922932
0.251569598087415 0.013960417971242
0.051966748388401 -0.017192114686504 -0.153720639803806 -0.039336539577630 -0.046750065088720
0.224892137329130 0.102966656575634
0.055879138587129 -0.003730987549589 -0.149222365611986 -0.055983826710450 0.019676377692321
0.198624536649443 0.117079718068755
0.060471239062319 0.013019907292427 -0.142937852935283 -0.061750334065300 0.078538720026971
0.173428284635628 0.080137878217519
0.065839034887040 0.033578351312719 -0.134559019879178 -0.054025144415820 0.127204372522009
0.149964869874957 0.015979412788842
0.072078511134363 0.058462125984462 -0.123777784549155 -0.030197340535647 0.163040745784214
0.128895780954702 -0.051557402450361
0.079285652877361 0.088189012780830 -0.110286065050694 0.012343994801578 0.183415250420365
0.110882506462137 -0.098634291733175
0.087556445189104 0.123276793174999 -0.093775779489278 0.076209778822216 0.185695297037241
0.096586534984534 -0.101412979292686
0.096986873142664 0.164243248640142 -0.073938845970391 0.164010928752627 0.167248296241622
0.086669355109166 -0.036055189361979
0.107672921811111 0.211606160649436 -0.050467182599513 0.278358361819173 0.125441658640286
0.081792455423305 0.121277353825864
0.048436222276813 -0.045255782739717 -0.149482599086073 0.026799255472543 -0.199718343799062
0.168977462999477 -0.172454792550092
0.051417285040516 -0.036783876877830 -0.146725592088496 -0.009525340046538 -0.132201756431037
0.146491021791059 0.015176515453707
0.054886087934537 -0.026057766257191 -0.142798510395071 -0.040190990240370 -0.066984090722962
0.123752952073967 0.104076370579321
0.058938616031948 -0.012559669404625 -0.137393272111279 -0.062586777882592 -0.006697936068057
0.101424742435473 0.118083048593662
0.063670854405819 0.004228195153042 -0.130201795342603 -0.074101785746844 0.046024118140456
0.080167881462851 0.081034825263648
0.069178788129222 0.024823608888984 -0.120915998194526 -0.072125096606765 0.088549482509357
0.060643857743373 0.016769976356191
0.075558402275227 0.049744353276378 -0.109227798772530 -0.054045793235995 0.118245567645425
0.043514159864312 -0.050873222361791
0.082905681916907 0.079508209788397 -0.094829115182097 -0.017252958408172 0.132479784155439
0.029440276412939 -0.098056495123385
0.091316612127332 0.114632959898216 -0.077411865528709 0.040864325103064 0.128619542646178
0.019083695976529 -0.100941566161675
0.100887177979573 0.155636385079010 -0.056667967917849 0.122916974524073 0.104032253724422
0.013105907142354 -0.035690159709747
0.111713364546702 0.203036266803954 -0.032289340454999 0.231515907081218 0.056085327996949
0.012168398497687 0.121535999999316

0.051436865214227 -0.055781530452077 -0.138162282436260 0.041675658547815 -0.199223458183017
0.078569000690050 -0.171334381767923
0.054558065876611 -0.047272654874539 -0.134498311346711 -0.000397437480668 -0.137847158941129
0.060021950522825 0.016190542757097
0.058167006669314 -0.036509574538250 -0.129664265561313 -0.036811588183902 -0.078769781359191
0.041223271846926 0.104984014403931
0.062359672665407 -0.022974507970033 -0.123352063185549 -0.064955876335526 -0.024623914830423
0.022834453249625 0.118884308939494
0.067232048937960 -0.006149673696716 -0.115253622324901 -0.082219384709181 0.021957851251953
0.005516983318196 0.081729702130700
0.072880120560044 0.014482709754877 -0.105060861084851 -0.085991196078504 0.058342927494717 -
0.010067649360089 0.017358469744464
0.079399872604731 0.039440423857921 -0.092465697570882 -0.073660393217135 0.081898724504648 -
0.023257956197958 -0.050391112452297
0.086887290145093 0.069241250085591 -0.077160049888477 -0.042616058898715 0.089992652888525 -
0.033392448608137 -0.097680768692670
0.095438358254199 0.104402969911061 -0.058835836143117 0.009752724103119 0.079992123253127 -
0.039809638003354 -0.100672223209740
0.105149062005122 0.145443364807506 -0.037184974440284 0.086056873014727 0.049264546205234 -
0.041848035796336 -0.035527200236591
0.116115386470933 0.192880216248100 -0.011899382885462 0.188907305062469 -0.004822667648375 -
0.038846153399811 0.121592575993693
0.054799087340280 -0.067893434874810 -0.124629850361424 0.060785914342294 -0.190280237568960
0.006770043408744 -0.170416041164830
0.058060425901347 -0.059347589581621 -0.120058915179902 0.012964317804409 -0.135044226453209 -
0.007837615717288 0.017002499881411
0.061809504592732 -0.048547539529681 -0.114317905302532 -0.029198333408227 -0.082107136997408 -
0.022696903351994 0.105689588049466
0.066142308487506 -0.034975503245814 -0.107098738834795 -0.063091122069254 -0.034101558594777 -
0.037146330908101 0.119483499106249
0.071154822658741 -0.018113699256846 -0.098093333882175 -0.086103130952310 0.006339919361462 -
0.050524409798338 0.082222508818676
0.076943032179507 0.002555653910398 -0.086993608550153 -0.095623442831035 0.036584707478090 -
0.062169651435430 0.017744892953661
0.083602922122876 0.027550337729093 -0.073491480944212 -0.089041140479069 0.054000216361884 -
0.071420567232105 -0.050111072721880
0.091230477561919 0.057388133672413 -0.057278869169834 -0.063745306670050 0.055953856619624 -
0.077615668601091 -0.097507112441032
0.099921683569707 0.092586823213533 -0.038047691332501 -0.017125024177618 0.039813038858089 -
0.080093466955115 -0.100604950436881
0.109772525219312 0.133664187825629 -0.015489865537696 0.053430624224588 0.002945173684059 -
0.078192473706905 -0.035566310942511
0.120878987583805 0.181138008981874 0.010702690109098 0.150532555762928 -0.057282328295687 -
0.071251200269186 0.121447081808994
0.058522888654975 -0.081591496007914 -0.108885302861565 0.084130022855980 -0.172888681956891 -
0.046419408844440 -0.169699770740813
0.061924365114723 -0.073008680999075 -0.103407403588071 0.030559925808693 -0.123792958967277 -
0.057087676929279 0.017612386826649
0.065813581704790 -0.062171661231484 -0.096759429618728 -0.017351225913345 -0.076996157637612 -
0.068007573522792 0.106193091515925
0.070286523498246 -0.048562655231967 -0.088633299059019 -0.056992515083774 -0.035130867361118 -
0.078517610037706 0.119880619093929
0.075439175568162 -0.031663881527348 -0.078720930014426 -0.085753024476232 -0.000829677531016 -
0.087956297886750 0.082513245327576
0.081367522987610 -0.010957558644453 -0.066714240590432 -0.101021836864359 0.023274822459474 -
0.095662148482649 0.017929245983782
0.088167550829661 0.014074094889892 -0.052305148892518 -0.100188035021794 0.034550043217131 -
0.100973673238131 -0.050033103170538
0.095935244167386 0.043948860548863 -0.035185573026168 -0.080640701722178 0.030363395348734 -
0.103229383565924 -0.097535526368469
0.104766588073856 0.079184519805634 -0.015047431096863 -0.039768919739148 0.008082289461063 -
0.101767790878755 -0.100739747843098
0.114757567622143 0.120298854133380 0.008417358789914 0.025038228153656 -0.034925863839104 -
0.095927406589352 -0.035807491827506
0.126004167885317 0.167809645005276 0.035516878528682 0.116391659182594 -0.101293653944987 -
0.085046742110440 0.121099517445219
0.062608269158309 -0.096875713851391 -0.090928639936683 0.111707984088873 -0.147048791346809 -
0.080999356069502 -0.169185570495871
0.066149883516739 -0.088255929126901 -0.084543776571216 0.052389386532184 -0.104093356483332 -
0.087728233113148 0.018020203592811
0.070179238005488 -0.077381939643660 -0.076988838509901 -0.001270265699256 -0.063436843279805 -
0.094708738665468 0.106494524803308
0.074792317697625 -0.063735963928492 -0.067955743858220 -0.046660055379086 -0.027711841129448 -
0.101279384139190 0.120075668902532

0.080085107666224	-0.046800220508222	-0.057136410721654	-0.081169065280946	0.000449060574518	-0.106778680947040	0.082601911657401
0.086153592984353	-0.026056927909677	-0.044222757205688	-0.102186378178476	0.018413272438871	-0.110545140501746	0.017911528834827
0.093093758725086	-0.000988304659681	-0.028906701415802	-0.107101076845313	0.023548205070391	-0.111917274216036	-0.050157203798272
0.101001589961493	0.028923430714940	-0.010880161457479	-0.093302244055098	0.013221269075857	-0.110233593502636	-0.097766010474983
0.109973071766645	0.064196059687362	0.010164944563799	-0.058178962581470	-0.015200124937951	-0.104832609774274	-0.101076615428390
0.120104189213613	0.105347363730759	0.034536698542548	0.000879684801931	-0.064348566364255	-0.095052834443677	-0.036250742891579
0.131490927375469	0.152895124318306	0.062543182373288	0.086484615321467	-0.136856644596275	-0.080232778923573	0.120549882902368
0.067055228850284	-0.113746088405240	-0.070759861586778	0.143519798040974	-0.112760565738716	-0.096969798266443	-0.168873440430006
0.070736981107396	-0.105089333965099	-0.063468034129339	0.078452699974883	-0.075945419001375	-0.099759284268896	0.018225950179898
0.074906473494826	-0.094178374766207	-0.055006131976051	0.019044547234041	-0.041429193923985	-0.102800398780023	0.106593887911615
0.079659691085646	-0.080495429335389	-0.045066073232397	-0.032093742955191	-0.011844479899765	-0.105431653212552	0.120068648532060
0.085092618952926	-0.063522716199469	-0.033339776003860	-0.072351253366454	0.010176133678064	-0.106991558979209	0.082488507808149
0.091301242169737	-0.042724534885273	-0.019519158395920	-0.099117066773385	0.022000057416280	-0.106818627492722	0.017691741506797
0.098381545809152	-0.017636860919626	-0.003296138514062	-0.109780265949624	0.020994701921663	-0.104251370165819	-0.050483374605082
0.106429514944240	0.012311844170646	0.015637365536233	-0.101729933668812	0.004527477800993	-0.098628298411226	-0.098198564760572
0.115541134648074	0.047621442858718	0.037589435649483	-0.072355152704586	-0.030034204338953	-0.089287923641671	-0.101615553192759
0.125812389993724	0.088809716617766	0.062868153720205	-0.019045005830586	-0.085322933891394	-0.075568757269881	-0.036896064134726
0.137339266054262	0.136394446920963	0.091781601642917	0.060811424179548	-0.163971300249551	-0.056809310708584	0.119798178180441
0.071863767730899	-0.132202619669461	-0.048378967811850	0.179565464712282	-0.070024005132610	-0.094330735435263	-0.168763380543217
0.080995913854630	-0.143514613772613	-0.014680202970515	0.143280885017902	0.005695460956575	-0.067992871496027	0.018031232816842
0.085445682039424	-0.132529715142419	-0.004404372633282	0.072375731258256	0.027931109781692	-0.063155203924768	0.106186403591000
0.090479175427607	-0.118772830280299	0.007349614294316	0.009740440050220	0.045235247553638	-0.057907676274911	0.119448397253887
0.096192379092251	-0.101726177713078	0.020889839706799	-0.042014071379846	0.054975284879192	-0.051588799959182	0.081655489572418
0.102681278106426	-0.080871975967581	0.036524385498683	-0.080276885805582	0.054518632365135	-0.043537086390309	0.016645956313507
0.110041857543204	-0.055692443570633	0.054561333564486	-0.102437086000625	0.041232700618244	-0.033091046981020	-0.051741926755930
0.118370102475656	-0.025669799049059	0.075308765798726	-0.105883754738616	0.012484900245300	-0.019589193144041	-0.099669883868978
0.127761997976853	0.009713739070314	0.099074764095921	-0.088005974793195	-0.034357358146919	-0.002370036292100	-0.103299639258724
0.138313529119867	0.050975952260663	0.126167410350588	-0.046192828937999	-0.101926663951634	0.019227912162076	-0.038792917158250
0.150120680977768	0.098634621995161	0.156894786457244	0.022166600053331	-0.192855606562065	0.045866140805759	0.117688558199359
0.082565583058051	-0.173874152329020	0.003019166013076	0.264358356212520	0.040794121073639	-0.033224094688537	-0.169149471306866
0.086667749011208	-0.165106488741927	0.013031885746432	0.182045756618223	0.059188403432568	-0.024195407567411	0.017630768866700
0.091257655094684	-0.154084620396083	0.024214680175637	0.105392102349175	0.075283764131548	-0.015418348954959	0.105679556162079
0.096431286381548	-0.140290765818313	0.036875631195208	0.037008310631736	0.086447613777357	-0.006231430263908	0.118835166346187
0.102284627944874	-0.123207143535441	0.051322820699662	-0.020494701307732	0.090047362976775	0.004026837093014	0.080935875185938
0.108913664857731	-0.102315972074293	0.067864330583519	-0.064506016242869	0.083450422336581	0.016017941703079	0.015819958448248
0.116414382193190	-0.077099469961694	0.086808242741294	-0.092414716947315	0.064024202463553	0.030403372153562	-0.052674308099968
0.124882765024324	-0.047039855724470	0.108462639067507	-0.101609886194708	0.029136113964472	0.047844617031734	-0.100708648691796

```
    0.134414798424203   -0.011619347889446   0.133135601456674   -0.089480606758688   -0.023846432553885
0.069003164924868   -0.104444787560320
    0.145106467465899   0.029679835016553   0.161135211803313   -0.053415961412895   -0.097556026484736
0.094540504420236   -0.040044448938626
    0.157053757222482   0.077375474466702   0.192769552001942   0.009194967069033   -0.194625257221304
0.125118124105113   0.116330642940204
    0.088458859504587   -0.197089153724357   0.032036406063073   0.313105581041450   0.108875686673781
0.025243483227008   -0.169645621957305
    0.092701163356426   -0.188284520421614   0.042956089888401   0.225044480937751   0.121129680906574
0.038211561389328   0.017028234737482
    0.097431207338583   -0.177225682360120   0.055045848409579   0.142642326159301   0.131084753479417
0.050928011042973   0.104970638554082
    0.102744976524130   -0.163394858066698   0.068613763521122   0.068510033932460   0.136108314999089
0.064054320775216   0.118019865259411
    0.108738455986137   -0.146274266068176   0.083967917117549   0.005258521483590   0.133567776072370
0.078251979173331   0.080014190620383
    0.115507630797676   -0.125346124891377   0.101416391093378   -0.044501293960949   0.120830547306038
0.094182474824590   0.014791890403913
    0.123148486031817   -0.100092653063128   0.121267267343126   -0.078158495174797   0.095264039306874
0.112507296316265   -0.053808759623082
    0.131757006761633   -0.069996069110254   0.143828627761311   -0.093102164931592   0.054235662681656
0.133887932235630   -0.101949483693689
    0.141429178060194   -0.034538591559579   0.169408554242450   -0.086721386004974   -0.004887171962837
0.158985871169957   -0.105792006040992
    0.152260985000571   0.006797561062071   0.198315128681062   -0.056405241168583   -0.084737054019826
0.188462601706519   -0.041498050898077
    0.164348412655836   0.054530170227871   0.230856432971664   0.000457186803943   -0.187946572882531
0.222979612432588   0.114770657501973

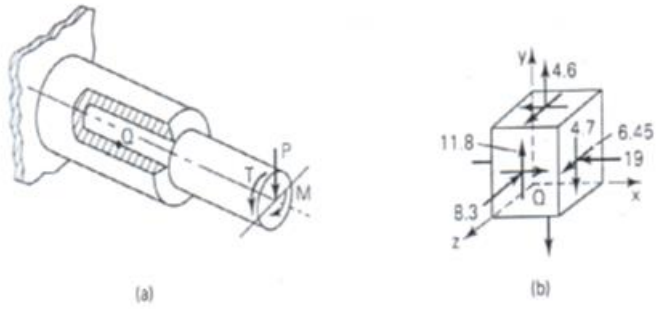
c=    32.563679591079040   -0.911800258359516   3.166225184988288   -0.619900546993170
0.186286693226662   0.043685351766361   -0.064977120190300   0.000000000000000
0.000000000000000   0.000000000000000   0.000000000000000
```

We would like to calculate principle stresses. The principle stresses are eigenvalues of the the three-dimensional stress field. Calculate the principle stresses for

$$\begin{bmatrix} 10 & 14 & 25 \\ 14 & 7 & 15 \\ 25 & 15 & 16 \end{bmatrix}$$

PROBLEM 2

A steel shaft is placed inside of a iron base as shown in the figure . T torsion and M beding moment is applied to the shaft. Under this forces stress matrix at point Q is as follows (MPa)



-19	***-4.7***	***6.45***
-4.7	**4.6**	**11.8**
6.45	**11.8**	**-8.3**

Calculate the principle stresses at point Q

Note : Principle stresses are the eigenvalues of the matrix

PROBLEM 3

$\boldsymbol{A} = \begin{bmatrix} \mathbf{1} & \mathbf{2} \\ \mathbf{3} & \mathbf{5} \end{bmatrix}$ Calculate the eigenvalues of the matrix

PROBLEM 4

$\boldsymbol{A} = \begin{bmatrix} \mathbf{1} & \mathbf{0} & \mathbf{-1} \\ \mathbf{-1} & \mathbf{1} & \mathbf{0} \\ \mathbf{0} & \mathbf{-1} & \mathbf{1} \end{bmatrix}$ Calculate the eigenvalues of the matrix

PROBLEM 5

$\boldsymbol{A} = \begin{bmatrix} \mathbf{3} & \mathbf{-2} & \mathbf{0} \\ \mathbf{2} & \mathbf{1} & \mathbf{4} \\ \mathbf{1} & \mathbf{-1} & \mathbf{2} \end{bmatrix}$ Calculate the eigenvalues of the matrix

PROBLEM 6

$A = \begin{bmatrix} 4 & -1 & 3 & 9 & -5 \\ -1 & 2 & 7 & 12 & 7 \\ 3 & 7 & 0 & -11 & 8 \\ 2 & -4 & 14 & 1 & -5 \\ 11 & -8 & 6 & -2 & 13 \end{bmatrix}$ Calculate the eigenvalues of the matrix

PROBLEM 7

$A = \begin{bmatrix} 8 & -1 & -5 \\ -4 & 4 & -2 \\ 18 & -5 & -7 \end{bmatrix}$ calculate eigenvalues and eigenvectors of he mtrix

PROBLEM 8

$\boldsymbol{A} = \begin{bmatrix} \boldsymbol{\sigma_x} & \boldsymbol{\tau_{xy}} & \boldsymbol{\tau_{xz}} \\ \boldsymbol{\tau_{yx}} & \boldsymbol{\sigma_y} & \boldsymbol{\tau_{yz}} \\ \boldsymbol{\tau_{zx}} & \boldsymbol{\tau_{zy}} & \boldsymbol{\sigma_z} \end{bmatrix} = \begin{bmatrix} \mathbf{10} & \mathbf{4} & \mathbf{-6} \\ \mathbf{4} & \mathbf{-6} & \mathbf{8} \\ \mathbf{-6} & \mathbf{8} & \mathbf{14} \end{bmatrix} \boldsymbol{MPa}$ Normal and shear stresses are given. Find the principle stresses

PROBLEM 9

Solve the following system of equation by using Singular value decomposition method.

$$\begin{bmatrix} \mathbf{1} & \mathbf{2} \\ \mathbf{2} & \mathbf{3} \\ \mathbf{4} & \mathbf{5} \\ \mathbf{6} & \mathbf{7} \end{bmatrix} \begin{bmatrix} \boldsymbol{x_0} \\ \boldsymbol{x_1} \\ \boldsymbol{x_2} \\ \boldsymbol{x_3} \end{bmatrix} = \begin{bmatrix} \mathbf{1} \\ \mathbf{1} \\ \mathbf{1} \\ \mathbf{1} \end{bmatrix}$$

PROBLEM 10 Solve the following system of equation by using SVD and Gauss elimination methods, compare the results

$$\begin{bmatrix} 8 & -1 & -5 \\ -4 & 4 & -2 \\ 18 & -5 & -7 \end{bmatrix} \begin{bmatrix} x_0 \\ x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix}$$

PROBLEM 11 Solve the roots of polynomial $f(x) = x^4 - 3x^3 + 2x^2 - x + 1$by using companion matrix and eigenvalue.

PROBLEM 12 Solve the roots of polynomial $f(x) = x^3 - 3x^2 + 3x - 1$by using companion matrix and eigenvalue.

PROBLEM 13 The following data is given. Fit the polynomial curve by using SVD

X	y
0	-2.20
0.1	-1.90
0.2	0
0.3	-0.1
0.4	-0.01
0.5	-0.3
0.6	-0.2
0.7	-0.09
0.8	0.4
0.9	-0.1
1.0	-1.1
1.1	-1.3
1.2	-2
1.3	-1.8
1.4	-0.02
1.5	1

PROBLEM 14 Fit the linear function **$f(x)=a_0*\exp(x)+a_1x+a_2/x$** the data given in problem 12 by using SVD

10.0 FINITE DIFFERENCE SOLUTIONS OF PARTIAL DIFFERENTIAL EQUATIONS

10.1 Introduction to finite difference method

Finite difference concept based on the difference concept introduced in the derivative section. We actually used finite differences to approximate to derivative of the function. In order to approximate derivation difference equations can be used. Diference equations can basically be derived from Taylor formulations

$$f(x_{i+1}) = f(x_i) + f'(x_i)h + \frac{f"(x_i)}{2}h^2 + \frac{f^{(3)}(x_i)}{3!}h^3 + \frac{f^{(4)}(x_i)}{4!}h^4 + \ldots$$

If the first derivative is solved from this equation it is found as

$$f'(x_i) = \frac{f(x_{i+1}) - f(x_i)}{h} - \frac{f"(x_i)}{2}h - \frac{f^{(3)}(x_i)}{3!}h^2 - \frac{f^{(4)}(x_i)}{4!}h^3 + O(h^4)\ldots$$

or

$$f'(x_i) = \frac{f(x_{i+1}) - f(x_i)}{h} + O(h)$$

Term O(h) is the order of error. Error is proportional to the difference parameter h. When h is getting smaller, error wil also shrinked. The second derivative can be written the same way:

$$f"(x_i) = \frac{f(x_{i+2}) - 2f(x_{i+1}) + f(x_i)}{h^2} + O(h)$$

In order to decrease the error, if the second derivative difference equation is substituted into thefirst derivative Taylor equation a first derivative equation with a second order error can be obtained.

$$f'(x_i) = \frac{f(x_{i+1}) - f(x_i)}{h} - \frac{f(x_{i+2}) - 2f(x_{i+1}) + f(x_i)}{2h^2}h + O(h^2)$$

$$f'(x_i) = \frac{2f(x_{i+1}) - 2f(x_i) - f(x_{i+2}) + 2f(x_{i+1}) - f(x_i)}{h} + O(h^2)$$

$$f'(x_i) = \frac{-f(x_{i+2}) + 4f(x_{i+1}) - 3f(x_i)}{2h} + O(h^2)$$

Difference equation can be written as forward or bakward way. The backward difference equation: $f'(x_i) = \frac{f(x_i) - f(x_{i-1})}{h} + O(h)$. By taken avarage of forward and backward formulas can create a central formula

$$f'(x_i) = \frac{f'(x_i)_{forward} + f'(x_i)_{backward}}{2} = \frac{\frac{f(x_i) - f(x_{i-1})}{h}}{2} + \frac{\frac{f(x_i) - f(x_{i-1})}{h}}{2}$$

$$= \frac{f(x_{i+1}) - f(x_{i-1})}{2h} + O(h^2)$$

Difference formulas for calculating derivatives are given (as coefficients) in the following table

Central differences

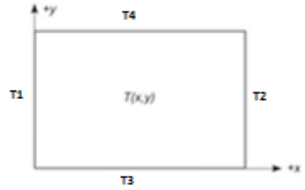
Derivative	Accuracy	−4	−3	−2	−1	0	1	2	3	4
1	2				−1/2	0	1/2			
	4			1/12	−2/3	0	2/3	−1/12		
	6		−1/60	3/20	−3/4	0	3/4	−3/20	21916	
	8	1/280	−4/105	1/5	−4/5	0	4/5	−1/5	4/105	−1/280
2	2				1	−2	1			
	4			−1/12	4/3	−5/2	4/3	−1/12		
	6		32874	−3/20	3/2	−49/18	3/2	−3/20	32874	
	8	−1/560	8/315	−1/5	8/5	−205/72	8/5	−1/5	8/315	−1/560
3	2			−1/2	1	0	−1	1/2		
	4		41122	−1	13/8	0	−13/8	1	−1/8	

	6	−7/240	41185	−169/120	61/30	0	−61/30	169/120	−3/10	7/240
4	2			1	−4	6	−4	1		
	4		−1/6	2	−13/2	28/3	−13/2	2	−1/6	
	6	7/240	−2/5	169/60	−122/15	91/8	−122/15	169/60	−2/5	7/240

Forward differences

Derivative	Accuracy	0	1	2	3	4	5	6	7	8
1	1	−1	1							
	2	−3/2	2	−1/2						
	3	−11/6	3	−3/2	1/3					
	4	−25/12	4	−3	4/3	−1/4				
	5	−137/60	5	−5	10/3	−5/4	41030			
	6	−49/20	6	−15/2	20/3	−15/4	41035	−1/6		
2	1	1	−2	1						
	2	2	−5	4	−1					
	3	35/12	−26/3	40958	−14/3	41254				
	4	41014	−77/6	107/6	−13	61/12	−5/6			
	5	203/45	−87/5	117/4	−254/9	33/2	−27/5	137/180		
	6	469/90	−223/10	879/20	−949/18	41	−201/10	1019/180	−7/10	
3	1	−1	3	−3	1					
	2	−5/2	9	−12	7	−3/2				
	3	−17/4	71/4	−59/2	49/2	−41/4	7/4			
	4	−49/8	29	−461/8	62	−307/8	13	−15/8		
	5	−967/120	638/15	−3929/40	389/3	−2545/24	268/5	−1849/120	29/15	
	6	−801/80	349/6	−18353/120	2391/10	−1457/6	4891/30	−561/8	527/30	−469/240
4	1	1	−4	6	−4	1				
	2	3	−14	26	−24	11	−2			
	3	35/6	−31	137/2	−242/3	107/2	−19	41077		
	4	28/3	−111/2	142	−1219/6	176	−185/2	82/3	−7/2	
	5	1069/80	−1316/15	15289/60	−2144/5	10993/24	−4772/15	2803/20	−536/15	967/240

Now let us consider a differential equation such as heat conduction equation in cartesian coordinates. It should be note that when a differential equation is given, in order to solve the differential equations boundary conditions should also be specified. In the example problem a steady state (independent of time) two dimensional heat conduction problem is defined with constant temperature boundary conditions.



$$\frac{\partial^2 T(x,y)}{\partial x^2} + \frac{\partial^2 T(x,y)}{\partial y^2} = 0$$

$T(x,y) = T_1$ x=0, $0 \le y \le W$
$T(x,y) = T_2$ x=L, $0 \le y \le W$
$T(x,y) = T_3$ y=0, $0 \le x \le L$
$T(x,y) = T_4$ y=W, $0 \le x \le L$

Now if second derivatives be replaced with second degree (linear) finite difference equations:

$$\frac{\partial^2 T(x,y)}{\partial x^2} \cong \frac{T_{m+1,n} - 2T_{m,n} + T_{m-1,n}}{\Delta x^2}$$

$$\frac{\partial^2 T(x,y)}{\partial y^2} \cong \frac{T_{m,n+1} - 2T_{m,n} + T_{m,n-1}}{\Delta y^2}$$

$$\frac{T_{m+1,n} - 2T_{m,n} + T_{m-1,n}}{\Delta x^2} + \frac{T_{m,n+1} - 2T_{m,n} + T_{m,n-1}}{\Delta y^2} = 0$$

It can be further simplified if $\Delta x = \Delta y$

$$T_{m+1,n} + T_{m-1,n} + T_{m,n+1} + T_{m,n-1} - 4T_{m,n} = 0$$

Now in the boundaries, boundary conditions can be applied.

x=0 (m=0) $T_{0,n} = T_1$
x=L (m=M) $T_{M,n} = T_2$
y=0 (n=0) $T_{m,0} = T_3$
y=W (n=M) $T_{m,N} = T_4$

Now this equations can be set as a linear system of equation and solved to find temperature profile.

Program 10.1 Finite difference solution of heat conduction problem with four temperature boundary layers.

```
import java.io.*;
import visad.*;
import visad.java3d.DisplayImplJ3D;
import visad.java2d.DisplayImplJ2D;
import java.rmi.RemoteException;

class HT_plane_FD_T
{
 public static double [] gauss_seidel(double a[][],double b[],double x[],double lambda)
 {
 //denklem sistemi çözümü b iteratif gauss_seidel metodu metodu
 //yakınsama faktörü lambda ile birlikte verilmiş formül
 int n=a.length;
 int imax=1000;//maksimum iterasyon sayısı
 double es=1.0e-10;
 double dummy=0;
 double ea=1e-1;
 double sum=0;
 double old=0;
 int iter=0;
 int sentinel=0;
 //System.out.print("iter = "+(iter+1)+" ");
 for(int i=0;i<n;i++)
 {  dummy=a[i][i];
   for(int j=0;j<n;j++)
     {a[i][j]/=dummy;}
   b[i]/=dummy;
 }
 for(int i=0;i<n;i++)
 {  sum=b[i];
   for(int j=0;j<n;j++)
     {if(i!=j)sum-=a[i][j]*x[j];}
   x[i]=sum;
   //System.out.print(x[i]+ "  ");
 }
  //System.out.println();
 iter++;
 do
```

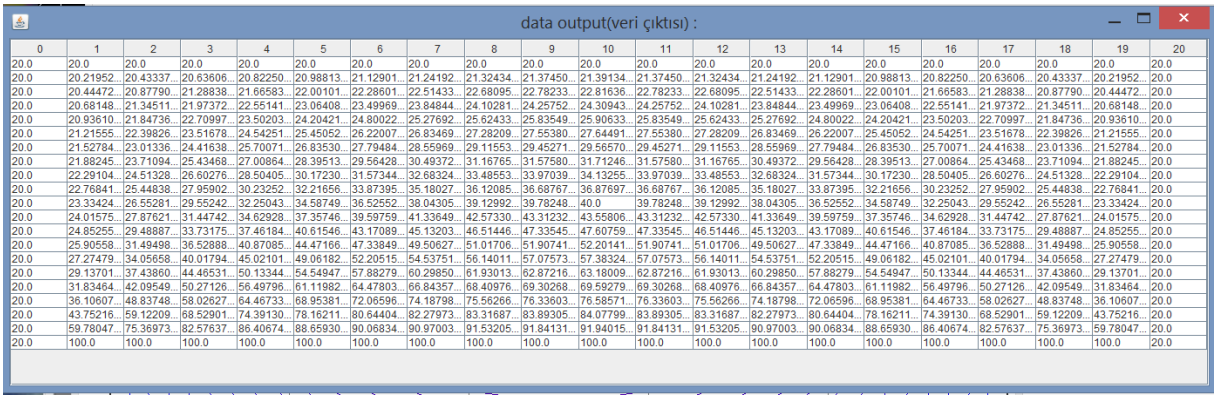
```
{
sentinel=1;
//System.out.print("iter = "+(iter+1)+" ");
for(int i=0;i<n;i++)
  {
              old=x[i];
              sum=b[i];
              for(int j=0;j<n;j++)
                 {if(i!=j) sum-=a[i][j]*x[j];}
                 x[i]=lambda*sum+(1.0-lambda)*old;
                 if(sentinel==1 && x[i]!=0.0 ) {ea=Math.abs((x[i]-old)/x[i])*100.0;}
                 if(ea<es) sentinel=0;
               //System.out.print(x[i]+ "  ");
  }
//System.out.println();
iter++;
}while(sentinel==1 && iter<imax);
if(iter>=imax) System.out.println("Maksimum iterasyon sayısı aşıldı sonuç geçerli olmayabilir iter = "+iter);
return x;
}

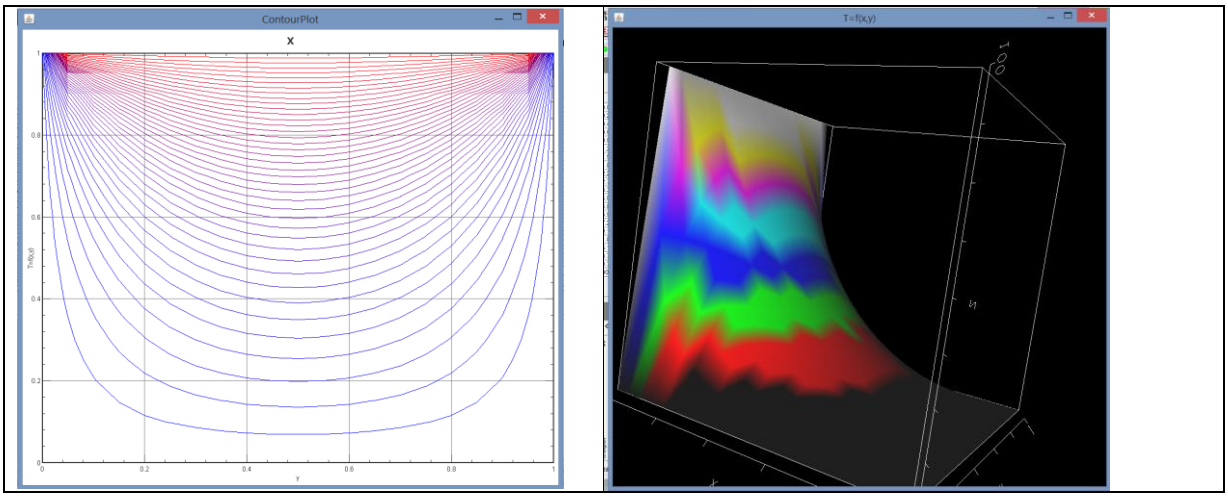
public static double[] pivotlugauss(double a[][],double b[])
{ //kısmi pivotlu gauss eleme yöntemi
int n=b.length;
double x[]=new double[n];
double carpan=0;
double toplam=0;
double buyuk;
double dummy=0;
//gauss eleme
int i,j,k,p,ii,jj;
//System.out.println("Orijinal Matris :\n"+Matrix.toString(a));
for(k=0;k<(n-1);k++)
{ //pivotlama
            p=k;
            buyuk=Math.abs(a[k][k]);
            for(ii=k+1;ii<n;ii++)
            {  dummy=Math.abs(a[ii][k]);
              if(dummy > buyuk) {buyuk=dummy;p=ii;}
            }
            if(p!=k)
            { for(jj=k;jj<n;jj++)
              { dummy=a[p][jj];
                a[p][jj]=a[k][jj];
                a[k][jj]=dummy;
              }
              dummy=b[p];
              b[p]=b[k];
              b[k]=dummy;
            }
              //System.out.println("pivotlu Matris : k="+k+"\n"+Matrix.toString(a));
            //gauss elemeyi çözme
  for(i=k+1;i<n;i++)
  {  carpan=a[i][k]/a[k][k];
     a[i][k]=0;
     for(j=k+1;j<n;j++)
     {   a[i][j]-=carpan*a[k][j]; }
        b[i]   =b[i]   -carpan*b[k];
            }
}
//back substitution
x[n-1]=b[n-1]/a[n-1][n-1];
for(i=n-2;i>=0;i--)
{
  toplam=0;
  for(j=i+1;j<n;j++)
  {  toplam+=a[i][j]*x[j];}
  x[i]=(b[i]-toplam)/a[i][i];
}
return x;
}


public static double [][] HT_planecoz( double L,double W,double dx, double Tx0,double Txn,double Ty0,double Tyn)
```

```
  {
  int nx=(int)(L/dx+0.01);
  int ny=(int)(W/dx+0.01);
  int mx=(nx-1);
  int my=(ny-1);
  int mt=mx*my;
  double A[][]=new double[mt][mt];
  double b[]=new double[mt];
  double x[]=new double[mt];
  double x1,x2,y1,y2;
  // boundary conditions
  int i=0,j=0;
  for(i=0;i<my;i++)
  {
    for(j=0;j<mx;j++)
    {
                x1=dx*j;
                x2=(L-x1);
                y1=dx*i;
                y2=(W-y1);
                int k=i*my+j;
                x[k]=((x1*Tx0+x2*Txn)/L+(y1*Ty0+y2*Tyn)/W)/2.0;
                int kp1=(i+1)*my+j;
                int km1=(i-1)*my+j;
                if(j==0)          {b[k]+=-Tx0;}
    if(j== (mx-1))     {b[k]+=-Txn;}
    if(i==0)          {b[k]+=-Ty0;}
    else if(i==(my-1))  {b[k]+=-Tyn;}
    A[k][k]+=-4.0;
    if((i==0)&&(j==0)) {A[kp1][k]+=1.0;A[k][k+1]+=1.0;}
    else if((i==0)&&(j==(mx-1))) {A[kp1][k]+=1.0;A[k][k-1]+=1.0;}
    else if((i==(my-1))&&(j==0)) {A[km1][k]+=1.0;A[k][k+1]+=1.0;}
    else if((i==(my-1))&&(j==(mx-1))) {A[km1][k]+=1.0;A[k][k-1]+=1.0;}
    else if(j==0) {A[kp1][k]+=1.0;A[km1][k]+=1.0;A[k][k+1]+=1.0;}
    else if(i==0) {A[k][k+1]+=1.0;A[k][k-1]+=1.0;A[kp1][k]+=1.0;}
    else if(j==(mx-1)) {A[kp1][k]+=1.0;A[km1][k]+=1.0;A[k][k-1]+=1.0;}
    else if(i==(my-1)) {A[km1][k]+=1.0;A[k][k-1]+=1.0;A[k][k+1]+=1.0;}
    else {A[km1][k]+=1.0;A[kp1][k]+=1.0;A[k][k-1]+=1.0;A[k][k+1]+=1.0;}
    }
  }
  Text.print(A,"A");
  Text.print(b,"B");
  double T[][]=new double[ny+1][nx+1];
  double a[]=conjugate_gradient1.conjugate_gradient(A,b);
  for(i=0;i<=ny;i++)
  { for(j=0;j<=nx;j++)
  {if(j==0) {T[i][j]=Tx0;}
   else if(i==0) {T[i][j]=Ty0;}
   else if(j==nx) {T[i][j]=Txn;}
   else if(i==ny) {T[i][j]=Tyn;}
   else
   {int k=(i-1)*my+(j-1);
    T[i][j]=a[k];
   }
   }

  }
   return T;
}
  public static void main(String args[]) throws IOException,RemoteException, VisADException
  {  double a[][]=HT_planecoz(1.0,1.0,0.05,20.0,20.0,20.0,100.0);
     Text.print(a);
                ContourPlot pp=new ContourPlot(a,0.0,1.0,0.0,1.0,40, false,"x","y","T=f(x,y)");
                pp.plot();
                Plot3D.data(a,0.0,1.0,0.0,1.0,400,"x","y","T=f(x,y)");
  }
}
```





Finite difference equations can be set for a higher degree, but in this case boundaries will be penitrate into more layers so the equations should be set more carefully. Considering the same differential equation with second order derivatives:

$$\frac{\partial^2 T(x,y)}{\partial x^2} \cong \frac{-T_{m-2,n} + 16T_{m-1,n} - 30T_{m,n} + 16T_{m+1,n} - T_{m+2,n}}{12\Delta x^2}$$

$$\frac{\partial^2 T(x,y)}{\partial y^2} \cong \frac{-T_{m,n-2} + 16T_{m,n-1} - 30T_{m,n} + 16T_{m,n+1} - T_{m,n+2}}{12\Delta y^2}$$

And differential equation becomes.

$$\frac{-T_{m-2,n} + 16T_{m-1,n} - 30T_{m,n} + 16T_{m+1,n} - 2T_{m+2,n}}{12\Delta x^2} + \frac{-T_{m,n-2} + 16T_{m,n-1} - 30T_{m,n} + 16T_{m,n+1} - 2T_{m,n+2}}{12\Delta y^2} = 0$$

The difficulty of using this equation is requirement of the two layers of coefficients existed in the equation. In the first layer external boundary conditions will apply, but for the second layer first degree difference equation can be used. So the boundary conditions for this equation will be:

x=0 (m=0) $T_{0,n} = T_1$

$x = \Delta x$ (m=1) $\frac{T_{2,n} - 2T_{1,n} + T_1}{\Delta x^2} + \frac{T_{1,n+1} - 2T_{1,n} + T_{1,n-1}}{\Delta y^2} = 0$

x=L (m=M) $T_{M,n} = T_2$

$x = L - \Delta x$ (m=M-1) $\frac{T_{M-2,n} - 2T_{M-1,n} + T_2}{\Delta x^2} + \frac{T_{M-1,n+1} - 2T_{M-1,n} + T_{M-1,n-1}}{\Delta y^2} = 0$

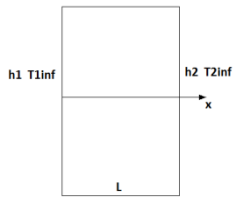
y=0 (n=0) $T_{m,0} = T_3$

$y = \Delta y$ (n=1) $\frac{T_{m+1,1} - 2T_{m,1} + T_{m-1,1}}{\Delta x^2} + \frac{T_{m,2} - 2T_{m,1} + T_3}{\Delta y^2} = 0$

y=W (n=N) $T_{m,N} = T_4$

$y = W - \Delta y$ (n=N-1) $\frac{T_{m+1,N} - 2T_{m,N} + T_{m-1,N}}{\Delta x^2} + \frac{T_{m,N-2} - 2T_{m,N-1} + T_4}{\Delta y^2} = 0$

As another example : In this example differential equation is function of x and t (time dependent heat transfer equation)



$$\frac{\partial^2 T(x,t)}{\partial x^2} = \frac{1}{\alpha}\frac{\partial T(x,t)}{\partial t}$$

Boundary conditions

$-k\frac{\partial T(x,t)}{\partial x} = h_1(T_{1\infty} - T(x,t))$ x=0 t >0

$-k\frac{\partial T(x,t)}{\partial x} = h_2(T(x,t) - T_{2\infty})$ x=L t >0

$T(x,t) = F(x)$ t=0 0<=x<=L

Where Tk and h_1 and h_2 , $T_{1\infty} and T_{2\infty}$are constants

$$\varphi\frac{T_{m+1}^p - 2T_m^p + T_{m-1}^p}{\Delta x^2} + (1-\varphi)\frac{T_{m+1}^p - 2T_m^p + T_{m-1}^p}{\Delta x^2} = \frac{1}{\alpha}\frac{T_m^{p+1} - T_m^p}{\Delta t}$$

$t = p\Delta t \quad x = m\Delta x Fo = \frac{\alpha\Delta t}{\Delta x^2} \; Bi = \frac{h\Delta x}{k} \; \alpha = \frac{k}{\rho C_p}$

By rearranging the equation

$$\varphi Fo T_{m+1}^p + (1 - 2\varphi Fo)T_m^p + \varphi Fo T_{m-1}^p = -(1-\varphi)Fo T_{m+1}^{p+1} + (1 + 2(1-\varphi)Fo)T_m^{p+1} - (1-\varphi)Fo T_{m-1}^{p+1}$$

$\varphi = 1$explicit solution (in this case solution of the system of equaton in each step is not required)

$\varphi = 0$implicit solution

Any other φmixed solution

Boundary conditions:

$$\varphi h_1 A(T_{1\infty} - T_0^p) + \varphi\frac{kA}{\Delta x}(T_1^p - T_0^p) + (1-\varphi)h_1 A(T_{1\infty} - T_0^{p+1}) + \varphi\frac{kA}{\Delta x}(T_1^{p+1} - T_0^{p+1}) = \rho C_p(A\frac{\Delta x}{2})\frac{T_0^{p+1} - T_0^p}{\Delta t}$$

$$\varphi h_2 A(T_{2\infty} - T_{k-1}^p) + \varphi\frac{kA}{\Delta x}(T_{k-2}^p - T_{k-1}^p) + (1-\varphi)h_2 A(T_{2\infty} - T_{k-1}^p) + (1-\varphi)\frac{kA}{\Delta x}(T_{k-2}^p - T_{k-1}^p) = \rho C_p(A\frac{\Delta x}{2})\frac{T_{k-1}^{p+1} - T_{k-1}^p}{\Delta t}$$

By rearranging the equations

$$2\varphi Bi_1 Fo(T_{1\infty} - T_0^p) + 2\varphi(T_1^p - T_0^p) + 2(1-\varphi)Bi_1 Fo(T_{1\infty} - T_0^{p+1}) + 2(1-\varphi)(T_1^{p+1} - T_0^{p+1}) = T_0^{p+1} - T_0^p$$

$$2\varphi Bi_2 Fo(T_{2\infty} - T_{k-1}^p) + 2\varphi(T_{k-2}^p - T_{k-1}^p) + 2(1-\varphi)Bi_2 Fo(T_{2\infty} - T_{k-1}^p) + 2(1-\varphi)(T_{k-2}^p - T_{k-1}^p) = T_{k-1}^{p+1} - T_{k-1}^p$$

Program 10.2 Finite difference solution of time dependent heat conduction problem with convective boundary layers.

```
import java.io.*;

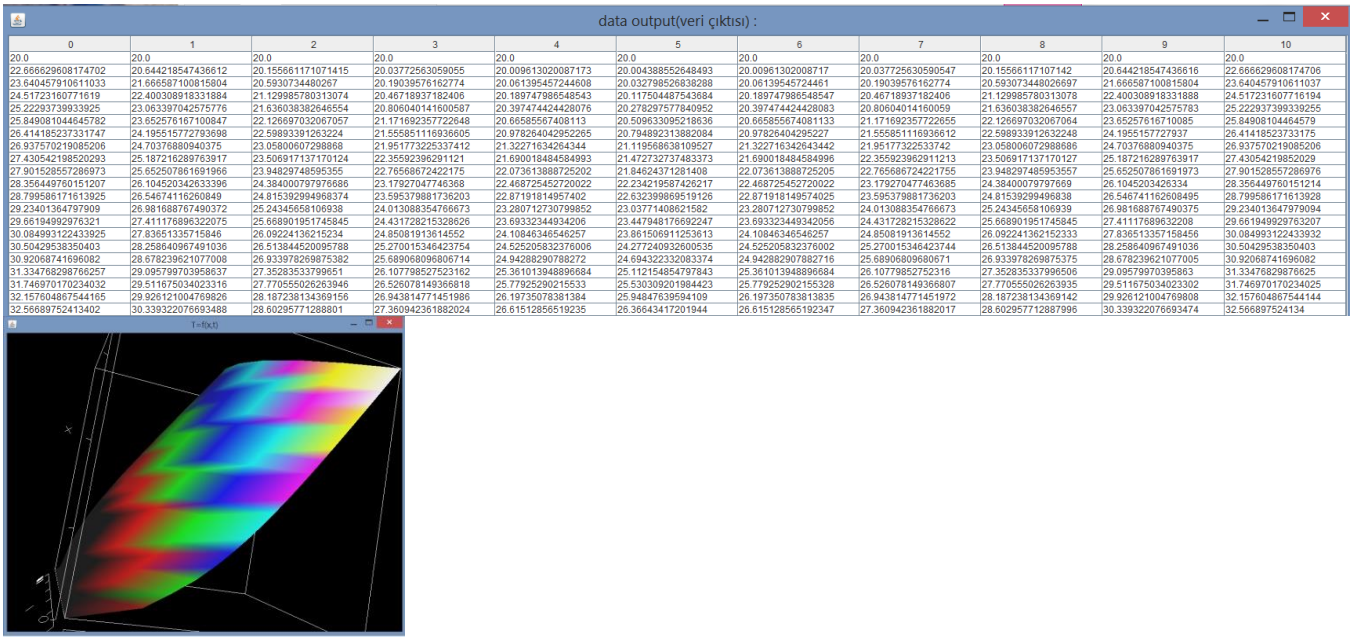
class HT_xt_FDB
{ public static double [] thomas(double a[][],double b[])
  {
  //3 band matrix
  int n=a.length;
  double f[]=new double[n];
  double e[]=new double[n];
  double g[]=new double[n];
  double x[]=new double[n];
```

```
  for(int i=0;i<n;i++)     {f[i]=a[i][i];}
  for(int i=0;i<(n-1);i++) {g[i]=a[i][i+1];}
  for(int i=0;i<(n-1);i++) {e[i+1]=a[i+1][i];}
  for(int k=1;k<n;k++)
     {e[k]=e[k]/f[k-1];
      f[k]=f[k]-e[k]*g[k-1];
     }
  for(int k=1;k<n;k++)
     {b[k]=b[k]-e[k]*b[k-1];
     }
  x[n-1]=b[n-1]/f[n-1];
  for(int k=(n-2);k>=0;k--)
      {x[k]=(b[k]-g[k]*x[k+1])/f[k];}
  return x;
  }
  public static double [][] q(double k, double dx,double T[][])
  {  //heat transfer
                int nx=T[0].length;
   int nt=T.length;
   double qi[][]=new double[nt][nx-1];
     for(int p=0;p<nt;p++)
     {  for(int m=1;m<nx;m++)
        { qi[p][m-1]=-k*(T[p][m]-T[p][m-1])/dx;}
     }
     return qi;
  }

  public static double [][] HT_xtcoz(double fi, double L,double t, double dt,double T0[],double alpha,double Bi1,double Bi2,double
T1inf,double T2inf)
  {
  //  L duvar kalınlığı
  //  t maksimum zaman
  //  dt zaman stebi
  //  T1 x=0 t>=0 boundary temperature
  //  T2 x=L t>=0 boundary temperature
  int nx=T0.length; //x step sayısı
  //System.out.println(nx="+nx)
  double A[][]=new double[nx][nx];
  double B[]=new double[nx];
  double dx=L/(nx-1); //x steb uzunluğu
  int nt=(int)(t/dt+0.001);
  double Fo=alpha*dt/(dx*dx);
  double T[][]=new double[nt+1][nx];
  for(int m=0;m<nx;m++) {T[0][m]=T0[m];}
  for(int p=0;p<nt;p++)
  { //t>0 wall temperatures
              for(int m=0;m<nx;m++)
    {
                 if(m==0) B[m]=(1.0-2.0*Fo*fi-2*Fo*Bi1*fi)*T[p][m]+2.0*fi*Fo*T[p][m+1]+2.0*Bi1*Fo*T1inf;
                 else if(m==(nx-1)) B[m]=(1.0-2.0*Fo*fi-2*Fo*Bi2*fi)*T[p][m]+2.0*fi*Fo*T[p][m-1]+2.0*Bi2*Fo*T2inf;
                 else  B[m]=fi*Fo*T[p][m-1]+(1.0-2.0*Fo*fi)*T[p][m]+fi*Fo*T[p][m+1];
                 if(m==0)
                 {A[m][m]=(1+2.0*Fo*Bi1*(1.0-fi)+2.0*Fo*(1.0-fi));
                  A[m][m+1]=-2.0*(1.0-fi)*Fo;
                 }
                 else if(m==(nx-1))
                 {A[m][m]=(1+2.0*Fo*Bi2*(1.0-fi)+2.0*Fo*(1.0-fi));
                  A[m][m-1]=-2.0*(1.0-fi)*Fo;
                 }
                 else
                 {A[m][m]=(2.0*Fo*(1.0-fi)+1);
                  A[m][m-1]=-(1.0-fi)*Fo;
                  A[m][m+1]=-(1.0-fi)*Fo;
                 }
```

```
  }
  T[p+1]=thomas(A,B);
 }
 return T;
 }

 public static void main(String args[])
 {
            double T0[]=new double[81];
            double L=0.8;
            // t=0 daki T0(x) sıcaklık profili
            int N=T0.length;
            for(int m=0;m<N;m++) {T0[m]=300-300.0*Math.sin(m*Math.PI/N);}
            double T1inf=100.0;  //x=0 da t>0 zamanı için duvar sıcaklığı sınır şartı
            double T2inf=25.0;  //x=L de t>0 zamanı için duvar sıcaklığı sınır şartı
            double h1=50.0;  //x=0 da t>0 zamanı için duvar sıcaklığı sınır şartı
            double h2=50.0;  //x=L de t>0 zamanı için duvar sıcaklığı sınır şartı
            double t=600.0; //sn toplam zaman
            double dt=1.0; //sn gerçek zaman adımı
            double dtplot=0.3; //sn her bir plotta bekleme zamanı
            double alpha=8.4e-5;
            double x0=0.0;
             double xn=L;
            double fi=0.5;
            double k=200.0;
            double dx=L/T0.length;
            double h=250.0;
            double Bi1=h*dx/k;
            double Bi2=h*dx/k;
            double T[][]=HT_xtcoz(fi,L,t,dt,T0,alpha,Bi1,Bi2,T1inf,T2inf);
            double b[][]=Matrix.T(T);
            double q[][]=q(k,dx,T);
            timeplot p=new timeplot(T,x0,xn,dtplot,dt);
            //timeplot p=new timeplot(q,x0,xn,dtplot,dt);
    p.start();
 }
}
```



11.0 FINITE ELEMENT SOLUTIONS OF DIFFERENTIAL EQUATIONS

11.1 Finite Element

The simplest finite element will be the linear one dimensional element. The linear temperature variation in the element is represented by:

$T(x,y) = \alpha_1 + \alpha_2 x$where T is independent variable (tempearture if heat transfer problems are considered). Since there are two arbitrary constants in the linear representation, it requires only two nodes to determine the values of α_1, and α_2, namely,

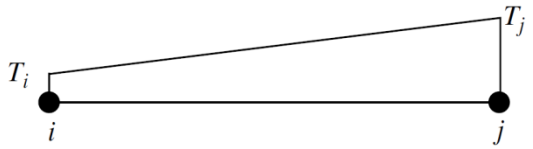


Figure 11.1 linear one dimensional finite element

$$T_i = \alpha_1 + \alpha_2 x_i$$
$$T_j = \alpha_1 + \alpha_2 x_j$$

From the above equation we get

$$\alpha_1 = \frac{T_i x_j - T_j x_i}{x_j - x_i}$$
$$\alpha_1 = \frac{T_j - T_i}{x_j - x_i}$$

On substituting the values α_1, and α_2, into the main equation, we get

$T = T_i\left[\frac{T_i x_j - T_j x_i}{x_j - x_i}\right] + T_j\left[\frac{T_j - T_i}{x_j - x_i}\right]$ or

$$T = N_i T_i + N_j T_j = [N_i \quad N_j]\begin{Bmatrix} T_i \\ T_j \end{Bmatrix}$$

Where N_i and $N_j$are called shape functions or interpolation functions. Equation can be rewritten as

$T = [N]\{T\}$ where $[N] = [N_i \quad N_j]$and $\{T\} = \begin{Bmatrix} T_i \\ T_j \end{Bmatrix}$This is basically linear interpolation formula that formulated in the shape function form. The derivative of the function can be easily formulated from this function

$$\frac{dT}{dx} = \frac{dN_i}{dx}T_i + \frac{dN_j}{dx}T_j = -\frac{1}{x_j - x_i}T_i + \frac{1}{x_j - x_i}T_j$$

$\frac{dT}{dx} = -\frac{1}{l}T_i + \frac{1}{l}T_j$ where $l = x_j - x_i$we can write the derivative equation as

$$g = \frac{dT}{dx} = \left[-\frac{1}{l} \quad \frac{1}{l}\right]\begin{Bmatrix} T_i \\ T_j \end{Bmatrix} = [B]\{T\}$$

If linear curve fitting representation will not be sufficent for the accuracy, degree of shape function can be increased. I a quadratic approximation is required:

$$T(x,y) = \alpha_1 + \alpha_2 x + \alpha_3 x^2$$

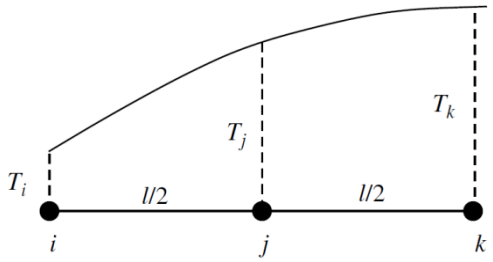


Figure 11.2 quadratic one dimensional finite element

$$T_i = \alpha_1$$
$$T_j = \alpha_1 + \alpha_2\left(\frac{l}{2}\right) + \alpha_3\left(\frac{l}{2}\right)^2$$

$T_k = \alpha_1 + \alpha_2 l + \alpha_3 l^2$substituting these values back into the equation gives:

$$T = N_i T_i + N_j T_j + N_k T_k = [N_i \quad N_j \quad N_k] \begin{Bmatrix} T_i \\ T_j \\ T_k \end{Bmatrix}$$

$$N_i = \left[1 - \frac{3x}{l} + \frac{2x^2}{l^2}\right]$$

$$N_j = \left[\frac{4x}{l} - \frac{4x^2}{l^2}\right]$$

$$N_k = \left[-\frac{x}{l} + \frac{2x^2}{l^2}\right]$$

The derivative of the function will be

$$g = \frac{dT}{dx} = \left[\left(\frac{4x}{l^2} - \frac{3}{l}\right) \quad \left(\frac{4}{l} - \frac{8x}{l^2}\right) \quad \left(\frac{4x}{l^2} - \frac{1}{l}\right)\right] \begin{Bmatrix} T_i \\ T_j \\ T_k \end{Bmatrix} = [B]\{T\}$$

By using a linear or non-linear simplex elements a surface curve fitting can be achieved. The simplest geometric shape that can be employed to approximate irregular surfaces is the triangle and it is one of the popular elements currently used in finite element calculations. The two-dimensional linear triangular element, also known as a *simplex element*, is represented by

$$T(x, y) = \alpha_1 + \alpha_2 x + \alpha_3 y$$

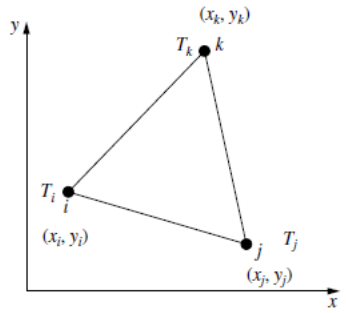


Figure 11.3 triangular finite element

where the polynomial is linear in *x* and *y* and contains three coefficients. Since a linear triangle has three nodes, the values of α_1, α_2 and α_3 are determined from

$T_i=\alpha_1+\alpha_2 x_i+\alpha_3 y_i$

$T_j=\alpha_1+\alpha_2 x_j+\alpha_3 y_j$

$T_k=\alpha_1+\alpha_2 x_k+\alpha_3 y_k$

which results in the following:

$$\alpha_1 = \frac{1}{2A}\left[(x_j y_k - x_k y_j)T_i + (x_k y_i - x_i y_k)T_j + (x_i y_j - x_j y_i)T_k\right]$$

$$\alpha_2 = \frac{1}{2A}\left[(y_j - y_k)T_i + (y_k - y_i)T_j + (y_i - y_j)T_k\right]$$

$$\alpha_3 = \frac{1}{2A}\left[(x_k - x_j)T_i + (x_i - x_k)T_j + (x_j - x_i)T_k\right]$$

where '*A*' is the area of the triangle given by

$$2A = det\begin{bmatrix} 1 & x_i & y_i \\ 1 & x_j & y_j \\ 1 & x_k & y_k \end{bmatrix} = (x_i y_j - x_j y_i) + (x_k y_i - x_i y_k) + (x_j y_k - x_k y_j)$$

Substituting the values of α_1, α_2 and α_3 into the equation and collating the coefficients of T_i, T_j and T_k, we get

$$T = N_i T_i + N_j T_j + N_k T_k$$

$$N_i = \frac{1}{2A}[a_i + b_i x + c_i y]$$

$$N_j = \frac{1}{2A}\left[a_j + b_j x + c_j y\right]$$

$$N_k = \frac{1}{2A}[a_k + b_k x + c_k y]$$

$$a_i = (x_j y_k - x_k y_j) a_j = (x_k y_i - x_i y_k) a_k = (x_i y_j - x_j y_i)$$
$$b_i = (y_j - y_k) b_j = (y_k - y_i) b_k = (y_i - y_j)$$
$$c_i = (x_k - x_j) c_j = (x_i - x_k) c_k = (x_j - x_i)$$

If we evaluate N_i at node i, where the coordinates are (x_i,y_i), then we obtain $N_i(x_i,y_i)=1$ and similarly $N_i(x_j,y_j)=0$, $N_i(x_k,y_k)=0$ Thus, we see that the shape functions have a value of unity at the designated vertex and zero at all other vertices. It is possible to show that $N_i+N_j+N_k=1$ everywhere in the element, including the boundaries. Linear simplex is given us a relatively simple way of interpolating any point in the surface (a linear interpolation). So the temperature profile of The simplex element can be calculated as:

$$T = N_i T_i + N_j T_j + N_k T_k = [N_i N_j N_k] \begin{bmatrix} T_i \\ T_j \\ T_k \end{bmatrix} = [N]\{T\}$$

The derivative of the temperature field can be given as:

$$\{g\} = \begin{bmatrix} \frac{\partial T}{\partial x} \\ \frac{\partial T}{\partial y} \end{bmatrix} = \frac{1}{2A} \begin{bmatrix} b_i & b_j & b_k \\ c_i & c_j & c_k \end{bmatrix} \begin{bmatrix} T_i \\ T_j \\ T_k \end{bmatrix} = [B]\{T\}$$Therefore heat transfer can be given as:

$$\begin{bmatrix} q_x \\ q_y \end{bmatrix} = -k[B]\{T\} = -\frac{k}{2A} \begin{bmatrix} b_i & b_j & b_k \\ c_i & c_j & c_k \end{bmatrix} \begin{bmatrix} T_i \\ T_j \\ T_k \end{bmatrix}$$

Integration of the finite element can be carried out as:

One dimensional integral: $\int_l N_i^a N_j^b N_k^c dl = \frac{a!b!c!}{(a+b+c+1)!} 2l$

Two dimensional integral $\int_A N_i^a N_j^b N_k^c dA = \frac{a!b!c!}{(a+b+c+2)!} 2A$

Different Finite elements can be defined the most common two dimensional element after tridigonal element is quadrilateral elements. Quadrilateral element is described by

$$T(x,y) = \alpha_1 + \alpha_2 x + \alpha_3 y + \alpha_4 xy$$

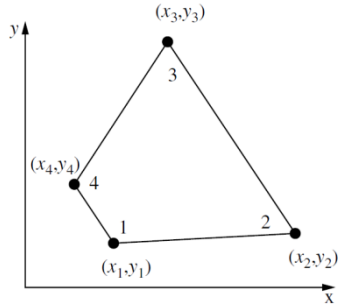


Figure 11.4 quadrilateral finite element

A special case of this element, rectangular element is used commonly.

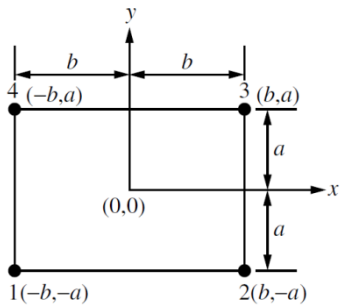


Figure 11.3 rectangular finite element

For this element shape function can be given as:

$$T(x,y) = N_1 T_1 + N_2 T_{21} + N_3 T_3 + N_4 T_4$$
$$N_1 = \frac{1}{4ab}(b-x)(a-y)$$
$$N_2 = \frac{1}{4ab}(b+x)(a-y)$$

$$N_3 = \frac{1}{4ab}(b+x)(a+y)$$
$$N_2 = \frac{1}{4ab}(b-x)(a+y)$$

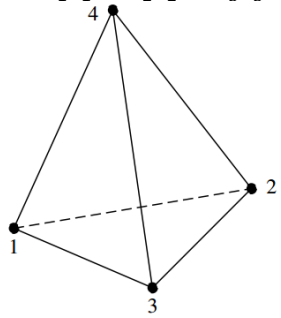
The derivatives can be expressed as follows:

$$\{g\} = \begin{bmatrix} \frac{\partial T}{\partial x} \\ \frac{\partial T}{\partial y} \end{bmatrix} = \frac{1}{4ab}\begin{bmatrix} -(a-y) & (a-y) & (a+y) & -(a+y) \\ -(b-x) & -(b+x) & (b+x) & (b-x) \end{bmatrix}\begin{bmatrix} T_1 \\ T_2 \\ T_3 \\ T_4 \end{bmatrix} = [B]\{T\}$$

As the last thing in this part, let us consider a three dimensional simplex element as well. The linear temperature representation of a tree-dimensional linear element (tetrahedron) is given by:

$T(x,y,z) = \alpha_1 + \alpha_2 x + \alpha_3 y + \alpha_4 z$The constatnt of equation can be solved in a simialar manner of 2D simplex

$T = N_1T_1 + N_2T_2 + N_3T_3 + N_4T_4$



Where

$$N_i = \frac{1}{6V}[a_i + b_i x + c_i y + d_i z]$$

The volume of the simplex element is:

$$6V = \det\begin{bmatrix} 1 & x_1 & y_1 & z_1 \\ 1 & x_2 & y_2 & z_2 \\ 1 & x_3 & y_3 & z_3 \\ 1 & x_4 & y_4 & z_4 \end{bmatrix}$$

$$\{g\} = \begin{bmatrix} \frac{\partial T}{\partial x} \\ \frac{\partial T}{\partial y} \\ \frac{\partial T}{\partial z} \end{bmatrix} = \frac{1}{6V}\begin{bmatrix} b_1 & b_1 & b_1 & b_1 \\ c_1 & c_2 & c_3 & c_4 \\ d_1 & d_2 & d_3 & d_4 \end{bmatrix}\begin{bmatrix} T_1 \\ T_2 \\ T_3 \\ T_4 \end{bmatrix} = [B]\{T\}$$Therefore heat transfer can be given as:

$$\begin{Bmatrix} q_x \\ q_y \\ q_z \end{Bmatrix} = -k[B]\{T\} = -\frac{k}{6V}\begin{bmatrix} b_1 & b_1 & b_1 & b_1 \\ c_1 & c_2 & c_3 & c_4 \\ d_1 & d_2 & d_3 & d_4 \end{bmatrix}\begin{bmatrix} T_1 \\ T_2 \\ T_3 \\ T_4 \end{bmatrix}$$

Integration of the finite element can be carried out as:

Two dimensional integral $\int_V N_1^a N_2^b N_3^c N_4^c dV = \frac{a!b!c!d!}{(a+b+c+d+3)!}6V$

11.2 Solution of the differential equation by using finite element

Now general steady state general heat conduction equation can be investigated and converted into an integral equation

$$\frac{\partial}{\partial x}\left(k_x \frac{\partial T}{\partial x}\right) + \frac{\partial}{\partial y}\left(k_y \frac{\partial T}{\partial y}\right) + g = 0$$

Boundary conditions:

T=T_b on surface S_1

$k_x \frac{\partial T}{\partial x}\vec{l} + k_y \frac{\partial T}{\partial y}\vec{m} + q = 0$ on surface S_2

$k_x \frac{\partial T}{\partial x}\vec{l} + k_y \frac{\partial T}{\partial y}\vec{m} + h(T - T_a) = 0$on surface S_3

There are several integration methods that can be applied into the differential equations to solve them. One of the most common is Galarkin method.. The method requires that the following expression will be satisfied $\int_\Omega w_k L(T)\, d\Omega = 0$where $\Omega$is the volume of the element. Where $w_k$is a node element and L(T) is the differential equation. When it is applied to the heat conduction problem above:

$$\int_\Omega N_k \left[\frac{\partial}{\partial x}\left(k_x \frac{\partial T}{\partial x}\right) + \frac{\partial}{\partial y}\left(k_y \frac{\partial T}{\partial y}\right) + g\right] d\Omega = 0$$

Note that for finite element T=[N]{T} and integral can be converted by partial integration rule

$$\int udv = uv - \int vdu$$

So

$$\int_\Omega \alpha_1 \frac{\partial \alpha_2}{\partial x_1} d\Omega = \int_s \alpha_1 \alpha_2 dS - \int_\Omega \frac{\partial \alpha_1}{\partial x_1} \alpha_2 d\Omega$$

$$\int_\Omega N_k \left[\frac{\partial}{\partial x}\left(k_x \frac{\partial T}{\partial x}\right)\right] d\Omega = \int_s N_k \left(k_x \frac{\partial T}{\partial x}\right) dS - \int_\Omega \frac{\partial N_k}{\partial x} k_x \frac{\partial N_m}{\partial x} T d\Omega$$

By applying these equation becomes

$$-\int_\Omega \left[k_x \frac{\partial N_k}{\partial x}\frac{\partial N_m}{\partial x} + k_y \frac{\partial N_k}{\partial y}\frac{\partial N_m}{\partial y}\right]\{T_m\} d\Omega + \int_\Omega g N_k d\Omega + \int_s N_k \left(k_x \frac{\partial T}{\partial x} + k_y \frac{\partial T}{\partial y}\right) dS = 0$$

From boundary conditions:

$$\int_s N_k \left(k_x \frac{\partial T}{\partial x} + k_y \frac{\partial T}{\partial y}\right) dS = -\int_s N_k (q + h(T - T_a) dS = -\int_s N_k q dS - \int_s N_k h N_m \{T_m\} dS + \int_s N_k h T_a dS$$

$-\int_\Omega \left[k_x \frac{\partial N_k}{\partial x}\frac{\partial N_m}{\partial x} + k_y \frac{\partial N_k}{\partial y}\frac{\partial N_m}{\partial y}\right]\{T_m\} d\Omega + \int_\Omega g N_k d\Omega + -\int_s N_k q dS - \int_s N_k h N_m \{T_m\} dS + \int_s N_k h T_a dS = 0$ If this equation is written in matrix form:

$[K]\{T\} = \{f\}$ where

$$[K] = -\int_\Omega \left[k_x \frac{\partial N_k}{\partial x}\frac{\partial N_m}{\partial x} + k_y \frac{\partial N_k}{\partial y}\frac{\partial N_m}{\partial y}\right] d\Omega + \int_S h N_k (N_m) dS$$

$$\{f\} = \int_\Omega g N_k d\Omega - \int_s N_k q dS + \int_S h N_k T_a dS$$

If equation put into the matrix form:

$$[K] = -\int_\Omega [B]^T [D][B]\, d\Omega + \int_S h[N]^T [N] dS$$

where

$$[D] = \begin{bmatrix} k_x & 0 \\ 0 & k_y \end{bmatrix}$$

$$\{f\} = \int_\Omega g[N]^T d\Omega - \int_s [N]^T q dS + \int_S h[N]^T T_a dS$$

$\int_l N_i^a N_j^b dl = \frac{a!b!}{(a+b+1)!} l \quad \int_l N_i^2 dl = \frac{2!0!}{(2+0+1)!} l = \frac{2l}{6} \quad \int_l N_i N_j dl = \frac{1!1!}{(1+1+1)!} l = \frac{l}{6}$

Matrix terms can be opened for two dimensional triangular element as:

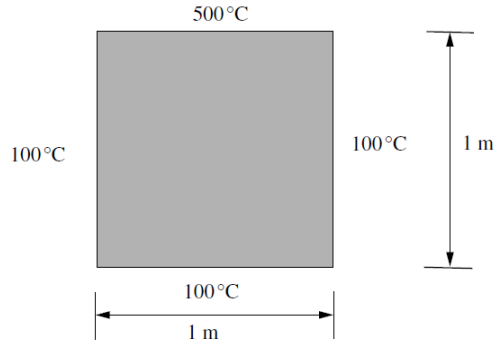
$$[K]=\frac{t}{4A}\left\{k_x\begin{bmatrix} b_i^2 & b_ib_j & b_ib_k \\ b_ib_j & b_j^2 & b_jb_k \\ b_ib_k & b_jb_k & b_k^2 \end{bmatrix}+k_y\begin{bmatrix} c_i^2 & c_ic_j & c_ic_k \\ c_ic_j & c_j^2 & c_jc_k \\ c_ic_k & c_jc_k & c_k^2 \end{bmatrix}\right\}+\frac{htl_{ij}}{6}\begin{bmatrix}2&1&0\\1&2&0\\0&0&0\end{bmatrix}+\frac{htl_{jk}}{6}\begin{bmatrix}0&1&0\\0&2&1\\0&1&0\end{bmatrix}$$

$$++\frac{htl_{ki}}{6}\begin{bmatrix}2&1&0\\0&0&0\\0&1&2\end{bmatrix}$$

$$\{f\}=\frac{gAt}{3}\begin{Bmatrix}1\\1\\1\end{Bmatrix}-\frac{qtl_{ij}}{2}\begin{Bmatrix}1\\1\\0\end{Bmatrix}-\frac{qtl_{jk}}{2}\begin{Bmatrix}0\\1\\1\end{Bmatrix}-\frac{qtl_{ik}}{2}\begin{Bmatrix}1\\0\\1\end{Bmatrix}+\frac{hT_atl_{ij}}{2}\begin{Bmatrix}1\\1\\0\end{Bmatrix}+\frac{hT_atl_{jk}}{2}\begin{Bmatrix}0\\1\\1\end{Bmatrix}+\frac{hT_atl_{ik}}{2}\begin{Bmatrix}1\\0\\1\end{Bmatrix}$$

PROBLEM: NON-HOMOGENEOUS, STEADY STATE SOLUTION; CARTESIAN COORDINATES, FINITE ELEMENT SOLUTION

A square plate with temperatures of 100 C in three sides and 500 C in the fourth side is given. Plate sides are given as 1 m. Calculate the temperature profile



Program 10.1 triangular finite element

```
public class triangular_element
{
public double t; //Thickness of two dimensional element
public double a[],b[],c[];
public double x[],y[];//coordinates of data points
public double xc,yc; //center point
public double A; //Area
public double N[],B[][]; //Shape factor and derivative
public double Ti[];      //Temperature nodes
public int n_point[]; //point numbers
public int n_element; //element number
public double l[];
public boolean isconv[];
public boolean isheatflux[];
public boolean isheatgen;
public double h[];
public double Tinf[];
public double qflux[];
public double qgen;
public triangular_element(int n_elementi,int n_pointi[],double xi[],double yi[])
{x=new double[3];
y=new double[3];
Ti=new double[3];
l=new double[3];//0:lij,1:ljk,2:lik
isconv=new boolean[3];
isheatflux=new boolean[3];
n_point=new int[3];
xc=0;yc=0;
isheatgen=false;
h=new double[3];
Tinf=new double[3];
qflux=new double[3];
n_element=n_elementi;
```

```
for(int i=0;i<3;i++)
{n_point[i]=n_pointi[i];
x[i]=xi[i];y[i]=yi[i];
xc+=x[i];yc+=y[i];
isconv[i]=false;
isheatflux[i]=false;
h[i]=0.0;
Tinf[i]=0.0;
qflux[i]=0.0;
qgen=0.0;
}
xc/=3.0;
yc/=3.0;
a=new double[3];
b=new double[3];
c=new double[3];
N=new double[3];
B=new double[2][3];
a[0]=x[1]*y[2]-x[2]*y[1];
a[1]=x[2]*y[0]-x[0]*y[2];
a[2]=x[0]*y[1]-x[1]*y[0];
b[0]=y[1]-y[2];
b[1]=y[2]-y[0];
b[2]=y[0]-y[1];
c[0]=x[2]-x[1];
c[1]=x[0]-x[2];
c[2]=x[1]-x[0];
A=0.5*((x[0]*y[1]-x[1]*y[0])+(x[2]*y[0]-x[0]*y[2])+(x[1]*y[2]-x[2]*y[1]));
for(int i=0;i<3;i++)
{ B[0][i]=1.0/(2.0*A)*b[i];
B[1][i]=1.0/(2.0*A)*c[i];
}
l[0]=Math.sqrt(b[2]*b[2]+c[2]*c[2]);
l[1]=Math.sqrt(b[0]*b[0]+c[0]*c[0]);
l[2]=Math.sqrt(b[1]*b[1]+c[1]*c[1]);
}

public double[][] K(double ti,double k[])
{t=ti;
double a0=t/(4.0*A)*k[0];
double a1=t/(4.0*A)*k[1];
double b0=t*h[0]*l[0]/6.0;
double b1=t*h[1]*l[1]/6.0;
double b2=t*h[2]*l[2]/6.0;
double K1[][]={{(a0*b[0]*b[0]+a1*c[0]*c[0]+2*b0+2.0*b2),
(a0*b[0]*b[1]+a1*c[0]*c[1]+b0),
(a0*b[0]*b[2]+a1*c[0]*c[2]+b2)},
{(a0*b[1]*b[0]+a1*c[1]*c[0]+b0),
(a0*b[1]*b[1]+a1*c[1]*c[1]+2*b0+2*b1),
(a0*b[1]*b[2]+a1*c[1]*c[2]+b1)},
{(a0*b[2]*b[0]+a1*c[2]*c[0]+b2),
(a0*b[2]*b[1]+a1*c[2]*c[1]+b1),
(a0*b[2]*b[2]+a1*c[2]*c[2]+2*b1+2*b2)}
};
return K1;
}
public double[] f(double ti,double k[])
{
t=ti;
double a0=qgen*A*t/3.0;
double b0=-qflux[0]*t*l[0]/2.0;
double b1=-qflux[1]*t*l[1]/2.0;
double b2=-qflux[2]*t*l[2]/2.0;
double c0= h[0]* Tinf[0]*t*l[0]/2.0;
```

```
double c1= h[1]* Tinf[1]*t*l[1]/2.0;
double c2= h[2]* Tinf[2]*t*l[2]/2.0;
double f1[]={(a0+b0+b2+c0+c2),(a0+b0+b1+c0+c1),(a0+b1+b2+c1+c2)};
return f1;
}


public int setI(int cond,double L0,double L,double W0,double W)
{
int i=100;
if(cond==0) //x=0
{if(x[0]==L0 && x[1]==L0) i=0;
else if(x[1]==L0 && x[2]==L0) i=1;
else if(x[0]==L0 && x[2]==L0) i=2;
else i=-1;
}
else if(cond==1) //x=L
{if(x[0]==L && x[1]==L) i=0;
else if(x[1]==L && x[2]==L) i=1;
else if(x[0]==L && x[2]==L) i=2;
else i=-1;
}
else if(cond==2) //y=0
{if(y[0]==W0 && y[1]==W0) i=0;
else if(y[1]==W0 && y[2]==W0) i=1;
else if(y[0]==W0 && y[2]==W0) i=2;
else i=-1;
}
else if(cond==3) //y=W
{if(y[0]==W && y[1]==W) i=0;
else if(y[1]==W && y[2]==W) i=1;
else if(y[0]==W && y[2]==W) i=2;
else i=-1;
}
return i;
}

public void setConvection(int i, double hi, double tinfi)
{if(i>=0 && i<=2)
{
h[i]=hi;Tinf[i]=tinfi;isconv[i]=true;
}
}

public void setHeatFlux(int i, double qi)
{if(i>=0 && i<=2 ){qflux[i]=qi;isheatflux[i]=true;}     }

public void setHeatGeneration(double qgeni)
{qgen=qgeni;isheatgen=true;}
public void setN(double x,double y)
{for(int i=0;i<3;i++)
{N[i]=1.0/(2.0*A)*(a[i]+b[i]*x+c[i]*y);}
}
public double[]  q(double k)
{ double qi[]=new double[2];
for(int i=0;i<3;i++)
{qi[0]+=-k*B[0][i]*Ti[i];
qi[1]+=-k*B[1][i]*Ti[i];
}
return qi;
}
public double[]  q(double k,double Ti[])
{  //heat transfer
setT(Ti);
```

```
double qi[]=new double[2];
for(int i=0;i<3;i++)
{qi[0]+=-k*B[0][i]*Ti[i];
qi[1]+=-k*B[1][i]*Ti[i];
}
return qi;
}

public void setN(double z[])
{setN(z[0],z[1]);}
public void setT(double TT[])
{for(int i=0;i<3;i++)
{Ti[i]=TT[i];}
}
public double T(double x,double y)
{ //interpolation
double TT=0;
setN(x,y);
for(int i=0;i<3;i++)
{TT+=N[i]*Ti[i];}
return TT;
}

public double T(double T[],double x,double y)
{ //set grid point temperatures and get interpolated value
setT(T);
setN(x,y);
double TT=0;
for(int i=0;i<3;i++)
{TT+=N[i]*Ti[i];}
return TT;
}
public double T(double T[],double z[])
{ return T(T,z[0],z[1]);}
public static void main(String arg[])
{int n_elementi=0;
int n_pointi[]={0,1,2};
double xi[]={0.0,4,0};
double yi[]={0,0,2.5};
triangular_element fe=new triangular_element(n_elementi,n_pointi,xi,yi);
double p[]={2.0,1.0};
fe.setN(p);
double Ti[]={50,70,100};
fe.setT(Ti);
double T=fe.T(Ti,p[0],p[1]);
double q[]=fe.q(200.0);
}

}
```

Program 10.2 Triangular finite element grid and solution of steady state heat conduction equation: element_number.java

```
public class element_number
{int number;
int ijk;
public element_number(int numberi,int ijki)
{
number=numberi;
if(ijki<=2) ijk=ijki;
else {ijk=0;System.out.println("wrong value");}
}
public boolean equals(element_number en)
{if(en.number==number && en.ijk==ijk) return true;
else return false;
}
```

```
public int getNumber() {return number;}
public int getIjk() {return ijk;}
public int[] output()
{int x[]={number,ijk};
return x;
}
}
```

Program 10.3 Triangular finite element grid and solution of steady state heat conduction equation: setElement.java

```
import java.util.*;
import java.awt.*;
import javax.swing.*;
import java.awt.event.*;

public class setElement
{   ArrayList<element_number> en;
int no[];
int ij[];
public setElement()
{en=new ArrayList<element_number>();}
public void add(int n1,int ij1)
{element_number en=new element_number(n1,ij1);
add(en);
}
public void add(element_number s1)
{ //int n=en.size();
boolean b=false;
element_number s2;
Iterator n2=en.iterator();
while(n2.hasNext()&& !b)
{ s2=(element_number)n2.next();
b=s2.equals(s1);
}
if(!b) en.add(s1);
}

public int[][] output()
{
int n=en.size();
int a[][]=new int[n][2];
Iterator i=en.iterator();
int j=0;
while(i.hasNext()) {element_number el=(element_number)i.next();a[j]=el.output();j++;}
return a;
}
public static void main(String arg[])
{setElement se=new setElement();
element_number en1=new element_number(11,0);
element_number en2=new element_number(11,1);
element_number en3=new element_number(13,2);
element_number en4=new element_number(13,0);
se.add(en1);
se.add(en2);
se.add(en3);
se.add(en4);
int a[][]=se.output();
Text.print(a);
}
}
```

Program 10.4 Triangular finite element grid and solution of steady state heat conduction equation: triangular_grid.java

```
import javax.swing.*;
import org.math.plot.*;
import static java.lang.Math.*;
import static org.math.array.DoubleArray.*;
import visad.*;
import visad.java3d.DisplayImplJ3D;
import visad.java2d.DisplayImplJ2D;
import java.rmi.RemoteException;
import java.awt.*;
import javax.swing.*;
import java.awt.event.*;
public class triangular_grid
{
public triangular_element te[];
public setElement se[];
public int N,M,K,K1,K2,Nm1,Mm1;
public double x[][];
public double y[][];
public double T[][];
public double Ti[];
public double xi[];
public double yi[];
public boolean bi[];
public double dx,dy;
public int Bx0[],BxM[],By0[],ByN[];
public double L,W;
public double min(double xi[][])
{ double min=1e200;
for(int i=0;i<xi.length;i++)
{for(int j=0;j<xi[0].length;j++)
{if(xi[i][j]<min) min=xi[i][j];}
}
return min;
}
public double max(double xi[][])
{double max=-1e200;
for(int i=0;i<xi.length;i++)
{for(int j=0;j<xi[0].length;j++)
{if(xi[i][j]<max) max=xi[i][j];}
}
return max;
}
public triangular_grid(double xi[][],double yi[][])
{x=xi;
y=yi;
N=x.length-1;
M=x[0].length-1;
T=new double[N+1][M+1];
L=max(xi);
W=max(yi);
setElements();
}

public void generateTriangularGrid(double Li, double Wi,int Ni, int Mi)
{int i,j;
N=Ni;M=Mi;
L=Li;
W=Wi;
x=new double[N+1][M+1];
y=new double[N+1][M+1];
T=new double[N+1][M+1];
setElements();
}
```

```
public void setElements()
{ int i,j;
dx=L/M;
dy=W/N;
K=M*N;
K1=K*2;
K2=K1+1;
//grid points
xi=new double[K2];
yi=new double[K2];
Ti=new double[K2];
bi=new boolean[K2];
se=new setElement[K2];
for(i=0;i<K2;i++) {se[i]=new setElement();}
te=new triangular_element[K1];
int k1=-1,k=0,n=0,m=0;
for(n=0;n<=N;n++)
{ for(m=0;m<=M;m++)
{k=(N+1)*n+m;
x[n][m]=m*dx;y[n][m]=n*dy;
xi[k]=x[n][m];
yi[k]=y[n][m];
bi[k]=false;
Ti[k]=0.0;
}
}

for(n=0;n<N;n++)
{for(m=0;m<M;m++)
{ k=(N+1)*n+m;
int kp1=(N+1)*(n+1)+m;
int kp1p1= kp1+1;
double xi1[]={x[n][m],x[n][m+1],x[n+1][m]};
double yi1[]={y[n][m],y[n][m+1],y[n+1][m]};

k1++;
int n_point1[]={k,(k+1),kp1};
se[k].add(k1,0);
se[k+1].add(k1,1);
se[kp1].add(k1,2);
te[k1]=new triangular_element(k1,n_point1,xi1,yi1);
double xi2[]={x[n][m+1],x[n+1][m+1],x[n+1][m]};
double yi2[]={y[n][m+1],y[n+1][m+1],y[n+1][m]};
k1++;
int n_point2[]={k+1,kp1p1,kp1};
se[k+1].add(k1,0);
se[kp1p1].add(k1,1);
se[kp1].add(k1,2);
te[k1]=new triangular_element(k1,n_point2,xi2,yi2);
}
}
}
public triangular_grid(double Li, double Wi,int Ni, int Mi)
{generateTriangularGrid(Li,Wi,Ni,Mi);}
public int find_element(double z[])
{ return find_element(z[0],z[1]);}
public int find_element(double x,double y)
{double R1=Math.sqrt(x*x+y*y);
int kmin=0;
double min=1.0e50;
for(int k=0;k<K1;k++)
{   te[k].setN(x,y);
double r0=Math.sqrt((te[k].x[0]-x)*(te[k].x[0]-x)+(te[k].y[0]-y)*(te[k].y[0]-y));
double r1=Math.sqrt((te[k].x[1]-x)*(te[k].x[1]-x)+(te[k].y[1]-y)*(te[k].y[1]-y));
```

```
double r2=Math.sqrt((te[k].x[2]-x)*(te[k].x[2]-x)+(te[k].y[2]-y)*(te[k].y[2]-y));
double dr=Math.sqrt(r0*r0+r1*r1+r2*r2);
boolean b1=(x>=te[k].x[0]) && (x<=te[k].x[2]);
boolean b2=(y>=te[k].y[0]) && (y<=te[k].y[1]);
boolean b= b1 && b2;
if(dr<min) {min=dr;kmin=k;}
}
return kmin;
}
public void setConvection(int cond,double L0,double L,double W0,double W, double hi, double tinfi)
{//cond 0: x==L0
//cond 1: x==L
//cond 2: y==W0
//cond 3: y==W
int j=0;
for(int i=0;i<K1;i++)
{  j=te[i].setI(cond,L0,L,W0,W);
te[i].setConvection(j,hi,tinfi);
}
}
public void setHeatFlux(int cond,double L0,double L,double W0,double W, double qi)
{//cond 0: x==L0
//cond 1: x==L
//cond 2: y==W0
//cond 3: y==W
int j=-1;
for(int i=0;i<K1;i++)
{  j=te[i].setI(cond,L0,L,W0,W);
te[i].setHeatFlux(j,qi);
}
}
public void setTemperature(int cond,double L0,double L,double W0,double W, double TTi)
{//cond 0: x==L0
//cond 1: x==L
//cond 2: y==W0
//cond 3: y==W
for(int i=0;i<K2;i++)
{  if(cond==0)
{if(xi[i]==L0) {Ti[i]=TTi;bi[i]=true;}}
else if(cond==1)
{if(xi[i]==L) {Ti[i]=TTi;bi[i]=true;}}
else if(cond==2)
{if(yi[i]==W0) {Ti[i]=TTi;bi[i]=true;}}
else if(cond==3)
{if(yi[i]==W) {Ti[i]=TTi;bi[i]=true;}}
}
}
public void setHeatGeneration(int node[],double qgeni)
{  int n=node.length;
for(int i=0;i<n;i++)
{te[node[i]].setHeatGeneration(qgeni);}
}

public void solve(double ti,double k[])
{
double K[][]=new double[K2][K2];
double KK[][]=new double[K2][K2];
double ff[]=new double[K2];
double KN[][]=new double[3][3];
double f[]=new double[K2];
double fN[]=new double[3];
for(int i=0;i<K1;i++)
{KN=te[i].K(ti,k);
fN=te[i].f(ti,k);
```

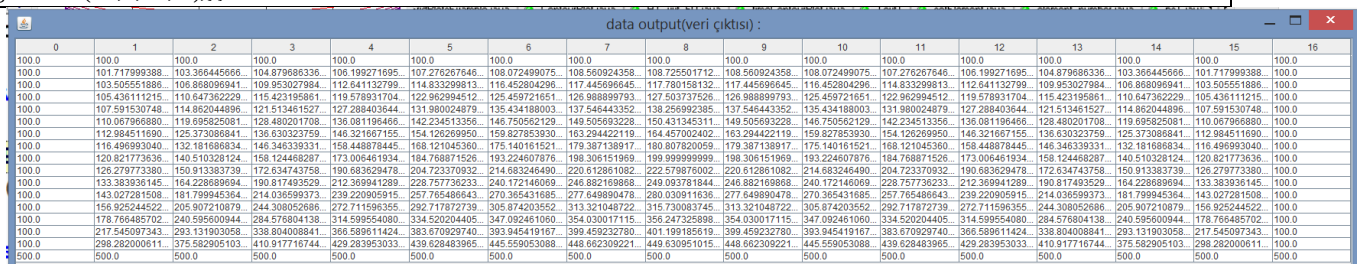
```
int n1=te[i].n_element;
int m[]=te[i].n_point;
for(int j=0;j<3;j++)
{ for(int k1=0;k1<3;k1++)
{K[m[j]][m[k1]]+=KN[j][k1];KK[m[j]][m[k1]]=K[m[j]][m[k1]];}
f[m[j]]+=fN[j];
ff[m[j]]=f[m[j]];
}
}
//redesign K matrix and f vector if any known
//temperature boundary is given
//int j=0;
for(int j=0;j<K2;j++)
{
for(int i=0;i<K2;i++)
{ if(bi[i])
{
if(i==j)
{f[i]=Ti[i];K[i][j]=1.0;}
else
{
K[i][j]=0.0;}
//System.out.println("i="+i+"j="+j+"bi="+bi[i]+"Ti="+Ti[i]+"fi="+f[i]);
}
}//end of for
} //end of for
//System.out.println("K =\n"+Matrix.toString(K));
//System.out.println("f =\n"+Matrix.toStringT(f));
double TT[]=AXB(K,f);
int k1=0;
for(int n=0;n<=N;n++)
{ for(int m=0;m<=M;m++)
{k1=(N+1)*n+m;
T[n][m]=TT[k1];
}
}
}

//LU decomposition method
public static double[][] LU(double c[][],int indx[],int d[])
{
//returns LU decomposition of matrix c and index indx
double a[][];
int n=c.length;
a=new double[n][n];
double vv[];
vv=new double[n];
double sum,dum,big,temp;
int i,j,k;
int imax;
int nmax=100;
double tiny=1.0e-40;
imax=0;
for(i=1;i<=n;i++)
{
for(j=1;j<=n;j++)
a[i-1][j-1]=c[i-1][j-1];
}
d[0]=1;
for(i=1;i<=n;i++)
{
big=0.0;
for(j=1;j<=n;j++)
{
```

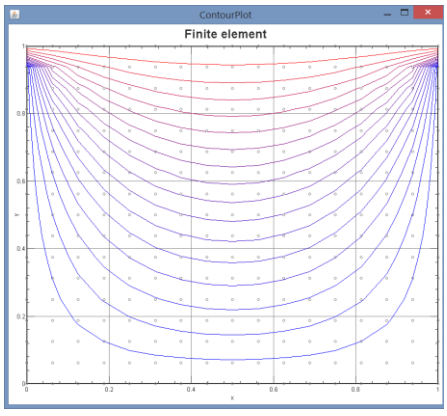
```
if(Math.abs(a[i-1][j-1])>big) big=Math.abs(a[i-1][j-1]);
}
if(big==0) {System.out.println("singular matrix");return a;}
vv[i-1]=1.0/big;
}
for(j=1;j<=n;j++)
{
for(i=1;i<j;i++)
{
sum=a[i-1][j-1];
for(k=1;k<i;k++)
{
sum-=a[i-1][k-1]*a[k-1][j-1];
}
a[i-1][j-1]=sum;
}
big=0;
for(i=j;i<=n;i++)
{
sum=a[i-1][j-1];
for(k=1;k<j;k++)
{
sum-=a[i-1][k-1]*a[k-1][j-1];
}
a[i-1][j-1]=sum;
dum=vv[i-1]*Math.abs(sum);
if(dum>=big)
{
imax=i;
big=dum;
}
} //end of i=0
if(j != imax)
{
for(k=1;k<=n;k++)
{
dum=a[imax-1][k-1];
a[imax-1][k-1]=a[j-1][k-1];
a[j-1][k-1]=dum;
}
d[0]=-d[0];
vv[imax-1]=vv[j-1];
} //end of if
indx[j-1]=imax;
if(a[j-1][j-1]==0) a[j-1][j-1]=tiny;
if(j!=n)
{
dum=1.0/a[j-1][j-1];
for(i=j+1;i<=n;i++)
a[i-1][j-1]*=dum;
}//endif
} //end for j=
return a;
}
public static double[] LUaxb(double a[][],double x[],int indx[])
{
//solves AX=B system of linear equation of LU decomposed matrix a
//(calculated by method LU)
int ii=0;
int i,j,ll=0;
double sum=0;
int n=a.length;
double b[];
b=new double[n];
```

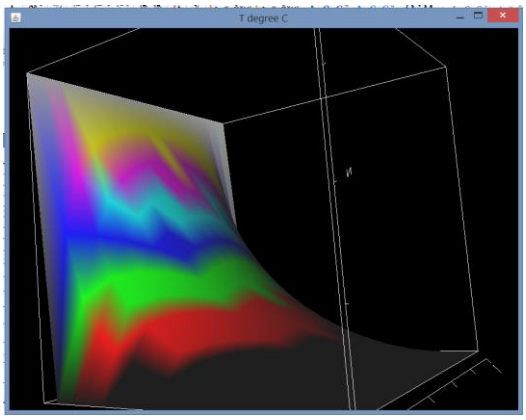
```
for(i=1;i<=n;i++)
{
b[i-1]=x[i-1];
}
for(i=1;i<=n;i++)
{
ll=indx[i-1];
sum=b[ll-1];
b[ll-1]=b[i-1];
if(ii!=0)
{
for(j=ii;j<=(i-1);j++)
{
sum-=a[i-1][j-1]*b[j-1];
}
}
else if(sum!=0) ii=i;
b[i-1]=sum;
}
for(i=n;i>=1;i--)
{
sum=b[i-1];
if(i<n)
{
for(j=(i+1);j<=n;j++)
{
sum-=a[i-1][j-1]*b[j-1];
}
}
b[i-1]=sum/a[i-1][i-1];
}
return b;
}
public void ContourPlot()
{ContourPlot pp=new ContourPlot( x,y,T,16,true,"Finite element","x","y");pp.plot();  }
public void Plot3D(double L0,double L,double W0,double W) throws RemoteException, VisADException
{
Plot3D.plot3D_CD1(T,L0,L,W0,W,200,"x","y","T degree C");
}
public void GridPlot(double L0,double L,double W0,double W)
{Plot3DPanel plot = new Plot3DPanel("SOUTH");
double dx=(L-L0)/M;
double dy=(W-W0)/N;
double[] xx = increment(L0, dx,L); // x = 0.0:0.1:1.0
double[] yy = increment(W0,dy,W);
plot.addGridPlot("Finite element Grid", xx, yy, T);
JFrame frame = new JFrame("a plot panel");
frame.setSize(600, 600);
frame.setContentPane(plot);
frame.setVisible(true);
}
public static double[] AXB(double a[][],double b[])
{
//Solution of system of linear equations by LU method
// note that the same calculation can be done by divide method.
int n=a.length;
double c[]=new double[n];
int d[]={1};
int indx[]=new int[n];
double e[][]=new double[n][n];
e=LU(a,indx,d);
c=LUaxb(e,b,indx);
return c;
} //end of AXB
```

```
public static void main(String arg[]) throws RemoteException, VisADException
{double L=1.0;
double W=1.0;
int N=16,M=16;
double hi=20.0;
triangular_grid tg=new triangular_grid(L,W,M,N);
int cond0=0;//x==0
int cond1=1;//x==L
int cond2=2;//y==0
int cond3=3;//y==W
double Ti0=100.0;
double Ti1=100.0;
double Ti2=100.0;
double Ti3=500.0;
tg.setTemperature(cond0,0.0,L,0.0,W,Ti0);
tg.setTemperature(cond1,0.0,L,0.0,W,Ti1);
tg.setTemperature(cond2,0.0,L,0.0,W,Ti2);
tg.setTemperature(cond3,0.0,L,0.0,W,Ti3);
double k[]={50,50};
tg.solve(1,k);
Text.print(tg.T);
tg.ContourPlot();
tg.Plot3D(0.0,L,0.0,W);
tg.GridPlot(0.0,L,0.0,W);}}
```



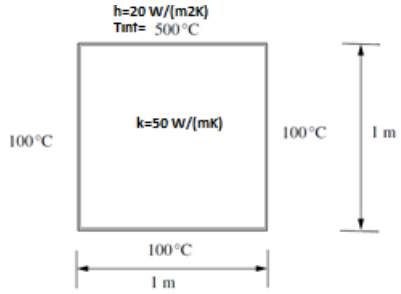
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
100.0	100.0	100.0	100.0	100.0	100.0	100.0	100.0	100.0	100.0	100.0	100.0	100.0	100.0	100.0	100.0	100.0
100.0	101.717999388...	103.366445666...	104.879686336...	106.199271695...	107.276267646...	108.072499075...	108.560924358...	108.725501712...	108.560924358...	108.072499075...	107.276267646...	106.199271695...	104.879686336...	103.366445666...	101.717999388...	100.0
100.0	103.505551886...	106.868096941...	109.953027984...	112.641132799...	114.833299813...	116.452804296...	117.445696645...	117.780158132...	117.445696645...	116.452804296...	114.833299813...	112.641132799...	109.953027984...	106.868096941...	103.505551886...	100.0
100.0	105.436111215...	110.647362229...	115.423195861...	119.578931704...	122.962994512...	125.459721651...	126.988899793...	127.503737526...	126.988899793...	125.459721651...	122.962994512...	119.578931704...	115.423195861...	110.647362229...	105.436111215...	100.0
100.0	107.591530748...	114.862044896...	121.513461527...	127.288403644...	131.980024879...	135.434188003...	137.546443352...	138.256992385...	137.546443352...	135.434188003...	131.980024879...	127.288403644...	121.513461527...	114.862044896...	107.591530748...	100.0
100.0	110.067966880...	119.695825081...	128.480201708...	136.081196466...	142.234513356...	146.750562129...	149.505693228...	150.431345311...	149.505693228...	146.750562129...	142.234513356...	136.081196466...	128.480201708...	119.695825081...	110.067966880...	100.0
100.0	112.984511690...	125.373086841...	136.630323759...	146.321667155...	154.126269950...	159.827853930...	163.294422119...	164.457002402...	163.294422119...	159.827853930...	154.126269950...	146.321667155...	136.630323759...	125.373086841...	112.984511690...	100.0
100.0	116.496993040...	132.181686834...	146.346339331...	158.448878445...	168.121045360...	175.140161521...	179.387138917...	180.807820059...	179.387138917...	175.140161521...	168.121045360...	158.448878445...	146.346339331...	132.181686834...	116.496993040...	100.0
100.0	120.821773636...	140.510328124...	158.124468287...	173.006461934...	184.768871526...	193.224607876...	198.306151969...	199.999999999...	198.306151969...	193.224607876...	184.768871526...	173.006461934...	158.124468287...	140.510328124...	120.821773636...	100.0
100.0	126.279773380...	150.913383739...	172.634743758...	190.683629478...	204.723370932...	214.683246490...	220.612861082...	222.579876002...	220.612861082...	214.683246490...	204.723370932...	190.683629478...	172.634743758...	150.913383739...	126.279773380...	100.0
100.0	133.383936145...	164.228689694...	190.817493529...	212.369941289...	228.757736233...	240.172146069...	246.882169868...	249.093781844...	246.882169868...	240.172146069...	228.757736233...	212.369941289...	190.817493529...	164.228689694...	133.383936145...	100.0
100.0	143.027281508...	181.799945364...	214.036599373...	239.220905915...	257.765486643...	270.365431685...	277.649890478...	280.030911636...	277.649890478...	270.365431685...	257.765486643...	239.220905915...	214.036599373...	181.799945364...	143.027281508...	100.0
100.0	156.925244522...	205.907210879...	244.308052686...	272.711596355...	292.717872739...	305.874203552...	313.321048722...	315.730083745...	313.321048722...	305.874203552...	292.717872739...	272.711596355...	244.308052686...	205.907210879...	156.925244522...	100.0
100.0	178.766485702...	240.595600944...	284.576804138...	314.599554080...	334.520204405...	347.092461060...	354.030017115...	356.247325898...	354.030017115...	347.092461060...	334.520204405...	314.599554080...	284.576804138...	240.595600944...	178.766485702...	100.0
100.0	217.545097343...	293.131903058...	338.804008841...	366.589611424...	383.670929740...	393.945419167...	399.459232780...	401.199185619...	399.459232780...	393.945419167...	383.670929740...	366.589611424...	338.804008841...	293.131903058...	217.545097343...	100.0
100.0	298.282000611...	375.582905103...	410.917716744...	429.283953033...	439.628483965...	445.559053088...	448.662309221...	449.630951015...	448.662309221...	445.559053088...	439.628483965...	429.283953033...	410.917716744...	375.582905103...	298.282000611...	100.0
500.0	500.0	500.0	500.0	500.0	500.0	500.0	500.0	500.0	500.0	500.0	500.0	500.0	500.0	500.0	500.0	500.0





EXAMPLE PROBLEM: NON-HOMOGENEOUS, TIME INDEPENDENT SOLUTION; CARTESİAN COORDINATES, FINITE ELEMENT SOLUTION

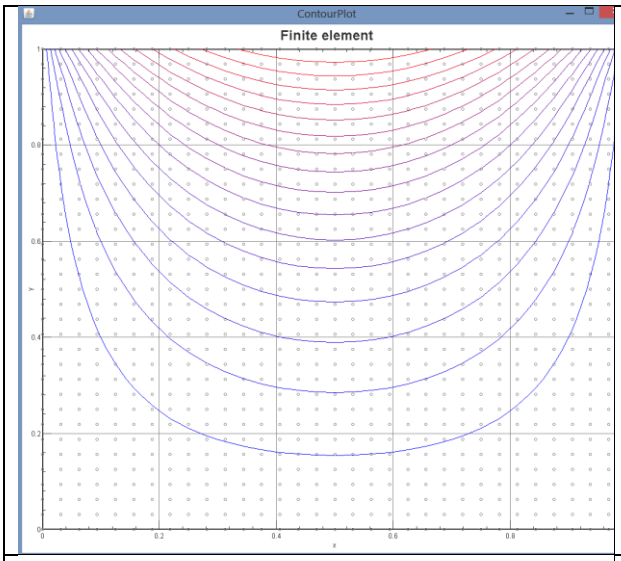
A second problem: In this one, three side temperatures specified as 100 C, and in the third side convective heat transfer is existed.

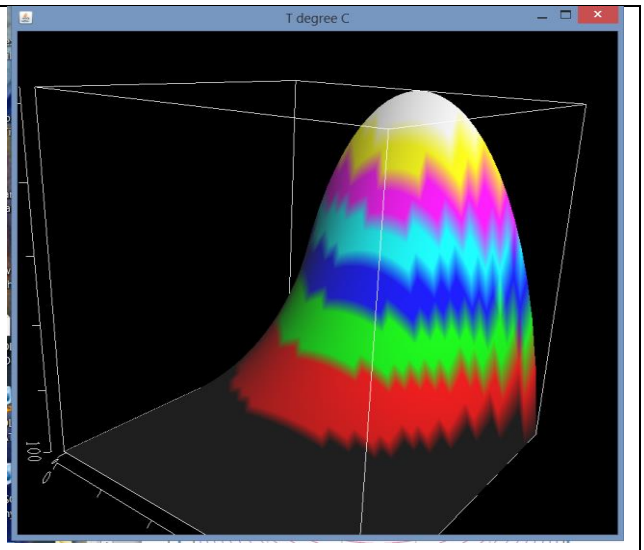


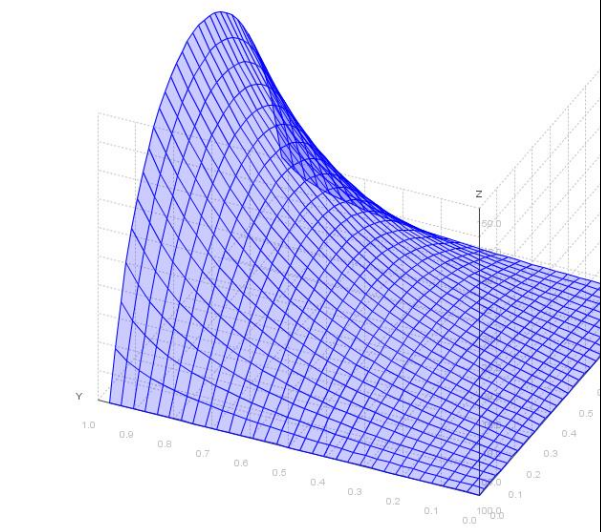

```
public static void main(String arg[]) throws RemoteException, VisADException
{double L=1.0;
double W=1.0;
int N=32,M=32;
double hi=20.0;
triangular_grid tg=new triangular_grid(L,W,M,N);
int cond0=0;//x==0
int cond1=1;//x==L
int cond2=2;//y==0
int cond3=3;//y==W
double Ti0=100.0;
double Ti1=100.0;
double Ti2=100.0;
double Ti3=500.0;
double qi=-5000.0;;
tg.setTemperature(cond0,0.0,L,0.0,W,Ti0);
tg.setTemperature(cond1,0.0,L,0.0,W,Ti1);
tg.setTemperature(cond2,0.0,L,0.0,W,Ti2);
tg.setConvection(cond3,0.0,L,0.0,W,hi,Ti3);
double k[]={50,50};
tg.solve(1,k);
Text.print(tg.T);
tg.ContourPlot();
tg.Plot3D(0.0,L,0.0,W);
tg.GridPlot(0.0,L,0.0,W);
tg.surfacePlot();
}
```

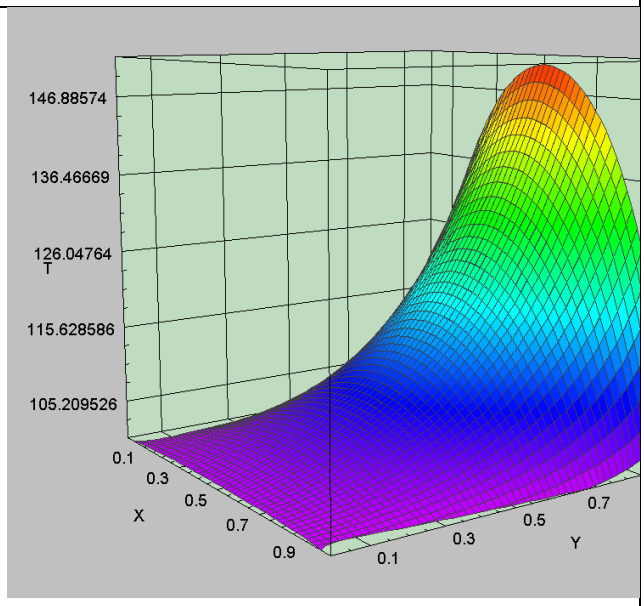
data output(veri çıktısı) :

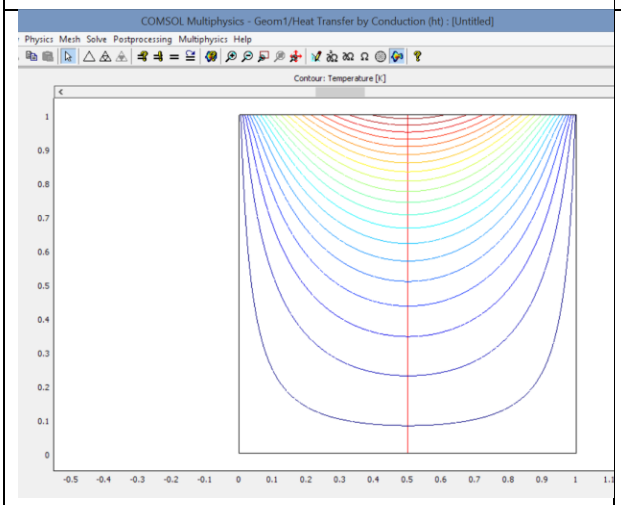
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
100.0	100.0	100.0	100.0	100.0	100.0	100.0	100.0	100.0	100.0	100.0	100.0	100.0	100.0	100.0	100.0	100.0	100.0	100.0	100.0	100.0	100.0	100.0	100.0	100.0	100.0	100.0	100.0	100.0	100.0	100.0	100.0
100.0	100.04...	100.09...	100.14...	100.18...	100.23...	100.27...	100.31...	100.34...	100.37...	100.40...	100.43...	100.45...	100.46...	100.47...	100.48...	100.48...	100.48...	100.47...	100.46...	100.45...	100.43...	100.40...	100.37...	100.34...	100.310...	100.27...	100.230...	100.18...	100.142...	100.09...	100.047...
100.0	100.09...	100.19...	100.28...	100.37...	100.46...	100.54...	100.62...	100.69...	100.75...	100.81...	100.86...	100.90...	100.93...	100.96...	100.97...	100.98...	100.97...	100.96...	100.93...	100.90...	100.86...	100.81...	100.75...	100.69...	100.623...	100.54...	100.463...	100.37...	100.285...	100.19...	100.096...
100.0	100.14...	100.29...	100.43...	100.56...	100.70...	100.82...	100.94...	101.05...	101.14...	101.23...	101.30...	101.37...	101.41...	101.45...	101.47...	101.48...	101.47...	101.45...	101.41...	101.37...	101.30...	101.23...	101.14...	101.05...	100.942...	100.82...	100.700...	100.56...	100.431...	100.29...	100.145...
100.0	100.19...	100.39...	100.58...	100.76...	100.94...	101.11...	101.27...	101.41...	101.54...	101.66...	101.76...	101.84...	101.91...	101.96...	101.98...	101.99...	101.98...	101.96...	101.91...	101.84...	101.76...	101.66...	101.54...	101.41...	101.270...	101.11...	100.944...	100.76...	100.582...	100.39...	100.196...
100.0	100.24...	100.49...	100.73...	100.97...	101.19...	101.41...	101.61...	101.79...	101.96...	102.10...	102.23...	102.34...	102.42...	102.48...	102.52...	102.53...	102.52...	102.48...	102.42...	102.34...	102.23...	102.10...	101.96...	101.79...	101.611...	101.41...	101.198...	100.97...	100.738...	100.49...	100.249...
100.0	100.30...	100.60...	100.90...	101.18...	101.46...	101.72...	101.96...	102.19...	102.39...	102.57...	102.73...	102.86...	102.96...	103.03...	103.07...	103.09...	103.07...	103.03...	102.96...	102.86...	102.73...	102.57...	102.39...	102.19...	101.967...	101.72...	101.463...	101.18...	100.901...	100.60...	100.304...
100.0	100.36...	100.72...	101.07...	101.41...	101.74...	102.05...	102.34...	102.61...	102.85...	103.06...	103.25...	103.40...	103.52...	103.61...	103.66...	103.68...	103.66...	103.61...	103.52...	103.40...	103.25...	103.06...	102.85...	102.61...	102.343...	102.05...	101.742...	101.41...	101.073...	100.72...	100.362...
100.0	100.42...	100.84...	101.25...	101.65...	102.03...	102.40...	102.74...	103.05...	103.33...	103.58...	103.80...	103.98...	104.12...	104.22...	104.28...	104.30...	104.28...	104.22...	104.12...	103.98...	103.80...	103.58...	103.33...	103.05...	102.741...	102.40...	102.039...	101.65...	101.256...	100.84...	100.424...
100.0	100.49...	100.97...	101.45...	101.91...	102.35...	102.77...	103.16...	103.52...	103.85...	104.14...	104.39...	104.59...	104.76...	104.87...	104.94...	104.97...	104.94...	104.87...	104.76...	104.59...	104.39...	104.14...	103.85...	103.52...	103.166...	102.77...	102.355...	101.91...	101.452...	100.97...	100.490...
100.0	100.56...	101.11...	101.66...	102.18...	102.69...	103.17...	103.62...	104.03...	104.40...	104.73...	105.02...	105.25...	105.44...	105.57...	105.65...	105.68...	105.65...	105.57...	105.44...	105.25...	105.02...	104.73...	104.40...	104.03...	103.623...	103.17...	102.695...	102.18...	101.661...	101.11...	100.561...
100.0	100.63...	101.26...	101.88...	102.48...	103.06...	103.60...	104.11...	104.58...	105.00...	105.37...	105.70...	105.96...	106.17...	106.32...	106.42...	106.45...	106.42...	106.32...	106.17...	105.96...	105.70...	105.37...	105.00...	104.58...	104.114...	103.60...	103.062...	102.48...	101.888...	101.26...	100.638...
100.0	100.72...	101.43...	102.13...	102.81...	103.46...	104.07...	104.64...	105.17...	105.65...	106.07...	106.43...	106.73...	106.97...	107.14...	107.24...	107.27...	107.24...	107.14...	106.97...	106.73...	106.43...	106.07...	105.65...	105.17...	104.647...	104.07...	103.460...	102.81...	102.134...	101.43...	100.721...
100.0	100.81...	101.61...	102.40...	103.16...	103.89...	104.58...	105.22...	105.81...	106.35...	106.82...	107.23...	107.56...	107.83...	108.02...	108.13...	108.17...	108.13...	108.02...	107.83...	107.56...	107.23...	106.82...	106.35...	105.81...	105.226...	104.58...	103.892...	103.16...	102.401...	101.61...	100.811...
100.0	100.91...	101.81...	102.69...	103.54...	104.36...	105.13...	105.85...	106.51...	107.11...	107.64...	108.09...	108.47...	108.76...	108.97...	109.10...	109.14...	109.10...	108.97...	108.76...	108.47...	108.09...	107.64...	107.11...	106.51...	105.856...	105.13...	104.364...	103.54...	102.693...	101.81...	100.910...
100.0	101.01...	102.02...	103.01...	103.96...	104.88...	105.74...	106.54...	107.28...	107.94...	108.53...	109.03...	109.45...	109.78...	110.01...	110.15...	110.20...	110.15...	110.01...	109.78...	109.45...	109.03...	108.53...	107.94...	107.28...	106.546...	105.74...	104.880...	103.96...	103.013...	102.02...	101.019...
100.0	101.13...	102.26...	103.36...	104.43...	105.44...	106.40...	107.30...	108.12...	108.85...	109.50...	110.06...	110.52...	110.89...	111.15...	111.30...	111.35...	111.30...	111.15...	110.89...	110.52...	110.06...	109.50...	108.85...	108.12...	107.301...	106.40...	105.448...	104.43...	103.365...	102.26...	101.138...
100.0	101.27...	102.52...	103.75...	104.94...	106.07...	107.13...	108.13...	109.03...	109.85...	110.57...	111.19...	111.70...	112.10...	112.38...	112.56...	112.61...	112.56...	112.38...	112.10...	111.70...	111.19...	110.57...	109.85...	109.03...	108.131...	107.13...	106.072...	104.94...	103.754...	102.52...	101.270...
100.0	101.41...	102.81...	104.18...	105.50...	106.76...	107.94...	109.04...	110.05...	110.95...	111.74...	112.42...	112.98...	113.42...	113.73...	113.92...	113.99...	113.92...	113.73...	113.42...	112.98...	112.42...	111.74...	110.95...	110.05...	109.045...	107.94...	106.762...	105.50...	104.184...	102.81...	101.416...
100.0	101.57...	103.13...	104.66...	106.13...	107.52...	108.83...	110.05...	111.16...	112.15...	113.02...	113.77...	114.38...	114.86...	115.21...	115.42...	115.49...	115.42...	115.21...	114.86...	114.38...	113.77...	113.02...	112.15...	111.16...	110.054...	108.83...	107.526...	106.13...	104.662...	103.13...	101.579...
100.0	101.76...	103.50...	105.19...	106.82...	108.37...	109.82...	111.16...	112.39...	113.48...	114.43...	115.25...	115.92...	116.45...	116.82...	117.05...	117.12...	117.05...	116.82...	116.45...	115.92...	115.25...	114.43...	113.48...	112.39...	111.169...	109.82...	108.375...	106.82...	105.195...	103.50...	101.761...
100.0	101.96...	103.90...	105.79...	107.60...	109.32...	110.92...	112.40...	113.74...	114.94...	115.99...	116.88...	117.61...	118.18...	118.59...	118.83...	118.91...	118.83...	118.59...	118.18...	117.61...	116.88...	115.99...	114.94...	113.74...	112.405...	110.92...	109.321...	107.60...	105.792...	103.90...	101.965...
100.0	102.19...	104.36...	106.46...	108.47...	110.37...	112.15...	113.77...	115.24...	116.55...	117.69...	118.66...	119.45...	120.07...	120.51...	120.78...	120.87...	120.78...	120.51...	120.07...	119.45...	118.66...	117.69...	116.55...	115.24...	113.777...	112.15...	110.379...	108.47...	106.464...	104.36...	102.196...
100.0	102.45...	104.88...	107.22...	109.46...	111.56...	113.52...	115.30...	116.91...	118.33...	119.57...	120.62...	121.48...	122.14...	122.62...	122.90...	123.00...	122.90...	122.62...	122.14...	121.48...	120.62...	119.57...	118.33...	116.91...	115.305...	113.52...	111.568...	109.46...	107.225...	104.88...	102.459...
100.0	102.76...	105.47...	108.09...	110.58...	112.90...	115.05...	117.01...	118.76...	120.30...	121.64...	122.77...	123.69...	124.40...	124.91...	125.22...	125.32...	125.22...	124.91...	124.40...	123.69...	122.77...	121.64...	120.30...	118.76...	117.010...	115.05...	112.909...	110.58...	108.093...	105.47...	102.761...
100.0	103.11...	106.16...	109.09...	111.85...	114.43...	116.78...	118.91...	120.81...	122.48...	123.92...	125.12...	126.11...	126.87...	127.41...	127.73...	127.84...	127.73...	127.41...	126.87...	126.11...	125.12...	123.92...	122.48...	120.81...	118.918...	116.78...	114.430...	111.85...	109.092...	106.16...	103.113...
100.0	103.52...	106.97...	110.25...	113.33...	116.16...	118.74...	121.05...	123.10...	124.89...	126.42...	127.71...	128.75...	129.55...	130.12...	130.47...	130.58...	130.47...	130.12...	129.55...	128.75...	127.71...	126.42...	124.89...	123.10...	121.056...	118.74...	116.167...	113.33...	110.255...	106.97...	103.528...
100.0	104.02...	107.93...	111.62...	115.04...	118.16...	120.96...	123.45...	125.65...	127.55...	129.17...	130.53...	131.63...	132.47...	133.07...	133.43...	133.55...	133.43...	133.07...	132.47...	131.63...	130.53...	129.17...	127.55...	125.65...	123.459...	120.96...	118.161...	115.04...	111.627...	107.93...	104.028...
100.0	104.65...	109.11...	113.27...	117.06...	120.46...	123.49...	126.16...	128.49...	130.49...	132.20...	133.61...	134.76...	135.64...	136.26...	136.63...	136.75...	136.63...	136.26...	135.64...	134.76...	133.61...	132.20...	130.49...	128.49...	126.163...	123.49...	120.467...	117.06...	113.273...	109.11...	104.651...
100.0	105.45...	110.60...	115.28...	119.46...	123.15...	126.38...	129.20...	131.64...	133.74...	135.51...	136.97...	138.15...	139.06...	139.69...	140.08...	140.20...	140.08...	139.69...	139.06...	138.15...	136.97...	135.51...	133.74...	131.64...	129.206...	126.38...	123.150...	119.46...	115.286...	110.60...	105.459...
100.0	106.58...	112.56...	117.80...	122.35...	126.28...	129.69...	132.62...	135.15...	137.30...	139.12...	140.62...	141.82...	142.74...	143.39...	143.78...	143.91...	143.78...	143.39...	142.74...	141.82...	140.62...	139.12...	137.30...	135.15...	132.629...	129.69...	126.287...	122.35...	117.804...	112.56...	106.580...
100.0	108.30...	115.25...	121.01...	125.85...	129.95...	133.46...	136.46...	139.03...	141.21...	143.05...	144.56...	145.77...	146.70...	147.35...	147.74...	147.87...	147.74...	147.35...	146.70...	145.77...	144.56...	143.05...	141.21...	139.03...	136.466...	133.46...	129.954...	125.85...	121.017...	115.25...	108.301...
100.0	111.36...	119.14...	125.15...	130.08...	134.21...	137.73...	140.73...	143.30...	145.47...	147.29...	148.80...	150.00...	150.92...	151.57...	151.96...	152.09...	151.96...	151.57...	150.92...	150.00...	148.80...	147.29...	145.47...	143.30...	140.737...	137.73...	134.218...	130.08...	125.159...	119.14...	111.368...







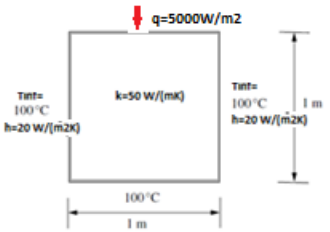




The same problem solved by comsol multiphysics finite element package:

EXAMPLE PROBLEM: NON-HOMOGENEOUS, TIME INDEPENDENT SOLUTION; CARTESIAN COORDINATES, FINITE ELEMENT SOLUTION

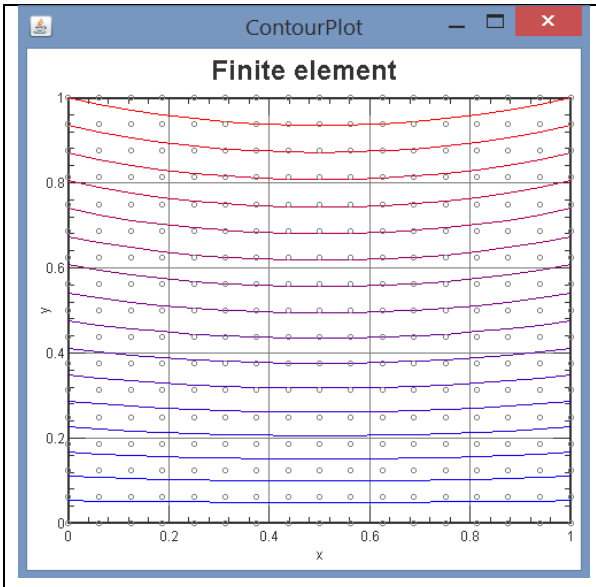
A third problem: In this one,one of the side temperatures specified as 100 C, and in second and third side convective heat transfer is existed. In the fourth side heat flux is given

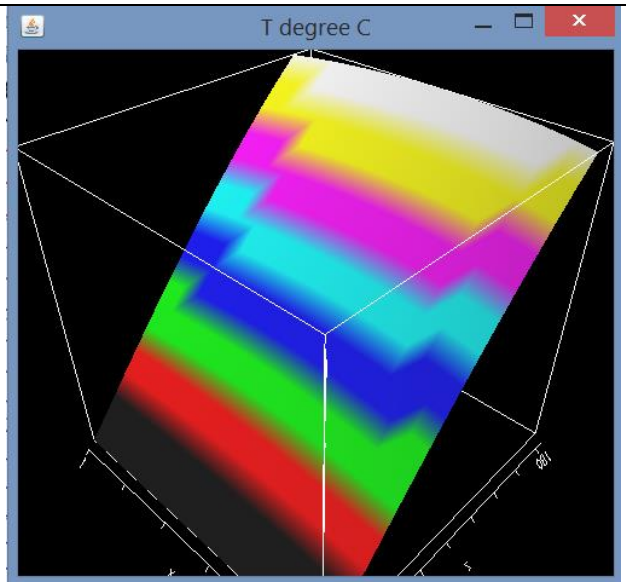



```
public static void main(String arg[]) throws RemoteException, VisADException
{double L=1.0;
double W=1.0;
int N=16,M=16;
double hi=20.0;
triangular_grid tg=new triangular_grid(L,W,M,N);
int cond0=0;//x==0
int cond1=1;//x==L
int cond2=2;//y==0
int cond3=3;//y==W
double Ti0=100.0; //degree C
double Ti1=100.0; //degree C
double Ti2=100.0; //degree C
double qi=-5000.0; //W/m2
tg.setTemperature(cond2,0.0,L,0.0,W,Ti0);
tg.setConvection(cond0,0.0,L,0.0,W,hi,Ti1);
tg.setConvection(cond1,0.0,L,0.0,W,hi,Ti2);
tg.setHeatFlux(cond3,0.0,L,0.0,W,qi);
double k[]={50,50};
tg.solve(1,k);
Text.print(tg.T);
tg.ContourPlot();
tg.Plot3D(0.0,L,0.0,W);
tg.GridPlot(0.0,L,0.0,W);
}
```

data output(veri çıktısı) :

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
100.0	100.0	100.0	100.0	100.0	100.0	100.0	100.0	100.0	100.0	100.0	100.0	100.0	100.0	100.0	100.0	100.0
104.1853...	104.2838...	104.3696...	104.4425...	104.5023...	104.5490...	104.5823...	104.6024...	104.6091...	104.6024...	104.5823...	104.5490...	104.5023...	104.4425...	104.3696...	104.2838...	104.1853...
108.3830...	108.5803...	108.7521...	108.8981...	109.0179...	109.1113...	109.1780...	109.2181...	109.2315...	109.2181...	109.1780...	109.1113...	109.0179...	108.8981...	108.7521...	108.5803...	108.3830...
112.6056...	112.9022...	113.1605...	113.3799...	113.5599...	113.7001...	113.8004...	113.8607...	113.8807...	113.8607...	113.8004...	113.7001...	113.5599...	113.3799...	113.1605...	112.9022...	112.6056...
116.8656...	117.2624...	117.6078...	117.9011...	118.1416...	118.3290...	118.4629...	118.5433...	118.5701...	118.5433...	118.4629...	118.3290...	118.1416...	117.9011...	117.6078...	117.2624...	116.8656...
121.1759...	121.6739...	122.1073...	122.4751...	122.7765...	123.0112...	123.1789...	123.2796...	123.3131...	123.2796...	123.1789...	123.0112...	122.7765...	122.4751...	122.1073...	121.6739...	121.1759...
125.5493...	126.1501...	126.6725...	127.1154...	127.4781...	127.7604...	127.9620...	128.0829...	128.1232...	128.0829...	127.9620...	127.7604...	127.4781...	127.1154...	126.6725...	126.1501...	125.5493...
129.9995...	130.7045...	131.3171...	131.8358...	132.2603...	132.5902...	132.8257...	132.9669...	133.0139...	132.9669...	132.8257...	132.5902...	132.2603...	131.8358...	131.3171...	130.7045...	129.9995...
134.5402...	135.3515...	136.0554...	136.6507...	137.1370...	137.5145...	137.7837...	137.9450...	137.9987...	137.9450...	137.7837...	137.5145...	137.1370...	136.6507...	136.0554...	135.3515...	134.5402...
139.1863...	140.1059...	140.9023...	141.5744...	142.1225...	142.5472...	142.8496...	143.0306...	143.0909...	143.0306...	142.8496...	142.5472...	142.1225...	141.5744...	140.9023...	140.1059...	139.1863...
143.9534...	144.9837...	145.8734...	146.6223...	147.2313...	147.7022...	148.0369...	148.2370...	148.3036...	148.2370...	148.0369...	147.7022...	147.2313...	146.6223...	145.8734...	144.9837...	143.9534...
148.8587...	150.0020...	150.9854...	151.8100...	152.4782...	152.9934...	153.3588...	153.5770...	153.6495...	153.5770...	153.3588...	152.9934...	152.4782...	151.8100...	150.9854...	150.0020...	148.8587...
153.9216...	155.1801...	156.2564...	157.1539...	157.8781...	158.4344...	158.8280...	159.0626...	159.1405...	159.0626...	158.8280...	158.4344...	157.8781...	157.1539...	156.2564...	155.1801...	153.9216...
159.1654...	160.5402...	161.7062...	162.6714...	163.4457...	164.0382...	164.4561...	164.7048...	164.7873...	164.7048...	164.4561...	164.0382...	163.4457...	162.6714...	161.7062...	160.5402...	159.1654...
164.6195...	166.1093...	167.3566...	168.3797...	169.1952...	169.8165...	170.2536...	170.5132...	170.5993...	170.5132...	170.2536...	169.8165...	169.1952...	168.3797...	167.3566...	166.1093...	164.6195...
170.3272...	171.9208...	173.2314...	174.2956...	175.1389...	175.7791...	176.2284...	176.4950...	176.5834...	176.4950...	176.2284...	175.7791...	175.1389...	174.2956...	173.2314...	171.9208...	170.3272...
176.3668...	178.0152...	179.3526...	180.4323...	181.2856...	181.9325...	182.3861...	182.6551...	182.7442...	182.6551...	182.3861...	181.9325...	181.2856...	180.4323...	179.3526...	178.0152...	176.3668...





Now general time dependent general heat conduction equation can be investigated as a finite element equation

$$\frac{\partial}{\partial x}\left(k_x\frac{\partial T}{\partial x}\right)+\frac{\partial}{\partial y}\left(k_y\frac{\partial T}{\partial y}\right)+g-\rho C_p\frac{\partial T}{\partial t}=0$$

Boundary conditions

$T=T_b$ on surface S_1

$k_x\frac{\partial T}{\partial x}\vec{l}+k_y\frac{\partial T}{\partial y}\vec{m}+q=0$ t>0 on surface S_2

$k_x\frac{\partial T}{\partial x}\vec{l}+k_y\frac{\partial T}{\partial y}\vec{m}+h(T-T_a)=0$t>0 on surface S_3

$T=T_0$ on x=0

Now that galarkin integration method will be applied to our differential equation . The method requires that the following expression will be satisfied

$\int_\Omega w_k L(T)\,d\Omega=0$where $\Omega$is the volume of the element.

$$\int_\Omega N_k\left[\frac{\partial}{\partial x}\left(k_x\frac{\partial T}{\partial x}\right)+\frac{\partial}{\partial y}\left(k_y\frac{\partial T}{\partial y}\right)+g-\rho C_p\frac{\partial T}{\partial t}\right]d\Omega=0$$

Note that for finite element T=[N]{T} $\frac{\partial T}{\partial t}=[N]\frac{d\{T\}}{dt}$and integral can be converted by partial integration rule

$\int udv=uv-\int vdu$ so $\int_\Omega \alpha_1\frac{\partial\alpha_2}{\partial x_1}d\Omega=\int_s \alpha_1\alpha_2 dS-\int_\Omega\frac{\partial\alpha_1}{\partial x_1}\alpha_2 d\Omega$

$\int_\Omega N_k\left[\frac{\partial}{\partial x}\left(k_x\frac{\partial T}{\partial x}\right)\right]d\Omega=\int_s N_k\left(k_x\frac{\partial T}{\partial x}\right)dS-\int_\Omega\frac{\partial N_k}{\partial x}k_x\frac{\partial N_m}{\partial x}T d\Omega$ By applying these equation becomes

$$-\int_\Omega\left[k_x\frac{\partial N_k}{\partial x}\frac{\partial N_m}{\partial x}+k_y\frac{\partial N_k}{\partial y}\frac{\partial N_m}{\partial y}\right]\{T_m\}d\Omega-\int_\Omega\rho C_p N_k N_m\left\{\frac{dT_m}{dt}\right\}d\Omega+\int_\Omega gN_k d\Omega$$
$$+\int_s N_k\left(k_x\frac{\partial T}{\partial x}+k_y\frac{\partial T}{\partial y}\right)dS=0$$

From boundary conditions:

$$\int_s N_k \left(k_x \frac{\partial T}{\partial x} + k_y \frac{\partial T}{\partial y}\right) dS = -\int_s N_k (q + h(T - T_a) dS$$

$$= -\int_s N_k q dS - \int_s N_k h N_m \{T_m\} dS + \int_s N_k h T_a dS$$

$$-\int_\Omega \left[k_x \frac{\partial N_k}{\partial x}\frac{\partial N_m}{\partial x} + k_y \frac{\partial N_k}{\partial y}\frac{\partial N_m}{\partial y}\right]\{T_m\} d\Omega - \int_\Omega \rho C_p N_k N_m \left\{\frac{dT_m}{dt}\right\} d\Omega - \int_s N_k h N_m \{T_m\} dS +$$

$+ \int_\Omega g N_k d\Omega - \int_s N_k q dS \int_s N_k h T_a dS = 0$

If this equation is written in matrix form :

$[C]\left\{\frac{dT}{dt}\right\} + [K]\{T\} = \{f\}$ where

$$[C] = \int_\Omega \rho C_p N_k N_m d\Omega$$

$$[K] = -\int_\Omega \left[k_x \frac{\partial N_k}{\partial x}\frac{\partial N_m}{\partial x} + k_y \frac{\partial N_k}{\partial y}\frac{\partial N_m}{\partial y}\right] d\Omega + \int_S h N_k (N_m) dS$$

$$\{f\} = \int_\Omega g N_k d\Omega - \int_s N_k q dS + \int_S h N_k T_a dS$$

$$[C] = \int_\Omega \rho C_p [N]^T [N] d\Omega$$

$[K] = -\int_\Omega [B]^T [D][B]\, d\Omega + \int_S h[N]^T [N] dS$ where

$$[D] = \begin{bmatrix} k_x & 0 \\ 0 & k_y \end{bmatrix}$$

$$\{f\} = \int_\Omega g[N]^T d\Omega - \int_s [N]^T q dS + \int_S h[N]^T T_a dS$$

$\int_l N_i^a N_j^b dl = \frac{a!b!}{(a+b+1)!} l \quad \int_l N_i^2 dl = \frac{2!0!}{(2+0+1)!} l = \frac{2l}{6} \quad \int_l N_i N_j dl = \frac{1!1!}{(1+1+1)!} l = \frac{l}{6}$

$\int_A N_i^a N_j^b N_k^c dA = \frac{a!b!c!}{(a+b+c+2)!} 2A \quad \int_A N_i^2 N_j^0 dA = \frac{2!0!}{(2+0+2)!} 2A = \frac{2A}{12} \quad \int_A N_i^1 N_j^1 dA =$

$\frac{1!1!}{(1+1+2)!} 2A = \frac{A}{12}$

The basic equation is still a differential equation in time, therefore should be solved by using finite difference method.

$[C]\left\{\frac{\{T\}^{p+1} - \{T\}^p}{\Delta t}\right\} + \varphi[K]\{T\}^p + (1-\varphi)[K]\{T\}^{p+1} = \{f\}$ where $t = p\Delta t$ and $\varphi$is implicit explicit sheme. If $\varphi = 1$

It is fully explicit scheme, If $\varphi = 0$, it is fully implicit scheme. $\varphi = 0.5$is called Crank-Nicolson method. Note that properties k,h,g, ρ, C_p and q can be changed in each step as a function of temperature, but for one step calculations can be considered constant.When equation is arranged for p+1 time step (unknown) temperatures, it becomes

$$([C] + (1-\varphi)\Delta t[K])\{T\}^{p+1} = ([C] - \varphi\Delta t[K])\{T\}^p + \Delta t\{f\}$$

Matrix terms can be opened for two dimensional triangular element as:

$$[C] = \frac{\rho C_p A t}{12}\begin{bmatrix} 4 & 1 & 1 \\ 1 & 4 & 1 \\ 1 & 1 & 4 \end{bmatrix}$$

$$[K] = \frac{t}{4A}\left\{k_x \begin{bmatrix} b_i^2 & b_i b_j & b_i b_k \\ b_i b_j & b_j^2 & b_j b_k \\ b_i b_k & b_j b_k & b_k^2 \end{bmatrix} + k_y \begin{bmatrix} c_i^2 & c_i c_j & c_i c_k \\ c_i c_j & c_j^2 & c_j c_k \\ c_i c_k & c_j c_k & c_k^2 \end{bmatrix}\right\} + \frac{h t l_{ij}}{6}\begin{bmatrix} 2 & 1 & 0 \\ 1 & 2 & 0 \\ 0 & 0 & 0 \end{bmatrix} + \frac{h t l_{jk}}{6}\begin{bmatrix} 0 & 1 & 0 \\ 0 & 2 & 1 \\ 0 & 1 & 0 \end{bmatrix}$$

$$++\frac{h t l_{ki}}{6}\begin{bmatrix} 2 & 1 & 0 \\ 0 & 0 & 0 \\ 0 & 1 & 2 \end{bmatrix}$$

$$\{f\} = \frac{gAt}{3}\begin{Bmatrix}1\\1\\1\end{Bmatrix} - \frac{qtl_{ij}}{2}\begin{Bmatrix}1\\1\\0\end{Bmatrix} - \frac{qtl_{jk}}{2}\begin{Bmatrix}0\\1\\1\end{Bmatrix} - \frac{qtl_{ik}}{2}\begin{Bmatrix}1\\0\\1\end{Bmatrix} + \frac{hT_a tl_{ij}}{2}\begin{Bmatrix}1\\1\\0\end{Bmatrix} + \frac{hT_a tl_{jk}}{2}\begin{Bmatrix}0\\1\\1\end{Bmatrix} + \frac{hT_a tl_{ik}}{2}\begin{Bmatrix}1\\0\\1\end{Bmatrix}$$

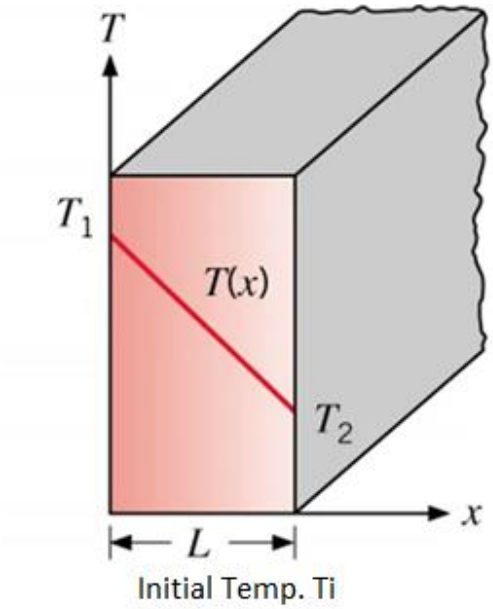
Initial temperature profile will be defined and loaded into the program, and then Matrix will be solved and temperature profiles for each time step will be calculated.

METHOD OF LINES SOLUTIONS OF PARTIAL DIFFERENTIAL EQUATIONS

A method combıne fınite difference and one dimensional numerical differential equation solutions

Method of lines, in general a method to combine finite difference method with one dimensional numerical differential eqaution solving methods such as Runge Kutta Method. In order to explain the methods let us consider two dimensional time dependent T(x,t) wall problem

Time dependent wall problem (conduction heat transfer)



Differential equation to be solved:

$$\frac{\partial^2 T(x,t)}{\partial x^2} = \frac{1}{\alpha}\frac{\partial T(x,t)}{\partial t}$$

Boundary conditions:

At x=0 t>0 $T(x,t) = d_0(t)$

At x=L t>0 $T(x,t) = d_1(t)$

At $0 \le x \le L$ $t = 0$ $T(x,t) = f(x)$

$$\alpha\frac{\partial^2 T(x,t)}{\partial x^2} = \frac{\partial T(x,t)}{\partial t}$$

$\frac{\partial^2 T(x,t)}{\partial x^2}$ term will be approximated by using finite difference

$$\frac{\partial^2 T(x,t)}{\partial x^2} \cong \frac{T(x_{n+1},t) - 2T(x_n,t) + T(x_{n-1},t)}{\Delta x^2} \quad n = 1..N-1$$

$$\alpha \frac{T(x_{n+1},t) - 2T(x_n,t) + T(x_{n-1},t)}{\Delta x^2} \cong \frac{\partial T(x_n,t)}{\partial t}$$

$$x_n = n\Delta x$$

Applying boundary condition yields:

$$n = 1 \quad \frac{T(x_{n+1},t) - 2T(x_n,t) + T(x_{n-1},t)}{\Delta x^2} = \frac{T(x_{n+1},t) - 2T(x_n,t) + d_0(t)}{\Delta x^2}$$

$$n = N-1 \quad \frac{T(x_{n+1},t) - 2T(x_n,t) + T(x_{n-1},t)}{\Delta x^2} = \frac{d_1(t) - 2T(x_n,t) + T(x_{n-1},t)}{\Delta x^2}$$

At $0 \le x \le L\ t = 0\ T(x_n,t) = f(x_n)\ n = 1..N-1$

This creates vector of ordinary differential equation. Set of ordinary differential equation can be solved by using a numerical method such as Runge-Kutta method.

PROBLEM 12.1 METHOD OF LINES, HOMOGENEOUS, TIME DEPENDENT SOLUTION; CARTESIAN COORDINATES, TWO DIMENSIONS

$$\alpha \frac{T(x_{n+1},t) - 2T(x_n,t) + T(x_{n-1},t)}{\Delta x^2} \cong \frac{\partial T(x_n,t)}{\partial t}$$

$$x_n = n\Delta x$$

Applying boundary condition yields:

$$n = 1 \quad \alpha \frac{T(x_{n+1},t) - 2T(x_n,t) + d_0(t)}{\Delta x^2} \cong \frac{\partial T(x_1,t)}{\partial t}$$

$$n = N-1 \quad \alpha \frac{d_1(t) - 2T(x_n,t) + T(x_{n-1},t)}{\Delta x^2} \cong \frac{\partial T(x_{n-1},t)}{\partial t}$$

At $0 \le x \le L\ t = 0\ T(x_n,t) = f(x_n)\ n = 1..N-1$

L=1 m
at x=0 t>0 T_1=20℃
at x=L t>0 T_2=100℃
initial temperature (t=0) T_2=50℃
COMPUTER CODES

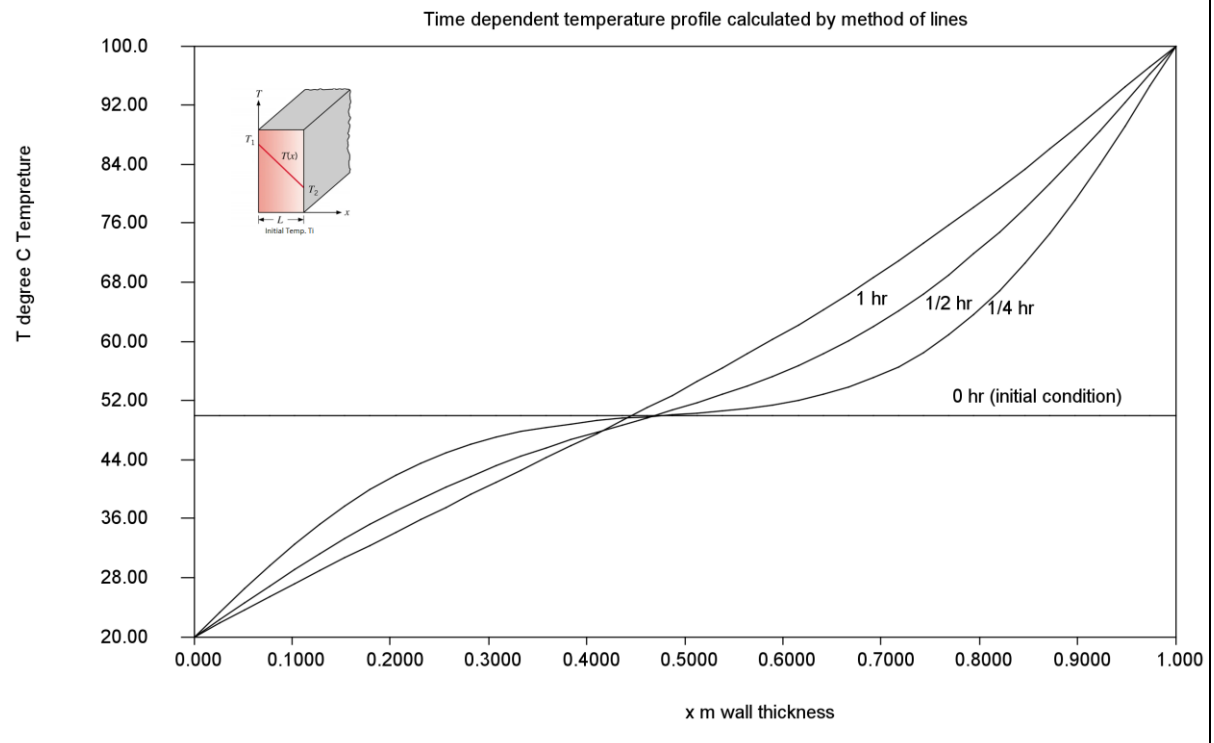
```
//=======================================================
// Numerical Analysis package in java
// Partial differential equation solution
// Method of lines combined with
// Runge-Kutta-Verner 13 constant step method with
// multible differential equations
// Dr. Turhan Coban
// =======================================================
// dy1/dx=f(x,y1,y2,..,yn)
// dy2/dx=f(x,y1,y2,..,yn)
```

```
// .....
// dyn/dx=f(x,y1,y2,..,yn)
// Method of lines
import java.io.*;

class fm2 extends f_xi
{ double L;
  int N;
  double alpha;
  double dx;
  double C;
  public fm2(double Li,int Ni,double alphai)
  {L=Li;N=Ni;alpha=alphai;dx=L/(N-1);C=alpha/(dx*dx);  }
//multivariable function
//lower limit x axis as a function of temperature
public double d0(double t)
{return 20.0;}
//upper limit x axis as a function of temperature
public double d1(double t)
{return 100.0;}
 //diferential equation set derived from finite difference formula
 double func(double T[],int n)
 {  double y=0;
   if(n==0)
   {y=C*(-2.0*T[1]+T[2])+C*d0(T[0]);}
   else if(n==(N-3) )
   {y=C*(-2.0*T[N-2]+T[N-3])+C*d1(T[0]);}
   else
    {y=C*(T[n]-2.0*T[n+1]+T[n+2]);}
    return y;
 }
}

class method_of_lines
{
public static void main(String args[]) throws IOException
  {
  double alpha=2e-5;
  double L=1.0;
 int N=40;
  double dx=L/(N-1);
  System.out.println("dx="+dx+"L="+L);
  fm2 b1=new fm2(L,N,alpha);
  double y[];
 double T[]=new double[N-2];
 double xx[]=new double[N];
 // initial (t=0) temperature gradient
  if_x f=x->40.0;
 //initial conditions
 // x axis
  for(int i=0;i<N;i++)
  {double x1=dx*i;
    xx[i]=x1;
  }
  //initial temperature profile
 for(int i=0;i<N-2;i++)
  {double x1=dx*(i+1);
    T[i]=f.func(x1);
  }
  //solution of partial differential equation
  Buthcher bb=new Buthcher("RKV13");
  Runge_Kutta rk=new Runge_Kutta(bb);
  double a[][]=rk.hesapla(b1,0.0,3600,T,1.0);
```

```
 double b[][]=Text.T(a);
 Text.printT(a[0]);
  int n=b.length;
 int m=b[0].length;
 System.out.println("n="+n+"m="+m);
 double c[][]=new double[n][m+1];
 for(int i=0;i<m+1;i++)
 {c[0][i]=f.func(xx[i]);}
 for(int j=1;j<n;j++)
 { c[j][0]=b1.d0(a[0][j]);
    c[j][m]=b1.d1(a[0][j]);
 }
 for(int i=1;i<m;i++)
 {for(int j=1;j<n;j++)
   {c[j][i]=b[j][i];}
 }
 double d[][]=Text.T(c);
 Text.print(c,"c");
 Text.print(c[0],"c0");
 Text.print(xx,"xx");
 Plot p=new Plot(xx,c[0]);
 p.addData(xx,c[900]);
 p.addData(xx,c[1800]);
 p.addData(xx,c[3600]);
 p.setPlabel("Heat Transfer wall T(x,t) method_of_lines solution");
 p.setXlabel("wall thickness x m");
  p.setYlabel("Temperature T degree C");
 p.plot();
}
}
```



PROBLEM 12.2 METHOD OF LINES, HOMOGENEOUS, TIME DEPENDENT SOLUTION; CARTESIAN COORDINATES, TWO DIMENSIONS

$$\alpha \frac{T(x_{n+1},t) - 2T(x_n,t) + T(x_{n-1},t)}{\Delta x^2} \cong \frac{\partial T(x_n,t)}{\partial t}$$

$$x_n = n\Delta x$$

Applying boundary condition yields:

$$n = 1 \quad \alpha \frac{T(x_{n+1},t) - 2T(x_n,t) + d_0(t)}{\Delta x^2} \cong \frac{\partial T(x_1,t)}{\partial t}$$

$$n = N-1 \quad \alpha \frac{d_1(t) - 2T(x_n,t) + T(x_{n-1},t)}{\Delta x^2} \cong \frac{\partial T(x_{n-1},t)}{\partial t}$$

At $0 \le x \le L\ t = 0\ T(x_n,t) = f(x_n)\ n = 1..N-1$

Numerical Example 2
L=1 m
at x=0 t>0 $T_1(t)$=20℃ +20t/3600
at x=L t>0 $T_2(t)$=100℃ - 20t/3600
initial temperature (t=0) $T_2^{(x)}$=40℃ + $50sin(\frac{\pi}{2}x)$

```
//=======================================================
// Numerical Analysis package in java
// Partial differential equation solution
// Method of lines combined with
// Runge-Kutta-Verner 13 constant step method with
// multible differential equations
// Dr. Turhan Coban
// =======================================================
// dy1/dx=f(x,y1,y2,..,yn)
// dy2/dx=f(x,y1,y2,..,yn)
// ......
// dyn/dx=f(x,y1,y2,..,yn)
// Method of lines
import java.io.*;

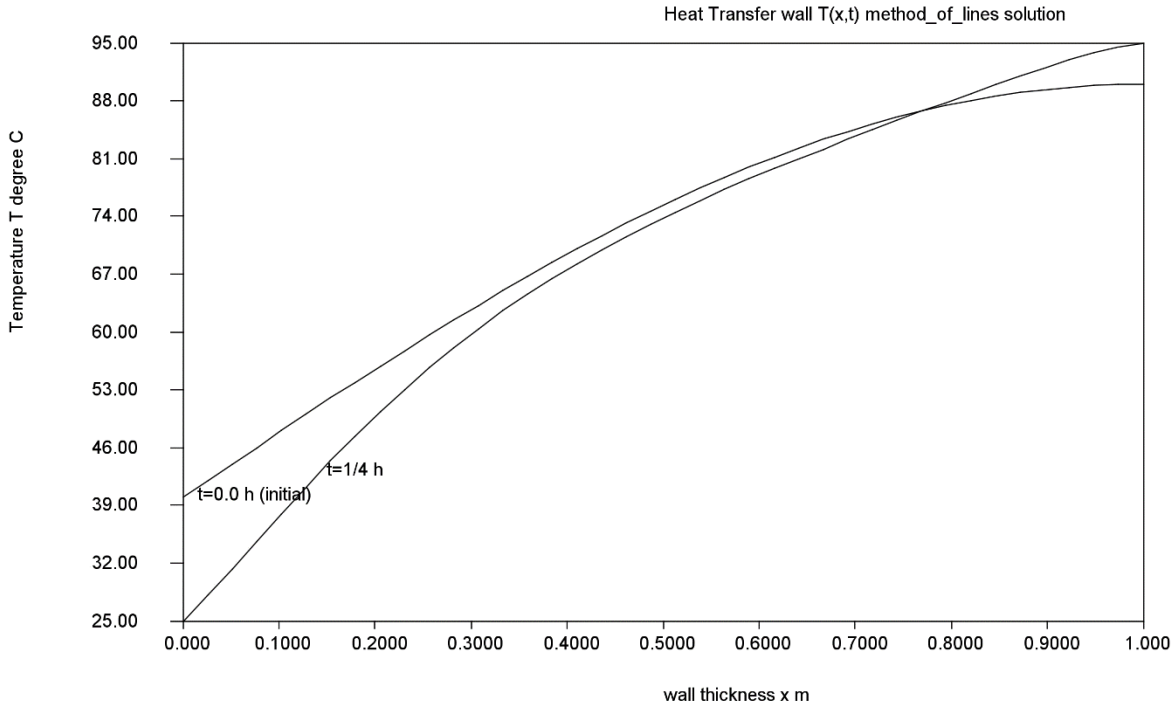
class fm2 extends f_xi
{ double L;
  int N;
  double alpha;
  double dx;
  double C;
  public fm2(double Li,int Ni,double alphai)
  {L=Li;N=Ni;alpha=alphai;dx=L/(N-1);C=alpha/(dx*dx);  }
//multivariable function
//lower limit x axis as a function of temperature
public double d0(double t)
{return 20.0+20.0*t/3600.0;}
//upper limit x axis as a function of temperature
public double d1(double t)
{return 100.0-20.0*t/3600.0;}
 //diferential equation set derived from finite difference formula
 double func(double T[],int n)
 {  double y=0;
   if(n==0)
   {y=C*(-2.0*T[1]+T[2])+C*d0(T[0]);}
```

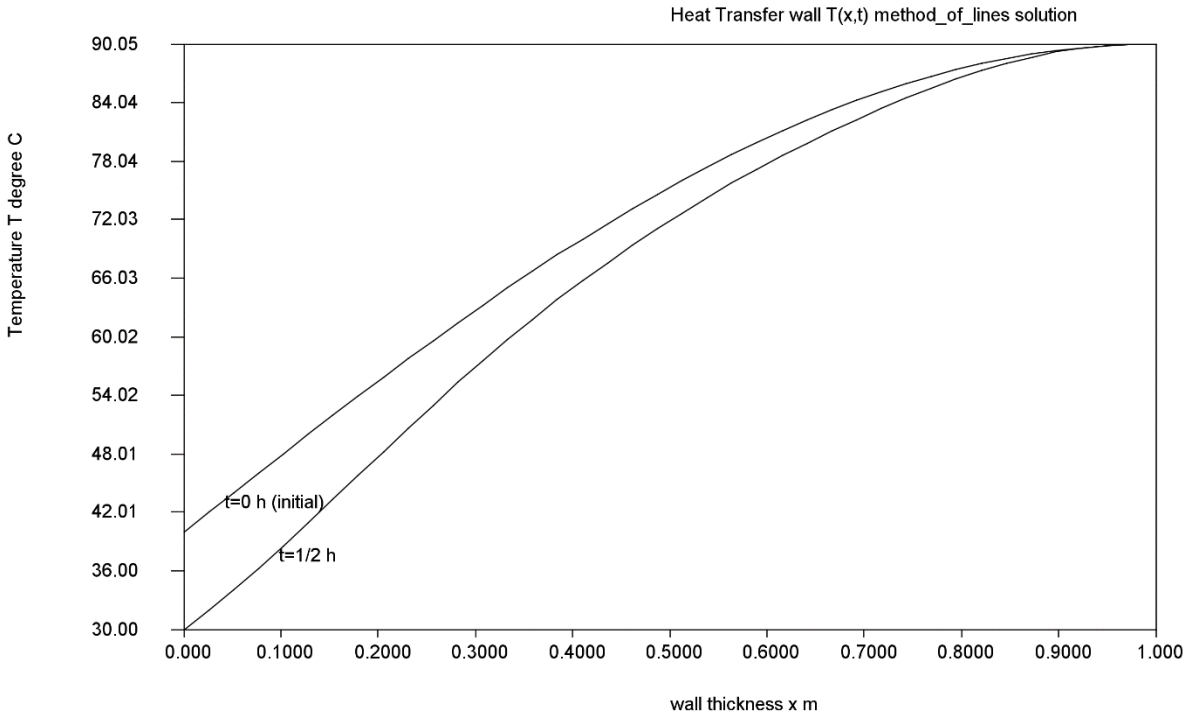
```
   else if(n==(N-3) )
    {y=C*(-2.0*T[N-2]+T[N-3])+C*d1(T[0]);}
    else
     {y=C*(T[n]-2.0*T[n+1]+T[n+2]);}
     return y;
  }
 }

 class method_of_lines
 {
 public static void main(String args[]) throws IOException
   {
   double alpha=2e-5;
   double L=1.0;
  int N=40;
   double dx=L/(N-1);
   System.out.println("dx="+dx+"L="+L);
   fm2 b1=new fm2(L,N,alpha);
   double y[];
  double T[]=new double[N-2];
  double xx[]=new double[N];
  // initial (t=0) temperature gradient
   if_x f=x->40.0+50*Math.sin(Math.PI*x/2.0);
  //initial conditions
  // x axis
   for(int i=0;i<N;i++)
   {double x1=dx*i;
     xx[i]=x1;
   }
   //initial temperature profile
  for(int i=0;i<N-2;i++)
   {double x1=dx*(i+1);
     T[i]=f.func(x1);
   }
   //solution of partial differential equation
   Buthcher bb=new Buthcher("RKV13");
   Runge_Kutta rk=new Runge_Kutta(bb);
   double a[][]=rk.hesapla(b1,0.0,3600,T,1.0);

   double b[][]=Text.T(a);
   Text.printT(a[0]);
    int n=b.length;
   int m=b[0].length;
   System.out.println("n="+n+"m="+m);
   double c[][]=new double[n][m+1];
   for(int i=0;i<m+1;i++)
   {c[0][i]=f.func(xx[i]);}
   for(int j=1;j<n;j++)
   { c[j][0]=b1.d0(a[0][j]);
     c[j][m]=b1.d1(a[0][j]);
   }
   for(int i=1;i<m;i++)
   {for(int j=1;j<n;j++)
     {c[j][i]=b[j][i];}
   }
   double d[][]=Text.T(c);
   Text.print(c,"c");
   Text.print(c[0],"c0");
   Text.print(xx,"xx");
   Plot p=new Plot(xx,c[0]);
   //p.addData(xx,c[900]);
   //p.addData(xx,c[1800]);
   p.addData(xx,c[3600]);
   p.setPlabel("Heat Transfer wall T(x,t) method_of_lines solution");
```

```
  p.setXlabel("wall thickness x m");
   p.setYlabel("Temperature T degree C");
  p.plot();
 }
}
```





13. APPROXIMATE METHODS AS SOLUTIONS OF PARTIAL DIFFERENTIAL EQUATIONS

Approximate methods used an approximation equation such as a polynomial to approximate partial differential equations. In example case problems, we will consider heat conduction problems. As a first example case let us consider semi-infinite wall with constant surface temperature. Let us solve the problem first as an exact solution by using Laplace transform method.

PROBLEM 13.1 SEMI-INFINITE WALL WITH CONSTANT TEMPERATURE APPROXIMATION METHODS

$$\frac{\partial^2 T(x,t)}{\partial x^2} = \frac{1}{\alpha}\frac{\partial T(x,t)}{\partial t}$$

with boundary condition:
$T(x,t)= T_0$ x=0 t > 0
$T(x,t)=T_i$ $x \to \infty$ t > 0
$T(x,t)= T_i$ at t=0
Defining $\theta(x,t) = T(x,t) - T_i$

$$\frac{\partial^2 \theta(x,t)}{\partial x^2} = \frac{1}{\alpha}\frac{\partial \theta(x,t)}{\partial t}$$

with boundary condition:
$\theta(x,t) = \theta_0 = T_0 - T_i$ x=0 t > 0
$\theta(x,t) = 0$ $x \to \infty$ t > 0
$\theta(x,t) = 0$ at t=0
Laplace transform of the equation is:
Laplace transform of the differential equation:

$$\frac{\partial^2 \theta(x,s)}{\partial x^2} - \frac{s}{\alpha}\theta(x,s) = 0$$

Boundary conditions:
$\theta(x,s) = T_0 - T_i$ x=0
$\theta(x,s) = 0$ $x \to \infty$
Solution of this ordinary differential equation is:

$$\theta(x,s) = C_1 e^{-x\sqrt{\frac{s}{\alpha}}} + C_2 e^{x\sqrt{\frac{s}{\alpha}}}$$

Boundary condition $\theta(x,s) = 0$ $x \to \infty$ $\rightarrow$ $C_2 = 0$
Boundary condition $\theta(x,s) = \frac{T_0 - T_i}{s}$ x=0 $\rightarrow$ $C_1 = \frac{T_0 - T_i}{s}$
So the solution is:

$$\theta(x,s) = \frac{T_0 - T_i}{s} e^{-x\sqrt{\frac{s}{\alpha}}}$$

Inverse laplace transfrom of $G(x,s) = \frac{e^{-x\sqrt{\frac{s}{\alpha}}}}{s}$ is $G(x,t) = erfc\left(\frac{x}{2\sqrt{\alpha t}}\right)$

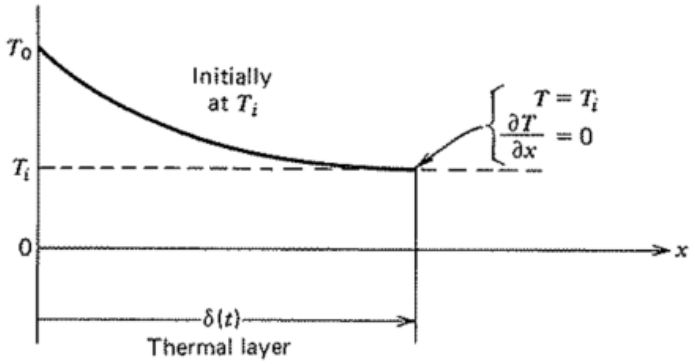
In the table it is given as case:

For $G(x,s) = \frac{e^{-k\sqrt{s}}}{s}$ $G(x,t) = erfc\left(\frac{k}{2\sqrt{t}}\right)$ and our case $k = \frac{x}{\sqrt{\alpha}}$

$$\theta(x,t) = T(x,t) - T_i = (T_0 - T_i)erfc\left(\frac{x}{2\sqrt{\alpha t}}\right) = (T_0 - T_i)\left[1 - erf\left(\frac{x}{2\sqrt{\alpha t}}\right)\right]$$

$$\frac{T(x,t) - T_0}{T_i - T_0} = erf\left(\frac{x}{2\sqrt{\alpha t}}\right)$$

Now that we have the exact solution let us look at the approximate solution method. It will be assumed that at time t, effect of surface temperature will be penetrate some thermal layer with penetration depth $\delta(t)$. For the remaining profile, temperature will remain constant at initial temperature, T_i.



Considering this condition differential equation can be integrate for the thermal layer only for integration depth $\delta(t)$. Therefore differential equation can be integrate for this integration depth.

$$\left.\frac{\partial T(x,t)}{\partial x}\right|_{x=\delta(t)} - \left.\frac{\partial T(x,t)}{\partial x}\right|_{x=0} = \frac{1}{\alpha}\int_{x=0}^{x=\delta(t)} \frac{\partial T(x,t)}{\partial t}\,dx$$

and considering

$$\frac{\partial}{\partial t}\left[\int_{x=0}^{x=\delta(t)} T(x,t)\,dx\right] = \int_{x=0}^{x=\delta(t)} \frac{\partial T(x,t)}{\partial t}\,dx + \left[T\frac{\partial x}{\partial t}\right]_{x=\delta(t)} - \left[T\frac{\partial x}{\partial t}\right]_{x=0}$$

$$\int_{x=0}^{x=\delta(t)} \frac{\partial T(x,t)}{\partial t}\,dx = \frac{\partial}{\partial t}\left[\int_{x=0}^{x=\delta(t)} T(x,t)\,dx\right] - T_i\frac{\partial \delta(t)}{\partial t}$$

Defining

$$\theta(x,t) = \int_{x=0}^{x=\delta(t)} T(x,t)\,dx$$

differential equation becomes:

$$-\alpha\left.\frac{\partial T(x,t)}{\partial t}\right|_{x=0} = \frac{\partial}{\partial t}[\theta(x,t) - T_i\delta(t)]$$

In order to solve this differential equation, an approximate temperature profile T(x,t) will be assumed. As a first case, let us take a cubic polynomial

$$T(x,t) = a + bx + cx^2 + dx^3$$

Coefficients can be found by applying boundary conditions:

$T(x,t)|_{x=0} = T_0$

$T(x,t)|_{x=\delta(t)} = T_i$

$\left.\frac{\partial T(x,t)}{\partial x}\right|_{x=\delta(t)} = 0$

$\left.\frac{\partial^2 T(x,t)}{\partial x^2}\right|_{x=\delta(t)} = 0$

$\left.\frac{\partial^2 T(x,t)}{\partial x^2}\right|_{x=0} = 0$

$$\frac{\partial T(x,t)}{\partial x} = b + 2cx + 3dx^2$$

$$\frac{\partial^2 T(x,t)}{\partial x^2} = 2c + 6dx$$

$\left.\frac{\partial^2 T(x,t)}{\partial x^2}\right|_{x=0} = 0 \quad \rightarrow \quad 2c = 0 \quad c = 0$

$T(x,t)|_{x=0} = T_0 \quad \rightarrow \quad a = T_0$

$\left.\frac{\partial T(x,t)}{\partial x}\right|_{x=\delta(t)} = 0 \quad \rightarrow \quad b + 3d\delta^2 = 0 \quad b = -3d\delta^2$

$T(x,t)|_{x=\delta(t)} = T_i \quad \rightarrow \quad T_i = T_0 + b\delta + d\delta^2 = T_0 + (-3d\delta^2)\delta + d\delta^2$

$d = \frac{T_0 - T_i}{2\delta^3} \qquad b = -3\frac{T_0 - T_i}{2\delta^3}\delta^2 = -\frac{3}{2}\frac{T_0 - T_i}{\delta}$

$$T(x,t) - T_i = (T_0 - T_i) - \frac{3}{2}(T_0 - T_i)\left(\frac{x}{\delta}\right) + \frac{1}{2}(T_0 - T_i)\left(\frac{x}{\delta}\right)^3$$

$$\frac{T(x,t) - T_i}{T_0 - T_i} = 1 - \frac{3}{2}\left(\frac{x}{\delta}\right) + \frac{1}{2}\left(\frac{x}{\delta}\right)^3$$

This equation can now be substituted into the differential equation

$$\frac{\partial T(x,t)}{\partial x} = \frac{\partial}{\partial x}\left[(T_0 - T_i) - \frac{3}{2}(T_0 - T_i)\left(\frac{x}{\delta}\right) + \frac{1}{2}(T_0 - T_i)\left(\frac{x}{\delta}\right)^3\right]$$

$$\frac{\partial T(x,t)}{\partial x} = -\frac{3}{2\delta}(T_0 - T_i) + \frac{3}{2}(T_0 - T_i)\frac{x^2}{\delta^3}$$

$$\left.\frac{\partial T(x,t)}{\partial x}\right|_{x=0} = -\frac{3}{2\delta}(T_0 - T_i)$$

$$\theta(x,t) = \int_{x=0}^{x=\delta(t)} T(x,t)\,dx = \int_{x=0}^{x=\delta(t)} \left[(T_0 - T_i) - \frac{3}{2}(T_0 - T_i)\left(\frac{x}{\delta}\right) + \frac{1}{2}(T_0 - T_i)\left(\frac{x}{\delta}\right)^3 + T_i\right] dx$$

$$\theta(x,t) = (T_0 - T_i)\left[x - \frac{3}{4}\frac{x^2}{\delta} + \frac{1}{8}\frac{x^4}{\delta^3}\right]_{x=0}^{x=\delta(t)} + T_i\delta$$

$$\theta(x,t) = (T_0 - T_i)\left[\delta - \frac{3}{4}\delta + \frac{1}{8}\delta\right] + T_i\delta = (T_0 - T_i)\left[\frac{3}{8}\delta\right] + T_i\delta$$

Substituting the results into the differential equation gives

$$-\alpha\left.\frac{\partial T(x,t)}{\partial t}\right|_{x=0} = \frac{\partial}{\partial t}[\theta(x,t) - T_i\delta(t)]$$

$$-\alpha\left[-\frac{3}{2\delta}(T_0 - T_i)\right] = \frac{\partial}{\partial t}\left[(T_0 - T_i)\left[\frac{3}{8}\delta\right] + T_i\delta - T_i\delta\right]$$

$$\left[\frac{3\alpha}{2\delta}\right] = \frac{\partial}{\partial t}\left[\frac{3}{8}\delta\right]$$

$$4\alpha\partial t = \delta\partial(\delta)$$

$$\int_{t=0}^{t} 4\alpha\partial t' = \int_{\delta'=0}^{\delta} \delta'\partial(\delta')$$

$$4\alpha t = \frac{\delta^2}{2}$$

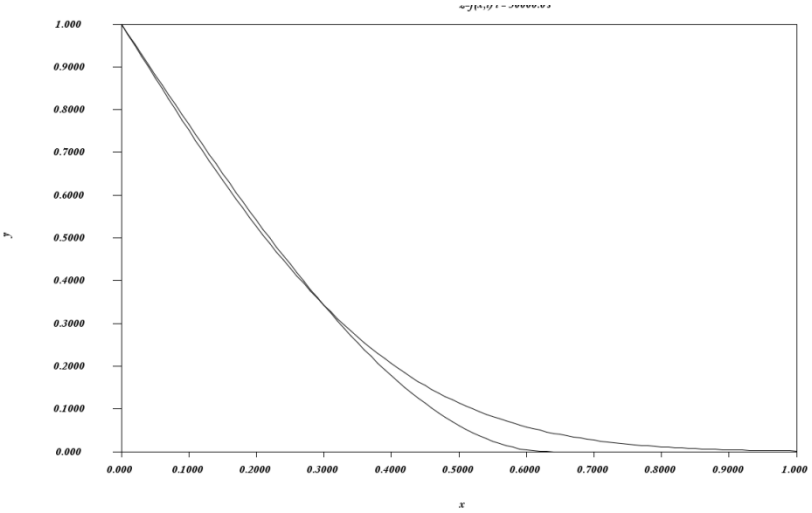
$$\delta(t) = \sqrt{8\alpha t}$$

By using this result, together with approximation temperature profile will give us approximation solution. The results can be compare by using a graphic program

Problem 13.1 java solution

```
 class i_wall extends fi_xt
{
public double[] func(double x,double t)
{ double a[]=new double[2];
                double alpha=1.0e-6;
  double delta=Math.sqrt(8.0*alpha*t);
  double eta=x/2/Math.sqrt(alpha*t);
  double xx=x/delta;

  a[0]=erfc(eta);
  if(xx<1) a[1]=1.0-1.5*xx+0.5*xx*xx*xx;
  else a[1]=0.0;
  return a;
}               }
public class timeplot_ntest1
{
public static void main(String args[])
 {
 i_wall ff=new i_wall();
  timeplot_n p=new timeplot_n(ff,0.0,1.0,0.01,0.0,50000.0,0.1,100.0);
  p.start();
 }}
```



Python solution

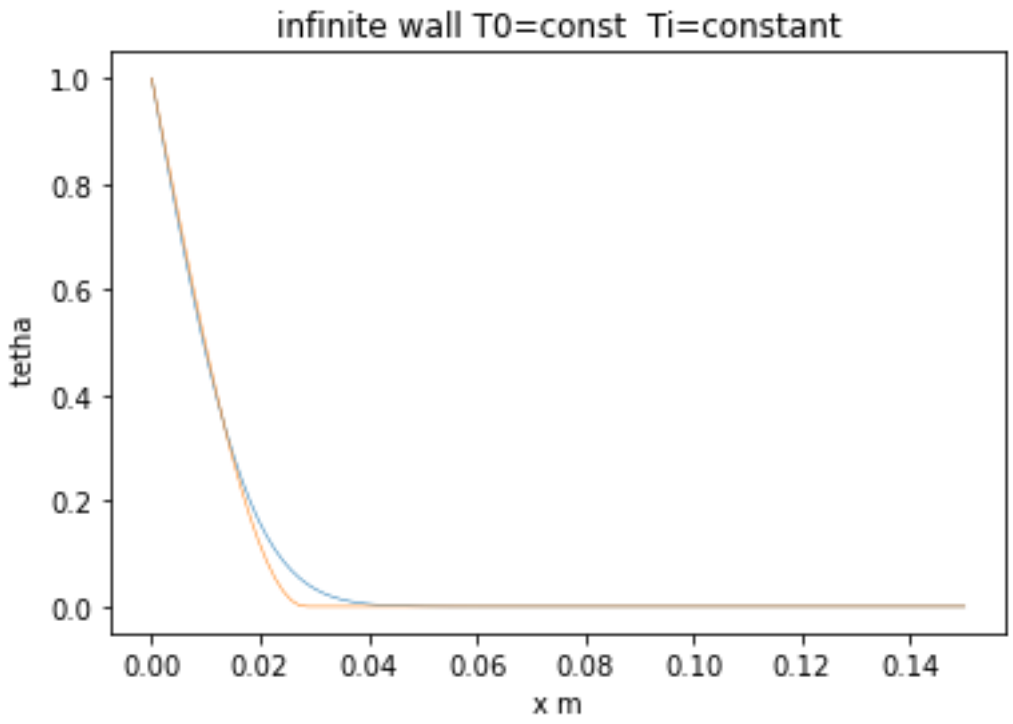
```
from math import *
from f_xj import *
import numpy as np;
import matplotlib.pyplot as plt;
#exact solution
class f1(f_xj):
    def func(self,z):
        x=z[0]
        t=z[1]
        eta=x/(2.0*sqrt(alpha*t))
        return erfc(eta)

#approximate solution
class f2(f_xj):
    def func(self,z):
        x=z[0]
        t=z[1]
        delta=sqrt(8.0*alpha*t)
        xx=x/delta
        if xx<1:
```

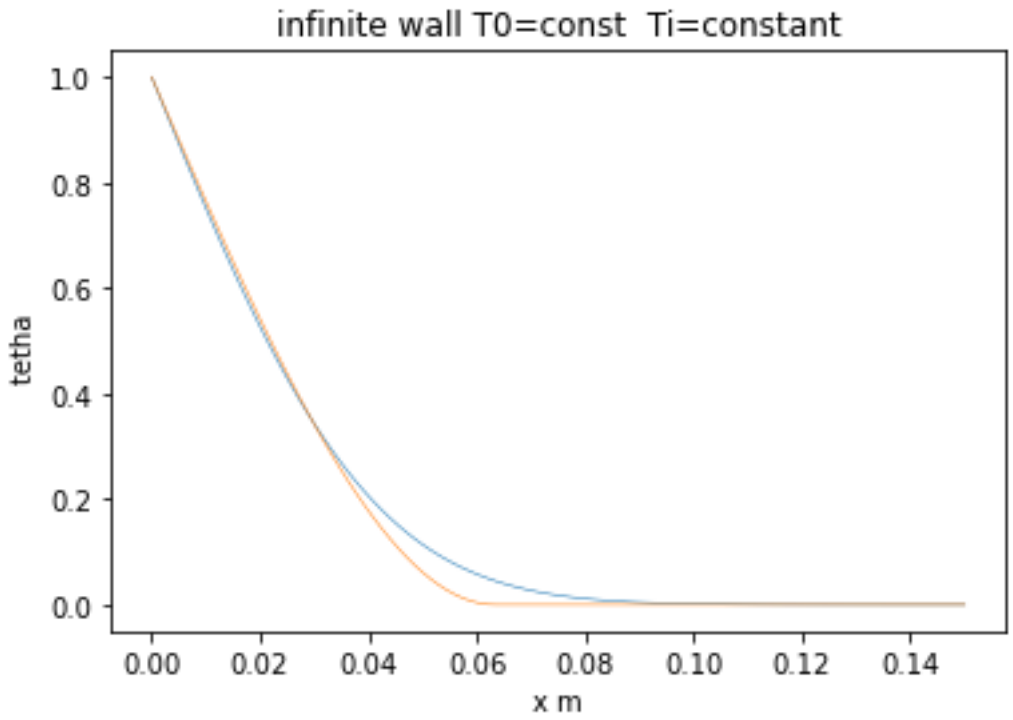
```
            T2=1.0-3.0/2.0*xx+0.5*xx*xx*xx
        else:
            T2=0
        return T2

np1=100
x=[0 for x in range(np1)]
dx=0.15/(len(x)-1)
t=[100]
np2=len(t)
T1= [[0 for x in range(np1)] for y in range(np2)]
T2= [[0 for x in range(np1)] for y in range(np2)]
T0=105.0
Ti=10.0
alpha=1e-6
s=""
f=f1()
ff=f2()
for i in range(np2):
    for j in range(np1):
        x[j]=dx*j
        z=[x[j],t[i]]
        T1[i][j]=f.func(z)
        T2[i][j]=ff.func(z)
plt.title('infinite wall T0=const  Ti=constant')
plt.xlabel('x m')
plt.ylabel('tetha ')
for i in range(np2):
    plt.plot(x,T1[i],linewidth=0.5)
    plt.plot(x,T2[i],linewidth=0.5)
```

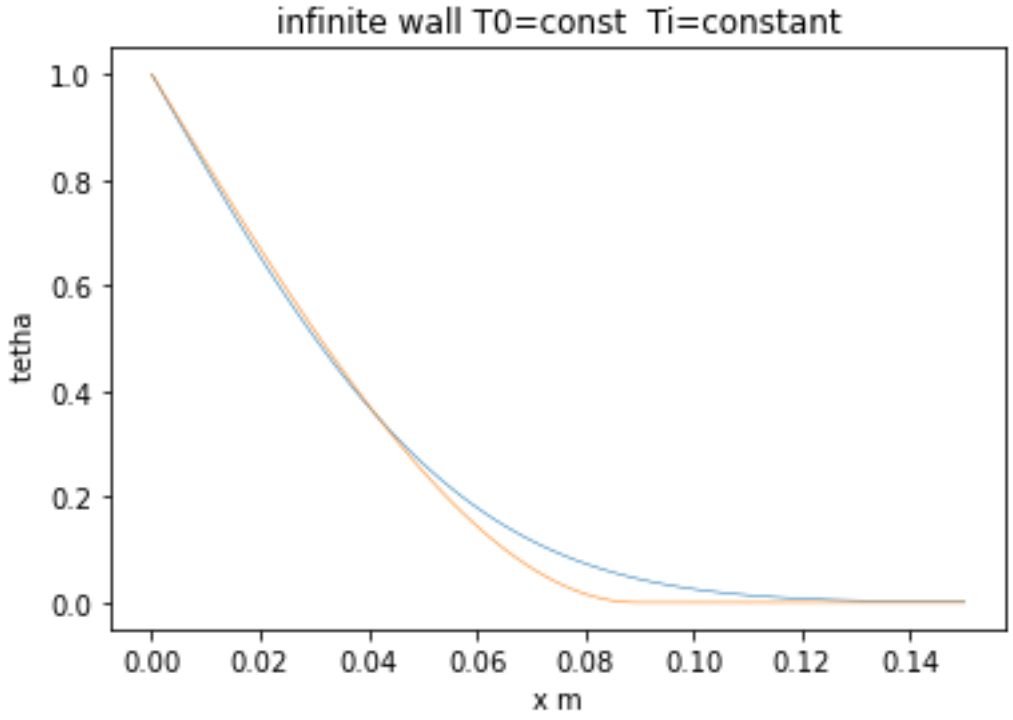
t=100 s



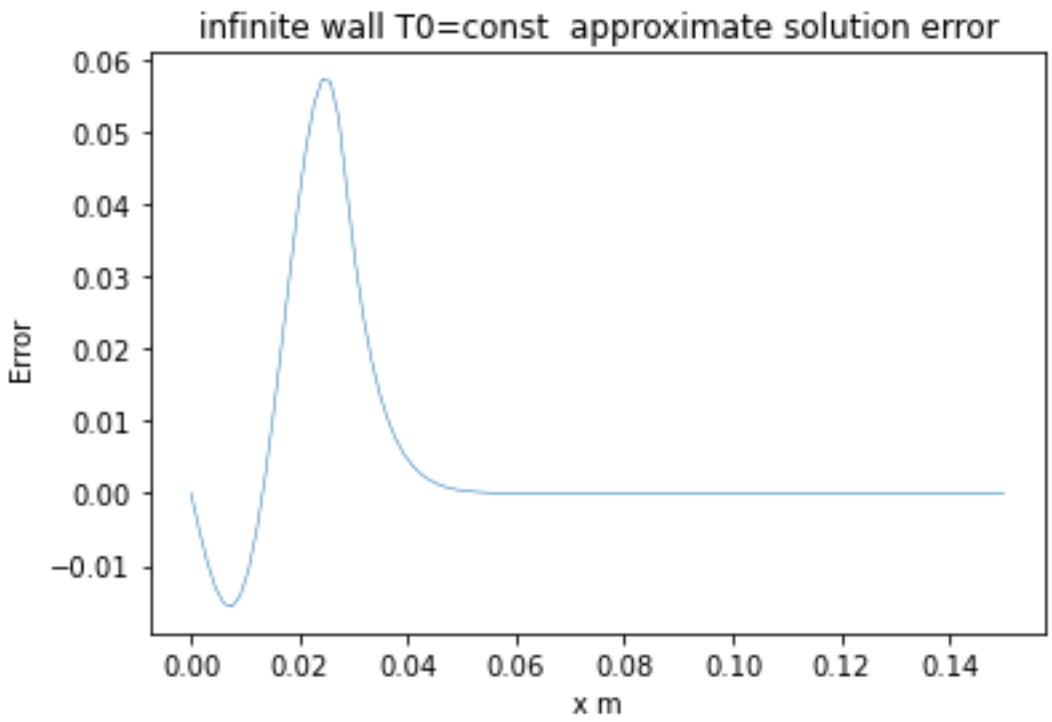
t=500 s



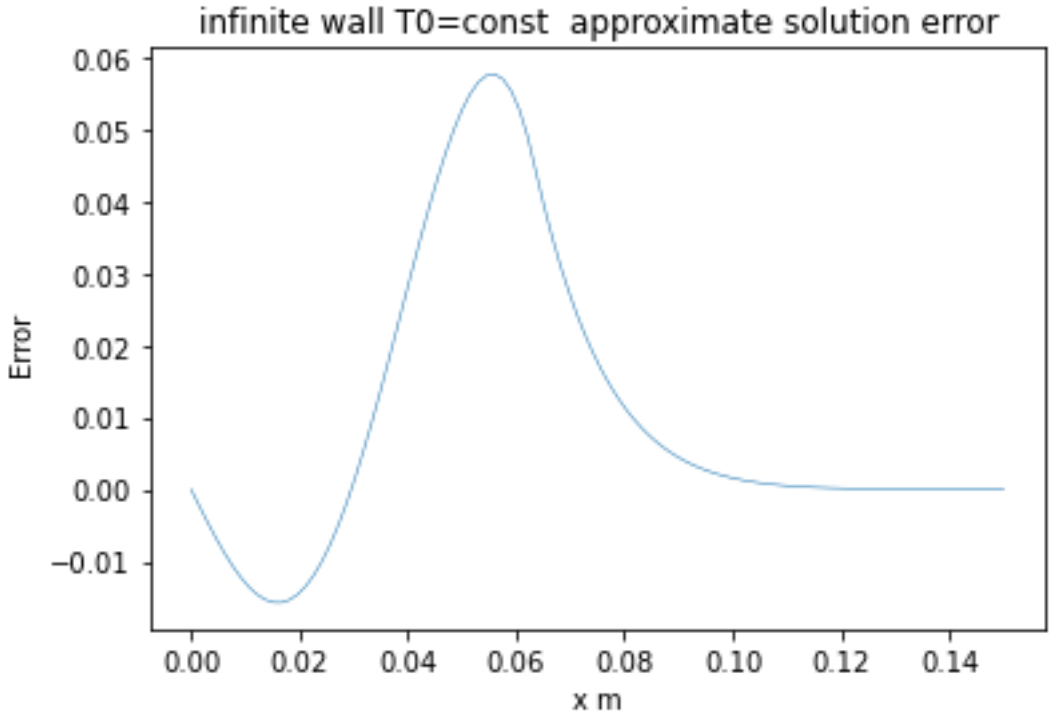
T=1000 s



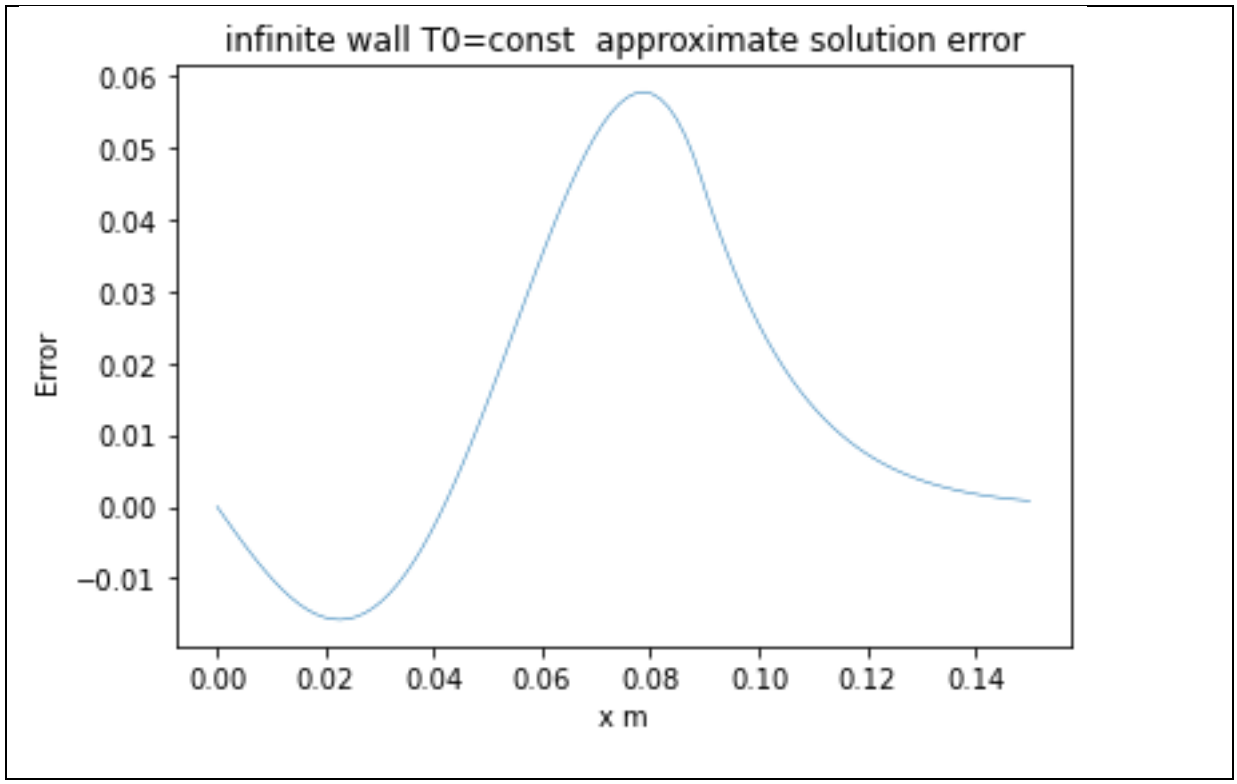
Error t=100 s



Error t=500 s



Error t=1000 s



For better accuracy, degree of polynomial can be increased. Of course additional boundary conditions to solve thic new equations is required. If a 4th degree polynomial is assumed:

$$T(x,t) = a + bx + cx^2 + dx^3 + ex^4$$

Coefficients can be found by applying boundary conditions:

$T(x,t)|_{x=0} = T_0$

$T(x,t)|_{x=\delta(t)} = T_i$

$\frac{\partial T(x,t)}{\partial x}\Big|_{x=\delta(t)} = 0$

$\frac{\partial^2 T(x,t)}{\partial x^2}\Big|_{x=\delta(t)} = 0$

$\frac{\partial^2 T(x,t)}{\partial x^2}\Big|_{x=0} = 0$

$$\frac{\partial T(x,t)}{\partial x} = b + 2cx + 3dx^2 + 4ex^3$$

$\frac{\partial^2 T(x,t)}{\partial x^2} = 2c + 6dx + 12ex^2$

$\frac{\partial^2 T(x,t)}{\partial x^2}\Big|_{x=0} = 0 \quad \rightarrow \quad 2c = 0 \quad c = 0$

$T(x,t)|_{x=0} = T_0 \quad \rightarrow \quad a = T_0$

$\frac{\partial^2 T(x,t)}{\partial x^2}\Big|_{x=\delta(t)} = 0 \quad \rightarrow \; 6d\delta + 12e\delta^2 = 0 \quad d = -2e\delta$

$\frac{\partial T(x,t)}{\partial x}\Big|_{x=\delta(t)} = 0 \quad \rightarrow \quad b + 3d\delta^2 + 4e\delta^3 = b + 3(-2e\delta)\delta^2 + 4e\delta^3 = 0 \quad b = 2e\delta^3$

$T(x,t)|_{x=\delta(t)} = T_i \quad \rightarrow \quad T_i = T_0 + b\delta + d\delta^3 + e\delta^3 = T_0 + (2e\delta^3)\delta + (-2e\delta)\delta^3 + e\,\delta^4$

$d = 2\frac{T_0 - T_i}{\delta^3} \quad e = -\frac{T_0 - T_i}{\delta^4} \quad b = -2\frac{T_0 - T_i}{\delta}$

$$T(x,t) - T_i = (T_0 - T_i) - 2(T_0 - T_i)\left(\frac{x}{\delta}\right) + 2(T_0 - T_i)\left(\frac{x}{\delta}\right)^3 - (T_0 - T_i)\left(\frac{x}{\delta}\right)^4$$

$$\frac{T(x,t) - T_i}{T_0 - T_i} = 1 - 2\left(\frac{x}{\delta}\right) + 2\left(\frac{x}{\delta}\right)^3 - \left(\frac{x}{\delta}\right)^4$$

We can apply this new temperature profile into the differential equation

$$T(x,t) = (T_0 - T_i) - 2(T_0 - T_i)\left(\frac{x}{\delta}\right) + 2(T_0 - T_i)\left(\frac{x}{\delta}\right)^3 - (T_0 - T_i)\left(\frac{x}{\delta}\right)^4 + T_i$$

$$\frac{\partial T(x,t)}{\partial x} = -2(T_0 - T_i)\left(\frac{1}{\delta}\right) + 6(T_0 - T_i)\left(\frac{x^2}{\delta^3}\right) - 4(T_0 - T_i)\left(\frac{x}{\delta}\right)^4$$

$$\left.\frac{\partial T(x,t)}{\partial x}\right|_{x=0} = -2(T_0 - T_i)\left(\frac{1}{\delta}\right)$$

$$\theta(x,t) = \int_{x=0}^{x=\delta(t)} T(x,t)\,dx = \int_{x=0}^{x=\delta(t)} \left[(T_0 - T_i) - 2(T_0 - T_i)\left(\frac{x}{\delta}\right) + 2(T_0 - T_i)\left(\frac{x}{\delta}\right)^3 - (T_0 - T_i)\left(\frac{x}{\delta}\right)^4 + T_i\right] dx$$

$$\theta(x,t) = (T_0 - T_i)\left[x - \left(\frac{x^2}{\delta}\right) + \frac{2}{4}\left(\frac{x^4}{\delta^3}\right) - \frac{1}{5}\left(\frac{x^5}{\delta^4}\right)\right]_0^{\delta} + T_i x|_0^{\delta} = (T_0 - T_i)\frac{3}{10}\delta + T_i\delta$$

differential equation becomes:

$$-\alpha\left.\frac{\partial T(x,t)}{\partial t}\right|_{x=0} = \frac{\partial}{\partial t}[\theta(x,t) - T_i\delta(t)]$$

$$-\alpha\left[-2(T_0 - T_i)\left(\frac{1}{\delta}\right)\right] = \frac{\partial}{\partial t}\left[(T_0 - T_i)\frac{3}{10}\delta + T_i\delta - T_i\delta\right]$$

$$\left(\frac{20\alpha}{3\delta}\right) = \frac{\partial}{\partial t}(\delta)$$

$$\left(\frac{20\alpha}{3}\right)\partial t = \delta\partial\delta$$

$$\left(\frac{20\alpha}{3}\right)t = \frac{\delta^2}{2}$$

$$\delta(t) = \sqrt{\frac{40\alpha t}{3}}$$

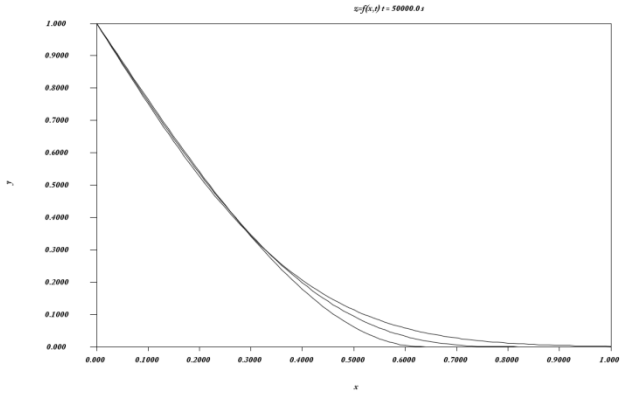
By using this result, together with approximation temperature profile will give us approximation solution. The results can be compare by using a graphic program

Problem 13.1a java solution

```
 class i_wall extends fi_xt
{
public double[] func(double x,double t)
{ double a[]=new double[3];
              double alpha=1.0e-6;
  double delta=Math.sqrt(8.0*alpha*t);
  double delta1=Math.sqrt(40.0/3.0*alpha*t);
  double eta=x/2/Math.sqrt(alpha*t);
  double xx=x/delta;
  double xx1=x/delta1;

  a[0]=erfc(eta);
  if(xx<1) a[1]=1.0-1.5*xx+0.5*xx*xx*xx;
  else a[1]=0.0;
  if(xx1<1) a[2]=1.0-2*xx1+2.0*xx1*xx1*xx1-xx1*xx1*xx1*xx1;
  else a[2]=0.0;
  return a;
}             }
public class timeplot_ntest
{
public static void main(String args[])
  {
 i_wall ff=new i_wall();
  timeplot_n p=new timeplot_n(ff,0.0,1.0,0.01,0.0,50000.0,0.1,100.0);
  p.start();
  }
```

```
}
```



Now let us consider 1 dimensional, time dependent (Lumped) solution

Python solution of the problem

```
from math import *
from f_xj import *
import numpy as np;
import matplotlib.pyplot as plt;
#exact solution
class f1(f_xj):
    def func(self,z):
        x=z[0]
        t=z[1]
        eta=x/(2.0*sqrt(alpha*t))
        return erfc(eta)

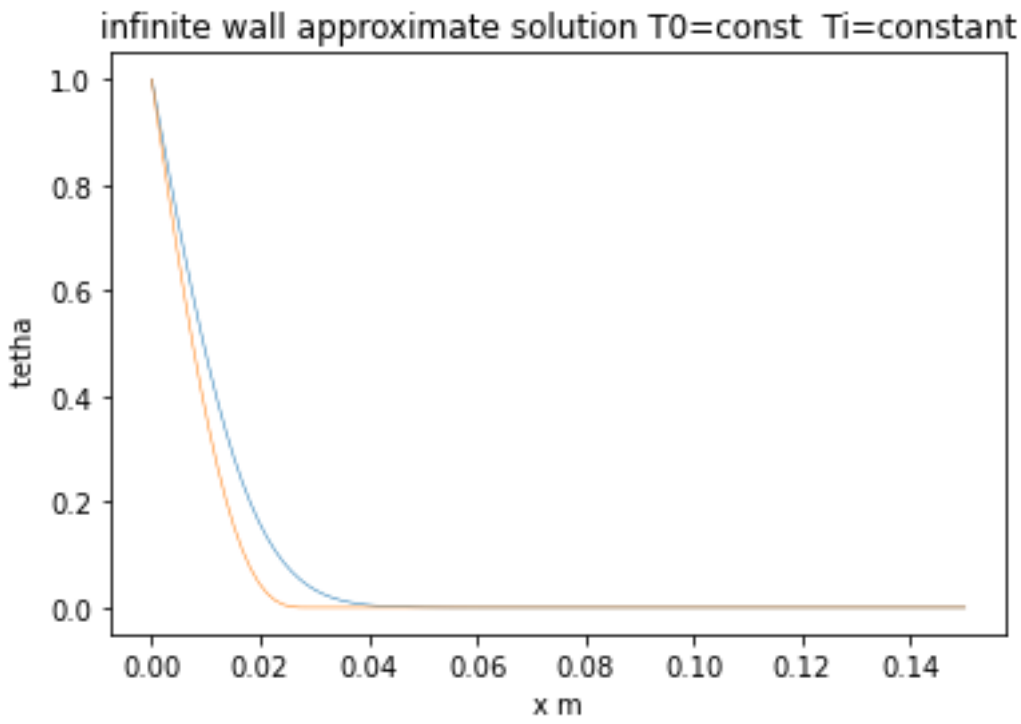
#approximate solution
class f2(f_xj):
    def func(self,z):
        x=z[0]
        t=z[1]
        delta=sqrt(8.0*alpha*t)
        xx=x/delta
        if xx<1:
            T2=1.0-2.0*xx+2.0*xx*xx*xx-xx*xx*xx*xx
        else:
            T2=0
        return T2
# difference between approximate and exact solutionapproximate solution
class f3(f_xj):
    def func(self,z):
        T1=f1()
        T2=f2()
        dT=T1.func(z)-T2.func(z)
        return dT

np1=100
x=[0 for x in range(np1)]
dx=0.15/(len(x)-1)
t=[100]
np2=len(t)
T1= [[0 for x in range(np1)] for y in range(np2)]
T2= [[0 for x in range(np1)] for y in range(np2)]
dT= [[0 for x in range(np1)] for y in range(np2)]
T0=105.0
Ti=10.0
alpha=1e-6
s=""
f=f1()
```

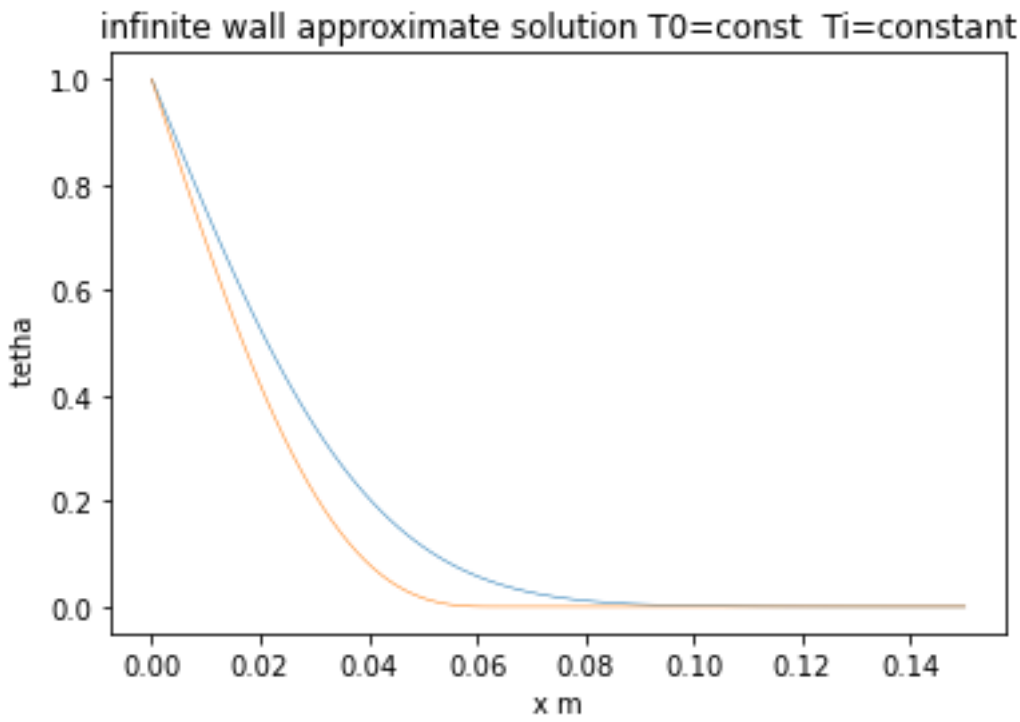
```
ff=f2()
df=f3()
for i in range(np2):
    for j in range(np1):
        x[j]=dx*j
        z=[x[j],t[i]]
        T1[i][j]=f.func(z)
        T2[i][j]=ff.func(z)
        dT[i][j]=df.func(z)

plt.title('infinite wall approximate solution T0=const  Ti=constant')
plt.xlabel('x m')
plt.ylabel('tetha ')
for i in range(np2):
    plt.plot(x,T1[i],linewidth=0.5)
    plt.plot(x,T2[i],linewidth=0.5)
"""
plt.title('infinite wall T0=const  approximate solution error')
plt.xlabel('x m')
plt.ylabel('Error ')
for i in range(np2):
    plt.plot(x,dT[i],linewidth=0.5)
"""
```

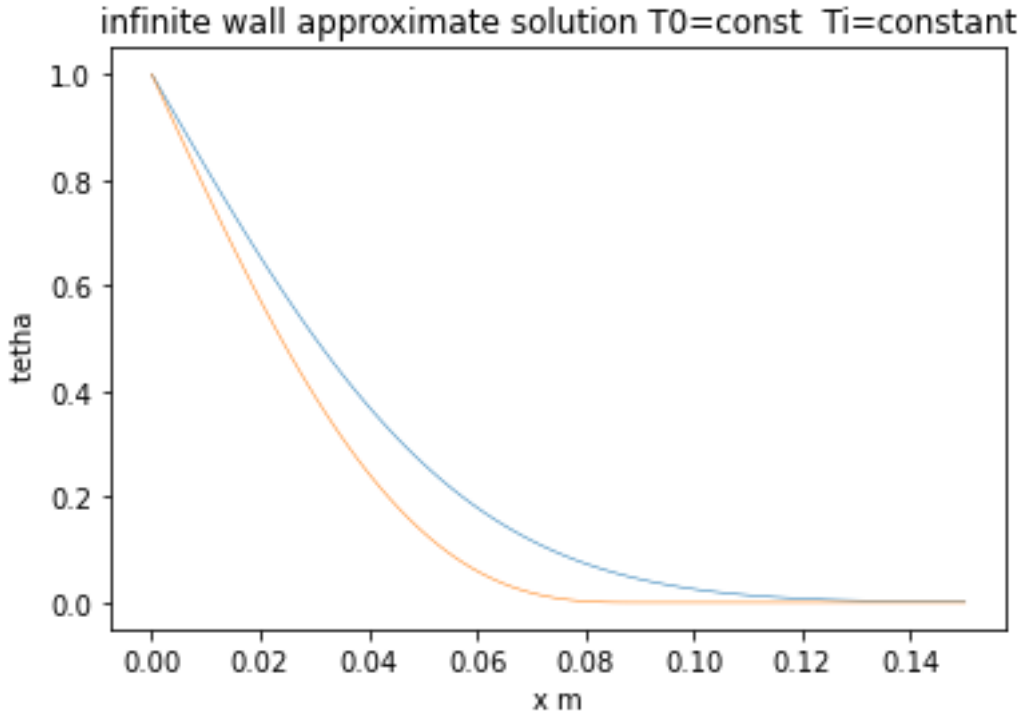
t=100 s



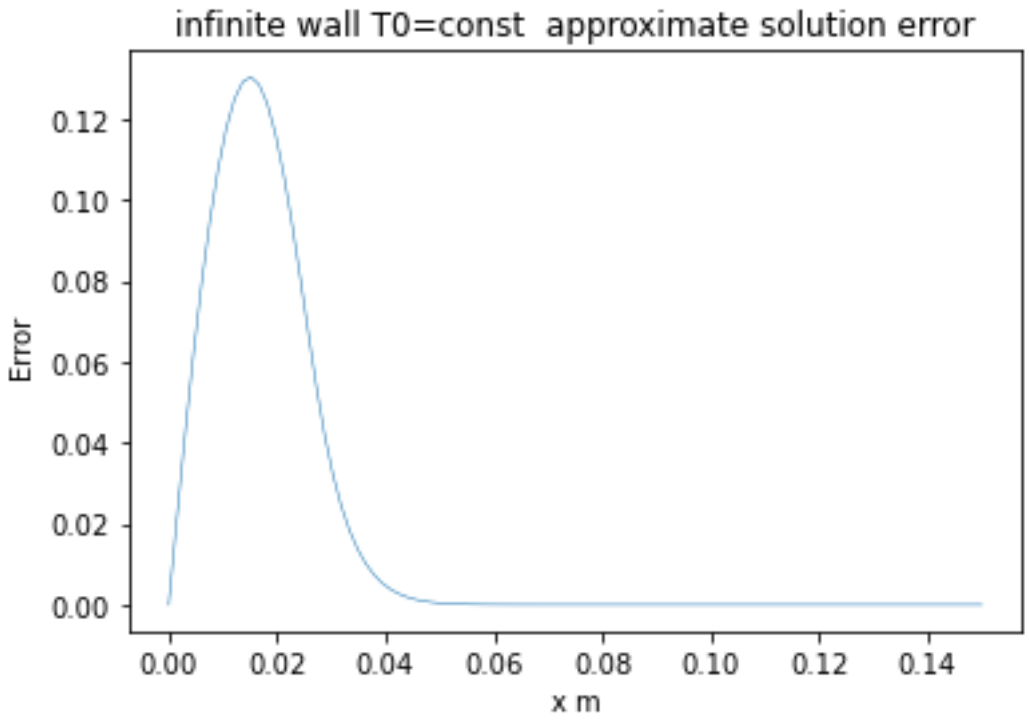
t=500 s



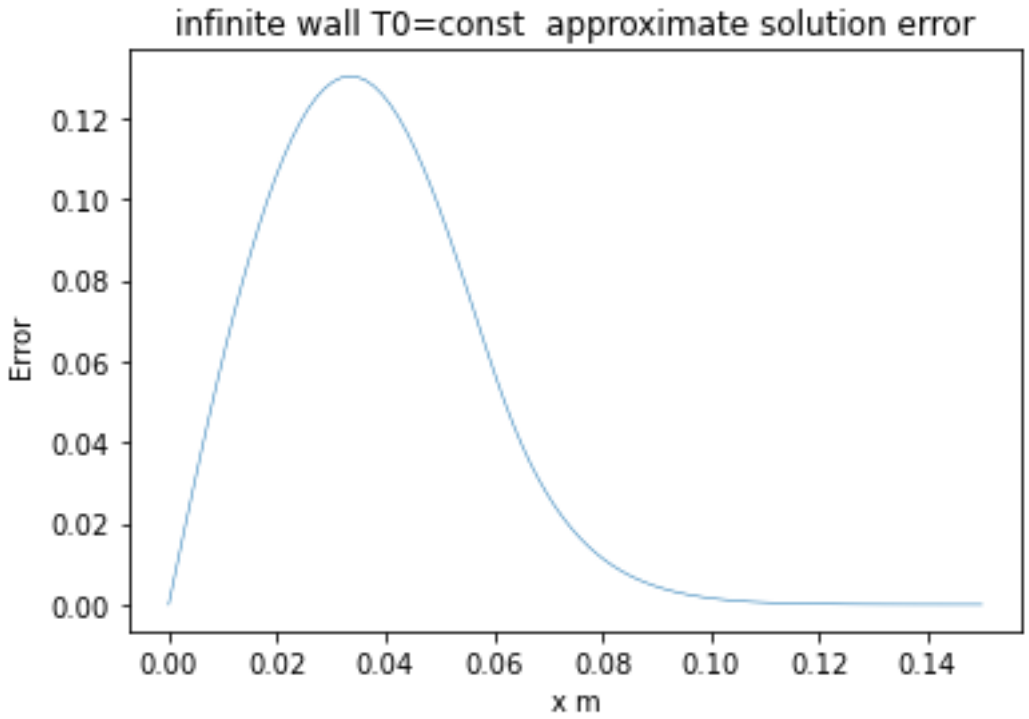
t=1000 s



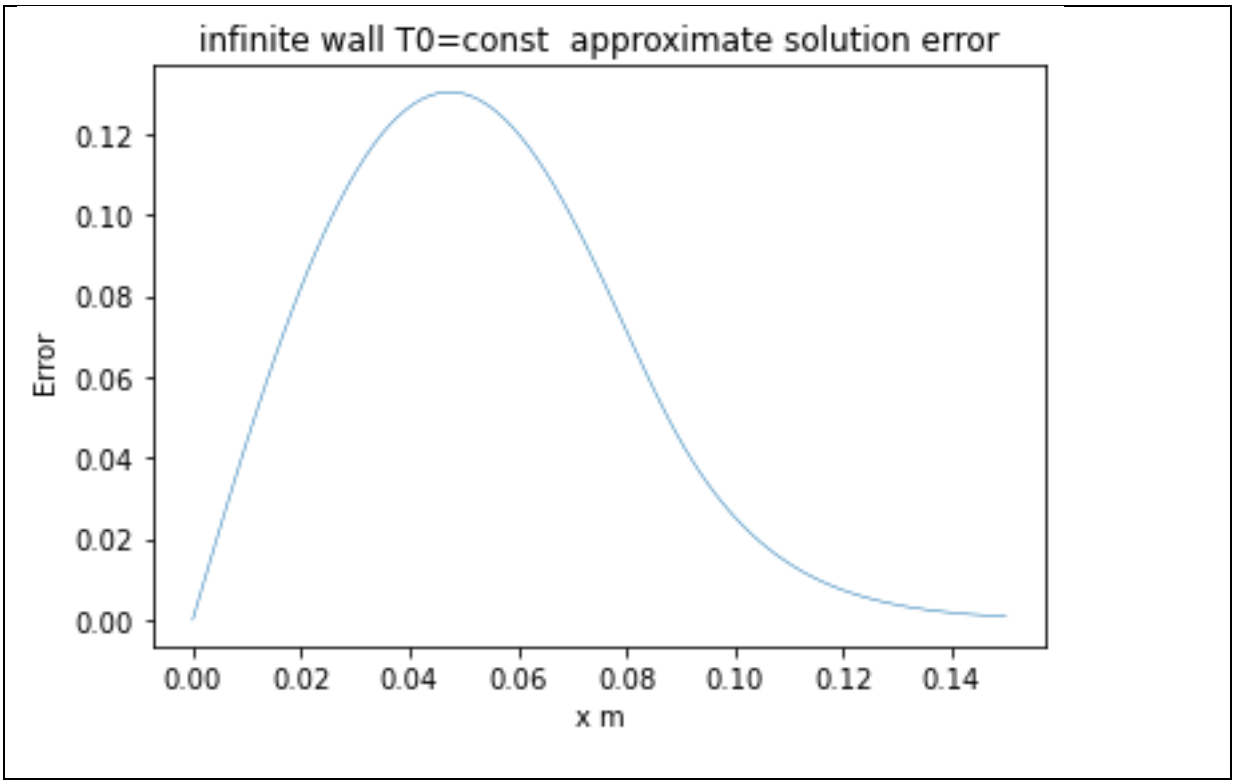
Error t=100 s



Error t=500 s



Error t=1000 s



In higher degree of polynomial version error is bigger. That indicates that increasing degree of polynomial not necessarily improve solution.

PROBLEM 13.2 TIME DEPENDENT LUMPED SOLUTION APPROXIMATION METHODS (COLLOCATION METHOD)

$$-hA_s(T(t)-T_\infty)=\rho VC_p\frac{dT(t)}{dt}$$

where A_s is surface area, V is volume, C_p is specific heat. By changing variables as:

$\theta(t)=\frac{T(t)-T_\infty}{T_i-T_\infty}\quad \tau=\frac{t}{\tau_{max}}\quad X=\frac{\rho VC_p}{\tau_{max}hA_s}$

equation becomes:

$$X\frac{d\theta(t)}{dt}+\theta(t)=0$$

Boundary condition t=0 θ =1so exact solution is:

$$\theta(t)=exp\left(-\frac{\tau}{X}\right)$$

Now let us create approximate solution, for this the following polynomial is assumed:

$$\theta(t)=1+c_1\tau+c_2\tau^2+c_3\tau^3+c_4\tau^4$$

This equation automatically satisfy the boundary condition but if it is applied to differential equation a residue is remained as:

$$\frac{d\theta(t)}{dt}=c_1\tau+2c_2\tau\ +3c_3\tau^2+4c_4\tau^3$$

$$X\frac{d\theta(t)}{dt}+\theta(t)=0$$

$$R=X(c_1\tau+2c_2\tau\ +3c_3\tau^2+4c_4\tau^3)+(1+c_1\tau+c_2\tau^2+c_3\tau^3+c_4\tau^4)$$

$$R=1+(X+\tau)c_1+(2\tau X+\tau^2)c_2+(3\tau^2X+\tau^3)c_3+(4\tau^3+\tau^4)c_4$$

Assuming that equation is valid in the range of $0\le\tau\le1$the equation will be evaluate at collocation locations such as:

$\tau_1=0.25\quad \tau_2=0.50\quad \tau_3=0.75\quad \tau_4=1.0\quad \tau_{max}=60\ s$

This method is called **Collocation method**

Coefficients can be obtained by solving the system of equations

$$\begin{bmatrix} (X+\tau_1) & (2\tau_1 X+\tau_1^2) & (3\tau_1^2 X+\tau_1^3) & (4\tau_1^3+\tau_1^4) \\ (X+\tau_2) & (2\tau_2 X+\tau_2^2) & (3\tau_2^2 X+\tau_2^3) & (4\tau_2^3+\tau_2^4) \\ (X+\tau_3) & (2\tau_3 X+\tau_3^2) & (3\tau_3^2 X+\tau_3^3) & (4\tau_3^3+\tau_3^4) \\ (X+\tau_4) & (2\tau_4 X+\tau_4^2) & (3\tau_4^2 X+\tau_4^3) & (4\tau_4^3+\tau_4^4) \end{bmatrix} \begin{Bmatrix} c_1 \\ c_2 \\ c_3 \\ c_3 \end{Bmatrix} = \begin{Bmatrix} -1 \\ -1 \\ -1 \\ -1 \end{Bmatrix}$$

Let us write a small program to do this task:

```
 import java.io.*;

class collocation
{
  public static double[] pivotlugauss(double a[][],double b[])
  { //gauss elimination with partial pivoting
  int n=b.length;
  double x[]=new double[n];
  double carpan=0;
  double toplam=0;
  double buyuk;
  double dummy=0;
  //gauss eleme
  int i,j,k,p,ii,jj;
  System.out.println("Original matrix :\n"+Matrix.toString(a));
  for(k=0;k<(n-1);k++)
  { //pivoting
              p=k;
              buyuk=Math.abs(a[k][k]);
              for(ii=k+1;ii<n;ii++)
              {  dummy=Math.abs(a[ii][k]);
                if(dummy > buyuk) {buyuk=dummy;p=ii;}
              }
              if(p!=k)
              { for(jj=k;jj<n;jj++)
                { dummy=a[p][jj];
                  a[p][jj]=a[k][jj];
                  a[k][jj]=dummy;
                }
                dummy=b[p];
                b[p]=b[k];
                b[k]=dummy;
              }
               System.out.println("pivoted matrix : k="+k+"\n"+Matrix.toString(a));
              //Solving gauss elimination
    for(i=k+1;i<n;i++)
    {  carpan=a[i][k]/a[k][k];
       a[i][k]=0;
       for(j=k+1;j<n;j++)
       {   a[i][j]-=carpan*a[k][j]; }
          b[i]   =b[i]   -carpan*b[k];
              }
  }
  System.out.println("final upper triangular matrix : \n"+Matrix.toString(a));
  //back substitution
  x[n-1]=b[n-1]/a[n-1][n-1];
  for(i=n-2;i>=0;i--)
  {
    toplam=0;
    for(j=i+1;j<n;j++)
    {  toplam+=a[i][j]*x[j];}
    x[i]=(b[i]-toplam)/a[i][i];
  }
  return x;
  }
```

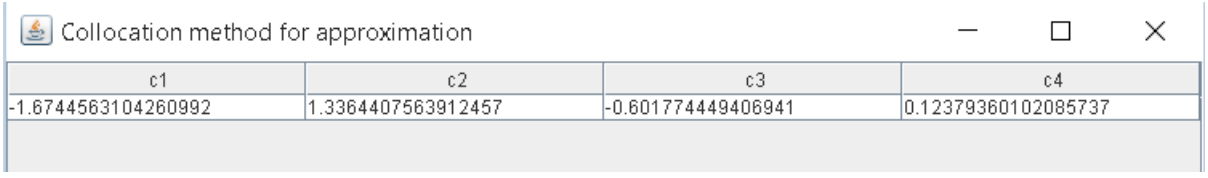
```
  public static double[] c(double X, double to[])
  {  double a[][]=new double[4][4];
    a[0][0]=(X+to[0]);a[1][0]=(X+to[1]);a[2][0]=(X+to[2]);a[3][0]=(X+to[3]);
    a[0][1]=(2.0*X*to[0]+to[0]*to[0]);a[1][1]=(2.0*X*to[1]+to[1]*to[1]);
    a[2][1]=(2.0*X*to[2]+to[2]*to[2]);a[3][1]=(2.0*X*to[3]+to[3]*to[3]);
    a[0][2]=(3.0*X*to[0]*to[0]+to[0]*to[0]*to[0]);a[1][2]=(3.0*X*to[1]*to[1]+to[1]*to[1]*to[1]);
              a[2][2]=(3.0*X*to[2]*to[2]+to[2]*to[2]*to[2]);a[3][2]=(3.0*X*to[3]*to[3]+to[3]*to[3]*to[3]);
    a[0][3]=(4.0*X*to[0]*to[0]*to[0]+to[0]*to[0]*to[0]*to[0]);
    a[1][3]=(4.0*X*to[1]*to[1]*to[1]+to[1]*to[1]*to[1]*to[1]);
    a[2][3]=(4.0*X*to[2]*to[2]*to[2]+to[2]*to[2]*to[2]*to[2]);
    a[3][3]=(4.0*X*to[3]*to[3]*to[3]+to[3]*to[3]*to[3]*to[3]);
              double b[]={-1,-1,-1,-1};
              Text.print(a,"a");
              Text.print(b,"b");
              double c[]=pivotlugauss(a,b);
              return c;
  }

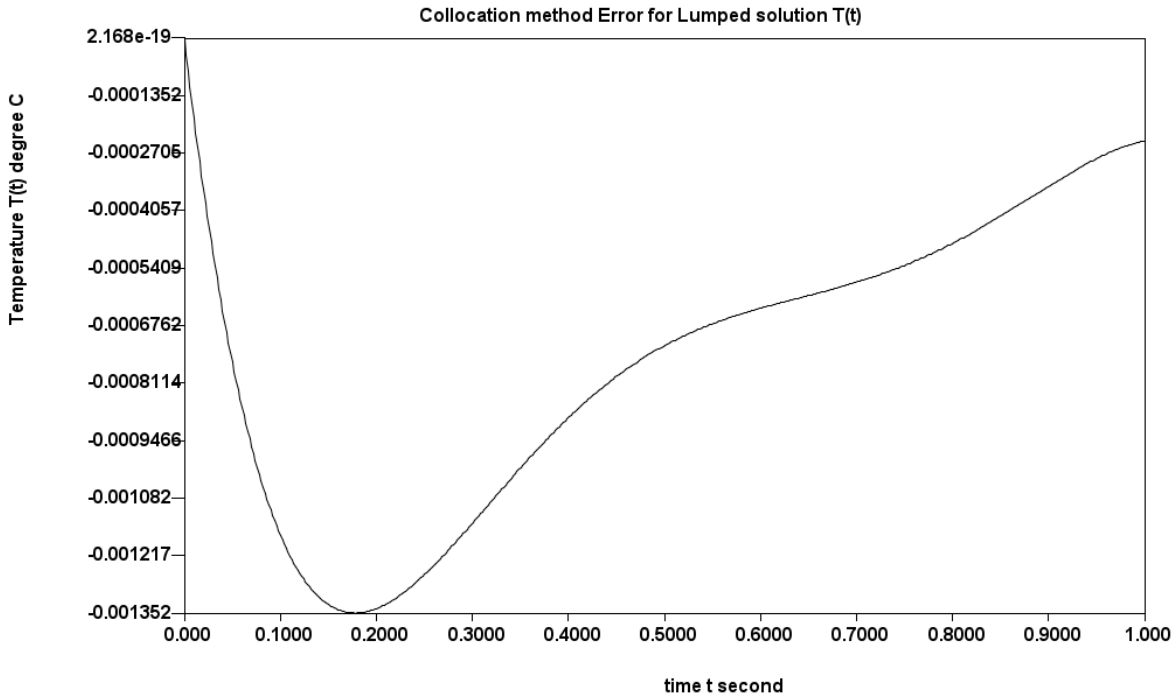
  public static void main(String args[]) throws IOException
  {
  double toi[]={0.25,0.5,0.75,1.0};
  double ro=8500; //kg/m^3
  double Cp=400; //J/kgK
  double h=80; //W/m2K
  double D=0.02;
  double k=20; //W/mK
  double V=Math.PI*D*D*D/6;
  double A=Math.PI*D*D;
  double Bi=h*D/6/k;
  System.out.println("Bi="+Bi);
  double tomax=240;
  double X=ro*V*Cp/(tomax*h*A);
  double c[]=c(X,toi);
  String c1[]={"c1","c2","c3","c4"};
  Text.print(c,c1,"Collocation method for approximation");
  if_x f=to->Math.exp(-to/X);
  if_x f1=to->1+c[0]*to+c[1]*to*to+c[2]*to*to*to+c[3]*to*to*to*to;
   if_x df=to->f.func(to)-f1.func(to);
  Plot pp=new Plot(df,0,1);
  pp.setPlabel("Collocation method Error for Lumped solution T(t)");
  pp.setXlabel("time t second");
  pp.setYlabel("Temperature T(t) degree C");
  pp.plot();
  }
}
```

0.840278	0.357639	0.126302	0.040799	C1
1.090278	0.840278	0.567708	0.357639	c2
1.340278	1.447917	1.417969	1.3125	c3
1.590278	2.180556	2.770833	3.361111	c4

-1
-1
-1
-1

c1	c2	c3	c4
-1.67446	1.336441	-0.60177	0.123794





Can ve increase the number of data points while keeping degree of polynomial constant. It is possible to use singular value decomposition method to solve the matrix system. Assuming that equation is valid in the range of $0 \leq \tau \leq 1$the equation will be evaluate at collocation locations such as:

$$\theta(t) = 1 + c_1\tau + c_2\tau^2 + c_3\tau^3 + c_4\tau^4$$

$\tau_1 = 0.1 \quad \tau_2 = 0.2 \quad \tau_3 = 0.3 \quad \tau_4 = 0.4 \quad \tau_5 = 0.5 \quad \tau_6 = 0.6 \quad \tau_7 = 0.7 \quad \tau_8 = 0.8 \quad \tau_8 = 0.9$
$\tau_{10} = 1.0 \quad \tau_{max} = 60\ s$

$$\begin{bmatrix} (X+\tau_1) & (2\tau_1 X+\tau_1^2) & (3\tau_1^2 X+\tau_1^3) & (4\tau_1^3+\tau_1^4) \\ (X+\tau_2) & (2\tau_2 X+\tau_2^2) & (3\tau_2^2 X+\tau_2^3) & (4\tau_2^3+\tau_2^4) \\ (X+\tau_3) & (2\tau_3 X+\tau_3^2) & (3\tau_3^2 X+\tau_3^3) & (4\tau_3^3+\tau_3^4) \\ (X+\tau_4) & (2\tau_4 X+\tau_4^2) & (3\tau_4^2 X+\tau_4^3) & (4\tau_3^3+\tau_3^4) \\ (X+\tau_5) & (2\tau_5 X+\tau_5^2) & (3\tau_5^2 X+\tau_5^3) & (4\tau_5^3+\tau_5^4) \\ (X+\tau_6) & (2\tau_6 X+\tau_6^2) & (3\tau_6^2 X+\tau_6^3) & (4\tau_6^3+\tau_6^4) \\ (X+\tau_7) & (2\tau_7 X+\tau_7^2) & (3\tau_7^2 X+\tau_7^3) & (4\tau_7^3+\tau_7^4) \\ (X+\tau_8) & (2\tau_8 X+\tau_8^2) & (3\tau_8^2 X+\tau_8^3) & (4\tau_8^3+\tau_8^4) \\ (X+\tau_9) & (2\tau_9 X+\tau_9^2) & (3\tau_9^2 X+\tau_9^3) & (4\tau_9^3+\tau_9^4) \\ (X+\tau_{10}) & (2\tau_{10} X+\tau_{10}^2) & (3\tau_{10}^2 X+\tau_{10}^3) & (4\tau_{10}^3+\tau_{10}^4) \end{bmatrix} \begin{Bmatrix} c_1 \\ c_2 \\ c_3 \\ c_3 \end{Bmatrix} = \begin{Bmatrix} -1 \\ -1 \\ -1 \\ -1 \\ -1 \\ -1 \\ -1 \\ -1 \\ -1 \\ -1 \end{Bmatrix}$$

Collocation with singular value decomposion **collocation_SVD.java**

```
import java.io.*;

class collocation_SVD
{

  public static double[] pivotlugauss(double a[][],double b[])
  { //gauss elimination with partial pivoting
  int n=b.length;
  double x[]=new double[n];
  double carpan=0;
  double toplam=0;
  double buyuk;
  double dummy=0;
```

```
//gauss eleme
int i,j,k,p,ii,jj;
System.out.println("Original matrix :\n"+Matrix.toString(a));
for(k=0;k<(n-1);k++)
{ //pivoting
            p=k;
            buyuk=Math.abs(a[k][k]);
            for(ii=k+1;ii<n;ii++)
            {  dummy=Math.abs(a[ii][k]);
              if(dummy > buyuk) {buyuk=dummy;p=ii;}
            }
            if(p!=k)
            { for(jj=k;jj<n;jj++)
              { dummy=a[p][jj];
                a[p][jj]=a[k][jj];
                a[k][jj]=dummy;
              }
              dummy=b[p];
              b[p]=b[k];
              b[k]=dummy;
            }
              System.out.println("pivoted matrix : k="+k+"\n"+Matrix.toString(a));
            //Solving gauss elimination
  for(i=k+1;i<n;i++)
  {  carpan=a[i][k]/a[k][k];
     a[i][k]=0;
     for(j=k+1;j<n;j++)
     {   a[i][j]-=carpan*a[k][j]; }
        b[i]   =b[i]   -carpan*b[k];
            }
}
System.out.println("final upper triangular matrix : \n"+Matrix.toString(a));
//back substitution
x[n-1]=b[n-1]/a[n-1][n-1];
for(i=n-2;i>=0;i--)
{
  toplam=0;
  for(j=i+1;j<n;j++)
  {  toplam+=a[i][j]*x[j];}
  x[i]=(b[i]-toplam)/a[i][i];
}
return x;
}
public static double[] c(double X, double to[])
{  double a[][]=new double[10][4];

a[0][0]=(X+to[0]);a[1][0]=(X+to[1]);a[2][0]=(X+to[2]);a[3][0]=(X+to[3]);a[4][0]=(X+to[4]);a[5][0]=(X+to[5]);a[6][0]=(X+to[6]);a[7][0]=(
X+to[7]);a[8][0]=(X+to[8]);a[9][0]=(X+to[9]);

a[0][1]=(2.0*X*to[0]+to[0]*to[0]);a[1][1]=(2.0*X*to[1]+to[1]*to[1]);a[2][1]=(2.0*X*to[2]+to[2]*to[2]);a[3][1]=(2.0*X*to[3]+to[3]*to[3]);
a[4][1]=(2.0*X*to[4]+to[4]*to[4]);a[5][1]=(2.0*X*to[5]+to[5]*to[5]);a[6][1]=(2.0*X*to[6]+to[6]*to[6]);a[7][1]=(2.0*X*to[7]+to[7]*to[7]);
a[8][1]=(2.0*X*to[8]+to[8]*to[8]);a[9][1]=(2.0*X*to[9]+to[9]*to[9]);

a[0][2]=(3.0*X*to[0]*to[0]+to[0]*to[0]*to[0]);a[1][2]=(3.0*X*to[1]*to[1]+to[1]*to[1]*to[1]);a[2][2]=(3.0*X*to[2]*to[2]+to[2]*to[2]*to[2]
);a[3][2]=(3.0*X*to[3]*to[3]+to[3]*to[3]*to[3]);a[4][2]=(3.0*X*to[4]*to[4]+to[4]*to[4]*to[4]);a[5][2]=(3.0*X*to[5]*to[5]+to[5]*to[5]*to[
5]);a[6][2]=(3.0*X*to[6]*to[6]+to[6]*to[6]*to[6]);a[7][2]=(3.0*X*to[7]*to[7]+to[7]*to[7]*to[7]);a[8][2]=(3.0*X*to[8]*to[8]+to[8]*to[8]*t
o[8]);a[9][2]=(3.0*X*to[9]*to[9]+to[9]*to[9]*to[9]);
    a[0][3]=(4.0*X*to[0]*to[0]*to[0]+to[0]*to[0]*to[0]*to[0]); a[1][3]=(4.0*X*to[1]*to[1]*to[1]+to[1]*to[1]*to[1]*to[1]);
a[2][3]=(4.0*X*to[2]*to[2]*to[2]+to[2]*to[2]*to[2]*to[2]); a[3][3]=(4.0*X*to[3]*to[3]*to[3]+to[3]*to[3]*to[3]*to[3]);
a[4][3]=(4.0*X*to[4]*to[4]*to[4]+to[4]*to[4]*to[4]*to[4]);a[5][3]=(4.0*X*to[5]*to[5]*to[5]+to[5]*to[5]*to[5]*to[5]);
a[6][3]=(4.0*X*to[6]*to[6]*to[6]+to[6]*to[6]*to[6]*to[6]);a[7][3]=(4.0*X*to[7]*to[7]*to[7]+to[7]*to[7]*to[7]*to[7]);
a[8][3]=(4.0*X*to[8]*to[8]*to[8]+to[8]*to[8]*to[8]*to[8]);a[9][3]=(4.0*X*to[9]*to[9]*to[9]+to[9]*to[9]*to[9]*to[9]);
              double b[]={-1,-1,-1,-1,-1,-1,-1,-1,-1,-1};
              Text.print(a);
```

```
            Text.print(b);
            SVD q=new SVD(a);
   double c[]=q.solve(b);
            return c;
  }

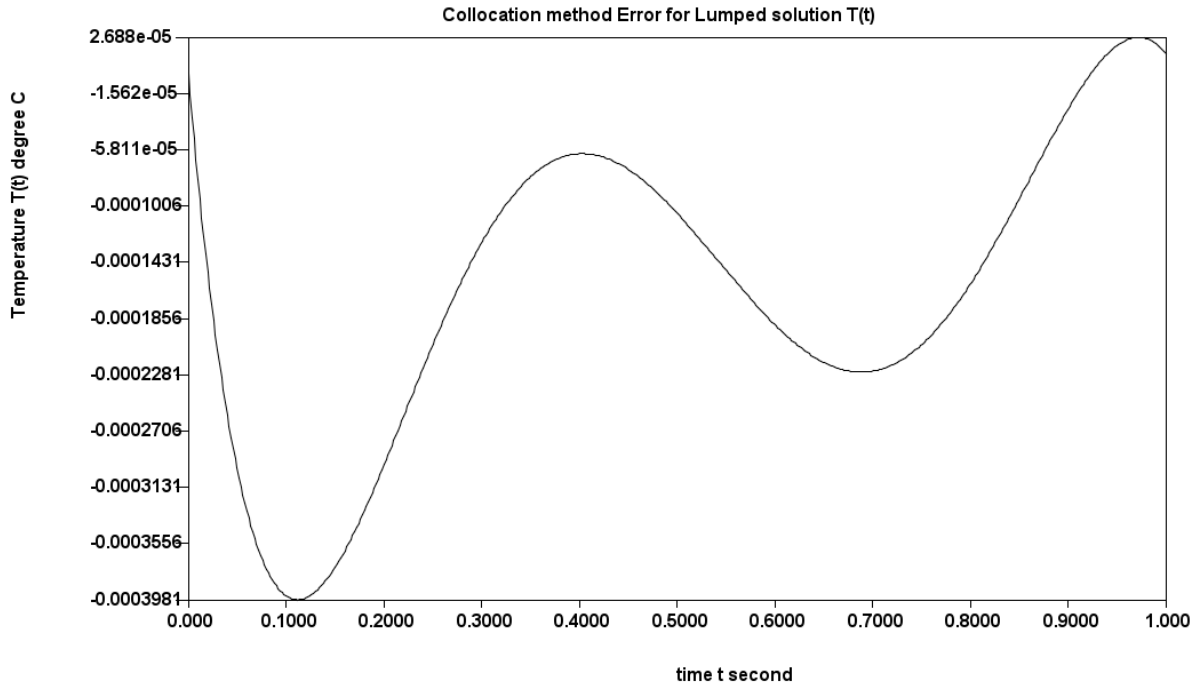
  public static void main(String args[]) throws IOException
  {
  double toi[]={0.1,0.2,0.3,0.4,0.5,0.6,0.7,0.8,0.9,1.0};
  double ro=8500; //kg/m^3
  double Cp=400; //J/kgK
  double h=80; //W/m2K
  double D=0.02;
  double k=20; //W/mK
  double V=Math.PI*D*D*D/6;
  double A=Math.PI*D*D;
  double Bi=h*D/6/k;
  System.out.println("Bi="+Bi);
  double tomax=240;
  double X=ro*V*Cp/(tomax*h*A);
  double c[]=c(X,toi);
  String c1[]={"c1","c2","c3","c4"};
  Text.print(c,c1,"Collocation method for approximation");
  if_x f=to->Math.exp(-to/X);
  if_x f1=to->1+c[0]*to+c[1]*to*to+c[2]*to*to*to+c[3]*to*to*to*to;
  Plot pp=new Plot(f,0,1);
  pp.addFunction(f1,0,1);
  pp.plot();
  }
}
```

0.690278	0.128056	0.018708	0.002461	c1
0.790278	0.276111	0.078833	0.020489	c2
0.890278	0.444167	0.186375	0.07185	c3
0.990278	0.632222	0.347333	0.176711	c4
1.090278	0.840278	0.567708	0.357639	
1.190278	1.068333	0.8535	0.6396	
1.290278	1.316389	1.210708	1.049961	
1.390278	1.584444	1.645333	1.618489	
1.490278	1.8725	2.163375	2.37735	
1.590278	2.180556	2.770833	3.361111	

-1
-1
-1
-1
-1
-1
-1
-1
-1
-1

Solution:

c1	c2	c3	c4
-1.68545	1.373155	-0.64461	0.140645

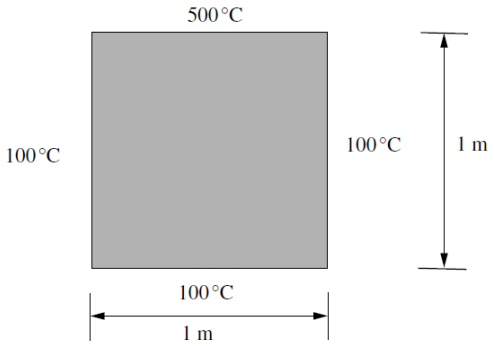


It is clear that Collocation method with SVD reduces the error. Maximum error in collocation (4 equation) is 0.001352. Maximum error in Collocation with SVD(10 equation) is 0.0003981

PROBLEM 12.3 TWO DIMENSIONAL STEADY-STATE PROBLEM APPROXIMATION WITH COLLOCATION METHOD

Now let us consider the following problem:

A square plate with temperatures of 100 C in three sides and 500 C in the fourth side is given. Plate sides are given as 1 m. Calculate the temperature profile



Differential equation:

$$\frac{\partial^2 T(x,y)}{\partial x^2} + \frac{\partial^2 T(x,y)}{\partial x^2} = 0$$

Polynomial equation

$$T(x,y) = \sum_{i=0}^{n} \sum_{j=0}^{m} a_{i(n+1)+j} x^i y^j$$

$$\frac{\partial T(x,y)}{\partial x} = \sum_{i=1}^{n} \sum_{j=0}^{m} i * a_{i(n+1)+j} x^{i-1} y^j$$

$$\frac{\partial T(x,y)}{\partial y} = \sum_{i=0}^{n} \sum_{j=1}^{m} j * a_{i(n+1)+j} x^i y^{j-1}$$

$$\frac{\partial^2 T(x,y)}{\partial x^2} = \sum_{i=2}^{n}\sum_{j=0}^{m} i(i-1)a_{i(n+1)+j}x^{i-2}y^{j}$$

$$\frac{\partial^2 T(x,y)}{\partial y^2} = \sum_{i=0}^{n}\sum_{j=2}^{m} j(j-1)a_{i(n+1)+j}x^{i}y^{j-2}$$

$$\frac{\partial^2 T(x,y)}{\partial x \partial y} = \sum_{i=1}^{n}\sum_{j=1}^{m} (i-1)(j-1)a_{i(n+1)+j}x^{i-1}y^{j-1}$$

Differential equation becomes:

$$R = \frac{\partial^2 T(x,y)}{\partial x^2} + \frac{\partial^2 T(x,y)}{\partial y^2}$$
$$= \sum_{i=2}^{n}\sum_{j=0}^{m} i(i-1)a_{i(n+1)+j}x^{i-2}y^{j} + \sum_{i=0}^{n}\sum_{j=2}^{m} j(j-1)a_{i(n+1)+j}x^{i}y^{j-2}$$

Equation have total K=(n+1)(m+1) coefficient
Equation divided to P=Px*Py pieces (greater than coefficient of polynomial surface)
x=dx*iPx $0 \le iPx \le (Px-1)$
y=dy*iPy $0 \le iPy \le (Py-1)$

Boundary conditions:
x=0 $0 \le y \le (Py-1)$ T=T_1
x=L $0 \le y \le (Py-1)$ T=T_2
y=0 $0 \le x \le (Px-1)$ T=T_3
y=w $0 \le x \le (Px-1)$ T=T_4

$$T_1 = \sum_{i=0}^{n}\sum_{j=0}^{m} a_{i(n+1)+j}y^{j}$$

$$T_2 = \sum_{i=0}^{n}\sum_{j=0}^{m} a_{i(n+1)+j}L^{i}y^{j}$$

$$T_3 = \sum_{i=0}^{n}\sum_{j=0}^{m} a_{i(n+1)+j}x^{i}$$

$$T_4 = \sum_{i=0}^{n}\sum_{j=0}^{m} a_{i(n+1)+j}x^{i}W^{j}$$

Not that number of equations are not equal to number of unkowns in the polynomial surface. Therefore **Singular value decomposition method** is used to solve the equations for finding the polynomial coefficients. For detailes of SVD method, please refer to eigenvalue problems chapter.

A code is given this program **poly2.java** , defines the two dimensional polynomial function and its coefficients and derivatives.

```
 //Polynomial with two independent variable
//polinomial of order k=n*m
public class poly2
{ public int k;
  public double func(double a[],double x,double y,int n,int m)
  {   k=n*m;
     int k1=0;
     double xx=0;
     double total=0.0;
     for(int i=0;i<=n;i++)
     {  for(int j=0;j<=m;j++)
        { k1=i*(m+1)+j;
          xx=pow(x,y,i,j);
          total+=a[k1]*xx;
        }
     }
                return total;
  }

  public double dfuncx(double a[],double x,double y,int n,int m)
  {   k=n*m;
     int k1=0;
     double total=0.0;
     for(int i=0;i<=n;i++)
     {for(int j=0;j<=m;j++)
      {k1=i*(m+1)+j;
       double xx=dpowx(x,y,i,j);
       total+=a[k1]*xx;
       System.out.println("i="+i+"j="+j+"k1="+k1+"xx="+xx+"a="+a[k1]+"total="+total);
      }
     }
                return total;
  }

  public double dfuncy(double a[],double x,double y,int n,int m)
  {   k=n*m;
     int k1=0;
     double total=0.0;
     for(int i=0;i<=n;i++)
     {for(int j=0;j<=m;j++)
      {k1=i*(m+1)+j;
      double xx=dpowy(x,y,i,j);
       total+=a[k1]*xx;
       System.out.println("i="+i+"j="+j+"k1="+k1+"xx="+xx+"a="+a[k1]+"total="+total);
      }
     }
                return total;
  }

  public double d2funcx2(double a[],double x,double y,int n,int m)
  {   k=n*m;
     int k1=0;
     double total=0.0;
     if(a.length<k) System.out.println("error");
     for(int i=0;i<=n;i++)
     {for(int j=0;j<=m;j++)
      {k1=i*(m+1)+j;
       total+=a[k1]*d2powx2(x,y,i,j);
      }
     }
                return total;
  }
  public double d2funcy2(double a[],double x,double y,int n,int m)
  {   k=n*m;
```

```
    int k1=0;
    double total=0.0;
    for(int i=0;i<=n;i++)
    {for(int j=0;j<=m;j++)
     {k=i*(m+1)+j;
      total+=a[k]*d2powy2(x,y,i,j);
     }
    }
            return total;
}
public double d2funcxy(double a[],double x,double y,int n,int m)
{   k=n*m;
    int k1=0;
    double total=0.0;
    if(a.length<k) System.out.println("error");
    for(int i=0;i<=n;i++)
    {for(int j=0;j<=m;m++)
     {k=i*(m-1)+j;
      total+=a[k]*d2powxy(x,y,i,j);
     }
    }
            return total;
}
//=====================================
public static double pow(double x,double y,int n,int m)
{ double pow=1.0;
            if(n==0 && m==0) return pow;
 else if(m==0 && n!=0)
 {for(int i=0;i<n;i++)
  {pow*=x;}
 }
 else if(n==0 && m!=0)
 {for(int j=0;j<m;j++)
  {pow*=y;}
 }
 else
 {for(int i=0;i<n;i++)
  {pow*=x;}
  for(int j=0;j<m;j++)
  {pow*=y;}
 }
 return pow;
}
public static double dpowx(double x,double y,int n,int m)
{ double powx=n;
  double powy=1.0;
  for(int i=1;i<n;i++)
  {powx*=x;}
  for(int j=0;j<m;j++)
  {powy*=y;}
  return powx*powy;
}
public static double dpowy(double x,double y,int n,int m)
{ double powx=1.0;
  double powy=m;
  for(int i=0;i<n;i++)
  {powx*=x;}
  for(int j=1;j<m;j++)
  {powy*=y;}
  return powx*powy;
}
public static double d2powx2(double x,double y,int n,int m)
{ double powx=n*(n-1);
  double powy=1.0;
```

```
   for(int i=2;i<n;i++)
   {powx*=x;}
   for(int j=0;j<m;j++)
   {powy*=y;}
   return powx*powy;
  }
  public static double d2powy2(double x,double y,int n,int m)
  { double powx=1.0;
   double powy=m*(m-1);
   for(int i=0;i<n;i++)
   {powx*=x;}
   for(int j=2;j<m;j++)
   {powy*=y;}
   return powx*powy;
   }

  public static double d2powxy(double x,double y,int n,int m)
  { double powx=n;
   double powy=m;
   for(int i=1;i<n;i++)
   {powx*=x;}
   for(int j=1;j<m;j++)
   {powy*=y;}
   return powx*powy;
  }
  public static void main(String arg[])
  { double x=0.1;double y=0.1;
              double a[]={1,2,3,4,5,6,7,8,9};
   poly2 p=new poly2();
   System.out.println(p.func(a,x,y,2,2));
   System.out.println(p.dfuncx(a,x,y,2,2));
   System.out.println(p.dfuncy(a,x,y,2,2));
  }
}
```

Now we can construct solution of the differential equation by using **collocation** method:
HT_poly2_T.java

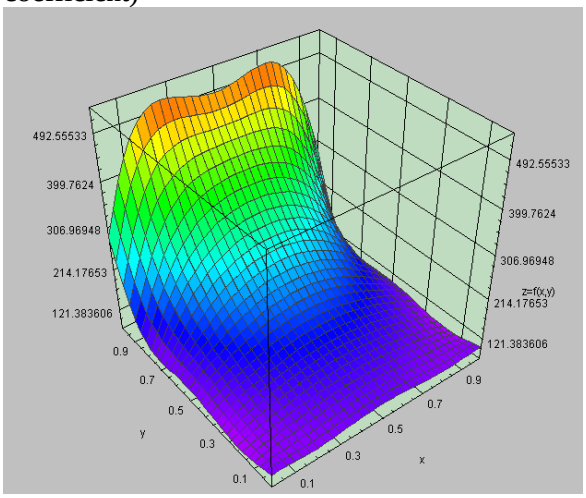
```
 public class HT_poly2_T
{ int P,Px,Py,m,n,k,ip;
double T1,T2,T3,T4;
double dx,dy;
double L,W;
double A[][];
double B[];
double TT;
double x,y;
 public HT_poly2_T(double T1i,double T2i,double T3i,double T4i,int ni,int mi, int Pxi,int Pyi,double Li,double Wi)
 {Px=Pxi;Py=Pyi;P=Px*Py;n=ni;m=mi;k=(n+1)*(m+1);T1=T1i;T2=T2i;T3=T3i;T4=T4i;W=Wi;L=Li;dx=L/Px;dy=W/Py;TT=0;
 A=new double[P][k];B=new double[P];
 int k1=0;
 for(int ipx=0;ipx<Px;ipx++)
 {for(int ipy=0;ipy<Py;ipy++)
  {   x=dx*ipx;y=dy*ipy;
     ip=ipx*Py+ipy;
              if(ipx==0)
              {TT=T3;
               for(int i=0;i<=n;i++)
               {for(int j=0;j<=m;j++)
                {k1=i*(m+1)+j;
                 A[ip][k1]=poly2.pow(x,y,i,j);
                 B[ip]=TT;
                }
```

```
                }
               }
               else if(ipx==(Px-1))
               {TT=T4;
                for(int i=0;i<=n;i++)
                {for(int j=0;j<=m;j++)
                 {k1=i*(m+1)+j;
                  A[ip][k1]=poly2.pow(x,y,i,j);
                  B[ip]=TT;
                 }
                }
               }
               else if(ipy==0)
               {TT=T1;
                for(int i=0;i<=n;i++)
                {for(int j=0;j<=m;j++)
                 {k1=i*(m+1)+j;
                  A[ip][k1]=poly2.pow(x,y,i,j);
                  B[ip]=TT;
                 }
                }
               }
               else if(ipy==(Py-1))
               {TT=T2;
                for(int i=0;i<=n;i++)
                {for(int j=0;j<=m;j++)
                 {k1=i*(m+1)+j;
                  A[ip][k1]=poly2.pow(x,y,i,j);
                  B[ip]=TT;
                 }
                }
               }
               else
               {for(int i=0;i<=n;i++)
                {for(int j=0;j<=m;j++)
                 {k1=i*(m+1)+j;
                  A[ip][k1]=poly2.d2powx2(x,y,i,j)+poly2.d2powy2(x,y,i,j);
                        B[ip]=0.0;
                 }
                }
               }
 }
}
Text.print(A,"A");
Text.printT(B,"B");
SVD q=new SVD(A);
double a[]=q.solve(B);
Text.printT(a,"a");
double T[][]=new double[Px][Py];
poly2 p=new poly2();
for(int ipx=0;ipx<Px;ipx++)
{for(int ipy=0;ipy<Py;ipy++)
 {   x=dx*ipx;y=dy*ipy;
     ip=ipx*Py+ipy;
     T[ipx][ipy]=p.func(a,x,y,n,m);
  } // end int ipy=0;ipy<Py;ipy++)
}//end for(int ipx=0;ipx<Px;ipx++)
 Text.print(T,"T");
    double b[][]=surfacePlot.dataformat(0.0,L,0.0,W,T);
    surfacePlot p1;
    p1=new surfacePlot(b);
    p1.plot();
}
public static void main(String arg[])
```

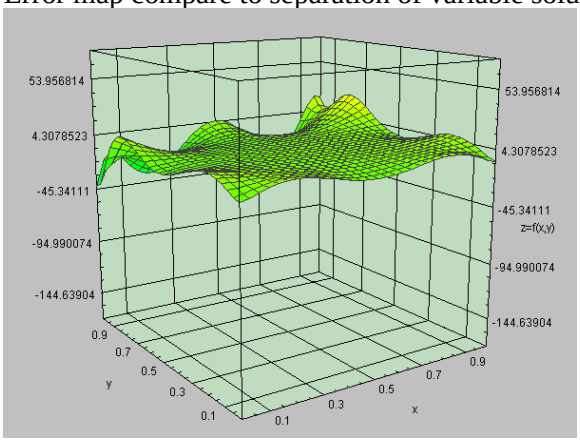
```
{HT_poly2_T h=new HT_poly2_T(100,500.0,100.0,100.0,5,5,80,80,1.0,1.0);

}
}
```

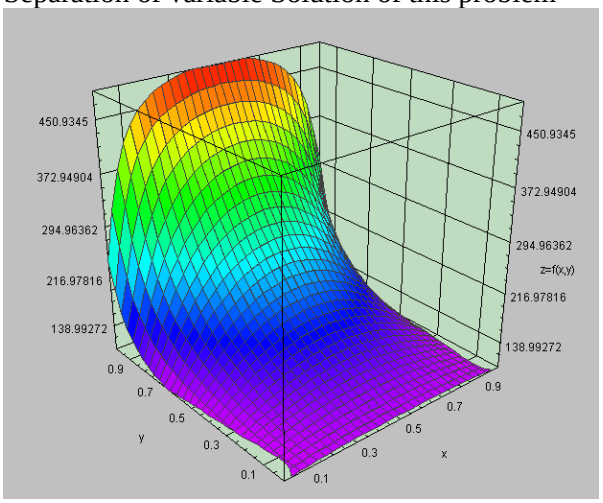
Output of the polynomial function (Temperature profile) n=7,m=7 Px=80,Py=80 (64 polynomial coefficient)



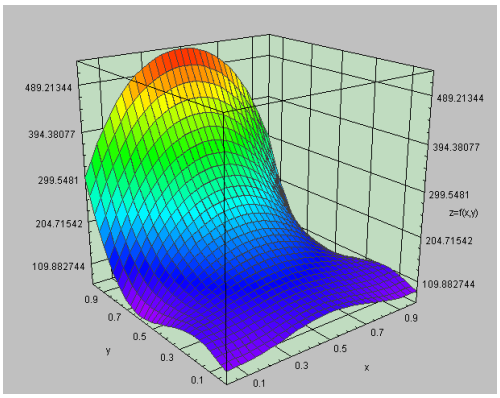
Error map compare to separation of variable solution



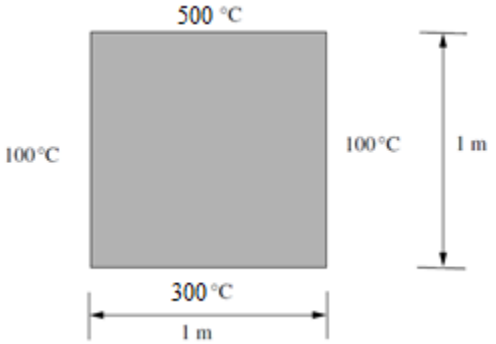
Separation of variable Solution of this problem



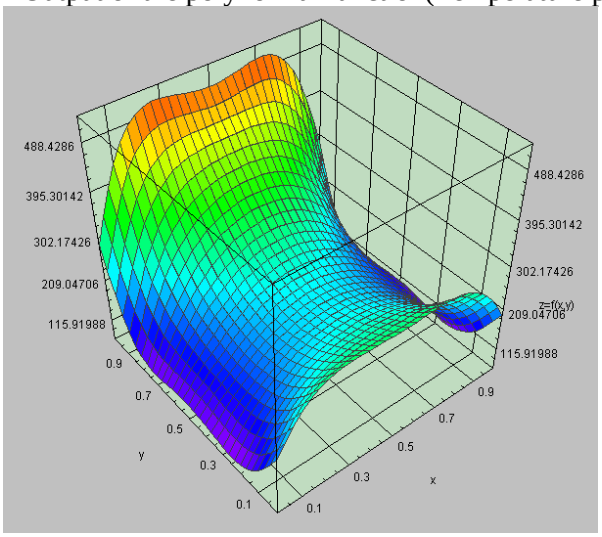
Output of the polynomial function(Temperature profile) n=5,m=5,Px=80,Py=80 (36 polynomial coefficient)



A square plate with temperatures as seen in the figures are given. Plate sides are given as 1 m. Calculate the temperature profile



Output of the polynomial function(Temperature profile) n=7,m=7,Px=80,Py=80



PROBLEM 12.4 TWO DIMENSIONAL STEADY-STATE PROBLEM APPROXIMATION WITH COLLOCATION METHOD

Now let us consider the following problem:

A square plate exposed to convective flux in each side.

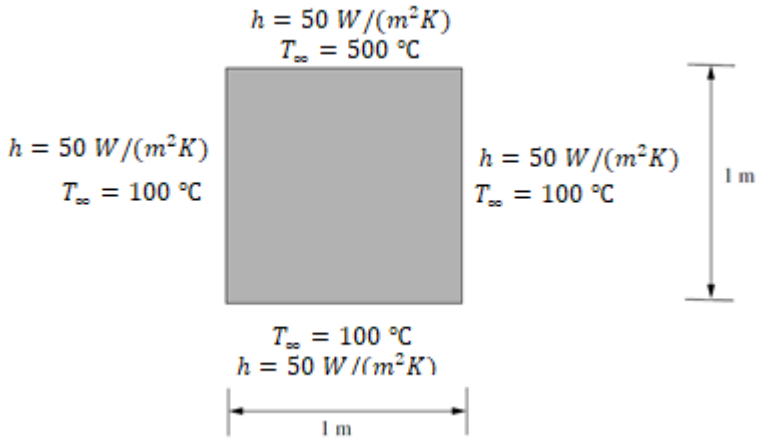
x=0 $h = 50\ W/(m^2K)$ $T_\infty = 100$ ℃

x=L=1 m $h = 50\ W/(m^2K)$ $T_\infty = 100$ ℃

y=0 $h = 50\ W/(m^2K)$ $T_\infty = 100$ ℃

y=W=1 m $h = 50\ W/(m^2K)$ $T_\infty = 500$ ℃

$k = 50\ W/(mK)$ Calculate the temperature profile



Differential equation:

$$\frac{\partial^2 T(x,y)}{\partial x^2} + \frac{\partial^2 T(x,y)}{\partial x^2} = 0$$

Polynomial equation

$$T(x,y) = \sum_{i=0}^{n}\sum_{j=0}^{m} a_{i(n+1)+j} x^i y^j$$

$$\frac{\partial T(x,y)}{\partial x} = \sum_{i=1}^{n}\sum_{j=0}^{m} i * a_{i(n+1)+j} x^{i-1} y^j$$

$$\frac{\partial T(x,y)}{\partial y} = \sum_{i=0}^{n}\sum_{j=1}^{m} j * a_{i(n+1)+j} x^i y^{j-1}$$

$$\frac{\partial^2 T(x,y)}{\partial x^2} = \sum_{i=2}^{n}\sum_{j=0}^{m} i(i-1) a_{i(n+1)+j} x^{i-2} y^j$$

$$\frac{\partial^2 T(x,y)}{\partial y^2} = \sum_{i=0}^{n}\sum_{j=2}^{m} j(j-1) a_{i(n+1)+j} x^i y^{j-2}$$

$$\frac{\partial^2 T(x,y)}{\partial x \partial y} = \sum_{i=1}^{n}\sum_{j=1}^{m} (i-1)(j-1) a_{i(n+1)+j} x^{i-1} y^{j-1}$$

Differential equation becomes:

$$R = \frac{\partial^2 T(x,y)}{\partial x^2} + \frac{\partial^2 T(x,y)}{\partial y^2}$$

$$= \sum_{i=2}^{n}\sum_{j=0}^{m} i(i-1) a_{i(n+1)+j} x^{i-2} y^j + \sum_{i=0}^{n}\sum_{j=2}^{m} j(j-1) a_{i(n+1)+j} x^i y^{j-2}$$

Equation have total K=(n+1)(m+1) coefficient

Equation divided to P=Px*Py pieces (greater than coefficient of polynomial surface)

x=dx*iPx $\quad 0 \le iPx \le (Px-1)$

y=dy*iPy $\quad 0 \le iPy \le (Py-1)$

Boundary conditions:

x=0 $\quad h_1 = 50\ W/(m^2K) \quad T_{1\infty} = 100$ ℃

$$-k\left.\frac{\partial T(x,y)}{\partial x}\right|_{x=0} = h_1(T(x,y)-T_{1\infty})|_{x=0}$$

x=L=1 m $h_2 = 50\ W/(m^2K)$ $T_{2\infty} = 100$ ℃

$$-k\left.\frac{\partial T(x,y)}{\partial x}\right|_{x=L} = h_2(T(x,y)-T_{2\infty})|_{x=L}$$

y=0 $h_3 = 50\ W/(m^2K)$ $T_{3\infty} = 100$ ℃

$$-k\left.\frac{\partial T(x,y)}{\partial x}\right|_{y=0} = h_3(T(x,y)-T_{3\infty})|_{y=0}$$

y=W=1 m $h_4 = 50\ W/(m^2K)$ $T_{4\infty} = 500$ ℃

$$-k\left.\frac{\partial T(x,y)}{\partial x}\right|_{y=W} = h_4T\big((x,y)-T_{4\infty}\big)\big|_{y=W}$$

x=0 $h_1 = 50\ W/(m^2K)$ $T_{1\infty} = 100$ ℃

$$T_1 = \sum_{i=0}^{n}\sum_{j=0}^{m} a_{i(n+1)+j}y^j$$

$$\frac{\partial T_1(x,y)}{\partial x} = \sum_{i=1}^{n}\sum_{j=0}^{m} i * a_{i(n+1)+j}y^j$$

$$-k\left[\sum_{i=1}^{n}\sum_{j=0}^{m} i * a_{i(n+1)+j}y^j\right] = h_1\left(\sum_{i=0}^{n}\sum_{j=0}^{m} a_{i(n+1)+j}y^j - T_{1\infty}\right)$$

x=L=1 m $h_2 = 50\ W/(m^2K)$ $T_{2\infty} = 100$ ℃

$$T_2 = \sum_{i=0}^{n}\sum_{j=0}^{m} a_{i(n+1)+j}L^iy^j$$

$$\frac{\partial T_2(x,y)}{\partial x} = \sum_{i=1}^{n}\sum_{j=0}^{m} i * a_{i(n+1)+j}L^{i-1}y^j$$

$$-k\left[\sum_{i=0}^{n}\sum_{j=0}^{m} a_{i(n+1)+j}L^iy^j\right] = h_2\left(\sum_{i=1}^{n}\sum_{j=0}^{m} i * a_{i(n+1)+j}L^{i-1}y^j - T_{2\infty}\right)$$

y=0 $h_3 = 50\ W/(m^2K)$ $T_{3\infty} = 100$ ℃

$$T_3 = \sum_{i=0}^{n}\sum_{j=0}^{m} a_{i(n+1)+j}x^i$$

$$\frac{\partial T_3(x,y)}{\partial y} = \sum_{i=0}^{n}\sum_{j=1}^{m} j * a_{i(n+1)+j}x^i$$

$$-k\left[\sum_{i=0}^{n}\sum_{j=0}^{m} a_{i(n+1)+j}x^i\right] = h_3\left(\sum_{i=0}^{n}\sum_{j=1}^{m} j * a_{i(n+1)+j}x^i - T_{2\infty}\right)$$

y=W=1 m $h_4 = 50\ W/(m^2K)$ $T_{4\infty} = 500\ ℃$

$$T_4 = \sum_{i=0}^{n}\sum_{j=0}^{m} a_{i(n+1)+j} x^i W^j$$

$$\frac{\partial T_4(x,y)}{\partial y} = \sum_{i=0}^{n}\sum_{j=1}^{m} j * a_{i(n+1)+j} x^i W^{j-1}$$

$$-k\left[\sum_{i=0}^{n}\sum_{j=0}^{m} a_{i(n+1)+j} x^i W^j\right] = h_4\left(\sum_{i=0}^{n}\sum_{j=1}^{m} j * a_{i(n+1)+j} x^i W^{j-1} - T_{2\infty}\right)$$

Not that number of equations are not equal to number of unkowns in the polynomial surface. Therefore Singular value decomposition method is used to solve the equations for finding the polynomial coefficients. **poly2.java** , defined in the previous example problem will also be used in here.

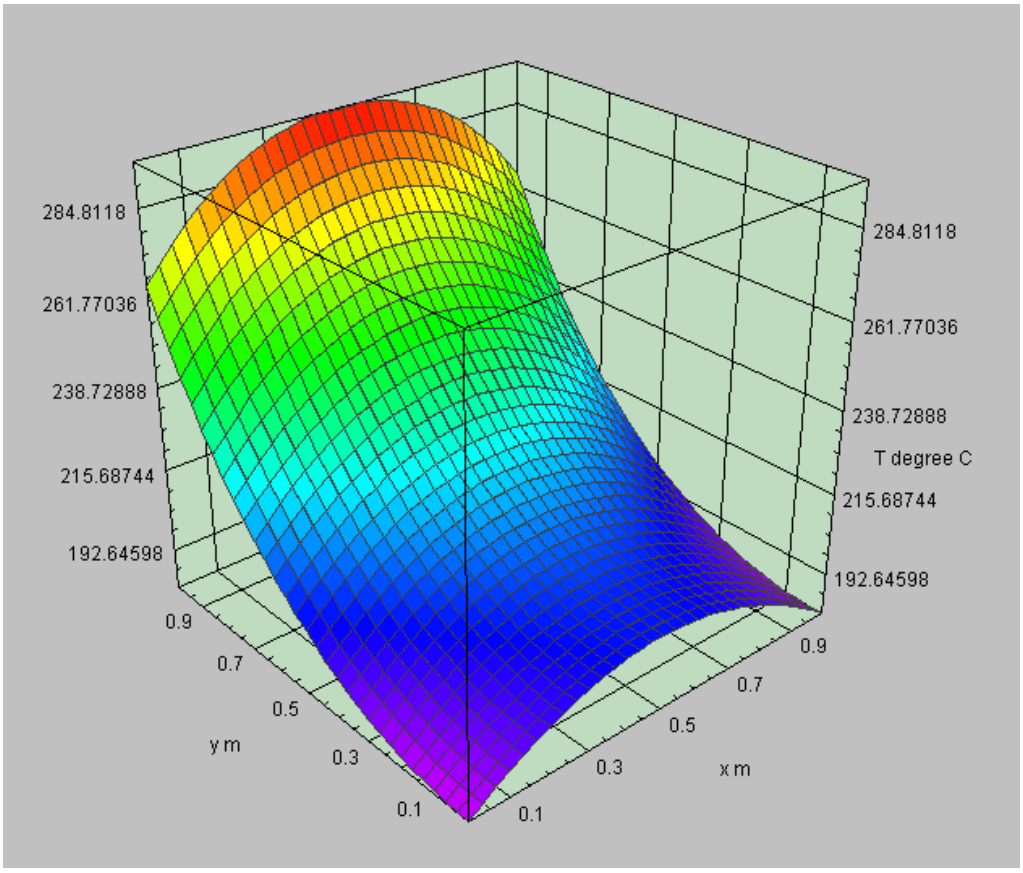
HT_poly2_hTinf.java

```
public class HT_poly2_hTinf
{ int P,Px,Py,m,n,k,ip;
double T1,T2,T3,T4;
double dx,dy;
double L,W;
double A[][];
double B[];
double TT;
double x,y;
double kcond;
double H1,H2,H3,H4;
 public HT_poly2_hTinf(double kcond1,double h1,double h2,double h3,double h4,double T1i,double T2i,double T3i,double T4i,int ni,int mi,
int Pxi,int Pyi,double Li,double Wi)

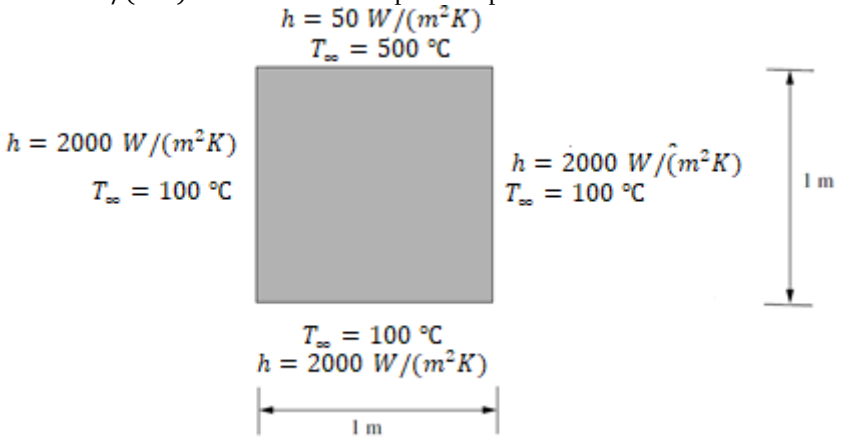
{kcond=kcond1;H1=h1/kcond;H2=h2/kcond;H3=H3/kcond;H4=h4/kcond;Px=Pxi;Py=Pyi;P=Px*Py;n=ni;m=mi;k=(n+1)*(m+1);T1=T1i;T2
=T2i;T3=T3i;T4=T4i;W=Wi;L=Li;dx=L/Px;dy=W/Py;TT=0;
 A=new double[P][k];B=new double[P];
 int k1=0;
 for(int ipx=0;ipx<Px;ipx++)
 {for(int ipy=0;ipy<Py;ipy++)
  {   x=dx*ipx;y=dy*ipy;
     ip=ipx*Py+ipy;
                if(ipx==0)
                {TT=T3;
                 for(int i=0;i<=n;i++)
                 {for(int j=0;j<=m;j++)
                  {k1=i*(m+1)+j;
                   A[ip][k1]=-poly2.dpowx(x,y,i,j)+H3*poly2.pow(x,y,i,j);
                   B[ip]=H3*TT;
                  }
                 }
                }
                else if(ipx==(Px-1))
                {TT=T4;
                 for(int i=0;i<=n;i++)
                 {for(int j=0;j<=m;j++)
                  {k1=i*(m+1)+j;
                   A[ip][k1]=poly2.dpowx(x,y,i,j)+H4*poly2.pow(x,y,i,j);
```

```
                B[ip]=H4*TT;
               }
              }
             }
             else if(ipy==0)
             {TT=T1;
              for(int i=0;i<=n;i++)
              {for(int j=0;j<=m;j++)
               {k1=i*(m+1)+j;
                A[ip][k1]=-poly2.dpowy(x,y,i,j)+H1*poly2.pow(x,y,i,j);
                B[ip]=H1*TT;
               }
              }
             }
             else if(ipy==(Py-1))
             {TT=T2;
              for(int i=0;i<=n;i++)
              {for(int j=0;j<=m;j++)
               {k1=i*(m+1)+j;
                A[ip][k1]=poly2.dpowy(x,y,i,j)+H2*poly2.pow(x,y,i,j);
                B[ip]=H2*TT;
               }
              }
             }
             else
             {for(int i=0;i<=n;i++)
              {for(int j=0;j<=m;j++)
               {k1=i*(m+1)+j;
                A[ip][k1]=poly2.d2powx2(x,y,i,j)+poly2.d2powy2(x,y,i,j);
                        B[ip]=0.0;
               }
              }
             }
 }
}
Text.print(A,"A");
Text.printT(B,"B");
SVD q=new SVD(A);
double a[]=q.solve(B);
Text.printT(a,"a");
double T[][]=new double[Px][Py];
poly2 p=new poly2();
for(int ipx=0;ipx<Px;ipx++)
{for(int ipy=0;ipy<Py;ipy++)
 {   x=dx*ipx;y=dy*ipy;
     ip=ipx*Py+ipy;
     T[ipy][ipx]=p.func(a,x,y,n,m);
  } // end int ipy=0;ipy<Py;ipy++)
}//end for(int ipx=0;ipx<Px;ipx++)
 Text.print(T,"T");
    double b[][]=surfacePlot.dataformat(0.0,L,0.0,W,T);
    surfacePlot p1;
    p1=new surfacePlot(b,"x m","y m","T degree C");
    p1.plot();
}
public static void main(String arg[])
{HT_poly2_hTinf h=new HT_poly2_hTinf(50,50,50,50,50,100,100.0,100.0,500.0,7,7,80,80,1.0,1.0);

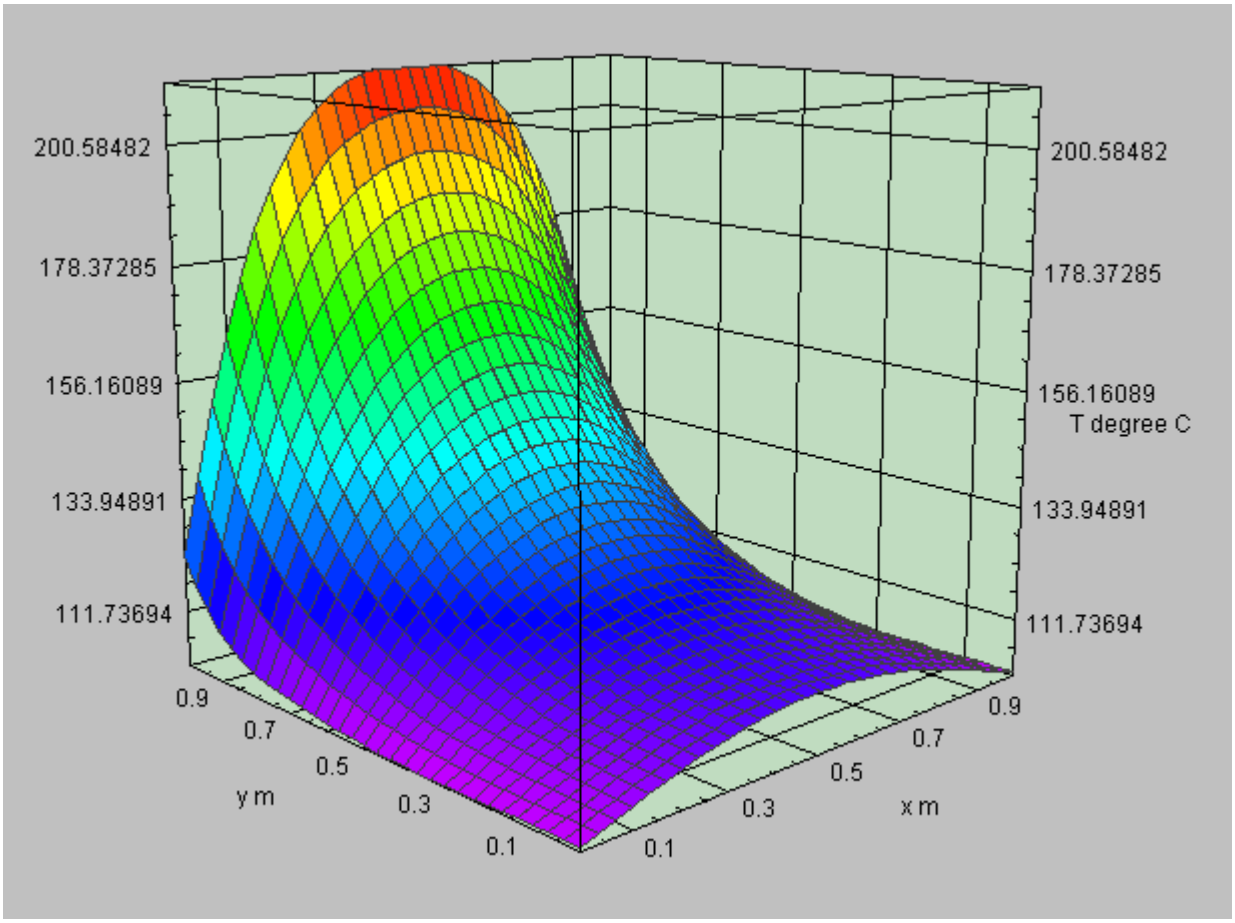
}
}
```



Now let us consider the following problem:
A square plate exposed to convective flux in each side.
x=0 $h = 2000\ W/(m^2K)$ $T_\infty = 100\ ℃$ (approaching to a constant temperature case)
x=L=1 m $h = 2000\ W/(m^2K)$ $T_\infty = 100\ ℃$ (approaching to a constant temperature case)
y=0 $h = 2000\ W/(m^2K)$ $T_\infty = 100\ ℃$ (approachingto a constant temperature case)
y=W=1 m $h = 50\ W/(m^2K)$ $T_\infty = 500\ ℃$
$k = 50\ W/(mK)$ Calculate the temperature profile




```
public static void main(String arg[])
{HT_poly2_hTinf h=new HT_poly2_hTinf(50,2000,2000,2000,50,100,100.0,100.0,500.0,7,7,80,80,1.0,1.0);
```



PROBLEM 12.5 TIME DEPENDENT LUMPED SOLUTION APPROXIMATION METHODS (LEAST SQUARE METHOD)

As a next approximation method we will consider **least square method**.

$$-hA_s(T(t)-T_\infty)=\rho V C_p\frac{dT(t)}{dt}$$

where A_s is surface area, V is volume, C_p is specific heat. By changing variables as:

$\theta(t)=\frac{T(t)-T_\infty}{T_i-T_\infty}\quad \tau=\frac{t}{\tau_{max}}\quad X=\frac{\rho V C_p}{\tau_{max}hA_s}$

equation becomes:

$$X\frac{d\theta(t)}{dt}+\theta(t)=0$$

Boundary condition t=0 θ =1so exact solution is:

$$\theta(t)=exp\left(-\frac{\tau}{X}\right)$$

Now let us create approximate solution, for this the following polynomial is assumed:

$$\theta(t)=1+c_1\tau+c_2\tau^2+c_3\tau^3+c_4\tau^4$$

This equation automatically satisfy the boundary condition but if it is applied to differential equation a residue is remained as:

$$\frac{d\theta(t)}{dt}=c_1\tau+2c_2\tau\ +3c_3\tau^2+4c_4\tau^3$$

$$X\frac{d\theta(t)}{dt}+\theta(t)=0$$

$$R=X(c_1\tau+2c_2\tau\ +3c_3\tau^2+4c_4\tau^3)+(1+c_1\tau+c_2\tau^2+c_3\tau^3+c_4\tau^4)$$

$$R=1+(X+\tau)c_1+(2\tau X+\tau^2)c_2+(3\tau^2X+\tau^3)c_3+(4\tau^3+\tau^4)c_4$$

It is possible to write least square equations as:

$$\frac{1}{2}\frac{\partial}{\partial c_i}\int_{\tau=0}^{\tau=1} R^2(\tau, c_1, c_2, c_3, c_4)d\tau = 0 \qquad i = 1,2,3,4$$

Considering R as

$$R = 1 + \phi_1 c_1 + \phi_2 c_2 + +\phi_3 c_3 + +\phi_4 c_4 = 1 + \sum_{i=1}^{4} \phi_i c_i$$

$$\frac{\partial R}{\partial c_i} = \phi_i \;\; i = 1,2,3,4$$

$$\int_{\tau=0}^{\tau=1} \phi_i \left(1 + \sum_{i=1}^{4} \phi_i c_i\right) d\tau = 0 \quad i = 1,2,3,4$$

So

$$\begin{bmatrix} \int \phi_1\phi_1 d\tau & \int \phi_1\phi_2 d\tau & \int \phi_1\phi_3 d\tau & \int \phi_1\phi_4 d\tau \\ \int \phi_2\phi_1 d\tau & \int \phi_2\phi_2 d\tau & \int \phi_2\phi_3 d\tau & \int \phi_2\phi_4 d\tau \\ \int \phi_3\phi_1 d\tau & \int \phi_3\phi_2 d\tau & \int \phi_3\phi_3 d\tau & \int \phi_3\phi_4 d\tau \\ \int \phi_4\phi_1 d\tau & \int \phi_4\phi_2 d\tau & \int \phi_4\phi_3 d\tau & \int \phi_4\phi_4 d\tau \end{bmatrix} \begin{Bmatrix} c_1 \\ c_2 \\ c_3 \\ c_4 \end{Bmatrix} = \begin{Bmatrix} -\int \phi_1 d\tau \\ -\int \phi_2 d\tau \\ -\int \phi_3 d\tau \\ -\int \phi_4 d\tau \end{Bmatrix}$$

Each integration should be calculated and carried out into the matrix to determine the coefficients.

$$\int_{\tau=0}^{\tau=0} \varphi_1\varphi_1 d\tau = \int_{\tau=0}^{\tau=1} (X^2 + 2X\tau + \tau^2)d\tau = \left(X^2\tau + X\tau^2 + \frac{\tau^3}{3}\right)\Big|_{\tau=0}^{\tau=1} = X^2 + X + \frac{1}{3}$$

$$\int_{\tau=0}^{\tau=0} \varphi_1\varphi_2 d\tau = \int_{\tau=0}^{\tau=1} (X+\tau)^*(2X\tau + \tau^2)d\tau = \int_{\tau=0}^{\tau=1} (2X^2\tau + 3X\tau^2 + \tau^3)d\tau$$

$$= \left(X^2\tau^2 + X\tau^3 + \frac{\tau^4}{4}\right)\Big|_{\tau=0}^{\tau=1} = X^2 + X + \frac{1}{4}$$

$$\int_{\tau=0}^{\tau=0} \varphi_1\varphi_3 d\tau = \int_{\tau=0}^{\tau=1} (X+\tau)^*(3X\tau^2 + \tau^3)d\tau = \int_{\tau=0}^{\tau=1} (3X^2\tau^2 + 4X\tau^3 + \tau^4)d\tau$$

$$= \left(X^2\tau^3 + X\tau^4 + \frac{\tau^5}{5}\right)\Big|_{\tau=0}^{\tau=1} = X^2 + X + \frac{1}{5}$$

$$\int_{\tau=0}^{\tau=0} \varphi_1\varphi_4 d\tau = \int_{\tau=0}^{\tau=1} (X+\tau)^*(3X\tau^3 + \tau^4)d\tau = \int_{\tau=0}^{\tau=1} (4X^2\tau^2 + 5X\tau^4 + \tau^5)d\tau$$

$$= \left(X^2\tau 4^3 + X\tau^5 + \frac{\tau^6}{6}\right)\Big|_{\tau=0}^{\tau=1} = X^2 + X + \frac{1}{6}$$

$$\int_{\tau=0}^{\tau=0} \varphi_2\varphi_2 d\tau = \int_{\tau=0}^{\tau=1} (2X\tau + \tau^2)*(2X\tau + \tau^2)d\tau = \int_{\tau=0}^{\tau=1} (4X^2\tau^2 + 4X\tau^3 + \tau^4)d\tau$$

$$= \left(\frac{4}{3}X^2\tau^2 + X\tau^4 + \frac{\tau^5}{5}\right)\Big|_{\tau=0}^{\tau=1} = \frac{4}{3}X^2 + X + \frac{1}{5}$$

$$\int_{\tau=0}^{\tau=0} \varphi_2\varphi_3 d\tau = \int_{\tau=0}^{\tau=1} (2X\tau + \tau^2)*(3X\tau^2 + \tau^3)d\tau = \int_{\tau=0}^{\tau=1} (6X^2\tau^3 + 5X\tau^4 + \tau^5)d\tau$$

$$= \left(\frac{6}{4}X^2\tau^4 + X\tau^5 + \frac{\tau^6}{6}\right)\Big|_{\tau=0}^{\tau=1} = \frac{6}{4}X^2 + X + \frac{1}{6}$$

$$\int_{\tau=0}^{\tau=0} \varphi_2\varphi_4 d\tau = \int_{\tau=0}^{\tau=1} (2X\tau + \tau^2)*(4X\tau^3 + \tau^4)d\tau = \int_{\tau=0}^{\tau=1} (8X^2\tau^4 + 5X\tau^5 + \tau^6)d\tau$$

$$= \left(\frac{8}{5}X^2\tau^5 + X\tau^6 + \frac{\tau^7}{7}\right)\Big|_{\tau=0}^{\tau=1} = \frac{8}{5}X^2 + X + \frac{1}{7}$$

$$\int_{\tau=0}^{\tau=0} \varphi_3\varphi_3 d\tau = \int_{\tau=0}^{\tau=1} (3X\tau^2 + \tau^3)*(3X\tau^2 + \tau^3)d\tau = \int_{\tau=0}^{\tau=1} (9X^2\tau^4 + 6X\tau^5 + \tau^6)d\tau$$

$$= \left(\frac{9}{5}X^2\tau^5 + X\tau^6 + \frac{\tau^7}{7}\right)\Big|_{\tau=0}^{\tau=1} = \frac{9}{5}X^2 + X + \frac{1}{7}$$

$$\int_{\tau=0}^{\tau=0} \varphi_3\varphi_4 d\tau = \int_{\tau=0}^{\tau=1} (3X\tau^2 + \tau^3)*(4X\tau^3 + \tau^4)d\tau = \int_{\tau=0}^{\tau=1} (12X^2\tau^5 + 7X\tau^6 + \tau^7)d\tau$$

$$= \left(\frac{12}{6}X^2\tau^6 + X\tau^7 + \frac{\tau^8}{8}\right)\Big|_{\tau=0}^{\tau=1} = 2X^2 + X + \frac{1}{8}$$

$$\int_{\tau=0}^{\tau=0} \varphi_4\varphi_4 d\tau = \int_{\tau=0}^{\tau=1} (4X\tau 3 + \tau^4)*(4X\tau^3 + \tau^4)d\tau = \int_{\tau=0}^{\tau=1} (16X^2\tau^6 + 8X\tau 7 + \tau^8)d\tau$$

$$= \left(\frac{16}{7}X^2\tau^7 + X\tau^8 + \frac{\tau 9}{8}\right)\Big|_{\tau=0}^{\tau=1} = \frac{16}{7}X^2 + X + \frac{1}{9}$$

$$-\int_{\tau=0}^{\tau=0} \varphi_1 d\tau = \int_{\tau=0}^{\tau=1} (X + \tau)d\tau = \left(X\tau + \frac{\tau^2}{2}\right)\Big|_{\tau=0}^{\tau=1} = -\left(X + \frac{1}{2}\right)$$

$$-\int_{\tau=0}^{\tau=0} \varphi_2 d\tau = \int_{\tau=0}^{\tau=1} (2X\tau + \tau^2)d\tau = \left(X\tau^2 + \frac{\tau^3}{3}\right)\Big|_{\tau=0}^{\tau=1} = -\left(X + \frac{1}{3}\right)$$

$$-\int_{\tau=0}^{\tau=0} \varphi_3 d\tau = \int_{\tau=0}^{\tau=1} (3X\tau^2 + \tau^3)d\tau = \left(X\tau^3 + \frac{\tau^4}{4}\right)\Big|_{\tau=0}^{\tau=1} = -\left(X + \frac{1}{4}\right)$$

$$-\int_{\tau=0}^{\tau=0} \varphi_4 d\tau = \int_{\tau=0}^{\tau=1} (3X\tau^4 + \tau^4)d\tau = \left(X\tau^4 + \frac{\tau^5}{5}\right)\Big|_{\tau=0}^{\tau=1} = -\left(X + \frac{1}{5}\right)$$

Now that all coefficients of the matrix is established, solution can be done by using a small application program **residue_least_square.java**

```
import java.io.*;

class residue_least_square
{
```

```
public static double[] pivotlugauss(double a[][],double b[])
{ //gauss elimination with partial pivoting
int n=b.length;
double x[]=new double[n];
double carpan=0;
double toplam=0;
double buyuk;
double dummy=0;
//gauss eleme
int i,j,k,p,ii,jj;
System.out.println("Original matrix :\n"+Matrix.toString(a));
for(k=0;k<(n-1);k++)
{ //pivoting
            p=k;
            buyuk=Math.abs(a[k][k]);
            for(ii=k+1;ii<n;ii++)
            {  dummy=Math.abs(a[ii][k]);
              if(dummy > buyuk) {buyuk=dummy;p=ii;}
            }
            if(p!=k)
            { for(jj=k;jj<n;jj++)
              { dummy=a[p][jj];
                a[p][jj]=a[k][jj];
                a[k][jj]=dummy;
              }
              dummy=b[p];
              b[p]=b[k];
              b[k]=dummy;
            }
              System.out.println("pivoted matrix : k="+k+"\n"+Matrix.toString(a));
            //Solving gauss elimination
  for(i=k+1;i<n;i++)
  {  carpan=a[i][k]/a[k][k];
     a[i][k]=0;
     for(j=k+1;j<n;j++)
     {   a[i][j]-=carpan*a[k][j]; }
        b[i]   =b[i]   -carpan*b[k];
            }
}
System.out.println("final upper triangular matrix : \n"+Matrix.toString(a));
//back substitution
x[n-1]=b[n-1]/a[n-1][n-1];
for(i=n-2;i>=0;i--)
{
  toplam=0;
  for(j=i+1;j<n;j++)
  {  toplam+=a[i][j]*x[j];}
  x[i]=(b[i]-toplam)/a[i][i];
}
return x;
}
public static double[] c(double X)
{  double a[][]=new double[4][4];
   a[0][0]=(X*X+X+1.0/3.0);
   a[1][0]=(X*X+X+1.0/4.0);
   a[2][0]=(X*X+X+1.0/5.0);
   a[3][0]=(X*X+X+1.0/6.0);
   a[0][1]=a[1][0];
   a[1][1]=(4.0/3.0*X*X+X+1.0/5.0);
   a[2][1]=(6.0/4.0*X*X+X+1.0/6.0);
   a[3][1]=(8.0/5.0*X*X+X+1.0/7.0);
   a[0][2]=a[2][0];
   a[1][2]=a[2][1];
             a[2][2]=(9.0/5.0*X*X+X+1.0/7.0);
```

```
            a[3][2]=(2.0*X*X+X+1.0/8.0);
    a[0][3]=a[3][0];
    a[1][3]=a[3][1];
    a[2][3]=a[3][2];
    a[3][3]=(16.0/7.0*X*X+X+1.0/9.0);;
            double b[]={-(X+1.0/2.0),-(X+1.0/3.0),-(X+1.0/4.0),-(X+1.0/5.0)};
            double c[]=pivotlugauss(a,b);
            return c;
  }

  public static void main(String args[]) throws IOException
  {
  double toi[]={0.25,0.5,0.75,1.0};
  double ro=8500; //kg/m^3
  double Cp=400; //J/kgK
  double h=80; //W/m2K
  double D=0.02;
  double k=20; //W/mK
  double V=Math.PI*D*D*D/6;
  double A=Math.PI*D*D;
  double Bi=h*D/6/k;
  System.out.println("Bi="+Bi);
  double tomax=240;
  double X=ro*V*Cp/(tomax*h*A);
  double c[]=c(X);
  String c1[]={"c1","c2","c3","c4"};
  Text.print(c,c1,"Least square method for approximation");
  if_x f=to->Math.exp(-to/X);
  if_x f1=to->1+c[0]*to+c[1]*to*to+c[2]*to*to*to+c[3]*to*to*to*to;
  Plot pp=new Plot(f,0,1);
  pp.addFunction(f1,0,1);
  pp.plot();
  }
}
```

```
---------- Capture Output ----------
> "C:\java\bin\javaw.exe" residue_least_square
Bi=0.013333333333333332
Original matrix :
      1.272038966049383      1.188705632716049      1.138705632716049      1.105372299382716
      1.188705632716049      1.254848251028807      1.279586226851852      1.290619488536155
      1.138705632716049      1.279586226851852      1.360305059523810      1.412133487654321
      1.105372299382716      1.290619488536155      1.412133487654321      1.497795414462081

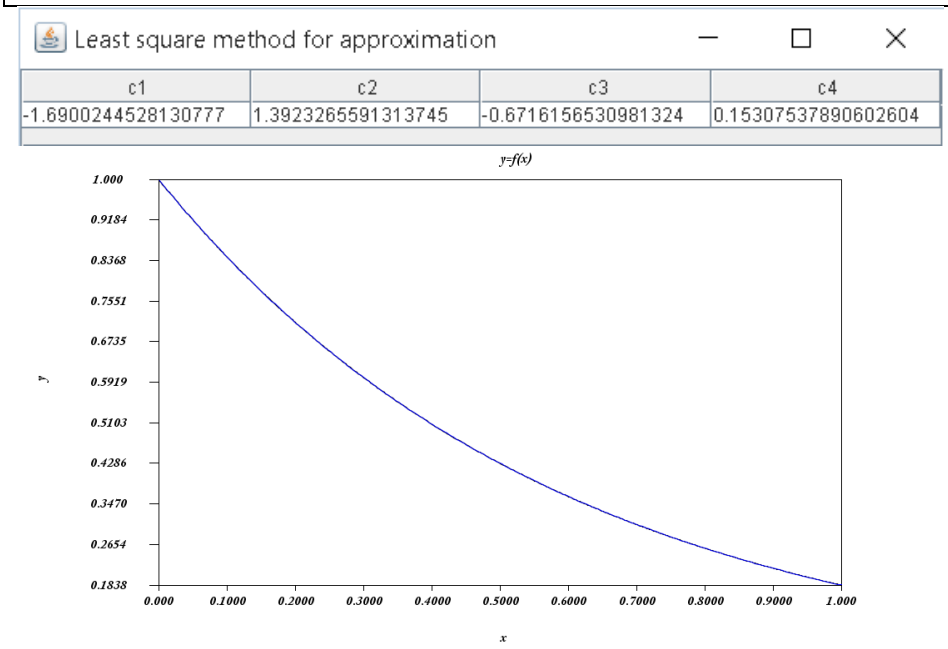
pivoted matrix : k=0
      1.272038966049383      1.188705632716049      1.138705632716049      1.105372299382716
      1.188705632716049      1.254848251028807      1.279586226851852      1.290619488536155
      1.138705632716049      1.279586226851852      1.360305059523810      1.412133487654321
      1.105372299382716      1.290619488536155      1.412133487654321      1.497795414462081

pivoted matrix : k=1
      1.272038966049383      1.188705632716049      1.138705632716049      1.105372299382716
      0.000000000000000      0.257661919158115      0.422624756279086      0.537252575088716
      0.000000000000000      0.215479044806210      0.340956947880412      0.422624756279086
      0.000000000000000      0.144016649981762      0.215479044806210      0.257661919158115

pivoted matrix : k=2
      1.272038966049383      1.188705632716049      1.138705632716049      1.105372299382716
      0.000000000000000      0.257661919158115      0.422624756279086      0.537252575088716
      0.000000000000000      0.000000000000000     -0.020741355156435     -0.042628152076166
      0.000000000000000      0.000000000000000     -0.012478201139771     -0.026672027964582

final upper triangular matrix :
      1.272038966049383      1.188705632716049      1.138705632716049      1.105372299382716
```

0.000000000000000	0.257661919158115	0.422624756279086	0.537252575088716
0.000000000000000	0.000000000000000	-0.020741355156435	-0.042628152076166
0.000000000000000	0.000000000000000	0.000000000000000	-0.001026516771543



The last method we will look for the approximation is **Galarkin Method**. This Method multiplies residue with a weight function and integrates similar to least square method

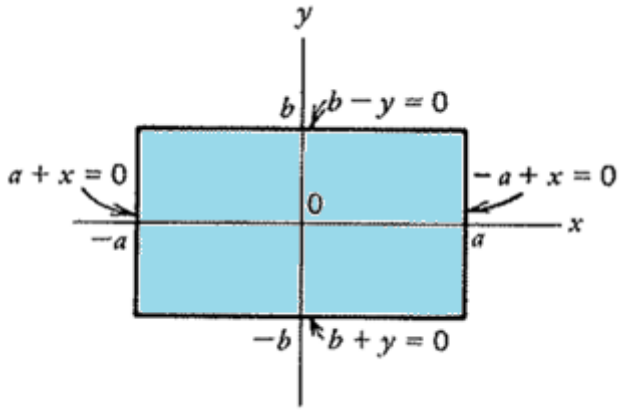
$$\int_{\tau=0}^{\tau=1} w_i R(\tau, c_1, c_2, c_3, c_4) d\tau = 0 \quad i = 1..4$$

$w_i$is called weight function. One possible way is defining the weight function in here is to utilised derivatives of residue with respect to c_i

$$w_i = \frac{\partial R(\tau, c_1, c_2, c_3, c_4)}{\partial c_i} \quad i = 1..4$$

in this case the results will be same as the previous least square case. Other considerations of weight functions are also possible. In fact selection of the weight factor is one of the most important criteria in solution. In general, weight functions should be selected so that it should satisfy the boundary conditions. As an example let us consider the following problem with a rectangular region:

PROBLEM 12.6 TWO DIMENSIONAL STEADY-STATE PROBLEM APPROXIMATION WITH GALARKIN METHOD



For differential equation: $\frac{\partial^2 T(x,y)}{\partial x^2} + \frac{\partial^2 T(x,y)}{\partial y} + \frac{1}{k} g(x,y) = 0$

Boundary conditions:

$T(x,y) = 0 \ \ at \ x = -a \qquad T(x,y) = 0 \ \ at \ y = -b$

$T(x,y) = 0 \ \ at \ x = \ \ a \qquad T(x,y) = 0 \ \ at \ y = \ b$

Where

$$\bar{T}(x,y) = \phi_0(x,y) + \sum_{j=1}^{n} C_j \, \phi_j(x,y)$$

when this function is substituted into the differential equation

$$\frac{\partial^2 \bar{T}(x,y)}{\partial x^2} + \frac{\partial^2 \bar{T}(x,y)}{\partial y^2} + \frac{1}{k} g(x,y) = R(C_1, C_2, C_3, C_4)$$

Then the Galarkin method to evalute the problem will be

$$\int_{x=-a}^{a} \int_{y=-b}^{b} \left[\frac{\partial^2 \bar{T}(x,y)}{\partial x^2} + \frac{\partial^2 \bar{T}(x,y)}{\partial y^2} + \frac{1}{k} g(x,y) \right] dxdy = \int_{x=-a}^{a} \int_{y=-b}^{b} [R(C_1, C_2, C_3, C_4)] \, dxdy$$

Before going into the details of approximate solution, let us solve the differential equation with exact solution so that the results can be compared. This equation can be solved by using separation of variable methods. In order to homogenize the equation, system is divided up into two equation as:

$$\bar{T}(x,y) = \theta(x,y) + \psi(x)$$

And

$$\frac{\partial^2 \psi(x)}{\partial x^2} + \frac{g}{k} = 0$$

Boundary conditions:

$$\psi(x) = 0 \ \ at \ \ x = -a$$
$$\psi(x) = 0 \ \ at \ \ x = \ \ a$$

$$\frac{\partial^2 \theta(x,y)}{\partial x^2} + \frac{\partial^2 \theta(x,y)}{\partial y^2} = 0$$

Boundary conditions:

$$\theta(x,y) = 0 \ \ at \ \ x = -a$$
$$\theta(x,y) = 0 \ \ at \ \ x = \ \ a$$
$$\theta(x,y) = -\psi(x) \ \ at \ \ y = -b$$
$$\theta(x,y) = -\psi(x) \ \ at \ \ x = \ \ b$$

For g(x,y)=g=const Solution of $\psi(x)$

$$\psi(x) = -\frac{g}{k}\frac{x^2}{2} + C_1 x + C_2$$

applying boundary conditions $C_2 = \frac{g}{k}\frac{a^2}{2} \quad C_1 = 0$

$$\psi(x) = \frac{g}{2k}(a^2 - x^2)$$

Solution of $\theta(x,y)$

$$\theta(x,y) = X(x)Y(y)$$

$$\frac{1}{X(x)}\frac{\partial^2 X(x)}{\partial x^2} = -\frac{1}{Y(y)}\frac{\partial^2 Y(y)}{\partial y^2} = -\beta^2$$

$$\frac{\partial^2 X(x)}{\partial x^2} + \beta^2 X(x) = 0$$

$$\frac{\partial^2 Y(y)}{\partial y^2} - \beta^2 Y(y) = 0$$

$$X(x) = C_1 \cos(\beta x) + C_2 \sin(\beta x)$$
$$Y(x) = C_3 \cos h(\beta y) + C_4 \sinh(\beta y)$$

Boundary conditions:

$$X(x) = 0 \ \ at\ x = -a$$
$$X(x) = 0 \ \ at\ x = \ \ a$$
$$X(x) = C_1 \cos(-\beta a) + C_2 \sin(-\beta a) = 0$$
$$X(x) = C_1 \cos(\beta a) + C_2 \sin(\beta a) = 0$$

Considering that $\cos(-\beta a) = \cos(\beta a)$ and $\sin(-\beta a) = -\sin(\beta a)$

$$X(x) = C_1 \cos(\beta a) - C_2 \sin(\beta a) = 0$$
$$X(x) = C_1 \cos(\beta a) + C_2 \sin(\beta a) = 0$$

$$C_1 \cos(\beta a) = 0$$
$$\beta a = (2n+1)\frac{\pi}{2}$$
$$\beta_n = (2n+1)\frac{\pi}{2a}$$
$$C_2 = 0$$

$$\theta(x,y) = \bar{T}(x,y) - \psi(x)$$
$$\theta(x,y) = \sum_{n=0}^{\infty} [C_3 \cos h(\beta_n y) + C_4 \sinh(\beta_n y)] \cos(\beta_n x)$$

Boundary conditions:

$$-\psi(x) = \sum_{n=0}^{\infty} [C_3 \cos h(-\beta_n b) + C_4 \sinh(-\beta_n b)] \cos(\beta_n x)$$
$$-\psi(x) = \sum_{n=0}^{\infty} [C_3 \cos h(\beta_n b) + C_4 \sinh(\beta_n b)] \cos(\beta_n x)$$

Considering that $\cosh(-\beta a) = \cos h(\beta a)$ and $\sinh(-\beta a) = -\sin h(\beta a)$

$$-\psi(x) = \sum_{n=0}^{\infty} [C_3 \cos h(\beta_n b) - C_4 \sinh(\beta_n b)] \cos(\beta_n x)$$
$$-\psi(x) = \sum_{n=0}^{\infty} [C_3 \cos h(\beta_n b) + C_4 \sinh(\beta_n b)] \cos(\beta_n x)$$

So $C_4 = 0$

$$-\psi(x) = \sum_{n=0}^{\infty} C_3 \cos h(\beta_n b) \cos(\beta_n x)$$
$$-\frac{\psi(x)}{\cos h(\beta_n b)} = \sum_{n=0}^{\infty} C_3 \cos(\beta_n x)$$

It is known that $\cos(\beta_n x)$is an orthogonal function so orthogonality can be applied.

$$-\int_{x=-a}^{a} \frac{\psi(x) cos(\beta_n x)}{\cos h(\beta_n b)} dx = C_n \int_{x=-a}^{a} \cos^2(\beta_n x) dx$$

To solve the integral variable change is aplied as

$\bar{x} = \frac{x}{a} \quad \psi(\bar{x}) = \frac{ga^2}{2k}(1-\bar{x}^2) \quad \bar{\beta}_n = (2n+1)\frac{\pi}{2}$

$$-\int_{\bar{x}=-1}^{1} \frac{\psi(\bar{x}) cos(\bar{\beta}_n \bar{x})}{\cos h\left(\bar{\beta}_n \frac{b}{a}\right)} d\bar{x} = C_n \int_{\bar{x}=-1}^{1} \cos^2(\bar{\beta}_n \bar{x}) d\bar{x}$$

$$\int_{\bar{x}=-1}^{1} \cos^2(\bar{\beta}_n\bar{x})d\bar{x} = \int_{\bar{x}=-1}^{1} \frac{1+\cos(2\bar{\beta}_n\bar{x})}{2} d\bar{x} = \left[\frac{x}{2} + \frac{1}{2\bar{\beta}_n}\sin(2\bar{\beta}_n)\right]_{-1}^{1} = 1$$

$$\frac{ga^2}{2k\cos h\left(\bar{\beta}_n\frac{b}{a}\right)} \int_{\bar{x}=-1}^{1} (1-\bar{x}^2)cos(\bar{\beta}_n\bar{x})d\bar{x}$$

$$\int_{\bar{x}=-1}^{1} \left[cos(\bar{\beta}_n\bar{x}) - \bar{x}^2 cos(\bar{\beta}_n\bar{x})\right]d\bar{x} = \left[\frac{sin(\bar{\beta}_n\bar{x})}{\bar{\beta}_n} - x^2\frac{sin(\bar{\beta}_n\bar{x})}{\bar{\beta}_n} - 2x\frac{cos(\bar{\beta}_n\bar{x})}{\bar{\beta}_n{}^2} + 2\frac{cos(\bar{\beta}_n\bar{x})}{\bar{\beta}_n{}^3}\right]_{-1}^{1}$$

$$\int_{\bar{x}=-1}^{1} \left[cos(\bar{\beta}_n\bar{x}) - \bar{x}^2 cos(\bar{\beta}_n\bar{x})\right]d\bar{x} = \frac{4(-1)^n}{\bar{\beta}_n{}^3}$$

$$C_n = \frac{ga^2}{2k\cos h\left(\bar{\beta}_n\frac{b}{a}\right)}\frac{4(-1)^n}{\bar{\beta}_n{}^3}$$

so the exact solution:

$$T(x,y) = \frac{g}{k}\left[\frac{a^2-x^2}{2} - 2a^2\sum_{n=0}^{\infty}\frac{(-1)^n}{\bar{\beta}_n^3}\frac{cosh\left(\bar{\beta}_n\frac{y}{a}\right)cos\left(\bar{\beta}_n\frac{x}{a}\right)}{cosh\left(\bar{\beta}_n\frac{b}{a}\right)}\right]$$

Now that exact solution formula is established, Galarkin approximation

$$\int_{x=-a}^{a}\int_{y=-b}^{b}\left[\frac{\partial^2\bar{T}(x,y)}{\partial x^2} + \frac{\partial^2\bar{T}(x,y)}{\partial y^2} + \frac{1}{k}g\right]\phi_1 dxdy = 0$$

Take

$$\varphi_1(x,y) = (a2 \quad -x^2)(b_2 - y^2)$$

$$\bar{T}_1(x,y) = C_1\varphi_1(x,y) = C_1(a_2 - x^2)(b_2 - y^2)$$

$$\frac{\partial^2\bar{T}_1(x,y)}{\partial x^2} = -2C_1(b2 \quad -y^2)$$

$$\frac{\partial^2\bar{T}_1(x,y)}{\partial y^2} = -2C_1(a2 \quad -x^2)$$

$$\int_{x=-a}^{a}\int_{y=-b}^{b}\left[-2C_1((a^2-x^2)+(b^2-y^2)) + \frac{g}{k}\right](a2 \quad -x^2)(b2 \quad -y^2)dxdy = 0$$

$$\int_{x=-a}^{a}\int_{y=-b}^{b}\left[-2C_1((a2 \quad -x^2)^2(b2 \quad -y^2) + (a2 \quad -x^2)(b2 \quad -y^2)^2) + \frac{g}{k}(a2 \quad -x^2)(b2 \quad -y^2)\right]dxdy = 0$$

$$\int_{x=-a}^{a}\int_{y=-b}^{b}\left[-2C_1((a4 \quad -2a^2x^2 + x^4)(b2 \quad -y^2) + (a2 \quad -x^2)(b4 \quad -2b^2y^2 + y^4)) + \frac{g}{k}(a2 \quad -x^2)(b2 \quad -y^2)\right]dxdy = 0$$

Variable conversion: $\bar{x} = \frac{x}{a}$ and $\bar{y} = \frac{y}{b}$

$$\int_{\bar{x}=-1}^{1}\int_{\bar{y}=-1}^{1}\left[-2C_1(a2 \quad (1-2\bar{x}^2+\bar{x}^4)(1-\bar{y}^2) + (1-2\bar{y}^2+\vec{y}^4)(1-\bar{x}^2)) + \frac{g}{k}(1-\bar{x}^2)(1-\bar{y}^2)\right]d\bar{x}d\bar{y} = 0$$

$$\int_{\bar{x}=-1}^{1}\int_{\bar{y}=-1}^{1}\left[-2C_1(a2\quad(1-2\bar{x}^2+\bar{x}^4)(1-\bar{y}^2)+(1-2\bar{y}^2+\vec{y}^4)(1-\bar{x}^2))+\frac{g}{k}(1-\bar{x}^2)(1\right.$$

$$\left.-\bar{y}^2)\right]d\bar{x}d\bar{y}=0$$

$$-2C_1\left(a^2\,(\bar{x}-\frac{2}{3}\bar{x}^3+\frac{1}{5}\bar{x}^5)\Big|_{-1}^{1}(\bar{y}-\frac{1}{3}\bar{y}^3)\Big|_{-1}^{1}\right\}+b^2\,(\bar{y}-\frac{2}{3}\bar{y}^3+\frac{1}{5}\vec{y}^5)\Big|_{-1}^{1}(\bar{x}-\frac{1}{3}\bar{x}^3)\Big|_{-1}^{1}\Big)$$

$$+\frac{g}{k}(\bar{x}-\frac{1}{3}\bar{x}^3)\Big|_{-1}^{1}(\bar{y}-\frac{1}{3}\bar{y}^3)\Big|_{-1}^{1}=0$$

$$-2C_1\left(a^2\left[\left(1-\frac{2}{3}+\frac{1}{5}\right)-\left(-1+\frac{2}{3}-\frac{1}{5}\right)\right]\left[\left(1-\frac{1}{3}\right)-\left(-1+\frac{1}{3}\right)\right]\right.$$

$$\left.+b^2\left[\left(1-\frac{2}{3}+\frac{1}{5}\right)-\left(-1+\frac{2}{3}-\frac{1}{5}\right)\right]\left[\left(1-\frac{1}{3}\right)-\left(-1+\frac{1}{3}\right)\right]\right)+$$

$$\frac{g}{k}\left[\left(1-\frac{1}{3}\right)-\left(-1+\frac{1}{3}\right)\right]\left[\left(1-\frac{1}{3}\right)-\left(-1+\frac{1}{3}\right)\right]=0$$

$$-2C_1\left(a^2\frac{16}{15}\frac{4}{3}+b^2\frac{16}{15}\frac{4}{3}\right)+\frac{g}{k}\frac{4}{3}\frac{4}{3}=0$$

$$C_1=\frac{5}{8}\frac{g/k}{a^2+b^2}$$

$$\bar{T}_1(x,y)=\frac{5}{8}\frac{g/k}{a^2+b^2}(a2\quad-x^2)(b2\quad-y^2)$$

Now let us apply a two coefficient approximation

$$\varphi_1=(a2\quad-x^2)(b2\quad-y^2)$$

$$\varphi_2=x^2(a2\quad-x^2)(b2\quad-y^2)$$

$$T_2(x,y)=C_1\varphi_1+C_2\varphi_2$$

$$T_2(x,y)=C_1(a2\quad-x^2)(b2\quad-y^2)+C_2(a^{\,2}x^2-x^4)(b2\quad-y^2)$$

$$\frac{\partial T_2(x,y)}{\partial x}=-2C_1x(b2\quad-y^2)+C_2(2a^2x-4x^3)(b2\quad-y^2)$$

$$\frac{\partial^2 T_2(x,y)}{\partial x^2}=-2C_1(b2\quad-y^2)+C_2(2a^2-12x^2)(b2\quad-y^2)$$

$$\frac{\partial T_2(x,y)}{\partial y}=-2C_1(a^2-x^2)y-2C_2(a^2x^2-x^4)y$$

$$\frac{\partial^2 T_2(x,y)}{\partial y^2}=-2C_1(a^2-x^2)-2C_2 2\quad(a^2x^2-x^4)$$

$$\int_{x=-a}^{a}\int_{y=-b}^{b}\left[\frac{\partial^2 T(x,y)}{\partial x^2}+\frac{\partial T(x,y)}{\partial y^2}+\frac{1}{k}g\right]\phi_1(x,y)dxdx=0$$

$$\int_{x=-a}^{a}\int_{y=-b}^{b}\left[\frac{\partial^2 T(x,y)}{\partial x^2}+\frac{\partial T(x,y)}{\partial y^2}+\frac{1}{k}g\right]\phi_2(x,y)dxdx=0$$

$$\int_{\bar{x}=-a}^{a}\int_{\bar{y}=-a}^{a}\left[(-2C_1(a^2-x^2)+C_2(2a^4-14a^2x^2+12x^4))(b^4-2b^2y^2+y^4)\right.$$

$$+((a^4-2a^2x^2+x^4)C_1+(a^4x^2-2a^2x^4+x^6)C_2)((-2)(b^2-y^2))+\frac{g}{k}(a^2$$

$$\left.-x^2)(b^2-y^2)\right]dxdy=0$$

$$\int_{\bar{x}=-a}^{a}\int_{\bar{y}=-a}^{a}\Big[(-2C_1(a^2x^4-x^2)+C_2(2a^4x^2-14a^2x^4+12x6))(b^4-2b^2y^2+y^4)$$
$$+((a^4x^2-2a^2x^4+x^6)C_1+(a^4x^4-2a^2x^6+x^8)C_2)((-2)(b^2-y^2))+\frac{g}{k}(a^2x^2$$
$$-x^4)(b^2-y^2)\Big]dxdy=0$$

$$\left(-2C_1(a^2x-\frac{x^3}{3})+C_2(2a^4x-\frac{14}{3}a^2x^3+\frac{12}{5}x^5)\right)\Big|_{-a}^{a}(b^4x-2b^2\frac{y^3}{3}+\frac{y^5}{5})\Big|_{-b}^{b}+$$
$$\left((a^4-\frac{2}{3}a^2x^3+\frac{1}{5}x^5)C_1+(a^4\frac{x^3}{3}-\frac{2}{5}a^2x^5+\frac{x^7}{7})C_2\right)\Big|_{-a}^{a}\left((-2)(b^2y-\frac{y^3}{3})\right)\Big|_{-b}^{b}$$
$$+\frac{g}{k}(a^2x-\frac{x^3}{3})\Big|_{-a}^{a}(b^2y-\frac{y^3}{3})\Big|_{-b}^{b}=0$$

$$\left(-2C_1(a^2\frac{x^3}{3}-\frac{x^5}{5})+C_2(2a^4\frac{x^3}{3}-\frac{14}{5}a^2x^5+\frac{12}{7}x^7)\right)\Big|_{-a}^{a}(b^4x-2b^2\frac{y^3}{3}+\frac{y^5}{5})\Big|_{-b}^{b}+$$
$$\left((a^4\frac{x^3}{3}-\frac{2}{5}a^2x^5+\frac{1}{7}x^7)C_1+(a^4\frac{x^5}{5}-\frac{2}{7}a^2x^7+\frac{x^9}{9})C_2\right)\Big|_{-a}^{a}\left((-2)(b^2y-\frac{y^3}{3})\right)\Big|_{-b}^{b}$$
$$+\frac{g}{k}(a^2\frac{x^3}{3}-\frac{x^5}{5})\Big|_{-a}^{a}(b^2y-\frac{y^3}{3})\Big|_{-b}^{b}=0$$

$$\left(-\frac{8}{3}a^3C_1-\frac{4}{15}a^5C_2\right)\frac{16}{15}b^5+\left(\frac{16}{15}a^5C_1+\frac{16}{105}a^7C_2\right)\frac{(-8)}{3}b^3+\frac{16}{9}\frac{g}{k}a^3b^3=0$$

$$\left(-\frac{8}{15}a^5C_1-\frac{88}{105}a^7C_2\right)\frac{16}{15}b^5+\left(\frac{16}{105}a^7C_1+\frac{158}{315}a^9C_2\right)\frac{(-8)}{3}b^3+\frac{16}{45}\frac{g}{k}a^5b^3=0$$

$$\begin{bmatrix}\frac{8b^2(a^2+b^2)}{5} & \left(\frac{4}{25}a^2b^2+\frac{8}{35}a^4\right)\\ 8(\frac{a^2}{7}+\frac{b^2}{5}) & \left(\frac{88}{35}a^2b^2+\frac{79}{21}a^4\right)\end{bmatrix}\begin{Bmatrix}C_1\\C_2\end{Bmatrix}=\begin{Bmatrix}\frac{g}{k}\\\frac{g}{k}\end{Bmatrix}$$

```
 class fa extends f_xj
{
double a,b,k,g;
public fa(double ai,double bi,double ki,double gi)
{a=ai;b=bi;k=ki;g=gi;}

public double func(double x[])
{
//çözümü istenen fonksiyon
double ff=0;
double sign=1.0;
double beta=1.0;
double x1,y1,y2,ffx;
for(int n=0;n<200;n++)
{   beta=(2.0*n+1.0)/2.0*PI;
   y1=Math.cosh(beta*x[1]/a);
   x1=Math.cos(beta*x[0]/a);
   y2=Math.cosh(beta*b/a);
            ff+=-2.0*a*a*sign/(beta*beta*beta)*y1*x1/y2;
            sign*=-1.0;
}
ff+=(a*a-x[0]*x[0])/2.0;
ff*=g/k;
```

```
double ff1=5.0/8.0*g/k/(a*a+b*b)*(a*a-x[0]*x[0])*(b*b-x[1]*x[1]);

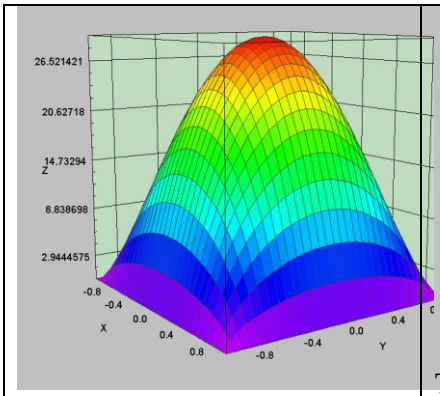
double aa[][]={{8.0/5.0*(b*b+a*a),(4.0/25.0*a*a*b*+8/35*a*a*a*a)},
            {(8.0/7.0*a*a+8.0/5.0*b*b),(88.0/35.0*a*a*b*b+79.0/21.0*a*a*a*a)}};
double bb[]={g/k,g/k};
double C[]=pivotlugauss(aa,bb);
double ff2=C[0]*(a*a-x[0]*x[0])*(b*b-x[1]*x[1])+C[1]*x[0]*x[0]*(a*a-x[0]*x[0])*(b*b-x[1]*x[1]);
return (ff-ff2); //maksimum testi
}

public static double[] pivotlugauss(double a[][],double b[])
  { //gauss elimination with partial pivoting
  int n=b.length;
  double x[]=new double[n];
  double carpan=0;
  double toplam=0;
  double buyuk;
  double dummy=0;
  //gauss eleme
  int i,j,k,p,ii,jj;
  for(k=0;k<(n-1);k++)
  { //pivoting
                p=k;
                buyuk=Math.abs(a[k][k]);
                for(ii=k+1;ii<n;ii++)
                {  dummy=Math.abs(a[ii][k]);
                  if(dummy > buyuk) {buyuk=dummy;p=ii;}
                }
                if(p!=k)
                { for(jj=k;jj<n;jj++)
                  { dummy=a[p][jj];
                    a[p][jj]=a[k][jj];
                    a[k][jj]=dummy;
                  }
                  dummy=b[p];
                  b[p]=b[k];
                  b[k]=dummy;
                }
                //Solving gauss elimination
    for(i=k+1;i<n;i++)
    {  carpan=a[i][k]/a[k][k];
       a[i][k]=0;
       for(j=k+1;j<n;j++)
       {   a[i][j]-=carpan*a[k][j]; }
          b[i]   =b[i]   -carpan*b[k];
                }
  }
  //back substitution
  x[n-1]=b[n-1]/a[n-1][n-1];
  for(i=n-2;i>=0;i--)
  {
    toplam=0;
    for(j=i+1;j<n;j++)
    {  toplam+=a[i][j]*x[j];}
    x[i]=(b[i]-toplam)/a[i][i];
  }
  return x;
  }

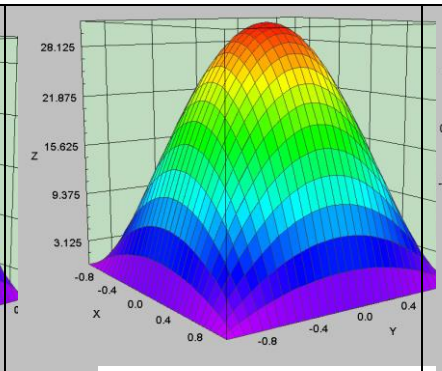
}

public class surfacePlotE4
{       public static void main(String arg[])
        {  double a=1.0;
```

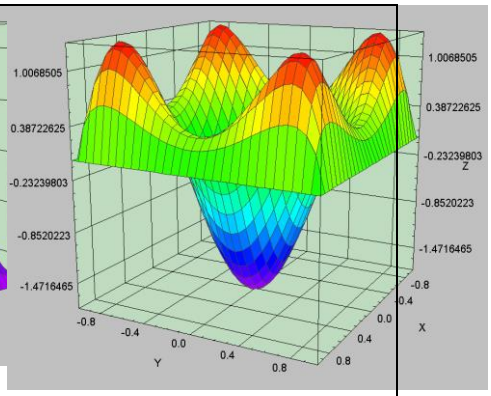
```
        double b=1.0;
        double k=100.0;
        double g=10000.0;
                fa f=new fa(a,b,k,g);
                double x[]={0,0};
                //System.out.println("T(0,0)="+f.func(x));
        surfacePlot p=new surfacePlot(f,-a,a,-b,b);
        p.plot();
      }
}
```



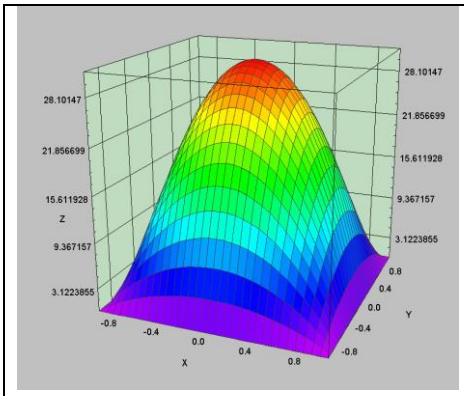
Exact solution T0



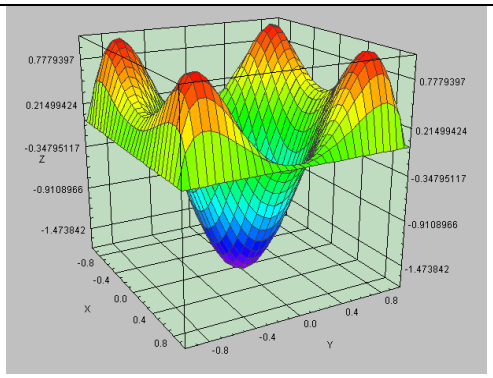
T1 Galarkin approximation



T0-T1 difference between solutions



T2 Galarkin approximation



T0-T2 difference between solutions

14. HOMEWORK PROBLEMS AND EXAMS

NUMERICAL ANALYSIS
10.5.2010 Midterm Examination

QUESTION 1
Determine the highest real root of $f(x)=0.95x^3 - 5.9x^2 +10.9x-6$ using the Newton-Raphson method with $x_0=3.5$. Carry out **3 iterations.**

QUESTION 2
Enthapy of a saturated water will be approximated by the following equation

$$h_f = a_0 + a_1 t + a_2 t^2$$

Following data is taken from the thermodynamic table. Find the polynomial coefficient by using **polinomial least square** by hand calculations

t, ^oC	10	30	50	70
h_f , kJ/kg	41.99	125.66	209.26	292.97

QUESTION 3
Perform first **two iteration** of the steepest descent method to find **minimum** of
$z = f(x,y) = -8x + x^2 + 12y + 4y^2 - 2xy$ Using initial guesses x=0 and y=0

NUMERICAL ANALYSIS FINAL EXAM
2009-2010 Spring Semester, 9/6/2010
Time : 2 hours

PROBLEM 1
Solve diferential equation
$\frac{dy}{dx} = \frac{1}{x+y}$ with initial conditions x_0=0 , y_0=2 , at the interval 0-0.4 with step size h=0.2,
with 4th order Runge-Kutta method

---------- Capture Output ----------
> "C:\turhan\java\bin\javaw.exe" NA87
k1=0.5
k2=0.46511627906976744
k3=0.465872156013001
k4=0.43607672682604176
i=0ai=2.0932684532330526
k1=0.43605884805601314
k2=0.41036174265959957
k3=0.4107949303893789
k4=0.38828506085481335
i=1ai=2.1754903617333454

```
//=====================================================
// Numerical Analysis package in java
// example to show differential equation solution
// Runge-Kutta 4rd order solution RK4
// Dr. Turhan Coban
// =====================================================
// dy/dx=f(x,y)

import java.io.*;
```

```
abstract class f_xy
{
  //single function single independent variable
  // example f=x*x
  abstract double func(double x,double y);
}

class fxy extends f_xy
{
  double func(double x,double y)
  {return 1.0/(x+y);}
}

class NAFP1
{
  public static double[][] RK4(f_xy f,double x0,double y0,double xn,double h)
  {
  // Runge-Kutta 4
  int n=(int)((xn-x0)/h);
  double a[][]=new double[2][n+1];
  int i=0;
  a[0][0]=x0;
  a[1][0]=y0;
  double k1,k2,k3,k4;
  for(double xx=x0;xx<xn;xx+=h)
   {a[0][i+1]=xx+h;
   k1=f.func(a[0][i],a[1][i]);
   k2=f.func(a[0][i]+0.5*h,a[1][i]+0.5*k1*h);
   k3=f.func(a[0][i]+0.5*h,a[1][i]+0.5*k2*h);
   k4=f.func(a[0][i]+h,a[1][i]+k3*h);
   a[1][i+1]=a[1][i]+(k1+2.0*k2+2.0*k3+k4)/6.0*h;
   System.out.println("k1="+k1);
   System.out.println("k2="+k2);
   System.out.println("k3="+k3);
   System.out.println("k4="+k4);
   System.out.println("i="+i+"ai="+a[1][i+1]);
   i++;
   }
  return a;
  }

  public static void main(String args[]) throws IOException
  {double x0=0;
   double y0=2.0;
   fxy f2=new fxy();
   double a[][]=RK4(f2,x0,y0,0.4,0.2);
   Text.print(Text.T(a));
   Plot p1=new Plot(a);
   p1.setPlotType(0,24);
   p1.plot();
  }
}
```

```
0.0  2.0
0.2  2.093268453233052
0.4  2.175490361733345
```

PROBLEM 2

Solve the following integral by n=4 Gauss-Legendre integral

$$I = \int_0^1 \frac{4dx}{1 + 256(x - 0.375)^2}$$

Coefficients:

N	x_k	c_k
4	-0.861136311594052	0.347854845137447
	-0.339981043584856	0.652145154862546
	0.339981043584856	0.652145154862546
	0.861136311594052	0.347854845137447

---------- Capture Output ----------
> "C:\turhan\java\bin\javaw.exe" NAFP2
k1=0.5k2=0.5x=0.06943184420297399y=0.16062189039772795z=0.02793655140499282
k1=0.5k2=0.5x=0.330009478207572y=2.634730839304772z=0.887050027014761
k1=0.5k2=0.5x=0.669990521792428y=0.1718436836502664z=0.9430835398578876
k1=0.5k2=0.5x=0.930568155797026y=0.049990045864377546z=0.9517781796891711
integral of class f1 : 0.9517781796891711
> Terminated with exit code 0.

```
//=====================================================
// Numerical Analysis package in java
// example to show utilisation of integration (integral)
// and differentiation (derivative) functions
// Gauss-Legendre 10 points integration formula
// Dr. Turhan Coban
// =====================================================
import java.io.*;


class f1 extends f_x
{
  public double func(double x)
  {
  double ff=4.0/(1+256*(x-0.375)*(x-0.375));
  return ff;
  }
}

class NAFP2
{
public static double integral(f_x f_xnt,double a,double b)
{
 double r[],c[];
 int n=4;
 r=new double[n];
 c=new double[n];
 r[ 0] = -0.861136311594052;
 r[ 1] = -0.339981043584856;
 r[ 2] = 0.339981043584856;
 r[ 3] = 0.861136311594052;
 c[ 0] = 0.347854845137447;
 c[ 1] = 0.652145154862546;
 c[ 2] = 0.652145154862546;
 c[ 3] = 0.347854845137447;
double z=0;
double x,y;
double k1=(b-a)/2.0;
double k2=(b+a)/2.0;
for(int i=0;i<n;i++)
{
```

```
x=k2+k1*r[i];
y=f_xnt.func(x);
z+=k1*c[i]*y;
System.out.println("k1="+k1+"k2="+k2+"x="+x+"y="+y+"z="+z);
}
return z;
}

  public static void main(String args[]) throws IOException
  {
  //Gauss-Legendre integration formula with n=10
  f1 b1=new f1();
  System.out.println("integral of class        f1 : "+integral(b1,0.0,1.0));
}
}
```

PROBLEM 3

Enthapy of a saturated water will be approximated by the following equation

$$h_f = A + Bt + Ct^2$$

Following data is taken from the thermodynamic table. Find the polynomial coefficient by using polinomial least square

t, ^oC	10	30	50
h_f , kJ/kg	41.99	125.66	209.26

---------- Capture Output ----------
> "C:\turhan\java\bin\javaw.exe" NAFP3
katsayilar :
0.128749999999834
4.187000000000012
-0.000087500000000

Polynomial Least Square Curve fitting
10.0000000000 41.9900000000
16.6666666667 69.8877777778
23.3333333333 97.7777777778
30.0000000000 125.6600000000
36.6666666667 153.5344444444
43.3333333333 181.4011111111
50.0000000000 209.2600000000

```
import java.io.*;
import javax.swing.*;

class NAFP3
{
// Polynomial least square
  public static double[] gausswithpartialpivot(double a[][],double b[])
  { //Gauss elimination with partial pivoting
  int n=b.length;
  double x[]=new double[n];
  double carpan=0;
  double toplam=0;
  double buyuk;
  double dummy=0;
  //gauss elimination
  int i,j,k,p,ii,jj;
  for(k=0;k<(n-1);k++)
  { //partial pivoting
            p=k;
            buyuk=Math.abs(a[k][k]);
```

```
            for(ii=k+1;ii<n;ii++)
            {  dummy=Math.abs(a[ii][k]);
              if(dummy > buyuk) {buyuk=dummy;p=ii;}
            }
            if(p!=k)
            { for(jj=k;jj<n;jj++)
              { dummy=a[p][jj];
                a[p][jj]=a[k][jj];
                a[k][jj]=dummy;
              }
              dummy=b[p];
              b[p]=b[k];
              b[k]=dummy;
            }
            //
   for(i=k+1;i<n;i++)
   {  carpan=a[i][k]/a[k][k];
      a[i][k]=0;
      for(j=k+1;j<n;j++)
      {   a[i][j]-=carpan*a[k][j]; }
         b[i]   =b[i]   -carpan*b[k];
            }
  }
  //backward substitution
  x[n-1]=b[n-1]/a[n-1][n-1];
  for(i=n-2;i>=0;i--)
  {
    toplam=0;
    for(j=i+1;j<n;j++)
    {  toplam+=a[i][j]*x[j];}
    x[i]=(b[i]-toplam)/a[i][i];
  }
  return x;
}

public static double[] PolynomialLSQ(double xi[],double yi[],int n)
{ //Polynomial least square
int l=xi.length;
int i,j,k;
int np1=n+1;
double A[][];
A=new double[np1][np1];
double B[];
B=new double[np1];
double X[];
X=new double[np1];
for(i=0;i<n+1;i++)
  {  for(j=0;j<n+1;j++)
     {if(i==0 && j==0) A[i][j]=l;
     else  for(k=0;k<l;k++) A[i][j] += Math.pow(xi[k],(i+j));
     }
     for(k=0;k<l;k++) { if(i==0)  B[i]+= yi[k];
                  else      B[i] += Math.pow(xi[k],i)*yi[k];}
  }
X=gausswithpartialpivot(A,B);
double max=0;
for(i=0;i<n+1;i++)
 if(Math.abs(X[i]) > max) max = Math.abs(X[i]);
for(i=0;i<n+1;i++)
 if((Math.abs(X[i]/max) > 0) && (Math.abs(X[i]/max) < 1.0e-100)) X[i]=0;
return X;
}

public static double funcPolynomialLSQ(double e[],double x)
{
// this function calculates the value of
```

```
// least square curve fitting function
int n=e.length;
double ff;
if(n!=0.0)
  { ff=e[n-1];
  for(int i=n-2;i>=0;i--)
   { ff=ff*x+e[i]; }
  }
else
  ff=0;
return ff;
}

public static double error(double x[],double y[],double e[])
{
//calculates absolute square root error of a least square approach
double n=x.length;
 int k;
 double total=0;
 for(k=0;k<n;k++)
 {
 total+=(y[k]-funcPolynomialLSQ(e,x[k]))*(y[k]-funcPolynomialLSQ(e,x[k]));
 }
total=Math.sqrt(total);
return total;
}

public static double[][]  funcPolynomialLSQ(double xi[],double yi[],int polinomkatsayisi,int aradegersayisi)
 {
             //aradegersayisi:  x--o--o--x--o--o--x zincirinde x deneysel noktalar ise
             // ara değer sayisi 2 dir
              int n=xi.length;
             int nn=(n-1)*(aradegersayisi+1)+1;
             double z[][]=new double[2][nn];
   double E[]=PolynomialLSQ(xi,yi,polinomkatsayisi);
   System.out.println("katsayilar :\n"+Matrix.toStringT(E));
   double dx=0;
   int k=0;
   int i;
   for(i=0;i<(n-1);i++)
      {z[0][k]=xi[i];z[1][k]=funcPolynomialLSQ(E,z[0][k]);k++;
      for(int j=0;j<aradegersayisi;j++)
      {dx=(xi[i+1]-xi[i])/((double)aradegersayisi+1.0);
      z[0][k]=z[0][k-1]+dx;z[1][k]=funcPolynomialLSQ(E,z[0][k]);k++;}
      }
      z[0][k]=xi[i];z[1][k]=funcPolynomialLSQ(E,z[0][k]);
   return z;
 }

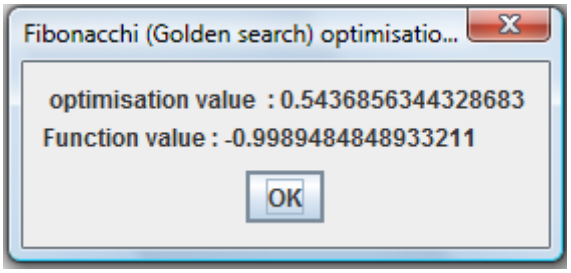

  public static void main(String args[]) throws IOException
  {
  double x[];
  double y[];
  String s1=JOptionPane.showInputDialog("Data file name (input1.txt)");
  // JFileChooser fc=new JFileChooser();
  // if (fc.showOpenDialog(null) == JFileChooser.APPROVE_OPTION) {File file =
fc.getSelectedFile();s1=file.getName(); }
  double a[][]=Text.readDoubleT(s1);
  x=a[0];
  y=a[1];
  double z[][]=funcPolynomialLSQ(x,y,2,2);
  System.out.println("Polynomial Least Square Curve fitting\n"+Matrix.toStringT(z));
  Plot pp=new Plot(a[0],a[1]);
  pp.setPlabel("Polynomial Least Square Curve fitting");
  pp.setXlabel("x");
```

```
  pp.setYlabel("y=f(x)");
  pp.setPlotType(0,0);
  pp.addData(z[0],z[1]);
  pp.setPlotType(6,22);
  pp.setGrid(1,1);
  pp.plot();
  }
}
```

PROBLEM 4

Find minimum of function by using Fibonnachi search(Golden search) method in the region $-1 \leq x \leq 4$

$$f(x) = \frac{-4.1xe^{-x}}{x^2+1}$$



x=0.5 -0.9947102819287187

x=1.5 -0.422230918434721

```
// OPTIMIZATION Golden search or fibonnachi method
import java.util.*;
import java.awt.*;
import java.applet.Applet;
import java.awt.event.*;
import javax.swing.*;
class fb extends f_x
{
public double func(double x)
{
//çözümü istenen fonksiyon
double ff;
ff=-4.1*x*Math.exp(-x)/(x*x+1);
return ff; //maksimum testi
}
}

public class NAFP4
{
public static double golden(f_x f,double a,double b,double epsilon)
{return golden(f,a,b,epsilon,1.0e-10,0);}
public static double golden(f_x f,double a,double b)
{return golden(f,a,b,1.0e-10,1.0e-10,0);}

public static double golden(f_x f,double a,double b,double epsilon,double delta,int print)
{
//  find the minimum of the function
```

```
// note  maximum f(x) = minimum (-f(x))
    double r1 = (Math.sqrt(5.0)-1.0)/2.0; // golden ratio
    double r2 = r1*r1;
    double h = b - a;
    double ya = f.func(a);
    double yb = f.func(b);
    double c = a + r2*h;
    double d = a + r1*h;
    double yc = f.func(c);
    double yd = f.func(d);
    int k = 1;
    double dp,dy,p,yp;
    while ((Math.abs(yb-ya)>epsilon) || (h>delta))
    {
      k++;
      if (yc<yd)
        {
        b = d;
        yb = yd;
        d = c;
        yd = yc;
        h = b - a;
        c = a + r2 * h;
        yc = f.func(c);
        }
      else
       {
         a = c;
         ya = yc;
         c = d;
         yc = yd;
         h = b - a;
         d = a + r1 * h;
         yd = f.func(d);
       }//end of if
    }//end of while
    dp = Math.abs(b-a);
    dy = Math.abs(yb-ya);
    p = a;
    yp = ya;
    if (yb<ya)
    {
      p = b;
      yp = yb;
    }
    if(print==1)
    {System.out.println("x min = "+p+"ymin = "+yp+"error of x ="+dp+"error of y"+dy); }
    return p;
}

     public static void main (String args[])
     {
               fb f=new fb();
     double a,b;
     a=Double.parseDouble(JOptionPane.showInputDialog(" enter lower limit of the function a : "));
     b=Double.parseDouble(JOptionPane.showInputDialog(" enter upper limit of the function b : "));
     double r;
```

```
    System.out.println("x=0.5"+f.func(0.5));
    System.out.println("x=1.5"+f.func(1.5));
    r= golden(f,a,b,1.0e-10,1.0e-5,0);
    JOptionPane.showMessageDialog(null," optimization value  : "+r+"\nFunction value : "+f.func(r),
    "Fibonacchi (Golden search) optimization : ",JOptionPane.PLAIN_MESSAGE);
    System.exit(0);
    }
}
```

WEEKLY EXERCISES AND HOMEWORKS
WEEK 1

EXERCISES WK1
EX1 Solve root of function f(x)=13*(x - x*x)*exp(-3*x/2) by using bisection method. Carry out 3 step hand calculation. Also solve by using computer algorithm.
EX2
Find the root of function $f(x)=x^{10}-1$. Use false position and modified false position methods to find the root of this function. Are the any difficulty to find it? Explain why? Not that normal false position method can not solve this function.
EX 3
Find the root of f(x)= $x-\sqrt{\frac{10}{x+4}}$ by using Fixed point iteration
HOMEWORKS WK1
HW1 : Write a program with a method **sqrt(x)** to calculate square root of a given number x, use bisection method to write it.
HW2 : Problem 5.3(Page131)
Determine the real root of $f(x)=-12-21x+18x^2-2.75x^3$ a) graphically b) using bi-section. Employ initial gueses of xl=0.5 and xu=1.0
HW3 : Problem 5.10(Page 131)
Find the positive real root of $f(x)=x^4-8x^3-35x^2+450x-1001$ using the false position method. Use initial gueses of xl=4.5 and xu=6
HW3:
Find the roots of f(x)=tan(x)*2x by using brent method.
WEEK 2
EXERCISES WK2
EX1 Re-write the program NA5(Newton Raphson method) to find root of $f(x)=x^2-2$ function. Note that you should also define the derivative

EX2 Re-write the program NA6 and NA7 (Secant method) to find root of $f(x)=x^2-2$ function.

EX3 Solve root of function f(x)=13*(x - x*x)*exp(-3*x/2) by using

a) Inverse lagrange interpolation method
b) Newton-bisection combined method
c) Brent method

HOMEWORKS WK2
WK2_HW1 : In round smooth tubes for turbulent flow region Fanning friction factor is given as:

$$\mathbf{1/\sqrt{f} = 1.737 * ln(Re*\sqrt{f}) - 0.4}$$

This equation is valid for $4 * 10^3 < Re < 3 * 10^6$ By using one of the root finding methods, calculate friction factor for Re=5000. You can utilize any root finding method you learned so far.

WK2_HW2: Determine the real root of $f(x)=-12-21x+18x^2-2.75x^3$ by using a) Newton-Raphson Method b) Brent method c) Muller method d) Secant method

WK2_HW3 : Find all real roots of function $\boldsymbol{f(x) = x^4 - 7.223x^3 + 13.447x^2 - 0.672x - 10.223}$

WEEK 3

LAB EXERCISES WK3

PROBLEM 1

$$\begin{bmatrix} 7 & 1 & 2 \\ -1 & 4 & -1 \\ 3 & 15 & 20 \end{bmatrix} \begin{bmatrix} X_0 \\ X_1 \\ X_2 \end{bmatrix} = \begin{bmatrix} 47 \\ 19 \\ 87 \end{bmatrix}$$

a) Solve the given system of equation by using Gauss elimination method by hand
b) By using computer code NA14 (Gauss elimination)
c) Solve the given system of equation by using Gauss elimination method with partial pivoting
d) By using computer code NA15 (Gauss elimination with partial pivoting)

PROBLEM 2

$$\begin{bmatrix} 4 & 3 & -1 \\ 7 & -2 & 3 \\ 5 & -18 & 13 \end{bmatrix} \begin{bmatrix} X_0 \\ X_1 \\ X_2 \end{bmatrix} = \begin{bmatrix} 6 \\ 9 \\ 3 \end{bmatrix}$$

a) Solve the given system of equation by using Gauss-Jordan elimination method by hand
b) By using computer code NA16 (Gauss elimination)
c) By using computer code NA17 (Gauss elimination with partial pivoting)

PROBLEM 3

$$\begin{bmatrix} 8 & 2 & 3 \\ 1 & -9 & 2 \\ 2 & 3 & 6 \end{bmatrix} \begin{bmatrix} X_0 \\ X_1 \\ X_2 \end{bmatrix} = \begin{bmatrix} 30 \\ 1 \\ 31 \end{bmatrix}$$

a) LU Decompose the given system of equation by using Gauss(Dolittle) decomposition method by hand and solve it
b) LU Decompose the given system of equation by using Gauss(Dolittle) decomposition method by using program NA18 and solve it
c) LU Decompose the given system of equation by using pivoted Gauss(Dolittle) decomposition method by using program NA19 and solve it
d) LU Decompose the given system of equation by using Craut decomposition method by using program NA20 and solve it

HOMEWORK PROBLEMS WK 3

PROBLEM 1

$$\begin{bmatrix} 1.1348 & 3.8326 & 1.1651 & 3.4017 \\ 0.5301 & 1.7875 & 2.5330 & 1.5435 \\ 3.4129 & 4.9317 & 8.7643 & 1.3142 \\ 1.2371 & 4.9998 & 10.6721 & 0.0147 \end{bmatrix} \begin{bmatrix} X_0 \\ X_1 \\ X_2 \\ X_3 \end{bmatrix} = \begin{bmatrix} 9.5342 \\ 6.3941 \\ 18.4231 \\ 16.9237 \end{bmatrix}$$

Exact solution : $\begin{bmatrix} X_0 \\ X_1 \\ X_2 \\ X_3 \end{bmatrix} = \begin{bmatrix} 1 \\ 1 \\ 1 \\ 1 \end{bmatrix}$

a) Solve the given system of equation by using Gauss elimination method by hand
b) By using computer code NA14 (Gauss elimination)
c) Solve the given system of equation by using Gauss elimination method with partial pivoting
d) By using computer code NA15 (Gauss elimination with partial pivoting)

PROBLEM 2

$$\begin{bmatrix} 3 & -5 & 47 & 20 \\ 11 & 16 & 17 & 10 \\ 56 & 22 & 11 & -18 \\ 17 & 66 & -12 & 7 \end{bmatrix} \begin{bmatrix} X_0 \\ X_1 \\ X_2 \\ X_3 \end{bmatrix} = \begin{bmatrix} 18 \\ 26 \\ 34 \\ 82 \end{bmatrix}$$

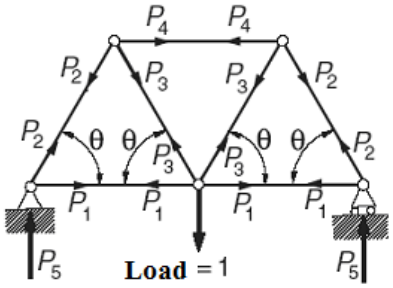
a) Solve the given system of equation by using Gauss-Jordan elimination method by hand
b) By using computer code NA16 (Gauss elimination)
c) By using computer code NA17 (Gauss elimination with partial pivoting)

PROBLEM 3

$3x_1 + 5x_2 + 4x_3 + 2x_4 = 11$
$6x_1 + 14x_2 + 11x_3 + 6x_4 = 26$
$9x_1 + 11x_2 + 16x_3 + 5x_4 = 68$
$3x_1 + 13x_2 + 17x_3 + 12x_4 = 43$

a) LU Decompose the given system of equation by using Gauss(Dolittle) decomposition method by hand and solve it
b) LU Decompose the given system of equation by using Gauss(Dolittle) decomposition method by using program NA18 and solve it
c) LU Decompose the given system of equation by using pivoted Gauss(Dolittle) decomposition method by using program NA19 and solve it
d) LU Decompose the given system of equation by using Craut decomposition method by using program NA20 and solve it

PROBLEM 4



Load balance of the system given in the figure can be formed into a system of equations as follows

$$\begin{bmatrix} c & 1 & 0 & 0 & 0 \\ 0 & s & 0 & 0 & 1 \\ 0 & 0 & 2s & 0 & 0 \\ 0 & -c & c & 1 & 0 \\ 0 & s & s & 0 & 0 \end{bmatrix} \begin{bmatrix} P_1 \\ P_2 \\ P_3 \\ P_4 \\ P_5 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 1 \\ 0 \\ 0 \end{bmatrix}$$

where c=cos(θ) s=sin(θ). For θ =60^o find the load distribution. Use any method (computer program) you desired for the solution

WEEK 4

LAB EXERCISES WK4

PROBLEM 1

$$\begin{bmatrix} 0.8 & -0.4 & 0 \\ -0.4 & 0.8 & -0.4 \\ 0 & -0.4 & 0.8 \end{bmatrix} \begin{bmatrix} X_0 \\ X_1 \\ X_2 \end{bmatrix} = \begin{bmatrix} 41 \\ 25 \\ 105 \end{bmatrix}$$

e) Solve the given system of equation by using Gauss-seidel method by using computer code NA21
f) Solve the given system of equation by using Thomas algorithm (3 band matrix) by using computer code NA23
g) Solve the given system of equation by using Thomas algorithm (3 band matrix) by hand

PROBLEM 2

$$\begin{bmatrix} 4 & 3 & -1 \\ 7 & -2 & 3 \\ 5 & -18 & 1 \end{bmatrix}$$

d) Find the Euclidian norm $\|A\|_e = \sqrt{\sum_{i=1}^{n}\sum_{j=1}^{n} a_{ij}^2}$ by hand

e) Find the inverse matrix by using gauss elimination by hand

f) Find the inverse matrix by using computer code NA28

PROBLEM 3

$$\begin{bmatrix} 8 & 20 & 15 \\ 20 & 80 & 50 \\ 15 & 50 & 60 \end{bmatrix}\begin{bmatrix} X_0 \\ X_1 \\ X_2 \end{bmatrix} = \begin{bmatrix} 50 \\ 250 \\ 100 \end{bmatrix}$$

Solve the given system of equation by using cholesky decomposition by computer code NA22

HOMEWORK PROBLEMS WK4

PROBLEM 1

$$\begin{bmatrix} -2 & 1 & & & & & & & \\ 1 & -2 & 1 & & & & & & \\ & 1 & -2 & 1 & & & & & \\ & & & & \cdots & & & & \\ & & & & 1 & -2 & 1 & & \\ & & & & & 1 & -2 & 1 & \\ & & & & & & 1 & -2 & \end{bmatrix}\begin{bmatrix} X_0 \\ X_1 \\ X_2 \\ \\ X_7 \\ X_8 \\ X_9 \end{bmatrix} = \begin{bmatrix} -0.5 \\ -1.5 \\ -1.5 \\ \cdots \\ -1.5 \\ -1.5 \\ 0.5 \end{bmatrix}$$

Solve the given system of equation by using Thomas algorithm (3 band matrix) computer code NA21

PROBLEM 2

$$\begin{bmatrix} 10 & 2 & -1 \\ -3 & -6 & 2 \\ 1 & 1 & 5 \end{bmatrix}\begin{bmatrix} X_0 \\ X_1 \\ X_2 \end{bmatrix} = \begin{bmatrix} 27 \\ -61.5 \\ -21.5 \end{bmatrix}$$

d) Solve the given system of equation by using Gauss-Seidel method by hand until the percent relative error falls below 5%

e) Solve the given system of equation by using Gauss-Seidel method by computer code

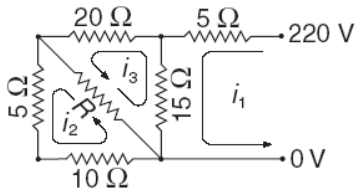
PROBLEM 3

$$\begin{bmatrix} 6 & 15 & 55 \\ 15 & 55 & 225 \\ 55 & 225 & 979 \end{bmatrix}\begin{bmatrix} X_0 \\ X_1 \\ X_2 \end{bmatrix} = \begin{bmatrix} 152.6 \\ 585.6 \\ 2488.8 \end{bmatrix}$$

f) Solve the given system of equation by using cholesky decomposition method by hand

g) Solve the given system of equation by using cholesky decomposition by computer code NA22

PROBLEM 4



For the circuit shown in the diagram following equations can be written

$$5i_1 + 15(i_1 - i_3) = 220\ \text{V}$$

$$R(i_2 - i_3) + 5i_2 + 10i_2 = 0$$

$$20i_3 + R(i_3 - i_2) + 15(i_3 - i_1) = 0$$

For the electrical resistance R=10 Ω calculate i current densities in Amper. Use any method you like (solving by hand or by computer code is left to you)

WEEK 5

LAB EXERCISES WK5

EX 1

$x_1(1- x_1) + 4x_3 – 12 =0$

$(x_1-2)^2+(2x_2-3)^2-25 =0$

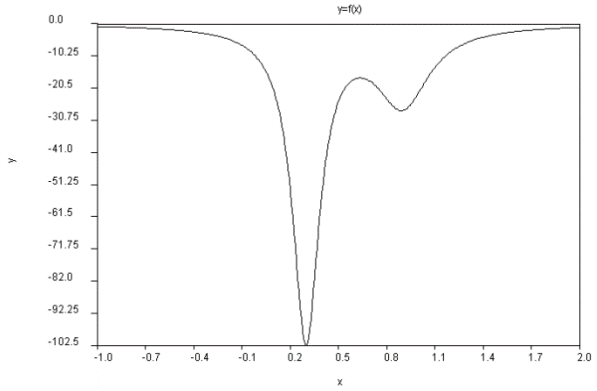
Solve the roots of nonlinear system of equation

a) by using Newton Raphson method by hand
b) by using Newton Raphson method bycomputer program

EX 2

Find the minimum of function

f(x)=-(1.0/((x-0.3)*(x-0.3)+0.01)+1.0/((x-0.9)*(x-0.9)+0.04))



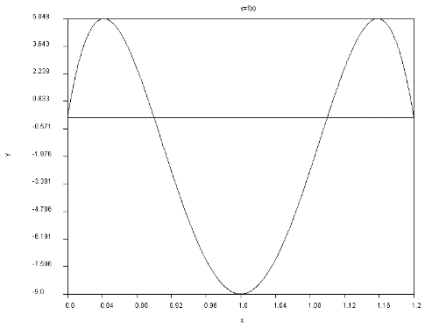
a) By using golden search method with computer
b) By using second degree polynomial with computer
c) By using Newton-Raphson method by computer

EX 3

Find the minimum of function

$f(x) = 2500x^6-15000x^5+14875x^4+40500x^3-97124x^2+73248x-19008$

between $x_0=0.8$, $x_1=1.2$



a) By using golden search method with computer
b) By using second degree polynomial with computer
c) By using Newton-Raphson method by computer

HOMEWORKS WK5

PROBLEM HW5.1

$10x_1-2x_2^2+x_2-2x_3-5=0$

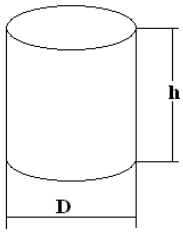
$8x_2^2+4x_3^2-9=0$

$8x_2x_3+4=0$

Solve the roots of nonlinear system of equation

c) by using Newton Raphson method by hand
d) by using Newton Raphson method bycomputer program

PROBLEM HW5.2



One of the basic optimization problem is to manufacture a box with a certain volume with minimum cost. The cost of a box is basically depend on the surface area. Therefore for a given volume surface area should be minimised. As an example if box volume, V is taken as 0.5 liters=$0.5x10^{-3}$ m^3

Volume $V = \frac{\pi D^2}{4} h$ and the height will be $h = \frac{4V}{\pi D^2}$.

Surface area of the cylinder : $A = 2\frac{\pi D^2}{4} + \pi D h = \frac{\pi D^2}{2} + \frac{4V}{D}$.

This minimisation problem can be solved analytically

$$\frac{dA}{dD} = \pi D - \frac{4V}{D^2} \rightleftarrows = 0$$

And D diameter is found as:

$D = \sqrt[3]{\frac{4V}{\pi}}$ $D = \sqrt[3]{\frac{4*0.5x10^{-3}}{\pi}}$=0.086025401 m .

If this value is substituted into h,

$h = \frac{4*0.5x10^{-3}}{\pi D^2}$= 0.086025401 m .

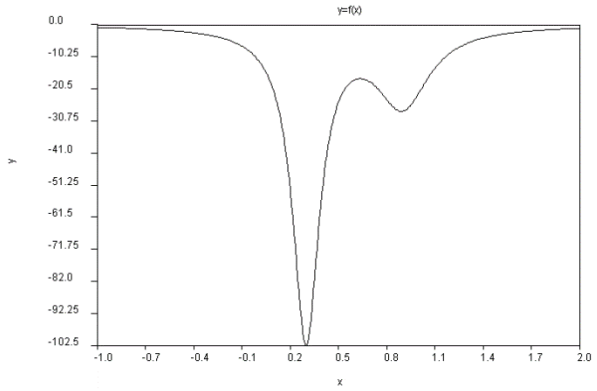
Solve the same problem with computer for : 0.01<=D<=0.2 arasinda

j) Graphic method
k) Golden ratio method
l) Newton-Raphson method
m) Quadratic(Second degree) polynomial
n) Cubic polynomial
o) Brent optimization method

PROBLEM HW5.3

Find the minimum of function

f(x)=-(1.0/((x-0.3)*(x-0.3)+0.01)+1.0/((x-0.9)*(x-0.9)+0.04))



a) By using golden search method by hand
b) By using second degree polynomial by hand
c) By using Newton-Raphson method by hand

WEEK 6

LAB EXERCISES

PROBLEM 1

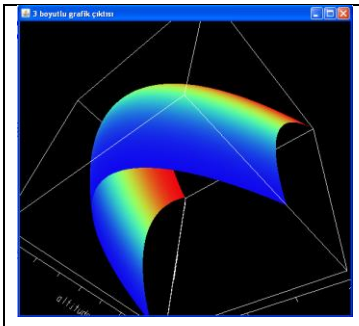
In a channel polutant concentration is given as

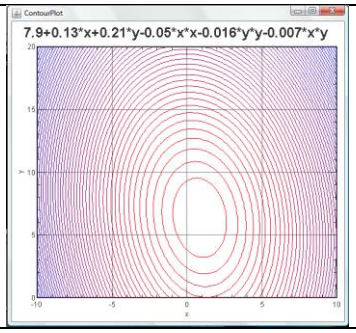
$C(x,y)=7.9+0.13x+0.21y-0.05x^2-0.016y^2-0.007xy$

Maximum concentration is in between -10 <= x <= 10 ve 0 <= y <= 20 limits. Find the maximum concentration

b) By using Newton raphson method
c) Nelder-Mead method
d) Steepest descent method

(use computer codes)



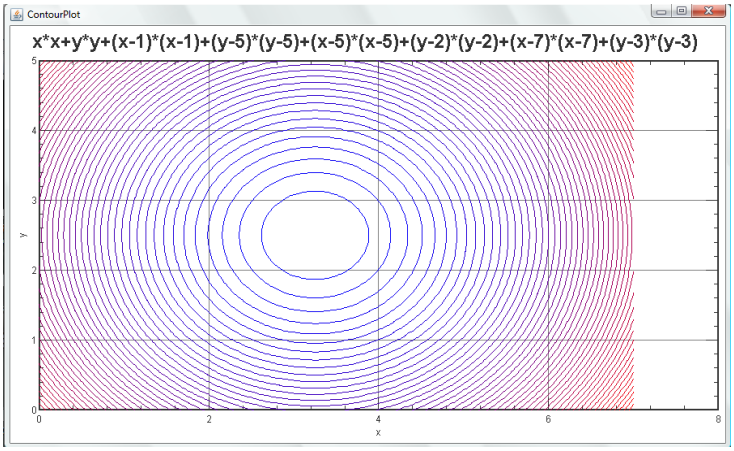


PROBLEM 2

Cities, A,B,C, and D are located at points (0,0),(1,5),(5,2) and (7,3) respectively. Roads from this four city is crossed at a point P. Find the point p so that total road distance will become minimum. (use computer codes)

$$D = \sum_i [(x - x_i)^2 + (y - y_i)^2]$$

$$D = (x-0)^2 + (y-0)^2 + (x-1)^2 + (y-5)^2 + (x-5)^2 + (y-2)^2 + (x-7)^2 + (y-3)^2$$



HOMEWORKS

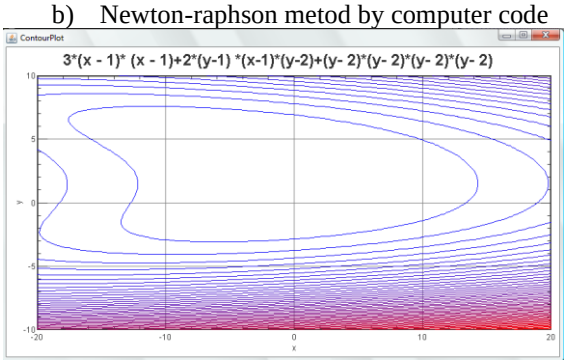
PROBLEM 1

Minimum of function f(x) is located at point(1,2)

$f(x_0, x_1) = 3(x_0 - 1)^2 + 2(x_1-1)(x_0-1)(x_1-2)+(x_1- 2)^4$

Starting from point (0,0) find the minimum by

a) Newton raphson metod calculated by hand
b) Newton-raphson metod by computer code



PROBLEM 2

Find the minimum of function

$f(x_0, x_1, x_2, x_3) = (x_0+10\ x_1)^2 + 5(x_2-10\ x_3)^2+(x_1-2\ x_2)^4+10(x_1- x_4)^4$

by starting from point (1,-1,0,1) .

a) By using Newton raphson method
b) Nelder-Mead method
c) Steepest descent method

(use computer codes)

PROBLEM 3

A petrochemical company is developing a new fuel additive for commercial airplanes. Additive has three different chemicals X,Y and Z. For a good performance additive amount in a liter of fuel should be 6 mL/L. Because of security concerns total of very flammable X and Y components should be less than 3 mL/L. In addition component X should be equal or greater than Y and Z component should be greater than half of Y. The cost of X,Y and Z components are 0.15,0.025 ve 0.05 TL respectively. Calculate the component percentages for the minimum fuel additive cost.

a) Solve by hand
b) Solve by computer code

WEEK 7

WEEK 7 LAB EXERCISES

PROBLEM 1

Use linear least square method (x=ao+a1*x) **by hand** to fit data to the following equation

X	2	3	4	7

Y	9	6	5	10

PROBLEM 2

Fit cubic spline interpolation Formula to the following equation **by computer**

X	Y
0	0
1	1
2	4
3	8

PROBLEM 3

Enthapy of a saturated water will be approximated by the equation

$$h_f = A + Bt + Ct^2 + Dt^3$$

Following data is taken from the thermodynamic table. Find the polynomial coefficient by **using least square computer program**

t, ^oC	10	30	50	70	90
h_f , kJ/kg	41.99	125.66	209.26	292.97	376.94

PROBLEM 4

Force aplied to a linear spring is given with the Formula F=kx according to Hooke law. In this equation k is the spring constant and x is the displacement. Experiment is being carried out and found the following results between displacement and Force. Estimate spring constant k (**use computer programs**)

a) By using Newton Interpolation Formula

b) By using Lagrange interpolation Formula

c) By using least square curve fitting

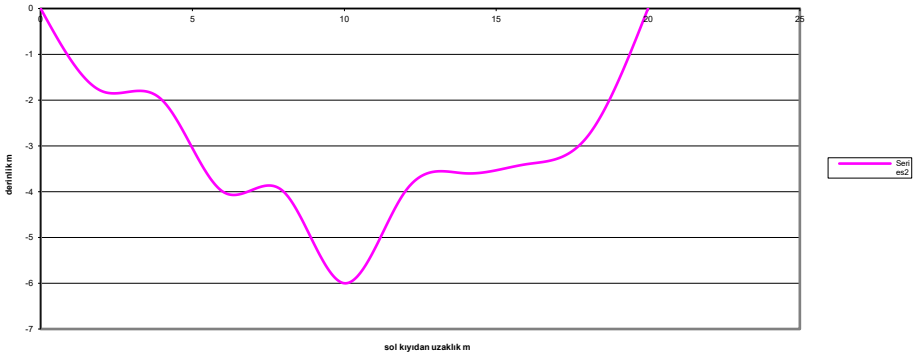
X, m	0.02	0.04	0.06	0.08	0.1
F, kN	3.1	6.1	9.2	12	15.1

WEEK 7 HOMEWORKS

PROBLEM 1

Depth (H) and velocity (U) profile of a channel is given below

x, m	0	2	4	6	8	10	12	14	16	18	20	
H, m	0	1.8	2	4	4	6	4	3.6	3.4	2.8	0	
U m/s	0	0.03	0.045	0.055	0.065	0.12	0.07	0.06	0.05	0.04	0	



Curve fit to H(x) and U(x) functions by using (Computer programming)

d) Cubic spline interpolation formula
e) Polinomial least sqare formula
f) Lagrange interpolation formula

PROBLEM 2

Thermal conductivity coefficient of water for different temperature values are given in the below table

Temperature ***T*** . [K]	275	290	305	320	340	360
Thermal conductivity coefficient ***k***. [W/(m·K)]	0.574	0.598	0.620	0.640	0.660	0.674

a-) curvefit the following equation to thermal conductivity-temperature relation by using hand calculations:

$$k = A \cdot T + B \quad .$$

b-) Calculate thermal conduvtivity for T=300 K .

PROBLEM 3

Enthapy of a saturated water will be approximated by the following equation

$$h_f = A + Bt + Ct^2 + Dt^3$$

Following data is taken from the thermodynamic table. Find the polynomial coefficient by using polinomial least square by hand calculations

t, ^oC	10	30	50	70	90
h_f , kJ/kg	41.99	125.66	209.26	292.97	376.94

PROBLEM 4

Force aplied to a linear spring is given with the Formula F=kx according to Hooke law. In this equation k is the spring constant and x is the displacement. Experiment is being carried out and found the following results between displacement and Force. Estimate spring constant k (by hand calculations)

d) By using Newton Interpolation Formula
e) By using Lagrange interpolation Formula
f) By using least square curve fitting

X, m	0.02	0.04	0.06	0.08	0.1
F, kN	3.1	6.1	9.2	12	15.1

WEEK 8

WEEK 8 LAB EXERCISES

PROBLEM 1

Calculate integral of **y=f(x)=1/(1 + x^2)** fonction between limits –1 ile 1 with 3 points Gauss-Legendre integration formula by hand and by computer program

Note: Required roots and coefficients :

coefficients	Roots
c_0 0.555555555555553	x_0 0.774596669241483
c_1 0.888888888888889	x_1 0.000000000000000
c_2 0.555555555555553	x_2 0.774596669241483

PROBLEM 2

$\int_{-3}^{3} \frac{2}{1+2x^2} dx$ solve with bole rule. Use n=2

Note: Bole rule for n=1

$$\boldsymbol{I = \int_{a}^{b} f(x)dx = \frac{(b-a)}{90}[7f(x_0) + 32f(x_1) + 12f(x_2) + 32f(x_3) + 7f(x_4)]}$$

Step size h=(b-a)/n

PROBLEM 3

$$\boldsymbol{I = \int_{0}^{1} \frac{2dx}{2 + sin(10\pi x)} = 1.15470054}$$

Solve the above integration by using

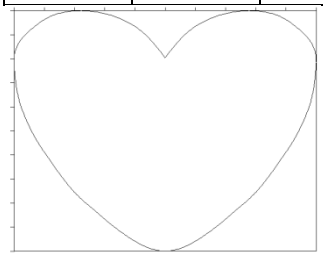
e) Gauss-Legendre integration by computer
f) Simpson 1/3 integration by computer
g) Romberg integration

WEEK 8 HOMEWORKS

PROBLEM 1

Coordinate of a hard shape is given in the table. Find the area of the hard.

Xi	Yi	xi	yi	xi	yi	xi	Yi
241	127	331	157	215	246	149	120
258	107	323	178	199	232	160	106
283	96	312	197	184	217	175	97
307	97	298	217	170	197	199	96
322	106	283	232	159	178	224	107
333	120	267	246	151	157	241	127
334	138	241	258	148	138		



PROBLEM 2

Depth(H) and velocity(U) profile of a channel is given in below table

x, m	0	2	4	6	8	10	12	14	16	18	20
H, m	0	1.8	2	4	4	6	4	3.6	3.4	2.8	0
U m/s	0	0.03	0.045	0.055	0.065	0.12	0.07	0.06	0.05	0.04	0

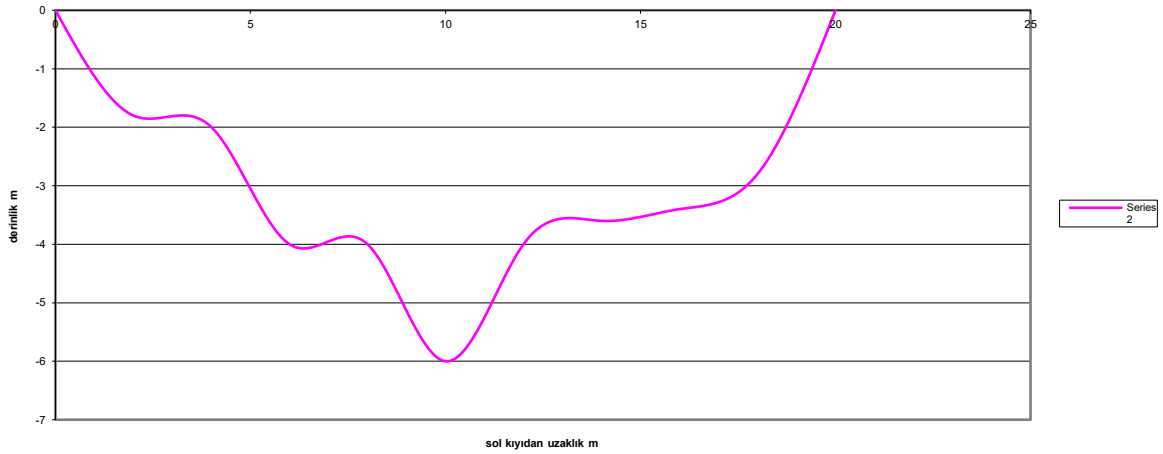
Cross sectional area of the channel can be calculated as :

$$A_c = \int_0^B H(x)dx$$

Flow rate of the channel will be

$$Q = \int_0^B U(x)H(x)dx$$

In this equations U is the velocity of the water measured at point x. Calculate cross sectional area and flow rate of the channel.



PROBLEM 3

Blackbody radiation function F(0- λT), λ(μm) T(degree Kelvin) can be calculated as

$$x = \frac{C_2}{\lambda T} = \frac{14387.69\ \mu mK}{\lambda T}$$

$$F(0-\lambda T) = 1 - \frac{15}{\pi^4}\int_0^x \frac{x^3}{e^x - 1}dx$$

It can also be calculated from the series solution

$$F(0-\lambda T) = \frac{15}{\pi^4}\sum_{n=1}^{\infty}\left[\frac{e^{-nx}}{n}(x^3 + \frac{3x^2}{n} + \frac{6x}{n^2} + \frac{6}{n^3})\right]$$

By creating a program that calculates the result with using series solution and integration. Compare the results. Resulting tables are given as a reference

λ T	F(0-λT)	λ T	F(0-λT)	λ T	F(0-λT)	λ T	F(0-λT)	λ T	F(0-λT)	λ T	F(0-λT)

μ mK		λ T		μ mK		μ mK		μ mK		μ mK	
1000	0.00032	2900	0.25056	4800	0.60754	6900	0.8022	10700	0.92709	18000	0.98081
1050	0.00056	2950	0.26191	4850	0.61428	7000	0.80808	10800	0.92872	18200	0.98137
1100	0.00091	3000	0.27323	4900	0.62089	7100	0.81374	10900	0.93031	18400	0.98191
1150	0.00142	3050	0.28453	4950	0.62737	7200	0.81918	11000	0.93185	18600	0.98243
1200	0.00213	3100	0.29578	5000	0.63373	7300	0.82443	11200	0.9348	18800	0.98293
1250	0.00308	3150	0.30697	5050	0.63996	7400	0.82949	11400	0.93758	19000	0.98341
1300	0.00432	3200	0.3181	5100	0.64608	7500	0.83437	11600	0.94021	19200	0.98387
1350	0.00587	3250	0.32915	5150	0.65207	7600	0.83907	11800	0.9427	19400	0.98431
1400	0.00779	3300	0.34011	5200	0.65795	7700	0.8436	12000	0.94505	19600	0.98474
1450	0.01011	3350	0.35097	5250	0.66371	7800	0.84797	12200	0.94728	19800	0.98516
1500	0.01285	3400	0.36173	5300	0.66937	7900	0.85219	12400	0.94939	20000	0.98555
1550	0.01605	3450	0.37238	5350	0.67491	8000	0.85625	12600	0.95139	20500	0.98649
1600	0.01972	3500	0.38291	5400	0.68034	8100	0.86018	12800	0.95329	21000	0.98735
1650	0.02388	3550	0.39332	5450	0.68566	8200	0.86397	13000	0.95509	21500	0.98814
1700	0.02854	3600	0.4036	5500	0.69089	8300	0.86763	13200	0.95681	22000	0.98886
1750	0.03369	3650	0.41375	5550	0.696	8400	0.87116	13400	0.95843	22500	0.98952
1800	0.03934	3700	0.42377	5600	0.70102	8500	0.87457	13600	0.95998	23000	0.99014
1850	0.04549	3750	0.43364	5650	0.70594	8600	0.87787	13800	0.96145	23500	0.9907
1900	0.05211	3800	0.44338	5700	0.71077	8700	0.88105	14000	0.96285	24000	0.99123
1950	0.0592	3850	0.45297	5750	0.7155	8800	0.88413	14200	0.96419	24500	0.99172
2000	0.06673	3900	0.46241	5800	0.72013	8900	0.88711	14400	0.96546	25000	0.99217
2050	0.07469	3950	0.47172	5850	0.72468	9000	0.88999	14600	0.96667	26000	0.99297
2100	0.08306	4000	0.48087	5900	0.72914	9100	0.89278	14800	0.96783	27000	0.99368
2150	0.0918	4050	0.48987	5950	0.73351	9200	0.89547	15000	0.96893	28000	0.99429
2200	0.10089	4100	0.49873	6000	0.73779	9300	0.89808	15200	0.96999	29000	0.99482
2250	0.11031	4150	0.50744	6050	0.74199	9400	0.9006	15400	0.971	30000	0.99529
2300	0.12003	4200	0.516	6100	0.74611	9500	0.90305	15600	0.97196	32000	0.99607
2350	0.13002	4250	0.52442	6150	0.75015	9600	0.90541	15800	0.97289	34000	0.99669
2400	0.14026	4300	0.53269	6200	0.75411	9700	0.9077	16000	0.97377	36000	0.99719
2450	0.15071	4350	0.54081	6250	0.758	9800	0.90992	16200	0.97461	38000	0.99759
2500	0.16136	4400	0.54878	6300	0.76181	9900	0.91207	16400	0.97542	40000	0.99792
2550	0.17217	4450	0.55662	6350	0.76554	10000	0.91416	16600	0.9762	45000	0.99851
2600	0.18312	4500	0.56431	6400	0.76921	10100	0.91618	16800	0.97694	50000	0.9989
2650	0.19419	4550	0.57186	6450	0.7728	10200	0.91814	17000	0.97765	55000	0.99917
2700	0.20536	4600	0.57927	6500	0.77632	10300	0.92004	17200	0.97834	60000	0.99935
2750	0.2166	4650	0.58654	6600	0.78317	10400	0.92188	17400	0.97899		
2800	0.22789	4700	0.59367	6700	0.78976	10500	0.92367	17600	0.97962		

WEEK 9

WEEK 9 LAB EXERCISES

PROBLEM 1

Calculate velocity and acceleration by using below data and creating excell tables with

g) cental difference method [with $O(h^4)$ error level],

h) forward difference method [with $O(h^2)$ error level],

i) backward difference method [with $O(h^2)$ error level],

At t=1.5,2.5 and 3.5 seconds

Time, t (s)	0	1	2	3	4	5	6	7	8
distance, x (m)	0	0.7	1.8	3.4	5.1	6.5	7.3	8.0	8.4

Note : Velocity $v = \frac{dx}{dt}$Acceleration : $a = \frac{d^2x}{dt^2}$

PROBLEM 2

Calculate velocity (first derivative) and acceleration (second derivative) by using function x=0.038073038073421 -0.005445480447162*t+0.645464257965281*t^2 -0.100887075887276*t^3+0.004574592074604

At t=1.5,2.5 and 3.5 seconds by using analytical hand derivative and by using related java program

WEEK 9 HOMEWORKS

PROBLEM 1

Calculate velocity and acceleration by using below data **directly** and **creating your own java program with**

a) cental difference method [with O(h^4) error level],
b) forward difference method [with O(h^2) error level],
c) backward difference method [with O(h^2) error level],

Time, t (s)	0	1	2	3	4	5	6	7	8
distance, x (m)	0	0.7	1.8	3.4	5.1	6.5	7.3	8.0	8.4

Note : Velocity $v = \frac{dx}{dt}$Acceleration : $a = \frac{d^2x}{dt^2}$

At t=1.5,2.5 and 3.5

PROBLEM 2

Calculate velocity and acceleration by using below data and **creating your own java program to curve-fit a 4th degree polynamial first and then apply**

a) cental difference method [with O(h^4) error level],
b) forward difference method [with O(h^2) error level],
c) backward difference method [with O(h^2) error level],

to the given polynomial function

Time, t (s)	0	1	2	3	4	5	6	7	8
distance, x (m)	0	0.7	1.8	3.4	5.1	6.5	7.3	8.0	8.4

Note : Velocity $v = \frac{dx}{dt}$Acceleration : $a = \frac{d^2x}{dt^2}$

At t=1.5,2.5 and 3.5 seconds

Note : Due to holiday of last week you are responsible both week 8 and week 9 exercises and homeworks, any exercises cannot be finished in class add them into the homework please

WEEK 10

WEEK 10 LAB EXERCISES

PROBLEM 1

Solve diferential equation

$\frac{dy}{dx} = yx^2 - y$ at the range x=0 ve x=2 , with initial condition y(0)=1 .

e) solve analytically from x=0 to x=2.
f) solve numerically by hand with h=0.5 step size with 4th order Runge_Kutte and calculate y(2)
g) solve numerically by computer with h=0.1 step size with 4th order Runge-Kutta
h) Plot to results and compare with analytical solution

PROBLEM 2

For diferential equation

$y' - 2(1 + y^2) = 0$ with initial condition, $y(0)=0$ and

a) h=0.25 step size find $y(1)$ with Heun method by hand

b) h=0.05 step size find y(1) with 6th order Runge Kutta by computer

PROBLEM 3

Solve diferential equation

$\frac{dy}{dx} = -5y + e^{-2x}$ with initial conditions **:** $x_0=0$ $y_0=1$. $y(1)$, by using Simpson 1/3 rule.

a) By hand with step size h=0.2

b) By computer with step size h=0.02

WEEK 10 HOMEWORKS

PROBLEM 1

Cooling rate of a substance are given with the following equation

$$\frac{dT}{dt} = -k(T - T_a)$$

In the equation T indicates the temperature of the substance in (ºC), T_a is environmental temperature (ºC) ,k is a constant (minute^{-1}) dir. Therefore cooling rate is function of temperature difference between body and environment. If a metal ball initially at 90 ºC drop into a water bath at T_a=20 ºC and if k=0.1 minute^{-1}, Find the time period required ball to reach 30 ºC temperature

c) By analytical methods

d) By a numerical method

PROBLEM 2

In a chemical reaction a chemical A and a chemical B reacts to give chemical C. Concentration of chemical C as a function of time, y(t) is given with the following equation

$y' = k(a - y)(b - y)$ $y(0)=0$

In this equation k is a positive constant, a and b is the initial concentration of chemicals A and B.

Assume that

k=0.01 , a= 70 milimole/liter , b= 50 milimole/liter .

a) By using 4th order Runge-Kutta method as hand calculation with step size h=0.5 estimate concentration at t=1 second.

b) By using 4th order Runge-Kutta method as computer calculation with step size h=0.05 estimate concentration at t=1 second.

PROBLEM 3

Solve diferential equation

$\frac{dy}{dx} = \frac{1}{x+y}$ with initial conditions $x_0=0$, $y_0=2$, at the interval 0-0.6 with step size h=0.2,

a-) with 4th order Runge-Kutta method by hand

b-) with Euler method by hand

NUMERİCAL ANALYSİS MIDTERM EXAM APRIL 2011

Student name	
Student id #	
Q1	
Q2	
Q3	
Q4	
TOTAL	

QUESTION 1 :

Find root of $f(x)=x^3-2x^2-3$, starting form x=2 by Secant method

Secant 2 f(x)=x*x*x-2*x*x-3

a	2					
x	f(x)=x*x-a	x+Δx	x-Δx	f(x+Δx)	f(x-Δx)	f'(x)
Δx	0.1					
2	-3	2.1	1.9	-2.559	-3.361	4.01
2.748129676	2.65004	2.84813	2.64813	3.87989	1.54507	11.6741
2.521128853	0.31234	2.62113	2.42113	1.26735	-0.5314	8.99376
2.486400067	0.00702	2.5864	2.3864	0.9227	-0.79949	8.61096
2.485585368	1.2E-05	2.58559	2.38559	0.91479	-0.80563	8.60206
2.485583999	1.4E-08	2.58558	2.38558	0.91477	-0.80564	8.60205
2.485583998	1.6E-11	2.58558	2.38558	0.91477	-0.80564	8.60205
2.485583998	2E-14	2.58558	2.38558	0.91477	-0.80564	8.60205
2.485583998	0	2.58558	2.38558	0.91477	-0.80564	8.60205
2.485583998	0	2.58558	2.38558	0.91477	-0.80564	8.60205
2.485583998	0	2.58558	2.38558	0.91477	-0.80564	8.60205

```
/ Numerical Analysis
//
//
// Root finding with secant method
// two initia root estimation
//
import java.util.*;
import java.awt.*;
import java.applet.Applet;
import java.awt.event.*;
import javax.swing.*;

//============== Function definition ================

class fb extends f_x
{
  public double func (double x)
  { return x*x*x-2*x*x-3;}
}

public class Q1_secant
{
public static double secant(f_x f,double x0,double x1)
{
int nmax=500;
double tolerance=1.0e-10;
double fx0,fx1;
double x;
fx0=f.func(x0);
for(int i=0;i<nmax;i++)
{
 fx1=f.func(x1);
 x=x1-fx1*(x1-x0)/(fx1-fx0);
 x0=x1;
 fx0=fx1;
 x1=x;
 System.out.println("x="+x+"i="+i);
 if(Math.abs(fx1)<tolerance) { return x1;}
}
JOptionPane.showMessageDialog(null,"Warning : MAximum number of iteration is exceeded \n"+
  " çözüm geçerli olmiyabilir","MAXİMUM ITERATION NUMBER WARNING",JOptionPane.WARNING_MESSAGE);
return x1;
}
```

```
    public static void main (String args[])
    {
            fb f=new fb();
    double x0,x1;
    x0=Double.parseDouble(JOptionPane.showInputDialog(" First guess for the root : "));
    x1=Double.parseDouble(JOptionPane.showInputDialog(" Second guess for the root : "));
    double r= secant(f,x0,x1);
    JOptionPane.showMessageDialog(null," the root  : "+r+"\nfunction value : "+f.func(r),
    "Secant root finding method : ",JOptionPane.PLAIN_MESSAGE);
    System.exit(0);
    }
}
```

First guess : 2.0, 2.5

```
---------- Capture Output ----------
> "D:\java\bin\javaw.exe" Q1_secant
x=2.48i=0
x=2.485533077172353i=1
x=2.485584178731823i=2
x=2.4855839976827454i=3
x=2.4855839976886003i=4
> Terminated with exit code 0.
```

QUESTION 2 :

$$\begin{bmatrix} 2 & 1 & 2 \\ 3 & 1 & -1 \\ 1 & 4 & 1 \end{bmatrix} \begin{bmatrix} X_0 \\ X_1 \\ X_2 \end{bmatrix} = \begin{bmatrix} 10 \\ 2 \\ 12 \end{bmatrix}$$

Solve the problem by hand by using Gauss-Jordan elimination

```
import java.io.*;

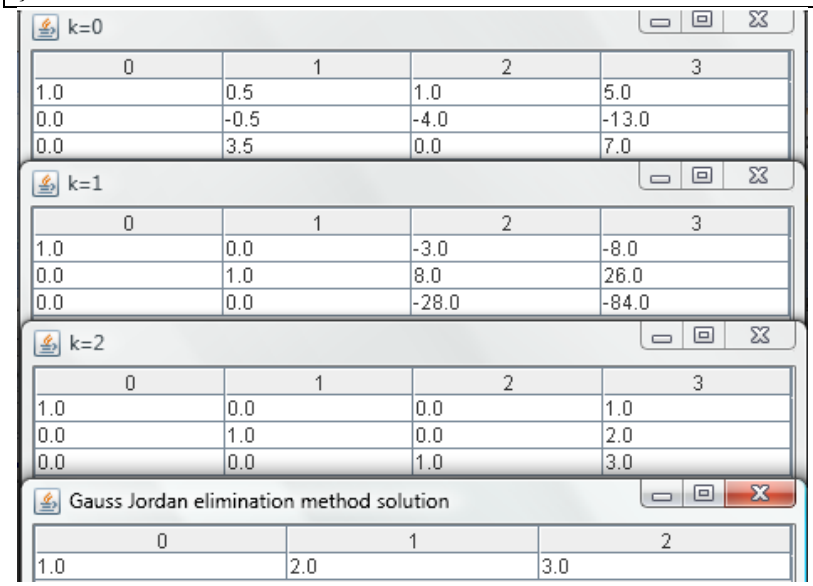
class Q2
{
//Gauss-Jordan elimination process
  public static double[] gaussjordan(double a[][],double b[])
  { //NO PİVOTİNG
  int n=b.length;
  int np1=n+1;
  double C[][]=new double[n][np1];
  double x[]=new double[n];
  double carpan=0;
  int i,j,k;

  for(i=0;i<n;i++)
  {for(j=0;j<n;j++){C[i][j]=a[i][j];};C[i][n]=b[i];}

  for(k=0;k<n;k++)
  {
            //gauss jordan elimination
            for(j=k+1;j<np1;j++)
            { C[k][j]/=C[k][k]; }
            for(i=0;i<n;i++)
```

```
   {  if(i!=k)
      {  carpan=C[i][k];
         C[i][k]=0;
         for(j=k+1;j<np1;j++)
         {   C[i][j]-=carpan*C[k][j]; }
      }
            }
            C[k][k]=1.0;
            Text.print(C,"k="+k);
  }
  for(i=0;i<n;i++)
  {x[i]=C[i][n];}
  return x;
  }

  public static void main(String args[]) throws IOException
  {
  double a[][]={{2,1,2},{3,1,-1},{1,4,1}};
  double b[]={10,2,12};
  double x[]=gaussjordan(a,b);
  Text.print(x,"Gauss Jordan elimination method solution");
  }
}
```



k=0

0	1	2	3
1.0	0.5	1.0	5.0
0.0	-0.5	-4.0	-13.0
0.0	3.5	0.0	7.0

k=1

0	1	2	3
1.0	0.0	-3.0	-8.0
0.0	1.0	8.0	26.0
0.0	0.0	-28.0	-84.0

k=2

0	1	2	3
1.0	0.0	0.0	1.0
0.0	1.0	0.0	2.0
0.0	0.0	1.0	3.0

Gauss Jordan elimination method solution

0	1	2
1.0	2.0	3.0

QUESTION 3

Find the minimum of function by Quadratic interpolation Method. Use x_0=0.3 x1=0.45, x2=0.6 as initial guesses

f(x)= -(1.0/(x*x-x+0.26)+1.0/(x*x-1.6*x+0.68))

```
// OPTIMISATION
//  2nd degree polynomial (Quadratic polynomial) optimisation
//  of function f(x)
import java.util.*;
import java.awt.*;
import java.applet.Applet;
import java.awt.event.*;
import javax.swing.*;
```

```
class fb extends f_x
{
   public double func (double x)
   { return -(1.0/(x*x-x+0.26)+1.0/(x*x-1.6*x+0.68))  ;}
}

public class Q3
{
public static double polinom2derece(f_x f,double x0,double x2)
{double x1=(x0+x2)/2.0;
 return polinom2derece(f,x0,x1,x2);
}
public static double polinom2derece(f_x f,double x0,double x1,double x2)
{
double epsilon=1.0e-10;
double delta=1.0e-5;
int print=0;
return polinom2derece(f,x0,x1,x2,epsilon,delta,print);
}
public static double polinom2derece(f_x f,double x0,double x1,double x2,double epsilon,double delta,int print)
{
int maxit=100;
double f0 = f.func(x0);
    double f1 = f.func(x1);
    double f2 = f.func(x2);
    double f3 = f2;
    double x3 = 0;
    double h = x1 - x0;
    double k=1;
    double dd=Math.abs(f1-f0);
    while ((dd >epsilon) || (h>delta))
    {
      k++;
      x3=(f0*(x1*x1-x2*x2)+f1*(x2*x2-x0*x0)+f2*(x0*x0-x1*x1))/
            (2*f0*(x1-x2)+2.0*f1*(x2-x0)+2.0*f2*(x0-x1));
      f3=f.func(x3);
      System.out.println("k="+k+"x3="+x3+"f3="+f3);
      if(x3 >= x0 && x3<x1)
        {x2=x1;f2=f1;x1=x3;f1=f3;}
      else if(x3 >= x1 && x3 <x2)
        {x0=x1;f0=f1;x1=x3;f1=f3; }
      else if(x3 >x2)
        {x0=x1;f0=f1;x1=x2;f1=f2;x2=x3;f2=f3; }
      else if(x3 < x0)
        {x0=x3;f0=f3;x1=x0;f1=f0;x2=x1;f2=f1; }
      dd=Math.abs(f1-f0);
      h=Math.abs(x1-x0);
      if(k>maxit) break;
    }//end of while
    if(print==1)
    {System.out.println("x  = "+x3+"f = "+f3+"error x ="+h+"error f(x)=y "+dd); }
    return x3;
}

     public static void main (String args[])
     {
             fb f=new fb();
     double x0,x1,x2;
     x0=Double.parseDouble(JOptionPane.showInputDialog(" enter lower limit of the function  x0 : "));
```

```
    x1=Double.parseDouble(JOptionPane.showInputDialog(" enter middle value of the function x1 : "));
    x2=Double.parseDouble(JOptionPane.showInputDialog(" enter upper limit of the function  x2 : "));
    double r;
    r= polinom2derece(f,x0,x1,x2,1.0e-10,1.0e-5,1);
   JOptionPane.showMessageDialog(null," optimisation value  : "+r+"\nFunction value : "+f.func(r),
    "Quadratic polynomial optimisation : ",JOptionPane.PLAIN_MESSAGE);
    System.exit(0);
    }
}
```

```
---------- Capture Output ----------
> "D:\java\bin\javaw.exe" Q3
k=2.0 x3=0.4839149965445751f3=-104.62544399249386
k=3.0 x3=0.5119679859449885f3=-106.72046366023359
k=4.0 x3=0.5054538576851278f3=-107.59252037998598
k=5.0 x3=0.5017979307593692f3=-107.72416408251658
k=6.0 x3=0.5017729838558391f3=-107.72415928235161
k=7.0 x3=0.5017953834615351f3=-107.72416415583007
k=8.0 x3=0.5017951927997668f3=-107.72416415616249
k=9.0 x3=0.5017949027816941f3=-107.72416415529139
k=10.0 x3=0.5017953553133061f3=-107.72416415592424
k=11.0 x3=0.5018013960819635f3=-107.7241637770342
k=12.0 x3=0.5017939121697447f3=-107.72416413978948
x  = 0.5017939121697447f = -107.72416413978948 error x = 0.0 error f(x)=y 0.0
```

QUESTION 4

Find the minimum of the function by steepest descent method
Use Starting point P(3.0,-1.0) apply at least two iterations

$f(x,y)=-8x+x^2+12y+4y^2-2xy;$

```
// Multidimensional non-Linear optimisation
// Steepest descent method
//
import javax.swing.*;
import java.util.*;
import java.awt.*;
import java.applet.Applet;
import java.awt.event.*;
import java.io.*;
/*
class f1 extends f_xj
{
public double func(double x[])
{
//optimisation function
double ff;
ff=3.0*x[0]*x[0]-4.0*x[0]*x[1]+2.0*x[1]*x[1]-x[0]-x[1];
return ff;
}
}*/
class f1 extends f_xj
{
public double func(double x[])
{
```

```
//optimisation function
double ff;
ff=3.5+0.12*x[0]+0.11*x[1]-0.05*x[0]*x[0]-0.012*x[1]*x[1]-0.002*x[0]*x[1];;
return ff;
}}

class f1dim extends f_x
{
  // pseudofunction to convert multidimensional function to
  // one dimensional function
  f_xj ff1;
  double pc[];
  double xc[];
  double xt[];
  int n;
  public f1dim(f_xj ff,double pi[],double xi[])
  { n=pi.length;
   ff1=ff;
            pc=new double[n];
   xc=new double[n];
   xt=new double[n];
            for(int i=0;i<n;i++)
              {pc[i]=pi[i];xc[i]=xi[i];}
  }

  public double func(double x)
  {
            for (int j=0;j<n;j++) xt[j]=pc[j]+x*xc[j];
            return ff1.func(xt);
  }
}


public class Q4
{
public static double SIGN(double a,double b) {return (b > 0.0 ? Math.abs(a) : -Math.abs(a));}
public static double MAX(double a,double b) {return (a > b ? a : b);}

// 3. brent one dimensional optimisation method

public  static double[] brentf(f_x f,double ax,double bx,double cx,double tol)
{
int ITMAX=100;
double CGOLD=(3.0-Math.sqrt(5))/2.0; //altin oran
double ZEPS=1.0e-10;
double xmin;
double aa[]=new double[2];
            int iter;
            double a,b,d,etemp,fu,fv,fw,fx,p,q,r,tol1,tol2,u,v,w,x,xm;
            double e=0.0;
            d=0.0;
            a=((ax < cx) ? ax : cx);
            b=((ax > cx) ? ax : cx);
            x=w=v=bx;
            fw=fv=fx=f.func(x);
            for (iter=1;iter<=ITMAX;iter++) {
                        xm=0.5*(a+b);
```

```
                tol2=2.0*(tol1=tol*Math.abs(x)+ZEPS);
                if (Math.abs(x-xm) <= (tol2-0.5*(b-a))) {
                        xmin=x;
            aa[0]=xmin;
            aa[1]=fx;
            return aa;
                }
                if (Math.abs(e) > tol1) {
                        r=(x-w)*(fx-fv);
                        q=(x-v)*(fx-fw);
                        p=(x-v)*q-(x-w)*r;
                        q=2.0*(q-r);
                        if (q > 0.0) p = -p;
                        q=Math.abs(q);
                        etemp=e;
                        e=d;
                        if (Math.abs(p) >= Math.abs(0.5*q*etemp) || p <= q*(a-x) || p >= q*(b-x))
                                d=CGOLD*(e=(x >= xm ? a-x : b-x));
                        else {
                                d=p/q;
                                u=x+d;
                                if (u-a < tol2 || b-u < tol2)
                                        d=SIGN(tol1,xm-x);
                        }
                } else {
                        d=CGOLD*(e=(x >= xm ? a-x : b-x));
                }
                u=(Math.abs(d) >= tol1 ? x+d : x+SIGN(tol1,d));
                fu=f.func(u);
                if (fu <= fx) {
                        if (u >= x) a=x; else b=x;
                        {v=w;w=x;x=u;}
                        {fv=fw;fw=fx;fx=fu;}
                } else {
                        if (u < x) a=u; else b=u;
                        if (fu <= fw || w == x) {
                                v=w;
                                w=u;
                                fv=fw;
                                fw=fu;
                        } else if (fu <= fv || v == x || v == w) {
                                v=u;
                                fv=fu;
                        }
                }
        }
        System.out.println("maximum iteration limit in BRENT method is exceeded");
        xmin=x;
        aa[0]=xmin;
        aa[1]=fx;
        return aa;
}

public  static double brent(f_x f,double ax,double bx,double cx,double tol)
{
 double aa[]=brentf(f,ax,bx,cx,tol);
 return aa[0];
```

```
}

// 3. Steepest descent
//_____________________________________________________________________
public static double[] steepestdescent( f_xj f,double x[])
{
// steepestdescent method
// uses linmin and brent methods
double ti=1.0;
int i,ii,jj;
int nmax=500;
double tolerance=1.0e-10;
int n=x.length;
double b[];
b=new double[n];
double lambda;
double dy[]=new double[n];
i=0;
for(i=0;i<n;i++)
{
b[i]=1.0;
}
i=0;
while( i++ < nmax && Matrix.abs(b) > tolerance )
{
   dy=derivative(f,x);
   lambda=linmin(f,x,dy);
   b=Matrix.multiply(dy,lambda);
   x=Matrix.add(x,b);
   System.out.println("i= "+i+"alfa="+lambda+"\n"+Matrix.toString(x));
}
if(i >= nmax) JOptionPane.showMessageDialog(null,"Uyari maximum iterasyon sayisi aşildi \n"+
  " çözüm geçerli olmiyabilir","MAKSİMUM ITERASYON SAYISI UYARISI",JOptionPane.WARNING_MESSAGE);
return x;
}
public  static double[] mnbrak(f_x f,double ax,double dx)
{
// dx ikinci derece polinom için gerekli iki ek noktayi oluşturmak için
// ana noktadan seçilen uzaklik
return mnbrak(f,ax,(ax-dx),(ax+dx));
}

public  static double[] mnbrak(f_x f,double ax)
{
// dx ikinci derece polinom için gerekli iki ek noktayi oluşturmak için
// ana noktadan seçiln uzaklik
return mnbrak(f,ax,(ax-0.1),(ax+0.1));
}

public  static double[] mnbrak(f_x f,double ax,double bx,double cx)
{
// ax,bx,cx üç adet tahmin noktasi verildiğinde içinde minimum bulunan ax,bx,cx üç adetnoktayi bize arama
// sonucu bulur
// f : fonksiyon (f_x abstract sinifinda tanimlanmiş)
// tol tolerans
int ITMAX=100;
double xmin;
```

```
double GOLD=(Math.sqrt(5.0)+1.0)/2.0;;
double GLIMIT=100.0;
double TINY=1.0e-20;
double fa,fb,fc;
fa=0;
fb=0;
fc=0;
        double ulim,u,r,q,fu,dum;
        double aa[]=new double[3];
        fa=f.func(ax);
        fb=f.func(bx);
        if (fb > fa) {
                //SHFT(dum,*ax,*bx,dum)
                {dum=ax;ax=bx;bx=dum;}
                //SHFT(dum,*fb,*fa,dum)
                {dum=fb;fb=fa;fa=dum;}
        }
        cx=(bx)+GOLD*(bx-ax);
        fc=f.func(cx);
                aa[0]=ax;
                aa[1]=bx;
                aa[2]=cx;
        while (fb > fc) {
                r=(bx-ax)*(fb-fc);
                q=(bx-cx)*(fb-fa);
                u=(bx)-((bx-cx)*q-(bx-ax)*r)/
                        (2.0*SIGN(MAX(Math.abs(q-r),TINY),q-r));
                ulim=(bx)+GLIMIT*(cx-bx);
                if ((bx-u)*(u-cx) > 0.0) {
                        fu=f.func(u);
                        if (fu < fc) {
                                ax=bx;
                                bx=u;
                                fa=fb;
                                fb=fu;
                                return aa;
                        } else if (fu > fb) {
                                cx=u;
                                fc=fu;
                                return aa;
                        }
                        u=(cx)+GOLD*(cx-bx);
                        fu=f.func(u);
                } else if ((cx-u)*(u-ulim) > 0.0) {
                        fu=f.func(u);
                        if (fu < fc) {
                                //SHFT(bx,cx,u,cx+GOLD*(cx-bx))
                                {bx=cx;cx=u;u=cx+GOLD*(cx-bx);}
                                //SHFT(fb,fc,fu,f.func(u))
                                {fb=fc;fc=fu;fu=f.func(u);}
                        }
                } else if ((u-ulim)*(ulim-cx) >= 0.0) {
                        u=ulim;
                        fu=f.func(u);
                } else {
                        u=cx+GOLD*(cx-bx);
                        fu=f.func(u);
```

```
                        }
                        //SHFT(ax,bx,cx,u)
                        {ax=bx;bx=cx;cx=u;}
                        //SHFT(fa,fb,fc,fu)
                        {fa=fb;fb=fc;fc=fu;}
                        aa[0]=ax;
                        aa[1]=bx;
                        aa[2]=cx;
            }
                        return aa;
}
public static double linmin(f_xj f,double p[],double xi[])
{
            double tol=2.0e-4;
   int n=p.length;
   double pcom[],xicom[];
            int j;
            double xx,xmin,bx,ax;
            f1dim f1=new f1dim(f,p,xi);
            ax=0.0;
            xx=1.0;
            bx=2.0;
            double aa[]=mnbrak(f1,ax,xx,bx);
            ax=aa[0];
            xx=aa[1];
            bx=aa[2];
            xmin=brent(f1,ax,xx,bx,tol);
            return xmin;
}

public static double[] derivative(f_xj f_deriv,double x[])
{
// df/dxj  j=0...x.length
// This method calculates derivative of a function with more than one independent variable.
// Accuracy of method can be adjusted by changing variables h0 and n
// function input should be in the form given in abstract class
// f_xj,j=0...x.length = df/dx(x_ref)
double a[]=new double[x.length];
for(int x_ref=0;x_ref<x.length;x_ref++)
  {
  a[x_ref]=derivative(f_deriv,x,x_ref);
  }
return a;
}

public static double derivative(f_xj f,double xxi[],int x_ref)
{
// df/dxj
double h0=0.256808;
int i,m;
int n=7;
double x1[];
x1=new double[xxi.length];
double x2[];
x2=new double[xxi.length];
for(i=0;i<xxi.length;i++)
{
```

```
x1[i]=xxi[i];
x2[i]=xxi[i];
}
//derivative of a simple function
double T[][];
T=new double[n][n];
double h[];
h=new double[n];
for(i=0;i<n;i++)
{
  h[i]=0;
  for(int j=0;j<n;j++)
    T[i][j]=0;
}
h[0]=h0;
double r=0.5;
for( i=1;i<n;i++)
{
//h[i]=h0*Math.pow(r,i);
h0/=2.0;
h[i]=h0;
}
//first derivative (difference formula)
for(i=0;i<n;i++)
{
x1[x_ref]+=h[i];
x2[x_ref]-=h[i];
T[i][0]=(f.func(x1)-f.func(x2))/(2.0*h[i]);
x1[x_ref]=xxi[x_ref];
x2[x_ref]=xxi[x_ref];
}

for(m=1;m<n;m++)
{
  for(i=0;i<n-m;i++)
  {
  T[i][m]=(h[i]*h[i]*T[i+1][m-1] -
  h[i+m]*h[i+m]*T[i][m-1])/(h[i]*h[i] - h[i+m]*h[i+m]);
  }
}
double xx=T[0][n-1];
return xx;
}

//input data method
//_____________________________________________________________________
    public static double[] inputdata(String s)
    {
    String s1=JOptionPane.showInputDialog(s);
    StringTokenizer token=new StringTokenizer(s1);
    int n=token.countTokens()-1;
    int m=n+1;
    double a[]=new double[m];
    int j=0;
    while(token.hasMoreTokens())
    {
    Double ax=new Double(token.nextToken());
```

```
    a[j++]=ax.doubleValue();
    }
    return a;
    }

    public static void main (String args[])
    {
    String s="enter optimum point first estimation values with one empty space in between: ";
    double [] x0=inputdata(s);
    f1 f_x=new f1();
    double [] r1= steepestdescent(f_x,x0);
    s=" çözüm seti  : \n"+Matrix.toStringT(r1);
    String s2="Steepest Descent multidimensional non-linear optimisation method:";
    JOptionPane.showMessageDialog(null,s,s2,JOptionPane.PLAIN_MESSAGE);
    System.exit(0);
    }
}
```

```
---------- Capture Output ----------
> "D:\java\bin\javaw.exe" Q4
i= 1alfa=-0.12499999999999663
      3.000000000000080          -0.749999999999960
i= 2alfa=-0.4999999999991899
      3.249999999999493          -0.750000000000232
i= 3alfa=-0.12500000000004022
      3.249999999999455          -0.687500000000146
i= 4alfa=-0.4999999999953388
      3.312499999999653          -0.687500000000252
i= 5alfa=-0.12499999999945541
      3.312499999999654          -0.671875000000081
i= 6alfa=-0.49999999999636197
      3.328124999999928          -0.671875000000081
i= 7alfa=-0.12500000000044853
      3.328125000000008          -0.667968749999971
i= 8alfa=-0.49999999999999994
      3.332031250000195          -0.667968749999919
```

SAYISAL ÇÖZÜMLEME FİNAL
NUMERICAL ANALYSIS FINAL
2009-2010 Spring Semester, 9/6/2010
Time : 2 hours

Öğrenci ismi **Student name**	
Öğrenci no **Student id #**	

This section is for instructor only (Bu bölüme bir şey yazmayınız)

Problem 1	
Problem 2	
Problem 3	
Problem 4	

PROBLEM 1
Solve diferential equation
$\frac{dy}{dx} = \frac{1}{x+y}$ with initial conditions x_0=0 , y_0=2 , at the interval 0-0.4 with step size h=0.2,
with 4th order Runge-Kutta method

---------- Capture Output ----------
> "C:\turhan\java\bin\javaw.exe" NA87
k1=0.5
k2=0.46511627906976744
k3=0.465872156013001
k4=0.43607672682604176
i=0ai=2.0932684532330526

k1=0.43605884805601314
k2=0.41036174265959957
k3=0.4107949303893789
k4=0.38828506085481335
i=1ai=2.1754903617333454

```
//=====================================================
// Numerical Analysis package in java
// example to show differential equation solution
// Runge-Kutta 4rd order solution RK4
// Dr. Turhan Coban
// =====================================================
// dy/dx=f(x,y)

import java.io.*;

abstract class f_xy
{
  //single function single independent variable
  // example f=x*x
  abstract double func(double x,double y);
}

class fxy extends f_xy
{
  double func(double x,double y)
  {return 1.0/(x+y);}
}



class NAFP1
{


  public static double[][] RK4(f_xy f,double x0,double y0,double xn,double h)
  {
  // Runge-Kutta 4
  int n=(int)((xn-x0)/h);
  double a[][]=new double[2][n+1];
  int i=0;
  a[0][0]=x0;
  a[1][0]=y0;
  double k1,k2,k3,k4;
  for(double xx=x0;xx<xn;xx+=h)
   {a[0][i+1]=xx+h;
   k1=f.func(a[0][i],a[1][i]);
   k2=f.func(a[0][i]+0.5*h,a[1][i]+0.5*k1*h);
   k3=f.func(a[0][i]+0.5*h,a[1][i]+0.5*k2*h);
```

```
    k4=f.func(a[0][i]+h,a[1][i]+k3*h);
    a[1][i+1]=a[1][i]+(k1+2.0*k2+2.0*k3+k4)/6.0*h;
    System.out.println("k1="+k1);
    System.out.println("k2="+k2);
    System.out.println("k3="+k3);
    System.out.println("k4="+k4);
    System.out.println("i="+i+"ai="+a[1][i+1]);
    i++;
    }
   return a;
   }

   public static void main(String args[]) throws IOException
   {double x0=0;
    double y0=2.0;
    fxy f2=new fxy();
    double a[][]=RK4(f2,x0,y0,0.4,0.2);
    Text.print(Text.T(a));
    Plot p1=new Plot(a);
    p1.setPlotType(0,24);
    p1.plot();
   }
}
```

0.0 2.0

0.2 2.0932684532330526

0.4 2.1754903617333454

PROBLEM 2

Solve the following integral by n=4 Gauss-Legendre integral

$$I = \int_0^1 \frac{4dx}{1 + 256(x - 0.375)^2}$$

Coefficients:

N	x_k	c_k
4	-0.861136311594052	0.347854845137447
	-0.339981043584856	0.652145154862546
	0.339981043584856	0.652145154862546
	0.861136311594052	0.347854845137447

---------- Capture Output ----------
> "C:\turhan\java\bin\javaw.exe" NAFP2
k1=0.5k2=0.5x=0.06943184420297399y=0.16062189039772795z=0.02793655140499282
k1=0.5k2=0.5x=0.330009478207572y=2.634730839304772z=0.887050027014761
k1=0.5k2=0.5x=0.669990521792428y=0.1718436836502664z=0.9430835398578876
k1=0.5k2=0.5x=0.930568155797026y=0.049990045864377546z=0.9517781796891711
integral of class f1 : 0.9517781796891711
> Terminated with exit code 0.

```
//=====================================================
// Numerical Analysis package in java
// example to show utilisation of integration (integral)
// and differentiation (derivative) functions
// Gauss-Legendre 10 points integration formula
// Dr. Turhan Coban
// =====================================================
import java.io.*;
```

```
class f1 extends f_x
{
  public double func(double x)
  {
  double ff=4.0/(1+256*(x-0.375)*(x-0.375));
  return ff;
  }
}

class NAFP2
{
public static double integral(f_x f_xnt,double a,double b)
{
 double r[],c[];
 int n=4;
 r=new double[n];
 c=new double[n];
 r[ 0] = -0.861136311594052;
 r[ 1] = -0.339981043584856;
 r[ 2] = 0.339981043584856;
 r[ 3] = 0.861136311594052;
 c[ 0] = 0.347854845137447;
 c[ 1] = 0.652145154862546;
 c[ 2] = 0.652145154862546;
 c[ 3] = 0.347854845137447;
double z=0;
double x,y;
double k1=(b-a)/2.0;
double k2=(b+a)/2.0;
for(int i=0;i<n;i++)
{
x=k2+k1*r[i];
y=f_xnt.func(x);
z+=k1*c[i]*y;
System.out.println("k1="+k1+"k2="+k2+"x="+x+"y="+y+"z="+z);
}
return z;
}

  public static void main(String args[]) throws IOException
  {
  //Gauss-Legendre integration formula with n=10
  f1 b1=new f1();
  System.out.println("integral of class         f1 : "+integral(b1,0.0,1.0));
}
}
```

PROBLEM 3

Enthapy of a saturated water will be approximated by the following equation

$$h_f = A + Bt + Ct^2$$

Following data is taken from the thermodynamic table. Find the polynomial coefficient by using polinomial least square

t, ^oC	10	30	50
h_f , kJ/kg	41.99	125.66	209.26

---------- Capture Output ----------
> "C:\turhan\java\bin\javaw.exe" NAFP3
katsayılar :

0.128749999999834
4.187000000000012
-0.000087500000000

Polynomial Least Square Curve fitting

10.0000000000 41.9900000000
16.6666666667 69.8877777778
23.3333333333 97.7777777778
30.0000000000 125.6600000000
36.6666666667 153.5344444444
43.3333333333 181.4011111111
50.0000000000 209.2600000000

```
import java.io.*;
import javax.swing.*;

class NAFP3
{
// Polynomial least square
  public static double[] gausswithpartialpivot(double a[][],double b[])
  { //Gauss elimination with partial pivoting
  int n=b.length;
  double x[]=new double[n];
  double carpan=0;
  double toplam=0;
  double buyuk;
  double dummy=0;
  //gauss elimination
  int i,j,k,p,ii,jj;
  for(k=0;k<(n-1);k++)
  { //partial pivoting
              p=k;
              buyuk=Math.abs(a[k][k]);
              for(ii=k+1;ii<n;ii++)
              {  dummy=Math.abs(a[ii][k]);
                 if(dummy > buyuk) {buyuk=dummy;p=ii;}
              }
              if(p!=k)
              { for(jj=k;jj<n;jj++)
                { dummy=a[p][jj];
                  a[p][jj]=a[k][jj];
                  a[k][jj]=dummy;
                }
                dummy=b[p];
                b[p]=b[k];
                b[k]=dummy;
              }
              //
    for(i=k+1;i<n;i++)
    {  carpan=a[i][k]/a[k][k];
       a[i][k]=0;
       for(j=k+1;j<n;j++)
       {   a[i][j]-=carpan*a[k][j]; }
          b[i]   =b[i]   -carpan*b[k];
              }
  }
  //backward substitution
  x[n-1]=b[n-1]/a[n-1][n-1];
  for(i=n-2;i>=0;i--)
  {
    toplam=0;
    for(j=i+1;j<n;j++)
    {  toplam+=a[i][j]*x[j];}
    x[i]=(b[i]-toplam)/a[i][i];
```

```
  }
  return x;
}

public static double[] PolynomialLSQ(double xi[],double yi[],int n)
{ //Polynomial least square
int l=xi.length;
int i,j,k;
int np1=n+1;
double A[][];
A=new double[np1][np1];
double B[];
B=new double[np1];
double X[];
X=new double[np1];
for(i=0;i<n+1;i++)
  {  for(j=0;j<n+1;j++)
     {if(i==0 && j==0) A[i][j]=l;
     else  for(k=0;k<l;k++) A[i][j] += Math.pow(xi[k],(i+j));
     }
     for(k=0;k<l;k++) { if(i==0)  B[i]+= yi[k];
                  else      B[i] += Math.pow(xi[k],i)*yi[k];}
  }
X=gausswithpartialpivot(A,B);
double max=0;
for(i=0;i<n+1;i++)
 if(Math.abs(X[i]) > max) max = Math.abs(X[i]);
for(i=0;i<n+1;i++)
 if((Math.abs(X[i]/max) > 0) && (Math.abs(X[i]/max) < 1.0e-100)) X[i]=0;
return X;
}

public static double funcPolynomialLSQ(double e[],double x)
{
// this function calculates the value of
// least square curve fitting function
int n=e.length;
double ff;
if(n!=0.0)
  { ff=e[n-1];
  for(int i=n-2;i>=0;i--)
   { ff=ff*x+e[i]; }
  }
else
  ff=0;
return ff;
}

public static double error(double x[],double y[],double e[])
{
//calculates absolute square root error of a least square approach
double n=x.length;
 int k;
 double total=0;
 for(k=0;k<n;k++)
 {
 total+=(y[k]-funcPolynomialLSQ(e,x[k]))*(y[k]-funcPolynomialLSQ(e,x[k]));
 }
total=Math.sqrt(total);
return total;
}

public static double[][]  funcPolynomialLSQ(double xi[],double yi[],int polinomkatsayisi,int aradegersayisi)
 {
            //aradegersayisi:  x--o--o--x--o--o--x zincirinde x deneysel noktalar ise
            // ara değer sayısı 2 dir
```

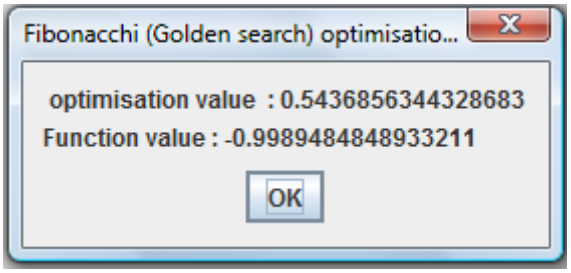
```
            int n=xi.length;
            int nn=(n-1)*(aradegersayisi+1)+1;
            double z[][]=new double[2][nn];
   double E[]=PolynomialLSQ(xi,yi,polinomkatsayisi);
   System.out.println("katsayılar :\n"+Matrix.toStringT(E));
   double dx=0;
   int k=0;
   int i;
   for(i=0;i<(n-1);i++)
     {z[0][k]=xi[i];z[1][k]=funcPolynomialLSQ(E,z[0][k]);k++;
     for(int j=0;j<aradegersayisi;j++)
     {dx=(xi[i+1]-xi[i])/((double)aradegersayisi+1.0);
     z[0][k]=z[0][k-1]+dx;z[1][k]=funcPolynomialLSQ(E,z[0][k]);k++;}
     }
     z[0][k]=xi[i];z[1][k]=funcPolynomialLSQ(E,z[0][k]);
   return z;
 }


  public static void main(String args[]) throws IOException
  {
  double x[];
  double y[];
  String s1=JOptionPane.showInputDialog("Data file name (input1.txt)");
  // JFileChooser fc=new JFileChooser();
  // if (fc.showOpenDialog(null) == JFileChooser.APPROVE_OPTION) {File file =
fc.getSelectedFile();s1=file.getName(); }
  double a[][]=Text.readDoubleT(s1);
  x=a[0];
  y=a[1];
  double z[][]=funcPolynomialLSQ(x,y,2,2);
  System.out.println("Polynomial Least Square Curve fitting\n"+Matrix.toStringT(z));
  Plot pp=new Plot(a[0],a[1]);
  pp.setPlabel("Polynomial Least Square Curve fitting");
  pp.setXlabel("x");
  pp.setYlabel("y=f(x)");
  pp.setPlotType(0,0);
  pp.addData(z[0],z[1]);
  pp.setPlotType(6,22);
  pp.setGrid(1,1);
  pp.plot();
  }
 }
```

PROBLEM 4

Find minimum of function by using Fibonnachi search(Golden search) method in the region $-1 \leq x \leq 4$

$$f(x) = \frac{-4.1xe^{-x}}{x^2+1}$$



---------- *Capture Output* ----------

> "C:\turhan\java\bin\javaw.exe" NAFP4

x=0.5-0.9947102819287187

x=1.5-0.422230918434721

```
// OPTIMISATION Golden search or fibonnachi method
import java.util.*;
import java.awt.*;
import java.applet.Applet;
import java.awt.event.*;
import javax.swing.*;
class fb extends f_x
{
public double func(double x)
{
//çözümü istenen fonksiyon
double ff;
ff=-4.1*x*Math.exp(-x)/(x*x+1);
return ff; //maksimum testi
}
}

public class NAFP4
{
public static double golden(f_x f,double a,double b,double epsilon)
{return golden(f,a,b,epsilon,1.0e-10,0);}
public static double golden(f_x f,double a,double b)
{return golden(f,a,b,1.0e-10,1.0e-10,0);}

public static double golden(f_x f,double a,double b,double epsilon,double delta,int print)
{
//  find the minimum of the function
// note  maximum f(x) = minimum (-f(x))
   double r1 = (Math.sqrt(5.0)-1.0)/2.0; // golden ratio
   double r2 = r1*r1;
   double h = b - a;
   double ya = f.func(a);
   double yb = f.func(b);
   double c = a + r2*h;
   double d = a + r1*h;
   double yc = f.func(c);
   double yd = f.func(d);
   int k = 1;
   double dp,dy,p,yp;
   while ((Math.abs(yb-ya)>epsilon) || (h>delta))
   {
     k++;
     if (yc<yd)
       {
       b = d;
       yb = yd;
       d = c;
       yd = yc;
       h = b - a;
```

```
        c = a + r2 * h;
        yc = f.func(c);
        }
      else
      {
        a = c;
        ya = yc;
        c = d;
        yc = yd;
        h = b - a;
        d = a + r1 * h;
        yd = f.func(d);
      }//end of if
   }//end of while
   dp = Math.abs(b-a);
   dy = Math.abs(yb-ya);
   p = a;
   yp = ya;
   if (yb<ya)
   {
    p = b;
    yp = yb;
   }
   if(print==1)
   {System.out.println("x min = "+p+"ymin = "+yp+"error of x ="+dp+"error of y"+dy); }
   return p;
}

    public static void main (String args[])
    {
            fb f=new fb();
    double a,b;
    a=Double.parseDouble(JOptionPane.showInputDialog(" enter lower limit of the function a : "));
    b=Double.parseDouble(JOptionPane.showInputDialog(" enter upper limit of the function b : "));
    double r;
    System.out.println("x=0.5"+f.func(0.5));
    System.out.println("x=1.5"+f.func(1.5));
    r= golden(f,a,b,1.0e-10,1.0e-5,0);
    JOptionPane.showMessageDialog(null," optimisation value  : "+r+"\nFunction value : "+f.func(r),
    "Fibonacchi (Golden search) optimisation : ",JOptionPane.PLAIN_MESSAGE);
    System.exit(0);
    }
}
```

SAYISAL ÇÖZÜMLEME
(NUMERICAL ANALYSIS)
Final Examination June 2011

Student Name	

Student id #	

QUESTION 1	
QUESTION 2	
QUESTION 3	
QUESTION 4	
TOTAL	

QUESTION 1

a) Find the inverse matrix of A
b) Multiply inverse matrix(A^{-1}) with the original matrix(A) to find unit matrix(I) and be ensure that you do not have an error
c) Calculate smallest eigenvalue of the given matrix by using inverse power method

$$\boldsymbol{A} = \begin{bmatrix} \mathbf{1} & \mathbf{1} \\ \mathbf{1} & \mathbf{2} \end{bmatrix}$$

QUESTION 2

Solve differential equation

$$\frac{dy}{dx} = x^2 - y$$

With range of x=0 and x=0.2. Initial value is given as y(0)=1 .Solve it with Runge-Kutta with second degree polynomial solution **(Ralston method)** with step size h=0.1

Runge-Kutta with second degree Ralston polynomial solution RK2:

$$y_{i+1} = y_i + \left(\frac{1}{3}k_1 + \frac{2}{3}k_2\right)h$$
$$k_1 = f(x_i, y_i)$$
$$k_2 = f\left(x_i + \frac{3}{4}h, y_i + \frac{3}{4}k_1 h\right)$$

QUESTION 3

Find the derivative of the function given in the table at x=3.1

x	1.2	3	3.2	5
f(x)	1.807	0.7468	0.6522	0.1684

QUESTION 4

Fit a second degree polynomial least square curve to the given data

x	1.2	3	3.2	5
f(x)	1.807	0.7468	0.6522	0.1684

$$\begin{Bmatrix} n & \sum_{i=1}^{n} x_i & \sum_{i=1}^{n} x_i^2 \\ \sum_{i=1}^{n} x_i & \sum_{i=1}^{n} x_i^2 & \sum_{i=1}^{n} x_i^3 \\ \sum_{i=1}^{n} x_i^2 & \sum_{i=1}^{n} x_i^3 & \sum_{i=1}^{n} x_i^4 \end{Bmatrix} \begin{bmatrix} a_0 \\ a_1 \\ a_2 \end{bmatrix} = \begin{bmatrix} \sum_{i=1}^{n} f_i \\ \sum_{i=1}^{n} x_i f_i \\ \sum_{i=1}^{n} x_i^2 f_i \end{bmatrix}$$

NUMERİCAL ANALYSİS MIDTERM EXAM APRIL 2012

Exam time : 2 hours

Student name	
Student id #	
Q1	
Q2	
Q3	
TOTAL	

QUESTION 1 : The concentration of pollutant of bacteria c in a lake decreases according to

$$c = 75e^{-1.5t} + 20e^{-0.07t}$$

Determine the time required for the bacteria concentration to be reduced to 15 by using the Newton-Raphson method with an initial guess of t=6.

---------- Capture Output ----------

> "D:\java\bin\javaw.exe" NA5

i=0x=3.693371481870112fx=-2.2381812322580323dfx = -0.9703258303910378

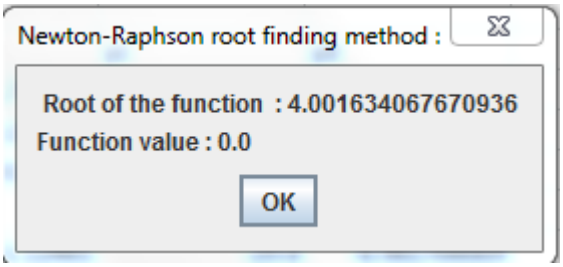
i=1x=3.9818962075928357fx=0.4555192261064658dfx = -1.57878748507754

i=2x=4.001563209176108fx=0.02751950754833743dfx = -1.399273164839888

i=3x=4.001634066766306fx=9.844270299552704E-5dfx = -1.3893035696110827

i=4x=4.001634067670936fx=1.2567724638756772E-9dfx = -1.3892680969074889

i=5x=4.001634067670936fx=0.0dfx = -1.3892680964571649



QUESTION 2 :

Fit a **second degree** polynomial least square curve to the given data

x	1.0	2.0	3.0	4.0
f(x)	1.8	0.75	0.65	0.17

$$\begin{Bmatrix} n & \sum_{i=1}^{n} x_i & \sum_{i=1}^{n} x_i^2 \\ \sum_{i=1}^{n} x_i & \sum_{i=1}^{n} x_i^2 & \sum_{i=1}^{n} x_i^3 \\ \sum_{i=1}^{n} x_i^2 & \sum_{i=1}^{n} x_i^3 & \sum_{i=1}^{n} x_i^4 \end{Bmatrix} \begin{bmatrix} a_0 \\ a_1 \\ a_2 \end{bmatrix} = \begin{bmatrix} \sum_{i=1}^{n} f_i \\ \sum_{i=1}^{n} x_i f_i \\ \sum_{i=1}^{n} x_i^2 f_i \end{bmatrix}$$

Note: Please double check your numbers be sure that your calculations are correct

	xi	fi	xi^2	xi^3	xi^4	xifi	xi^2fi
	1	1.8	1	1	1	1.8	1.8
	2	0.75	4	8	16	1.5	3
	3	0.65	9	27	81	1.95	5.85
	4	0.17	16	64	256	0.68	2.72
total	**10**	**3.37**	**30**	**100**	**354**	**5.93**	**13.37**

	A			B
	4	10	30	3.37
	10	30	100	5.93
	30	100	354	13.37

c	4	10	30	3.37
2.5	0	5	25	-2.495
7.5	0	25	129	-11.905

c	4	10	30	3.37
5	0	5	25	-2.495
	0	0	4	0.57

x1	0.1425
x2	-1.2115
x3	2.8025

QUESTION 3

Find the minimum of function

$$f(x) = 4x - 1.8x^2 + 1.2x^3 - 0.3x^4$$

By using Golden-section search (xl = -2 ; xu=4)

R	0.618033988749895000		function f(x)=x^2*3*x				
a	b	h	c	d	yc	yd	
-2.000000000000000000	4.000000000000000000	6	0.291796	1.708204	1.041562423	5.007504445	
-2.000000000000000000	1.708203932499370000	3.708204	-0.58359	0.291796	-	1.041562423	
-2.000000000000000000	0.291796067500631000	2.291796	-2	-0.58359	-29.6	-3.220722137	

-2.000000000000000000	-0.583592135001261000	1.416408	-2	-2	-29.6	-29.6	
-2.000000000000000000	-0.583592135001261000	1.416408	-2	-1.12461	-29.6	-8.961704909	
-2.000000000000000000	-1.124611797498110000	0.875388	-2	-2	-29.6	-29.6	
-2.000000000000000000	-1.124611797498110000	0.875388	-2	-1.45898	-29.6	-14.75349945	

W1 NUMERICAL ANALYSIS 2019-2020 FALL

Grading:

Class exercises : %30

Homework exercises %20

Final exam (written in class exam) open book %50

Books:

Numerical Analysis with example programs. M. Turhan ÇOBAN

http://www.turhancoban.com/kitap/NA_with%20example%20problems.pdf

Numerical Methods for Engineers, Steven C.Chapra, Raymond P. Canale, Mc Graw Hill publication

Numerical Analysis, Richard L. Burden J. Douglas Faires, Thomson Brooks/Cole publications

SERIES AND ALGORİTHMS

CLASS EXERCISES

Class exercises will be completed and graded in class

1. $\sin(x) = \sum_{k=0}^{\infty}(-1)^k \frac{x^{2k+1}}{(2k+1)!}$
2. $\sin h(x) = \sum_{k=0}^{\infty} \frac{x^{2k+1}}{(2k+1)!}$
3. $\cos(x) = \sum_{k=0}^{\infty}(-1)^k \frac{x^{2k}}{(2k)!}$

Sin(x) function is defined in java,c++, python and octave write them down and run the programs

EX 1

Java version

```
public class sin
{
  public static double sin(double x)
  {double total=0;
   double factorial=1;
   double power=x;
   int plusminus=1;
   int i=0;
   for(int k=1;k<=200;k++)
   {total=total+power/factorial*plusminus;
    power=power*x*x;
    i=2*k+1;
    factorial=factorial*(i-1)*i;
    plusminus*=-1;
   }
   return total;
  }
  public static void main(String arg[])
  {   double x=Math.PI/3.0;
              System.out.println("sin = "+sin(x)+" Math.sin = "+Math.sin(x));}
  }
```

C++ version

```
#include <stdio.h>
#include <iostream>
```

```
#include <math.h>
using namespace std;

  double sin1(double x)
  {double total=0;
   double factorial=1;
   double power=x;
   int plusminus=1;
   int i=0;
   for(int k=1;k<=200;k++)
   {total=total+power/factorial*plusminus;
    power=power*x*x;
    i=2*k+1;
    factorial=factorial*(i-1)*i;
    plusminus*=-1;
   }
   return total;
  }
  int main()
  {             double PI=atan(1.0)*4.0;
                 double x=PI/3.0;
                 cout<<"sin = "<<sin1(x)<<" Math.sin = "<<sin(x);
  }
```

Python version

```
 import math;

def sin(x):
    total=0;
    factorial=1;
    power=x;
    plusminus=1;
    i=0;
    for k in range(1,80):
        total=total+power/factorial*plusminus;
        power=power*x*x;
        i=2*k+1;
        factorial=factorial*(i-1)*i;
        plusminus*=-1;
    return total;

pi=math.pi;
x=pi/3.0;
print ("sin="+str(sin(x))+"math.sin = "+str(math.sin(x)));
```

Octave(matlab) version

Octave editor (write the fuction sin1)

```
 function total=sin1(x)
% sin function
    total=0;
    factorial=1;
    power=x;
    plusminus=1;
    for k=1:100
        total=total+power/factorial*plusminus;
        power=power*x*x;
        i=2*k+1;
        factorial=factorial*(i-1)*i;
        plusminus=-1*plusminus;
    end
end
```

Octave command window

```
>> format long
>> x=pi/3
x =  1.047197551196598
>> sin1(x)
ans =    8.660254037844385e-01
>> sin(x)
ans =    8.660254037844386e-01
```

EX2

As a second example write cos(x) in java,c++, python and octave write them down and run the programs (code is not iven to you, but please note that it is very similar to sinüs function)

HOMEWORK EXERCISES

Homework exercises will be done at home and will bring to next Thursday class printed no late exercises will be excepted. Each code should include student name id#, code plus results should be given. Homeworks will be accepted in written format plus a computer copy in pdf format will be sent to numerical_analysis@turhancoban.com adress your file name should be "group"+"week#"+studentname+studentid#.pdf

A_W1_turhan_coban_0101333.pdf

B_W3_ali_veli_02335646.pdf

W1HW1 : write arcsin(x) in java,c++, python and octave write them down and run the programs

1. $\arcsin(x) = \sum_{k=0}^{\infty} \frac{(2k)!x^{2k+1}}{(2^k k!)^2(2k+1)} \qquad x^2 \leq 1$

W1HW2 : write erf(x) in java,c++, python and octave write them down and run the programs

2. $\text{erf}(x) = \frac{2}{\sqrt{\pi}} \sum_{n=0}^{\infty} (-1)^n \frac{x^{2n+1}}{n!(2n+1)}$

W2 NUMERICAL ANALYSIS 2019-2020 FALL

Grading:

Class exercises : %30

Homework exercises %20

Final exam (written in class exam) open book %50

Books:

Numerical Analysis with example programs. M. Turhan ÇOBAN

http://www.turhancoban.com/kitap/NA_with%20example%20problems.pdf

Numerical Methods for Engineers, Steven C.Chapra, Raymond P. Canale, Mc Graw Hill publication

Numerical Analysis, Richard L. Burden J. Douglas Faires, Thomson Brooks/Cole publications

ROOT FINDING f(x)=0 x=?

CLASS EXERCISES

Class exercises will be completed and graded in class

EX 1

Java version

A reference empty function Shell (this program will be given to you it is existed in SCO1 directory.

interface if_x

```
import static java.lang.Math.*;
import java.io.*;
import java.util.*;
import javax.swing.*;
import javax.swing.table.*;
import java.awt.*;
  // single function single independent variable
  // example f=x*x
  // includes full set of derivatives
  // Reference :"Generation of Finite Difference Formulas on Arbitrary Spaced Grids",
  // Bength Fornberg, Mathematics of Computation, Volume 51, Number 184, October 1988
  // pages 699-706
  //interface version
@FunctionalInterface
interface if_x
{  public double func(double x);
//first order derivative
default double dfunc(double x)
{double h=1.0e-3;
 int n=1;
 int M=10;
 return dfunc(x,n,M,h);
}
//second order derivative
default double dfunc2(double x)
{double h=1.0e-3;
 int n=2;
 int M=10;
 return dfunc(x,n,M,h);
}
default double dfunc(double x,int N,int Mi,double hi)
{// order of the maximum derivative
 // N order of derivative
 // M degree of difference formula
double c[][][];
double alpha[];
double h;
int M;
double a[]=new double[0];
h=0.01;
double x0=0;
M=20;
double alphai[]={0,1,-1,2,-2,3,-3,4,-4,5,-5,6,-6,7,-7,8,-8,9,-9,10,-10,11,-11,12,-12,13,-13,14,-14,15,
-15,16,-16,17,-17,18,-18,19,-19,20,-20,21,-21,22,-22,23,-23,24,-24,25,-25,26,-26,27,-27,28,-28,29,-29,30,-30,
31,-31,32,-32,33,-33,34,-34,35,-35,36,-36,37,-37,38,-38,39,-39,40,-40,41,-41,42,-42,43,-43,44,-44,45,-45,46,
-46,47,-47,48,-48,49,-49,50,-50,51,-51,52,-52,53,-53,54,-54,55,-55,56,-56,57,-57,58,-58,59,-59,60,-60,
-61,61,-62,62,-63,63,-64,64,-65,65,-66,66,-67,67,-68,68,-70,70,-71,71,-72,72,-73,73,-74,74,-75,75,
-76,76,-77,77,-78,78,-79,79,-80,80,-81,81,-82,82,-83,83,-84,84,-85,85,-86,86,-87,87,
-88,88,-89,89,-90,90,-91,91,-92,92,-93,93,-94,94,-95,95,-96,96,-97,97,-98,98,-99,99,-100,100};
alpha=alphai;
int N1=alpha.length-1;
// M   degree of highest derivative
// N+1 number of coefficients
double delta[][][]=new double[N1+1][N1+1][M+1];
double c1,c2,c3;
delta[0][0][0]=1.0;
c1=1.0;
for(int n=1;n<=N1;n++)
{  c2=1;
```

```
   for(int nu=0;nu<=(n-1);nu++)
   {c3=alpha[n]-alpha[nu];
    c2=c2*c3;
    if(n<=M) delta[n-1][nu][n]=0.0;
    for(int m=0;m<=Math.min(n,M);m++)
    {
       if(m==0)
       { delta[n][nu][m]=((alpha[n]-x0)*delta[n-1][nu][m])/c3;}
       else
       {delta[n][nu][m]=((alpha[n]-x0)*delta[n-1][nu][m]-m*delta[n-1][nu][m-1])/c3;}
    }//next m
   }//next nu
   for(int m=0;m<=Math.min(n,M);m++)
   {   if(m==0)
                {delta[n][n][m]=c1/c2*(-(alpha[n-1]-x0)*delta[n-1][n-1][m]);}
                else
                {delta[n][n][m]=c1/c2*(m*delta[n-1][n-1][m-1]-(alpha[n-1]-x0)*delta[n-1][n-1][m]);}
   }//next m
   c1=c2;
}//next n
c=delta;
  if(Mi<N) M=N;
  else M=Mi;
  h=hi;
  double deriv=0;
  double h1=1/h;
  double h2=1;
  for(int j=0;j<N;j++)
  {h2*=h1;}
  for(int i=0;i<c[0].length;i++)
  {   deriv+=c[M][i][N]*func(x+alpha[i]*h);}
  return deriv*h2;}
default double dfunc(double x,int N)
{int M=30;double h=0.05*N;return dfunc(x,N,M,h);}
}
```

Bisection method(java version)

```
 import java.io.*;
import javax.swing.*;
import java.awt.*;
class bisection
{
//Bisection root finding method f(x)=0 x=? given initial guess limits a and b
public static double bisection(if_x f,double a,double b)
  {double b1=1.1*b;
   double r=(a+b)/2.0;
   double eps=1.0e-6;
   int nmax=100;
   int i=0;
   while(Math.abs(f.func(r))>eps && i<nmax)
   {if(f.func(a)*f.func(r)<0) b=r;
    else a=r;
    r=(a+b)/2.0;;
    i++;
   }
   if(i>=nmax) r=bisection(f,a,b1);
   return r;
  }
public static void main (String args[]) throws java.io.IOException
{    double a,b;
     a=Double.parseDouble(JOptionPane.showInputDialog(" INPUT LOWER LİMİT OF SEARCH AREA a : "));
     b=Double.parseDouble(JOptionPane.showInputDialog(" INPUT UPPER LIMIT OF SEARCH AREA b : "));
     double r;
     if_x f1=x->x*x-2.0;
     r= bisection(f1,a,b);
```

```
    String s1=" ROOT  : "+r+"\nFUNCTION VALUE : "+f1.func(r);
    JOptionPane.showMessageDialog(null,s1);
    System.exit(0);
}
}
```

C++ version

```
 #include <stdio.h>
#include <iostream>
#include <math.h>
#include <iomanip>
using namespace std;
// interface if_x
  class if_x{
  public: virtual double func(double x)=0;
  public: double dfunc(double x)
  { double h=0.0001;
    double hh=1.0/h;
    double df=(3.0*func(x-4.0*h)-32.0*func(x-3.0*h)+168.0*func(x-2.0*h)-672.0*func(x-h)+672.0*func(x+h)-168.0*func(x+2.0*h)
  +32.0*func(x+3.0*h) - 3.0*func(x+4.0*h))/840.0*hh;
    return df;
  }};

double bisection(if_x &f,double a,double b)
  {double b1=1.1*b;
   double r=(a+b)/2.0;
   double eps=1.0e-6;
   int nmax=100;
   int i=0;
   while(abs(f.func(r))>eps && i<nmax)
   {if(f.func(a)*f.func(r)<0) b=r;
    else a=r;
    r=(a+b)/2.0;
    i++;
   }
   if(i>=nmax) r=bisection(f,a,b1);
   return r;
  }
int main()
{  double a,b,r;
   class f1:public if_x{public:double func(double x){ return x*x-2; };};
   f1 f;
   cout << " INPUT LOWER LIMIT OF SEARCH AREA a : ";
   cin >>a;
   cout << " INPUT UPPER LIMIT OF SEARCH AREA b : ";
   cin >>b;
   r=bisection(f,a,b);
   cout<<setprecision(20);
   cout<<"r="<<r;
   return 0;
}
```

Python version

interface(abstract method) if_x

```
from abc import ABC, abstractmethod
class if_x(ABC):

    @abstractmethod
    def func(self,x):
        pass;

    def dfunc(self,x,n):
        dx=0.001;
```

```
        if n==0: df=self.func(x);
        elif n==1: df=(3.0*self.func(x-4.0*dx)-32.0*self.func(x-3.0*dx)+168.0*self.func(x-2.0*dx)-672.0*self.func(x-
dx)+672.0*self.func(x+dx)-168.0*self.func(x+2.0*dx)+32.0*self.func(x+3.0*dx)-3.0*self.func(x+4.0*dx))/840.0/dx;
        elif n==2: df=(-14350.0*self.func(x)-9.0*self.func(x-4.0*dx)+128.0*self.func(x-3.0*dx)-1008.0*self.func(x-
2.0*dx)+8064.0*self.func(x-dx)+8064.0*self.func(x+dx)-1008.0*self.func(x+2.0*dx)+128.0*self.func(x+3.0*dx)-
9.0*self.func(x+4.0*dx))/5040.0/dx/dx;
        elif n==3: df=(-7.0*self.func(x-4.0*dx)+72.0*self.func(x-3.0*dx)-338.0*self.func(x-2.0*dx)+488.0*self.func(x-dx)-
488.0*self.func(x+dx)+338.0*self.func(x+2.0*dx)-72.0*self.func(x+3.0*dx)+7.0*self.func(x+4.0*dx))/240.0/dx/dx/dx;
        elif n==4: df=(2730.0*self.func(x)+7.0*self.func(x-4.0*dx)-96.0*self.func(x-3.0*dx)+676.0*self.func(x-2*dx)-1952.0*self.func(x-
dx)-1952.0*self.func(x+dx)+676.0*self.func(x+2.0*dx)-96.0*self.func(x+3.0*dx)+7.0*self.func(x+4.0*dx))/240.0/dx/dx/dx/dx;
        elif n==5: df=(self.func(x-4.0*dx)-9.0*self.func(x-3.0*dx)+26.0*self.func(x-2.0*dx)-29.0*self.func(x-dx)+29.0*self.func(x+dx)-
26.0*self.func(x+2.0*dx)+9.0*self.func(x+3.0*dx)-self.func(x+4.0*dx))/6.0/dx/dx/dx/dx/dx;
        elif n==6: df=(-150.0*self.func(x)-self.func(x-4.0*dx)+12.0*self.func(x-3.0*dx)-52.0*self.func(x-2.0*dx)+116.0*self.func(x-
dx)+116.0*self.func(x+dx)-52.0*self.func(x+2.0*dx)+12.0*self.func(x+3.0*dx)-self.func(x+4.0*dx))/4.0/dx/dx/dx/dx/dx/dx;
        elif n==7: df=(-self.func(x-4.0*dx)+6.0*self.func(x-3.0*dx)-14.0*self.func(x-2.0*dx)+14.0*self.func(x-dx)-
14.0*self.func(x+dx)+14.0*self.func(x+2.0*dx)-6.0*self.func(x+3.0*dx)+self.func(x+4.0*dx))/2.0/dx/dx/dx/dx/dx/dx/dx;
        elif n==8: df=(70.0*self.func(x)+self.func(x-4.0*dx)-8.0*self.func(x-3.0*dx)+28.0*self.func(x-2.0*dx)-56.0*self.func(x-dx)-
56.0*self.func(x+dx)+28.0*self.func(x+2.0*dx)-8.0*self.func(x+3.0*dx)+self.func(x+4.0*dx))/dx/dx/dx/dx/dx/dx/dx/dx;
        else: df=0;
        return df;
```

```
 import math;

def sin(x):
    total=0;
    factorial=1;
    power=x;
    plusminus=1;
    i=0;
    for k in range(1,80):
        total=total+power/factorial*plusminus;
        power=power*x*x;
        i=2*k+1;
        factorial=factorial*(i-1)*i;
        plusminus*=-1;
    return total;

pi=math.pi;
x=pi/3.0;
print ("sin="+str(sin(x))+"math.sin = "+str(math.sin(x)));
```

Octave(matlab) version

Octave editor (write the fuction sin1)

```
 function xr=bisection(f,xl,xu)
% bisection root finding method
maxit=100;
iter=0;
es=0.0000001;
ea=1.1*es;
while((ea>es)&&(iter<maxit))
    xr=(xl+xu)/2.0;
    iter=iter+1;
    if xr~=0 ea=abs((xu-xl)/(xu+xl))*100;end
    fxl= f(xl);
    fxr= f(xr);
    test= fxl*fxr;
    if test == 0.0      ea=0;
    elseif test < 0.0  xu=xr;
    else               xl=xr;
    end
end
if(iter>=maxit) fprintf('Maximum number of iteration is exceeded result might not be valid","MAKSİMUM NUMBER OF ITERATION
```

```
WARNING');
end
end
```

Octave command window

```
>> format long
>> f=@(x)x*x-2;
>> bisection(f,1,3)
ans =  1.414213561452925
```

EX2: Secant method Java version

```
// ROOT FINDING Secant method
import java.util.*;
import java.awt.*;
import java.applet.Applet;
import java.awt.event.*;
import javax.swing.*;

public class secant
{
public static double secant(if_x f,double x)
{ double eps=1.0e-10;
  double y=f.dfunc(x);
  double dy=f.dfunc2(x);
  int miter=100;
  int i=0;
  while(Math.abs(y)>eps && i<miter)
  {x=x-y/dy;
   y=f.func(x);
   dy=f.dfunc(x);
   i++;
  }
  if(i>=miter)
    System.out.println("i="+i+"results may not be valid");
  return x;
}
    public static void main (String args[])
    { if_x f=x->x*x-2;
    double x=Double.parseDouble(JOptionPane.showInputDialog(" enter lower limit of the function a : "));
    double r= secant(f,x);
    JOptionPane.showMessageDialog(null," optimisation value  : "+r+"\nFunction value : "+f.func(r),
    "secant optimisation : ",JOptionPane.PLAIN_MESSAGE);
    Plot p=new Plot(f,0,3);
    p.plot();
    }
}
```

Secant method C++ version

```
#include <stdio.h>
#include <iostream>
#include <math.h>
#include <iomanip>
using namespace std;

class if_x{
public: virtual double func(double x)=0;
public: double dfunc(double x)
{ double h=0.0001;
  double hh=1.0/h;
  double df=(3.0*func(x-4.0*h)-32.0*func(x-3.0*h)+168.0*func(x-2.0*h)-672.0*func(x-h)+672.0*func(x+h)-168.0*func(x+2.0*h)
+32.0*func(x+3.0*h) - 3.0*func(x+4.0*h))/840.0*hh;
  return df;
```

```
}
};

double secant(if_x &f,double x)
{
int nmax=100;
double tolerance=1.0e-8;
double fx,dfx;
for(int i=0;i<nmax;i++)
{ fx=f.func(x);
  dfx=f.dfunc(x);
  x-=fx/dfx;
  cout <<"i="<<i<<"x="<<x<<"fx="<<fx<<"dfx="<<dfx<<"\n";
  if(fabs(fx)<tolerance) { return x;}
}
cout <<("Maximum number of iteration is exceeded \n result might not be valid");
return x;
}
int main()
{  double x,r;
   class f1:public if_x{public:double func(double x){ return x*x-2; };};
   f1 f;
   cout << " input initial guess x : ";
   cin >>x;
   r=secant(f,x);
   cout<<setprecision(20);
   cout<<"r="<<r;
   return 0;
}
```

Secant method python version

```
from math import *
from if_x import *;

def secant(f,x):
   # secant root finding method
  nmax=100
  tolerance=1.0e-10
  for i in range(0,nmax):
      fx= f.func(x);
      dfx=f.dfunc(x,1);
      x=x-fx/dfx
      if abs(fx)<tolerance: return x
  return x

class f1(if_x):func=lambda self,x:x*x-2;
f=f1();
x=float(input("enter initial guess : "))
r=secant(f,x);
print(" ROOT  : ",r,"\nFUNCTION VALUE : ",f.func(r));
```

Secant method octave version editor

```
function p=secant(f,x,tolerance,nmax)
% secant method for root finding
if nargin<4 nmax=100;end
if nargin<3 tolerance=1e-10;end
i=0;
fx=f(x);
while (abs(fx) > tolerance) && i<nmax
    fx=f(x);
    dfx=df(f,x);
```

```
   x=x-fx/dfx;
   i=i+1;
end
p=x;
if i>=nmax fprintf('Maximum number of iterations is exceeded the result may not be valid');end
end
```

Command window

```
>> format long
>> f=@(x)x*x-2;
>> secant(f,1)
ans =  1.414213562373095
>>
```

HOMEWORK EXERCISES

Homework exercises will be done at home and will bring to next Thursday class printed no late exercises will be excepted. Each code should include student name id#, code plus results should be given. Homeworks will be accepted in written format plus a computer copy in pdf format will be sent to numerical_analysis@turhancoban.com adress your file name should be "group"+"week#"+studentname+studentid#.pdf

A_W1_turhan_coban_0101333.pdf

B_W3_ali_veli_02335646.pdf

W1HW1 : function f(x) $= x * x - 5 * x + 1$ is given find the root by using

a) Newton raphson method by hand
b) Secant method by hand
c) Secant method by computer program (you can select your language)
d) Bisection method by computer program (you can select your language)

W1HW2 : function f(x) $= \sin(x)$ is given find the root in the range of $0.2 \leq x \leq 2.5$ by using

a) Newton raphson method by hand
b) Bisection method by hand
c) Secant method by computer program (you can select your language)
d) Bisection method by computer program (you can select your language)

W3 NUMERICAL ANALYSIS 2019-2020 FALL

ROOT FINDING f(x)=0 x=?

CLASS EXERCISES

Class exercises will be completed and graded in class

EX 1 Horner's Method (uses synthetic division) Roots of degree n polynomial

$\boldsymbol{f(x) = \sum_{i=0}^{n} a_i x^i = a_0 + a_1 x + a_2 x^2 + a_3 x^3 + ..}$

Java version c.java class contains complex class and complex algebra

```
 import javax.swing.*;
import java.util.*;
public class horner
{
c CA[];//complex polynomial coefficient
public static c f_poly(c c1[],c x)
{
// this function calculates the value of
// c least square curve fitting function
```

```
int n=c1.length;
c ff;
if(n!=0.0)
  { ff=new c(c1[n-1]);
  for(int i=n-2;i>=0;i--)
   { ff=c.p(c.p(ff,"*",x),"+",c1[i]); }
  }
else
  ff=new c(0,0);
return ff;
}
public double[] toDouble()
{int n=CA.length;
 double d[]=new double[n];
 for(int i=0;i<n;i++)
    {   d[i]=c.toDouble(CA[i]);}
    return d;}
public c horner(double a[], c x)
{   int n=a.length;
                c c1[]=new c[n];
    for(int i=0;i<n;i++) c1[i]=new c(a[i]);
    return horner(c1,x);}
public c horner(c c3[], c x)
{   int n=c3.length;
    int iter=0;
    int nmax=200;
    boolean b=true;
    int j=0;
                c c1[]=new c[n];
                int n1=n-1;
                c cp[]=new c[n1];
                c1[n-1]=new c(c3[n-1]);
    for(int i=n-2;i>=0;i--)
    {   c c2=new c(c3[i]);
                    c1[i]=c.p(c2,"+",c.p(x,"*",c1[i+1]));
    }
                c P=c1[0];
                for(int i=n1-1;i>=0;i--)
                {cp[i]=c1[i+1];}
                CA=cp;
                c Q=f_poly(cp,x);
                //Newton's method
                x=c.p(x,"-",c.p(P,"/",Q));//x=x-P/Q
                iter++;
                if(c.abs(P)<1.0e-20 || c.abs(P)==0 ||iter>nmax)
                {return x;}
                else
                {
                while(b)
                {
                c1[n-1]=new c(c3[n-1]);
    for(int i=n-2;i>=0;i--)
    {   c c2=new c(c3[i]);
                    c1[i]=c.p(c2,"+",c.p(x,"*",c1[i+1]));
                }
                P=c1[0];
                for(int i=n1-1;i>=0;i--)
                {cp[i]=c1[i+1];}
                CA=cp;
    Q=f_poly(cp,x);
                if(c.abs(P)<1.0e-20||iter>nmax)
                {b=false;
                 return x;
                }
```

```
                //Newton's method
                x=c.p(x,"-",c.p(P,"/",Q));
                iter++;
   }
}
   return x;
}

public String toString()
{int n=CA.length;
 String s="";
 for(int i=n-1;i>=0;i--) {s+=CA[i]+" ";}
 return s;
}
public static String toString(c x[])
{int n1=x.length;
 String s="";
 for(int i=n1-1;i>=0;i--) {s+="x["+i+"] = "+x[i]+"\n";}
 return s;
}
public c[] root(double a[])
{   int n=a.length;
    c y[]=new c[n-1];
                c x=new c(4.0);
                c x1=horner(a, x);
                int i=n-2;
    y[i]=x1;
    i--;
    int nn=CA.length;
    while(i>=0)
    {  if(nn!=CA.length)
       {c D[]=new c[nn];
       for(int j=0;j<nn;j++) {D[j]=CA[j];}
       CA=D;
       }
     {c z=horner(CA,x);
      y[i]=z;
     i--;
     nn--;
     }
     }
    return y;
}
     public static double[] enterdata(String s)
     {
     String s1=JOptionPane.showInputDialog(s);
     StringTokenizer token=new StringTokenizer(s1);
     int n=token.countTokens()-1;
     int m=n+1;
     double a[]=new double[m];
     int j=0;
     while(token.hasMoreTokens())
     {
     Double ax=new Double(token.nextToken());
     a[j++]=ax.doubleValue();
     }
     return a;
     }
public static void calculate_Horner()
{   horner h=new horner();
                String s="enter coefficients of the n degree polynomial a[0] a[1]..a[n] y=a[0]+a[1]*x+a[2]*x^2+a[3]*x^3+..+a[n]*x^n ";
    double [] x0=enterdata(s);
    int n=x0.length;
    String s1[]=new String[n];
```

```
    String s2=" coefficients of the n degree polynomial y=a[0]+a[1]*x+a[2]*x^2+a[3]*x^3+..+a[n]*x^n ";
    for(int i=0;i<n;i++)
    {s1[i]="a["+i+"]";}
    JOptionPane.showMessageDialog(null,toString(h.root(x0)),
    "roots of the nth degree polynomial by Horner's synthetic division & Newton-Raphson method : ",JOptionPane.PLAIN_MESSAGE);

}
public static void main(String arg[])
{   calculate_Horner();}
}
```

Horner method python version

```
from math import *

def f_poly(c1,x):
     # this function calculates the value of
     # c least square curve fitting function
     n=len(c1);
     if n!=0.0:
       ff=c1[n-1];
       for i in range(n-2,-1,-1):
         ff=ff*x+c1[i];
     else:
       ff=0,0;
     return ff;

def horner(a):
  # roots of a polynomial by using synthetic division process
     n=len(a);
     x1=complex(1.56435,0.1);
     n1=n;
     nmax=150;
     tolerance=1.0e-10;
     n1=n+1;
     ref=3.1;
     y=[complex(0.0,0.0) for i in range(n1-2)];
     n2=n1-2;
     for k in range(0,n2):
           n=len(a);
           x=x1;
           xold=3.4878;
           ref=abs(xold-x);
           b=[complex(0.0,0) for i in range(n)];
           c=[complex(0.0,0.0) for i in range(n-1)];
           b[n-1]=a[n-1];
           ii=0;
           while ref>tolerance and ii<nmax:
             for i in range(n-2,-1,-1):
                b[i]=a[i]+b[i+1]*x;
             for i in range(n-2,-1,-1):  c[i]=b[i+1];
             Q=f_poly(c,x);
             P=b[0];
             x=x-P/Q;
             ref=abs(xold-x);
             ii=ii+1;
             xold=x;
           a=c;
           n=len(a);
           y[k]=x;
     return y;

a=[1,3,3,1];
print(horner(a));
```

EX2 Halley root finding method

$$x_{n+1} = x_n - \frac{2f(x_n)f'(x_n)}{2f'(x_n)f'(x_n) - f(x_n)f"(x_n)}$$

Java version

```
// ROOT FINDING Halley method
import java.util.*;
import java.awt.*;
import java.applet.Applet;
import java.awt.event.*;
import javax.swing.*;
public class Halley2
{public static double Halley(if_x f,double x)
{
int nmax=500;
double tolerance=1.0e-8;
double fx,dfx,fxm,dfxm,d2fx;
for(int i=0;i<nmax;i++)
{  fx=f.func(x);
   dfx=f.dfunc(x);
   d2fx=f.dfunc(x,2);
   x-=2.0*fx*dfx/(2.0*dfx*dfx-fx*d2fx);
  if(Math.abs(fx)<tolerance) { return x;}
}
JOptionPane.showMessageDialog(null,"Maximum number of iteration is exceeded in Halley method\n"+
   " results may not be valid","MAXIMUM ITERATION WARNING",JOptionPane.WARNING_MESSAGE);
return x;
}
public static void main(String arg[])
{    if_x f=x->x*x-2;
     double x=Double.parseDouble(JOptionPane.showInputDialog(" enter lower limit of the function a : "));
     double r= Halley(f,x);
     JOptionPane.showMessageDialog(null," root  : "+r+"\nFunction value : "+f.func(r));
     Plot p=new Plot(f,0,10);
     p.plot();
}
}
```

Halley method python version

```
#halley.py
from math import *
from graphics import *
from f_x import *;

class f1(f_x):
  def func(self,x):
      y=x*x+2;
      return y;

def halley(f,x):
   # halley root finding method
  nmax=100
  tolerance=1.0e-10
  for i in range(0,nmax):
      fx= f.func(x);
      dfx=f.dfunc(x,1);
      d2fx=f.dfunc(x,2);
      x=x-2.0*fx*dfx/(2.0*dfx*dfx-fx*d2fx)
      print("fx",fx,"dfx",dfx,"x=",x)
      if abs(fx)<tolerance: return x
  return x
```

```
def read(s1,s2):
   # reads x1 and x2 values as float then
   # converts to a float array and returned
   s="read data "+s1+" and "+s2+" as float push mouse to exit";
   win = GraphWin(s, 450, 100)
   win.setCoords(0.0, 0.0, 3.0, 4.0)
   # Draw the interface
   Text(Point(1,2), s1).draw(win)
   Text(Point(1,1),s2).draw(win)
   i1 = Entry(Point(2,2), 20)
   i1.setText("")
   i1.draw(win);
   i2 = Entry(Point(2,1), 20)
   i2.setText("")
   i2.draw(win);
   # wait for a mouse click
   win.getMouse()
   # convert input
   x1 = float(i1.getText());
   x2 = float(i2.getText());
   x=[x1,x2];
   return x;
def main():
   x1=read("x1:","x2:");
   x=complex(x1[0],x1[1]);
   f=f1();
   r=halley(f,x);
   print(" ROOT  : ",r,"\nFUNCTION VALUE : ",f.func(r));
main();
```

EX3 Olver's root finding method

$$x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)} - \frac{1}{2}\frac{f(x_n)f(x_n)f''(x_n)}{f'(x_n)f'(x_n)f'(x_n)}$$

Write the code in your selected language and find the root of $f(x) = x^3 + 3.6x - 36.4$

HOMEWORK EXERCISES

Homework exercises will be done at home and will bring to next Thursday class printed no late exercises will be excepted. Each code should include student name id#, code plus results should be given. Homeworks will be accepted in written format plus a computer copy in pdf format will be sent to numerical_analysis@turhancoban.com adress your file name should be "group"+"week#"+studentname+studentid#.pdf
A_W1_turhan_coban_0101333.pdf
B_W3_ali_veli_02335646.pdf

W3HW1

W1HW1 : function f(x) $= x * x - 5 * x + 1$ is given find the root by using Horner's method of synthetic division

e) by hand (finding just one root is enough)
f) by using java program
g) by using python program

W3HW2: function f(x) $= x * x - 5 * x + 1$ is given find the root by using Halley's method by hand (finding just one root is enough)

a) by hand (finding just one root is enough)
b) by using java program
c) by using python program

W3HW3 : function f(x) $= \sin(x)$ is given find the root in the range of $0.2 \leq x \leq 2.5$ by using

e) Olver's method by hand
f) Olver's method by computer program (you can select your language)

W4 NUMERICAL ANALYSIS 2019-2020 FALL

LINEAR SYSTEM OF EQUATIONS AX=B

CLASS EXERCISES

Class exercises will be completed and graded in class

EX 1 Following excel files(programs) are given investigate solutions and solve the following system of equation:

$$\begin{bmatrix} 1 & 2 & 3 \\ 2 & 1 & 2 \\ 3 & 2 & 1 \end{bmatrix} \begin{Bmatrix} x_0 \\ x_1 \\ x_2 \end{Bmatrix} = \begin{Bmatrix} 1 \\ 1 \\ 2 \end{Bmatrix}$$

gauss.xls

	A			B
	1	2	3	1
	2	1	2	1
	3	2	1	2
c	1	2	3	1
2	0	-3	-4	-1
3	0	-4	-8	-1
c	1	2	3	1
1.333333	0	-3	-4	-1
	0	0	-2.66667	0.333333
x0				0.375
x1				0.5
x2				-0.125

Gauss_jordan.xls

	A			B	
	1	2	3	1	
	2	1	2	1	
	3	2	1	2	
c	1	2	3	1	Normalisation
	2	1	2	1	1
	3	2	1	2	
	1	2	3	1	
2	0	-3	-4	-1	
3	0	-4	-8	-1	
	1	2	3	1	Normalisation
	0	1	1.333333	0.333333	-3
	0	-4	-8	-1	

2	1	0	0.333333	0.333333	
	0	1	1.333333	0.333333	
-4	0	0	-2.66667	0.333333	
	1	0	0.333333	0.333333	Normalisation
	0	1	1.333333	0.333333	-2.66667
	0	0	1	-0.125	
0.333333	1	0	0	0.375	
1.333333	0	1	0	0.5	
	0	0	1	-0.125	

Jacobi.xls

$$\begin{bmatrix} 3 & -0.1 & -0.2 \\ 0.1 & 7 & -0.3 \\ 0.3 & -0.2 & 10 \end{bmatrix} \begin{Bmatrix} x_0 \\ x_1 \\ x_2 \end{Bmatrix} = \begin{Bmatrix} 7.85 \\ -19.3 \\ 1.4 \end{Bmatrix}$$

iterative method

3	-0.1	-0.2	x1	7.85
0.1	7	-0.3	x2	-19.3
0.3	-0.2	10	x3	71.4

Jacobi		**Gauss-Seidel**	
Initial guess		Initial guess	
x1	0	x1	0
x2	0	x2	0
x3	0	x3	0
iter	1	iter	1
x1	2.616667	x1	2.616667
x2	-2.75714	x2	-2.79452
x3	7.14	x3	7.00561
iter	2	iter	2
x1	3.000762	x1	2.990557
x2	-2.48852	x2	-2.49962
x3	7.006357	x3	7.000291
iter	3	iter	3
x1	3.000806	x1	3.000032
x2	-2.49974	x2	-2.49999
x3	7.000207	x3	6.999999
iter	4	iter	4
x1	3.000022	x1	3
x2	-2.5	x2	-2.5
x3	6.999981	x3	7

EX2 Following Octave(Matlab) files are given investigate codes and solve the following system of equation:

$$\begin{bmatrix} 1 & 2 & 3 \\ 2 & 1 & 2 \\ 3 & 2 & 1 \end{bmatrix} \begin{Bmatrix} x_0 \\ x_1 \\ x_2 \end{Bmatrix} = \begin{Bmatrix} 1 \\ 1 \\ 2 \end{Bmatrix}$$

gauss1.m

```
function x=gauss1(a,b)
% Gauss elimination
n=length(b);
carpan=0;
toplam=0;
```

```
  for k=1:n
   %pivoting
   p=k;
   buyuk=abs(a(k,k));
   for ii=k+1:n
    dummy=abs(a(ii,k));
     if dummy > buyuk buyuk=dummy;p=ii;end
   end
   if p~=k
    for jj=k:n
     dummy=a(p,jj);
     a(p,jj)=a(k,jj);
     a(k,jj)=dummy;
     end
    dummy=b(p);
    b(p)=b(k);
    b(k)=dummy;
   end
   %gauss eleme
   for i=k+1:n
    carpan=a(i,k)/a(k,k);
     a(i,k)=0;
     for j=k+1:n
       a(i,j)=a(i,j)-carpan*a(k,j);
     end
        b(i)   =b(i)   -carpan*b(k);
   end
  end
  %back substituting
  x(n)=b(n)/a(n,n);
  for i=n-1:-1:1
   toplam=0;
   for j=i+1:n
    toplam=toplam+a(i,j)*x(j);end
   x(i)=(b(i)-toplam)/a(i,i);
  end
end
```

```
>> a=[1 2 3;2 1 2;3 2 1]
a =
   1   2   3
   2   1   2
   3   2   1
>> b=[1 1 2]'
b =
   1
   1
   2
>> gauss1(a,b)
ans =

   0.37500   0.50000  -0.12500
>>
```

Gaussjordan.m

```
function x=gaussjordan(a,b)
  n=length(b);
  np1=n+1;
  carpan=0;
  C=a;
  for i=1:n
     C(i,n+1)=b(i);
  end
  for k=1:n
```

```
%gauss jordan elimination
for j=k+1:np1
 C(k,j)=C(k,j)/C(k,k);
end
for i=1:n
 if i~=k
   carpan=C(i,k);
    C(i,k)=0;
    for j=k+1:np1
      C(i,j)=C(i,j)-carpan*C(k,j); end
  end
end
end
for i=1:n
C(i,i)=1;
x(i)=C(i,np1);
end
end
```

EX3: Following java programs are given investigate solutions and solve the following system of equation(s):

$$\begin{bmatrix} 1 & 2 & 3 \\ 2 & 1 & 2 \\ 3 & 2 & 1 \end{bmatrix} \begin{Bmatrix} x_0 \\ x_1 \\ x_2 \end{Bmatrix} = \begin{Bmatrix} 1 \\ 1 \\ 2 \end{Bmatrix}$$

NA15.java (gauss elimination with pivoting)

```
import java.io.*;
import java.util.Locale;
class NA15
{ public static String toString(double a[])
 {int n=a.length;
  String s="";
  for(int i=0;i<n;i++)
  {s+=String.format(Locale.US,"%10g",a[i])+" ";}
  return s;
 }
 public static String toString(double a[][])
 {int n=a.length;
  int m=a[0].length;
  String s="";
  for(int i=0;i<n;i++)
  { for(int j=0;j<m;j++)
            {s+=String.format(Locale.US,"%10g",a[i][j])+" ";}
            s+="\n";
  }
  return s;
 }

 public static double[] pivotlugauss(double a[][],double b[])
 { //gauss elimination with partial pivoting
 int n=b.length;
 double x[]=new double[n];
 double carpan=0;
```

```
double toplam=0;
double buyuk;
double dummy=0;
//gauss eleme
int i,j,k,p,ii,jj;
System.out.println("Original matrix :\n"+toString(a));
for(k=0;k<(n-1);k++)
{ //pivoting
        p=k;
        buyuk=Math.abs(a[k][k]);
        for(ii=k+1;ii<n;ii++)
        {  dummy=Math.abs(a[ii][k]);
          if(dummy > buyuk) {buyuk=dummy;p=ii;}
        }
        if(p!=k)
        { for(jj=k;jj<n;jj++)
          { dummy=a[p][jj];
            a[p][jj]=a[k][jj];
            a[k][jj]=dummy;
          }
          dummy=b[p];
          b[p]=b[k];
          b[k]=dummy;
        }
        System.out.println("pivoted matrix : k="+k+"\n"+toString(a));
        // Solving gauss elimination
  for(i=k+1;i<n;i++)
  {  carpan=a[i][k]/a[k][k];
     a[i][k]=0;
     for(j=k+1;j<n;j++)
     {   a[i][j]-=carpan*a[k][j]; }
        b[i]   =b[i]   -carpan*b[k];
          }
}
System.out.println("final upper triangular matrix : \n"+toString(a));
// back substitution
x[n-1]=b[n-1]/a[n-1][n-1];
for(i=n-2;i>=0;i--)
{
  toplam=0;
  for(j=i+1;j<n;j++)
  {  toplam+=a[i][j]*x[j];}
  x[i]=(b[i]-toplam)/a[i][i];
}
return x;
}
```

```
  public static void main(String args[]) throws IOException
  {
  double a[][]={{1,2,3},{2,1,2},{3,2,1}};
  double b[]={1,1,2};
  double x[]=pivotlugauss(a,b);
  System.out.println(toString(x));
  }
}
```

```
 ---------- Capture Output ----------
> "E:\co\java\bin\java.exe" NA15
Original matrix :
   1.00000    2.00000    3.00000
   2.00000    1.00000    2.00000
   3.00000    2.00000    1.00000

pivoted matrix : k=0
   3.00000    2.00000    1.00000
   2.00000    1.00000    2.00000
   1.00000    2.00000    3.00000

pivoted matrix : k=1
   3.00000    2.00000    1.00000
   0.00000    1.33333    2.66667
   0.00000  -0.333333    1.33333

final upper triangular matrix :
   3.00000    2.00000    1.00000
   0.00000    1.33333    2.66667
   0.00000    0.00000    2.00000

  0.375000   0.500000  -0.125000

> Terminated with exit code 0.
```

NA17.java Gauss-Jordan elimination with pivoting

```
 import java.io.*;

class NA17
{ public static double[][] C(double a[][],double b[])
  {int n=a.length;int m=a[0].length;double c[][]=new double[n][m+1];
  for(int i=0;i<n;i++){for(int j=0;j<m;j++){c[i][j]=a[i][j];};c[i][m]=b[i];}
  return c;
  }

  public static double[][] C(double a[][],double b[][])
  {int n=a.length;int o=b[0].length;int m=n+o;double c[][]=new double[n][m];
  for(int i=0;i<n;i++)
  {for(int j=0;j<n;j++){c[i][j]=a[i][j];};
   for(int j=n;j<m;j++){c[i][j]=b[i][j];}
  }
  return c;
  }

  public static double[][] gaussjordanwithpivot(double a[][])
  { //pivotlu
  int n=a.length;
  int m=a[0].length;
```

```
  int o=m-n;
  double x[][]=new double[n][o];
  double carpan=0;
  double toplam=0;
  double buyuk;
  double dummy=0;
  int i,j,k,p,ii,jj;
  i=0;
  for(k=0;k<n;k++)
  { //pivoting
              p=k;
              buyuk=Math.abs(a[k][k]);
              for(ii=0;ii<n;ii++)
              {  if(i!=k)
                 {
                 dummy=Math.abs(a[ii][k]);
                 if(dummy > buyuk) {buyuk=dummy;p=ii;}
     }
              }
              if(p!=k)
              { for(jj=k;jj<m;jj++)
                { dummy=a[p][jj];
                  a[p][jj]=a[k][jj];
                  a[k][jj]=dummy;
                }
              }
              //gauss jordan elimination
              for(j=k+1;j<m;j++)
              { a[k][j]/=a[k][k]; }
              a[k][k]=1.0;
    for(i=0;i<n;i++)
    {  if(i!=k)
       {
                    carpan=a[i][k];
         a[i][k]=0;
         for(j=k+1;j<m;j++)
           {a[i][j]-=carpan*a[k][j]; }
       }
              }
  }
  for(jj=0;jj<o;jj++)
  {for(i=0;i<n;i++) {x[i][jj]=a[i][n+jj];}}
  return x;
  }
  public static double[] gaussjordan(double a[][],double b[])
  {double c[][]=C(a,b);
   double d[][]=gaussjordanwithpivot(c);
   return d[0];
  }

  public static void main(String args[]) throws IOException
  { double a[][]={{1,2,3,1},{2,1,2,1},{3,2,1,2}};
    double x[][]=gaussjordanwithpivot(a);
    System.out.println(NA15.toString(x));
  }
}
```

```
---------- Capture Output ----------
> "E:\co\java\bin\java.exe" NA17
 0.375000
 0.500000
-0.125000
```

```
> Terminated with exit code 0.
```

LU decomposition(Gauss-dolittle)

```
 ---------- Capture Output ----------
> "E:\co\java\bin\java.exe" NA18
d=
           0.375000000000000
           0.500000000000000
          -0.125000000000000

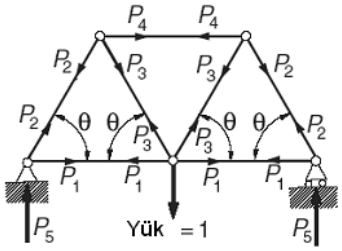

Solution of Gauss(Doolittle) decomposition and elimination
Matrix a=
   1.00000    2.00000    3.00000
   2.00000    1.00000    2.00000
   3.00000    2.00000    1.00000
Matrix b =
   1.00000    1.00000    2.00000
Solution Matrix=
  0.375000   0.500000  -0.125000

> Terminated with exit code 0.
```

HOMEWORK EXERCISES

Homework exercises will be done at home and will bring to next Thursday class printed no late exercises will be excepted.

W3HW1: Solving method is not specified

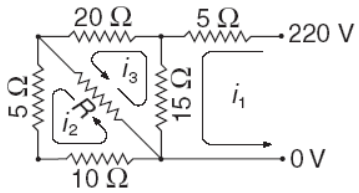


Load Matrix of the figure can be given with the following system of eqautions

$$\begin{bmatrix} c & 1 & 0 & 0 & 0 \\ 0 & s & 0 & 0 & 1 \\ 0 & 0 & 2s & 0 & 0 \\ 0 & -c & c & 1 & 0 \\ 0 & s & s & 0 & 0 \end{bmatrix} \begin{bmatrix} P_1 \\ P_2 \\ P_3 \\ P_4 \\ P_5 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 1 \\ 0 \\ 0 \end{bmatrix}$$

Where c=cos(θ),s=sin(θ) . For θ =60º find the load distribution profile

W3HW2: **Solving method is not specified**



For the given electrical circuit current density-Voltage eqautions can be given as:

$$5i_1 + 15(i_1 - i_3) = 220\,\mathrm{V}$$

$$R(i_2 - i_3) + 5i_2 + 10i_2 = 0$$

$$20i_3 + R(i_3 - i_2) + 15(i_3 - i_1) = 0$$

Calculate current densities for R=10 Ω.

W3HW3 :

Solve system of equation by using

g) By using Gauss elimination
h) By using Gauss-Jordan elimination
i) LU decomposition (Dolittle)

$$x_1 + 3x_2 + x_3 + 5x_4 = 4$$
$$2x_1 + x_2 + 3x_4 = 5$$
$$4x_1 + 2x_2 + 2x_3 + x_4 = 11$$
$$-3x_1 + x_2 + 3x_3 + 2x_4 = 3$$

W3HW4:

Find the inverse matrix

$$\begin{bmatrix} 1.1348 & 3.8326 & 1.1651 & 3.4017 \\ 0.5301 & 1.7875 & 2.5330 & 1.5435 \\ 3.4129 & 4.9317 & 8.7643 & 1.3142 \\ 1.2371 & 4.9998 & 10.6721 & 0.0147 \end{bmatrix}$$

W5 NUMERICAL ANALYSIS 2019-2020 FALL

OPTIMISATION I

CLASS EXERCISES

Class exercises will be completed and graded in class

EX 1 Following excel files(programs) are given investigate solutions

opt_golden_search.xlsx **opt_secant.xlsx** **opt_bisection.xlsx**	

EX2 following one dimensional nonlinear optimiation programs are given investigate solutions.

```
public class opt_test
{
public static double golden(if_x f,double a,double b)
{
//  find the minimum of the function
// note  maximum f(x) = minimum (-f(x))
   double epsilon;
   double delta;
```

```
    int print;
    epsilon=0.001;
    delta=0.002;
    print=0;
    double r1 = (Math.sqrt(5.0)-1.0)/2.0; // golden ratio
    double r2 = r1*r1;
    double h = b - a;
    double ya = f.func(a);
    double yb = f.func(b);
    double c = a + r2*h;
    double d = a + r1*h;
    double yc = f.func(c);
    double yd = f.func(d);
    int k = 1;
    double dp,dy,p,yp;
    while ((Math.abs(yb-ya)>epsilon) || (h>delta))
    {
      k++;
      if (yc<yd)
        {
        b = d;
        yb = yd;
        d = c;
        yd = yc;
        h = b - a;
        c = a + r2 * h;
        yc = f.func(c);
        }
      else
       {
         a = c;
         ya = yc;
         c = d;
         yc = yd;
         h = b - a;
         d = a + r1 * h;
         yd = f.func(d);
       }//end of if
    }//end of while
    dp = Math.abs(b-a);
    dy = Math.abs(yb-ya);
    p = a;
    yp = ya;
    if (yb<ya)
    {
      p = b;
      yp = yb;
    }
    if(print==1)
    {System.out.println("x min = "+p+"ymin = "+yp+"error of x ="+dp+"error of y"+dy); }
    return p;
}
public static double quadratic_polynomial(if_x f,double x0,double x2)
{double x1=(x0+x2)/2.0;
 return quadratic_polynomial(f,x0,x1,x2);
}
//secant optimisation method
public static double secant(if_x f,double x)
{ double eps=1.0e-10;
  double y=f.func(x);
  double dy=f.dfunc(x);
  int miter=100;
  int i=0;
  while(Math.abs(y)>eps && i<miter)
```

```
  {x=x-y/dy;
   y=f.dfunc(x);
   dy=f.dfunc2(x);
   i++;
  }
  if(i>=miter)
    System.out.println("i="+i+"results may not be valid");
  return x;
}
//Bisection optimisation method f'(x)=0 x=? given initial guess limits a and b
public static double bisection(if_x f,double a,double b)
  {double b1=1.1*b;
   double r=(a+b)/2.0;
   double eps=1.0e-6;
   int nmax=100;
   int i=0;
   while(Math.abs(f.dfunc(r))>eps && i<nmax)
   {if(f.dfunc(a)*f.dfunc(r)<0) b=r;
    else a=r;
    r=(a+b)/2.0;
    System.out.println("r="+r+"i="+i);
    i++;
   }
   return r;
  }
public static double quadratic_polynomial(if_x f,double x0,double x1,double x2)
{
double epsilon=1.0e-10;
double delta=1.0e-5;
int print=0;
return quadratic_polynomial(f,x0,x1,x2,epsilon,delta,print);
}
public static double quadratic_polynomial(if_x f,double x0,double x1,double x2,double epsilon,double delta,int print)
{
int maxit=100;
double f0 = f.func(x0);
    double f1 = f.func(x1);
    double f2 = f.func(x2);
    double f3 = f2;
    double x3 = 0;
    double h = x1 - x0;
    double k=1;
    double dd=Math.abs(f1-f0);
    while ((dd >epsilon) || (h>delta))
    {
      k++;
      x3=(f0*(x1*x1-x2*x2)+f1*(x2*x2-x0*x0)+f2*(x0*x0-x1*x1))/
            (2*f0*(x1-x2)+2.0*f1*(x2-x0)+2.0*f2*(x0-x1));
      f3=f.func(x3);
      if(x3 >= x0 && x3<x1)
        {x2=x1;f2=f1;x1=x3;f1=f3;}
      else if(x3 >= x1 && x3 <x2)
        {x0=x1;f0=f1;x1=x3;f1=f3; }
      else if(x3 >x2)
        {x0=x1;f0=f1;x1=x2;f1=f2;x2=x3;f2=f3; }
      else if(x3 < x0)
        {x0=x3;f0=f3;x1=x0;f1=f0;x2=x1;f2=f1; }
      dd=Math.abs(f1-f0);
      h=Math.abs(x1-x0);
      if(k>maxit) break;
    }//end of while
    if(print==1)
    {System.out.println("x  = "+x3+"f = "+f3+"hata x ="+h+"hata f(x)=y "+dd); }
    return x3;
```

```
}
public static void main(String arg[])
{if_x ff=(x)->1.0/3.0*x*x*x-2*x+5;
 double r1=secant(ff,1.0);
 double r2=bisection(ff,1.0,3.0);
 double r3=golden(ff,1.0,3.0);
 double r4=quadratic_polynomial(ff,1.0,3.0);
 System.out.println("r1="+r1+"r2="+r2+"r3="+r3+"r4="+r4);
 Plot pp=new Plot(ff,0.0,3.0);
 pp.plot();
 if_x df=(x)->ff.dfunc(x);
 Plot pp1=new Plot(df,0.0,3.0);
 pp1.plot();
}
}
```

EX3 following multi dimensional nonlinear optimiation programs are given investigate solutions.

```
import java.util.Locale;
public class opt1_test
{
public static double[] gauss(double a[][],double b[])
  { //gauss elimination with partial pivoting
  int n=b.length;
  double x[]=new double[n];
  double carpan=0;
  double toplam=0;
  double buyuk;
  double dummy=0;
  //gauss elimination
  int i,j,k,p,ii,jj;
  for(k=0;k<(n-1);k++)
  { //pivoting
              p=k;
              buyuk=Math.abs(a[k][k]);
              for(ii=k+1;ii<n;ii++)
              {  dummy=Math.abs(a[ii][k]);
                if(dummy > buyuk) {buyuk=dummy;p=ii;}
              }
              if(p!=k)
              { for(jj=k;jj<n;jj++)
                { dummy=a[p][jj];
                  a[p][jj]=a[k][jj];
                  a[k][jj]=dummy;
                }
                dummy=b[p];
                b[p]=b[k];
                b[k]=dummy;
              }
              //Solving gauss elimination
    for(i=k+1;i<n;i++)
    {  carpan=a[i][k]/a[k][k];
       a[i][k]=0;
       for(j=k+1;j<n;j++)
       {   a[i][j]-=carpan*a[k][j]; }
          b[i]   =b[i]   -carpan*b[k];
              }
  }
  //back substitution
  x[n-1]=b[n-1]/a[n-1][n-1];
  for(i=n-2;i>=0;i--)
  {
    toplam=0;
```

```
    for(j=i+1;j<n;j++)
    {  toplam+=a[i][j]*x[j];}
    x[i]=(b[i]-toplam)/a[i][i];
  }
  return x;
  }
public static double[] newton_opt(if_xj f,double x[])
{ int k=x.length;
  int nmax=100;
  double tolerance=1.0e-10;
  double fx[];
  double dfx[][];
  double dx[];
  double total=0;
  for(int i=0;i<nmax;i++)
  { fx=f.dfunc(x);
    dfx=f.d2func(x);
    dx=gauss(dfx,fx);
    for(int j=0;j<k;j++)
      x[j]-=dx[j];
    total=0;
    for(int j=0;j<k;j++)
      total+=fx[j];
    if(Math.abs(total)<tolerance) return x;
  }
  return x;
}
public static String toString(double a[])
  {int n=a.length;
   String s="";
   for(int i=0;i<n;i++)
   {s+=String.format(Locale.US,"%10g",a[i])+" ";}
   return s;
  }
  public static String toString(double a[][])
  {int n=a.length;
   int m=a[0].length;
   String s="";
   for(int i=0;i<n;i++)
   { for(int j=0;j<m;j++)
                {s+=String.format(Locale.US,"%10g",a[i][j])+" ";}
                s+="\n";
   }
   return s;
  }
//6-2. Nelder and Mead Simplex multivariable nonlinear optimization method
//  Nelder & Mead 1965 Computer J, v.7, 308-313.
//_____________________________________________________________________
public static double[] nelder(if_xj fnelder,double a[],double da[],int maxiteration,double tolerance,int printlist)
{
              int i,j;
double x[][]=new double[a.length+1][a.length];
double p[][]=new double[a.length+1][a.length+1];
for(i=0;i<x.length;i++)
   {for(j=0;j<x[0].length;j++)
     {if(i==j){x[i][j]=a[j]+da[j];p[i][j]=x[i][j];}
      else    {x[i][j]=a[j];p[i][j]=x[i][j]; }
     }
     p[i][j] = fnelder.func(p[i]);
   }

   // Inlet variable definitions
   // fnelder : abstract multivariable function f(x)
   // x : independent variable set of n+1 simplex elements
```

```
// maxiteration : maximum iteration number
// tolerance :
    int NDIMS = x.length-1;
    int NPTS = x.length;
    int FUNC = NDIMS;
    int ncalls = 0;
      ////// construct the starting simplex //////////////////
      //double p[][]=new double[NPTS][NPTS]; // [row][col] = [whichvx][coord,FUNC]
      double z[]=new double[NDIMS];
      double best = 1E99;
      /////////////// calculate the first function values for the simplex ////////////////
      int iter=0;
      for (iter=1; iter<maxiteration; iter++)
      {
                  ///////////  define lo, nhi, hi (low high next_to_high //////////////
        int  ilo=0, ihi=0, inhi = -1; // -1 means missing
        double flo = p[0][FUNC];
        double fhi = flo;
        double pavg,sterr;
        for (i=1; i<NPTS; i++)
        {
          if (p[i][FUNC] < flo)
            {flo=p[i][FUNC]; ilo=i;}
          if (p[i][FUNC] > fhi)
            {fhi=p[i][FUNC]; ihi=i;}
        }
        double fnhi = flo;
        inhi = ilo;
        for (i=0; i<NPTS; i++)
          if ((i != ihi) && (p[i][FUNC] > fnhi))
            {fnhi=p[i][FUNC]; inhi=i;}
        ////////// exit criteria //////////////
        if ((iter % 4*NDIMS) == 0)
        {
          // calculate the avarage (including maximum value)
          pavg=0;
          for(i=0;i<NPTS;i++)
            pavg+=p[i][FUNC];
          pavg/=NPTS;
          double tot=0;
          if(printlist!=0)
          {  System.out.print(iter);
            for (j=0; j<=NDIMS; j++)
              { System.out.print(p[ilo][j]+" ");}
            System.out.println("");
          }
          for(i=0;i<NPTS;i++)
          { tot=(p[i][FUNC]-pavg)*(p[i][FUNC]-pavg);}
          sterr=Math.sqrt(tot/NPTS);
          //if(sterr < tolerance)
            { for (j=0; j<NDIMS; j++)
                { z[j]=p[ilo][j];}
              //break;
            }
          best = p[ilo][FUNC];
        }

        ///// calculate avarage without maximum value //////

        double ave[] = new double[NDIMS];
        for (j=0; j<NDIMS; j++)
          ave[j] = 0;
        for (i=0; i<NPTS; i++)
          if (i != ihi)
```

```
      for (j=0; j<NDIMS; j++)
        ave[j] += p[i][j];
    for (j=0; j<NDIMS; j++)
      ave[j] /= (NPTS-1);
    ///////// reflect ////////////////
    double r[] = new double[NDIMS];
    for (j=0; j<NDIMS; j++)
     r[j] = 2*ave[j] - p[ihi][j];
    double fr = fnelder.func(r);

    if ((flo <= fr) && (fr < fnhi))  // in zone: accept
    {
      for (j=0; j<NDIMS; j++)
        p[ihi][j] = r[j];
      p[ihi][FUNC] = fr;
      continue;
    }
    if (fr < flo)  //// expand
    {
      double e[] = new double[NDIMS];
      for (j=0; j<NDIMS; j++)
        e[j] = 3*ave[j] - 2*p[ihi][j];
      double fe = fnelder.func(e);
      if (fe < fr)
      {
        for (j=0; j<NDIMS; j++)
          p[ihi][j] = e[j];
        p[ihi][FUNC] = fe;
        continue;
      }
      else
      {
        for (j=0; j<NDIMS; j++)
          p[ihi][j] = r[j];
        p[ihi][FUNC] = fr;
        continue;
      }
    }
    ////////////// shrink:
    if (fr < fhi)
    {
      double c[] = new double[NDIMS];
      for (j=0; j<NDIMS; j++)
        c[j] = 1.5*ave[j] - 0.5*p[ihi][j];
      double fc = fnelder.func(c);
      if (fc <= fr)
      {
        for (j=0; j<NDIMS; j++)
          p[ihi][j] = c[j];
        p[ihi][FUNC] = fc;
        continue;
      }
      else   /////// shrink
      {
        for (i=0; i<NPTS; i++)
          if (i != ilo)
          {
            for (j=0; j<NDIMS; j++)
              p[i][j] = 0.5*p[ilo][j] + 0.5*p[i][j];
            p[i][FUNC] = fnelder.func(p[i]);
          }
        continue;
      }
    }
```

```
            if (fr >= fhi)   ///
            {
               double cc[] = new double[NDIMS];
               for (j=0; j<NDIMS; j++)
                 cc[j] = 0.5*ave[j] + 0.5*p[ihi][j];
               double fcc = fnelder.func(cc);
               if (fcc < fhi)
               {
                  for (j=0; j<NDIMS; j++)
                    p[ihi][j] = cc[j];
                  p[ihi][FUNC] = fcc;
                  continue;
               }
               else    /////////
               {
                  for (i=0; i<NPTS; i++)
                    if (i != ilo)
                    {
                       for (j=0; j<NDIMS; j++)
                         p[i][j] = 0.5*p[ilo][j] + 0.5*p[i][j];
                       p[i][FUNC] = fnelder.func(p[i]);
                    }
               }
            }
         }
      return z;
}
     public static double[] nelder(if_xj fnelder,double a[],double da[],double tolerance)
     {return nelder(fnelder,a,da,500,tolerance,0);}

     public static double[] nelder(if_xj fnelder,double a[],double da[])
     {return nelder(fnelder,a,da,500,1.0e-10,0);}

     public static double[] nelder(if_xj fnelder,double a[])
     {
                       double [] da=new double[a.length];
            for(int i=0;i<a.length;i++) da[i]=0.1*a[i];
            return nelder(fnelder,a,da);
     }


public static void main(String arg[])
{if_xj ff=(double x[])->3.0*x[0]*x[0]-4.0*x[0]*x[1]+2.0*x[1]*x[1]-x[0]-x[1];
 double xx[]={1,2};
 double r1[]=newton_opt(ff,xx);
 double r2[]=nelder(ff,xx);
 System.out.println("r1=\n"+toString(r2)+"\nr2=\n"+toString(r2));
 surfacePlot pp1=new surfacePlot(ff,-2.0,2.0,-2.0,2.0);
 pp1.plot();
}
}
```

```
/*
 * To change this template, choose Tools | Templates
 * and open the template in the editor.
 */

/*
 * Example.java
 *
 * Created on Mar 13, 2010, 2:46:54 PM
 */
/**
```

```
 *
 * @author siva
 */
//changed by M. Turhan Coban
import java.awt.*;
import javax.swing.*;
import javax.swing.JPanel;
import javax.swing.JLabel;
import java.awt.BorderLayout;
import java.awt.FlowLayout;

public class surfacePlot2 extends javax.swing.JFrame {
                ExampleSurfaceModel2 model;
                SurfaceCanvas canvas;
                JPanel southPanel;
                Container c;
    public void plot()
    {
                c=this.getContentPane();
    c.setLayout(new BorderLayout());
    southPanel = new JPanel(new FlowLayout(FlowLayout.CENTER, 50, 5));
    southPanel.add(new JLabel("Rotate: Mouse Click & Drag"));
    southPanel.add(new JLabel("Zoom: Shift Key + Mouse Click & Drag"));
    southPanel.add(new JLabel("Move: Control Key + Mouse Click & Drag"));
    add(southPanel, BorderLayout.SOUTH);
                this.setSize(1000,1000);
                add(canvas, BorderLayout.CENTER);
                this.repaint();
                this.setVisible(true);
  }

  public surfacePlot2(if_xj f,double x0,double xn,double y0,double yN,double z0,double zN,
            String xLabel, String yLabel,String zLabel)
  {   model = new ExampleSurfaceModel2(f,x0,xn,y0,yN,z0,zN,xLabel,yLabel,zLabel);
                canvas = new SurfaceCanvas();
                canvas.setModel(model);
                canvas.setSize(1000,1000);
                canvas.setVisible(true);

            }
  public surfacePlot2(if_xj f,double x0,double xN,double y0,double yN,double z0,double zN)
  {   String xLabel="x";
    String yLabel="y";
    String zLabel="z";
                model = new ExampleSurfaceModel2(f,x0,xN,y0,yN,z0,zN,xLabel,yLabel,zLabel);
                canvas = new SurfaceCanvas();
                canvas.setModel(model);
                canvas.setSize(1000,1000);
                canvas.setVisible(true);
            }
            public surfacePlot2(if_xj f,double x0,double xN,double y0,double yN)
  {   String xLabel="x";
    String yLabel="y";
    String zLabel="z";
    double z0=9.9e50;
    double zN=-9.9e50;
    int n=100;
    double dx=(xN-x0)/n;
    double dy=(yN-y0)/n;
    double zz;
    double x1=0,y1=0;
    for(int i=0;i<n;i++)
    {for(int j=0;j<n;j++)
     {   x1=x0+i*dx;y1=y0+j*dy;
```

```
        double xx[]={x1,y1};
                zz=f.func(xx);
                if(zz<z0) z0=zz;
                if(zz>zN) zN=zz;
    }
    }
            model = new ExampleSurfaceModel2(f,x0,xN,y0,yN,z0,zN,xLabel,yLabel,zLabel);
            canvas = new SurfaceCanvas();
            canvas.setModel(model);
            canvas.setSize(1000,1000);
            canvas.setVisible(true);
        }
        public surfacePlot2(if_xj f,double x0,double xN,double y0,double yN,String xLabel,String yLabel,String zLabel,int N)
  {
    double z0=9.9e50;
    double zN=-9.9e50;
    int n=100;
    double dx=(xN-x0)/n;
    double dy=(yN-y0)/n;
    double zz;
    double x1=0,y1=0;
    for(int i=0;i<n;i++)
    {for(int j=0;j<n;j++)
     {   x1=x0+i*dx;y1=y0+j*dy;
        double xx[]={x1,y1};
                zz=f.func(xx);
                if(zz<z0) z0=zz;
                if(zz>zN) zN=zz;
    }
    }
            model = new ExampleSurfaceModel2(f,x0,xN,y0,yN,z0,zN,xLabel,yLabel,zLabel);
            canvas = new SurfaceCanvas();
            canvas.setModel(model);
            canvas.setSize(1000,1000);
            canvas.setVisible(true);
        }
  public static void main(String args[]) {
            if_xj f=(double x[])->x[0]*x[0]+x[1]*x[1];
            surfacePlot2 p=new surfacePlot2(f,-5.0,5.0,-5.0,5.0);
    p.plot();
  }
}
```

```
/*
* To change this template, choose Tools | Templates
* and open the template in the editor.
*/
//package net.sf.surfaceplot;
/**
 * @author salagarsamy
*/
// Adapted by M. Turhan Coban
public class ExampleSurfaceModel2 implements ISurfacePlotModel
{   float x[],y[],z[];
   String xlabel,ylabel,zlabel;
   float xmin,xmax,ymin,ymax,zmin,zmax;
   boolean boxed,mesh,scalebox, displayxy,displayz,displaygrid;
   int calcdiv,dispdiv;
   if_xj f;
   public ExampleSurfaceModel2(if_xj f1,double xmini,double xmaxi,double ymini,double ymaxi,double zmini,double zmaxi)
            {f=f1;xmin=(float)xmini;xmax=(float)xmaxi;ymin=(float)ymini;ymax=(float)ymaxi;
             zmin=(float)zmini;zmax=(float)zmaxi;
             boxed=true;
             mesh=true;
```

```
            scalebox=false;
            displayxy=true;
            displayz=true;
            displaygrid=true;
            calcdiv=80;
            dispdiv=80;
            xlabel="X";
            ylabel="Y";
            zlabel="Z";
           }
  public ExampleSurfaceModel2(if_xj f1,double xmini,double xmaxi,double ymini,double ymaxi,double zmini,double zmaxi,String
lx,String ly,String lz)
           {f=f1;xmin=(float)xmini;xmax=(float)xmaxi;ymin=(float)ymini;ymax=(float)ymaxi;
            zmin=(float)zmini;zmax=(float)zmaxi;
            xlabel=lx;
            ylabel=ly;
            zlabel=lz;
            boxed=true;
            mesh=true;
            scalebox=false;
            displayxy=true;
            displayz=true;
            displaygrid=true;
            calcdiv=100;
            dispdiv=100;
           }
           public float calculateZ(float x, float y)
           {   double z[]={x,y};
                    return (float)f.func(z);
           }

  public double calculateZ(double x, double y)
  {        double z[]={x,y};
              return f.func(z);}
  public double calculateZ(double z[])
  {         return f.func(z);
           }
           public int getPlotMode()
           {return ISurfacePlotModel.PLOT_MODE_SPECTRUM;}

           public boolean isBoxed()
           {return boxed;}

           public boolean isMesh()
           {return mesh;}

           public boolean isScaleBox()
           {return scalebox;}

           public boolean isDisplayXY()
           {return displayxy;}

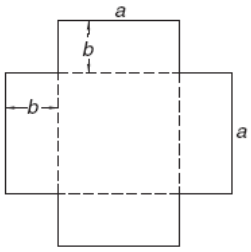
           public boolean isDisplayZ()
           {return displayz;}

           public boolean isDisplayGrids()
           {return displaygrid;}

           public void setCalcDivisions(int calcdivi )
           {calcdiv=calcdivi;}
           public int getCalcDivisions()
           {return calcdiv;}
  public void setDispDivisions(int dispdivi )
           {dispdiv=dispdivi;}
```

```
            public int getDispDivisions()
            {return dispdiv;}
    public void setXMin(double xmini)
            {xmin=(float)xmini;}
            public float getXMin()
            {return xmin;}
    public void setXMax(double xmaxi)
            {xmax=(float)xmaxi;}
            public float getXMax()
            {return xmax;}
    public void setYMin(double ymini)
            {ymin=(float)ymini;}
            public float getYMin()
            {return ymin;}
    public void setYMax(double ymaxi)
            {ymax=(float)ymaxi;}
            public float getYMax()
            {return ymax;}
            public void setZMin(double zmini)
            {zmin=(float)zmini;}
            public float getZMin()
            {return zmin;}
            public void setZMax(double zmaxi)
            {zmax=(float)zmaxi;}
            public float getZMax()
            {return zmax;}
            public void setXAxisLabel(String xlabeli)
            {xlabel=xlabeli;}
            public String getXAxisLabel()
            {return xlabel;}
            public void setYAxisLabel(String ylabeli)
            {ylabel=ylabeli;}
            public String getYAxisLabel()
            {return ylabel;}
            public void setZAxisLabel(String zlabeli)
            {zlabel=zlabeli;}
            public String getZAxisLabel()
            {return zlabel;}
}
```

EX5:



In order to make a cartoon box, the shape above should be used.After bending from the dotted line the volume of the box will be 1 m^3. In order to spent the minimum amount of the cartoon, what a and b dimensions should be?

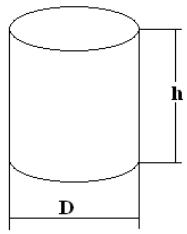
EX 6: Find the minimum of function

$$f(x) = \frac{15x}{(4x^2 - 3x + 4)}$$

f is in the range of 0 to10.

HOMEWORK EXERCISES

Homework exercises will be done at home and will bring to next friday class printed no late exercises will be excepted.



W5HW1:

One of the very basic optimization problem is the minimum cost of container problem. The cost of a box usually is a function of the surface area. Therefore we should minimize the area for a given volume For example if the volume of the container V=0.5 liter=$0.5x10^{-3}$ m^3 :

Volume $V = \frac{\pi D^2}{4} h$ or from this equation h, height is obtained as

$$h = \frac{4V}{\pi D^2}.$$

Surface area of the cylinder :

$$A = 2\frac{\pi D^2}{4} + \pi D h = \frac{\pi D^2}{2} + \frac{4V}{D}.$$

Analytical solution of the minimization problem

$$\frac{dA}{dD} = \pi D - \frac{4V}{D^2} \rightleftarrows = 0$$

$D = \sqrt[3]{\frac{4V}{\pi}} \rightleftarrows$. From here solution is $D = \sqrt[3]{\frac{4*0.5x10^{-3}}{\pi}} \rightleftarrows$=0.086025401 m and .

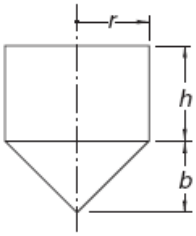
$h = \frac{4*0.5x10^{-3}}{\pi D^2}$= 0.086025401 m.

Now obtain this results by using numerical optimization methods.

For the range of 0.01<=D<=0.2

p) Graphic method
q) Golden ratio (Fibonnachi)
r) Secant method
s) Second degree polynomials

W5HW2:



It is desired to have volume of the cone base shape shown in the figure as 1 m^3. Calculate r,h and b dimensions to minimise the surface area.

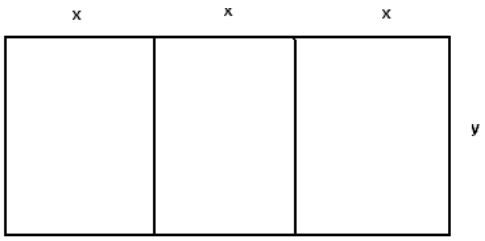
$$\boldsymbol{V = \pi r^2 \left(\frac{b}{3} + h\right)}$$
$$\boldsymbol{S = \pi r \left(2h + \sqrt{b^2 + r^2}\right)}$$

W5HW3 :

Find the minimum of

$f = 25x^2 - 12x^4 + 6xy + 25y^2 - 24x^2y^2 - 12y^4$

W3HW4: A farmer needs 150 meters of fencing to fence three adjacent gardens. What would be maximum area of each garden, and what would be values of x and y



$$4y + 6x = 150$$
$$y = 37.5 - 1.5x$$
$$A(x) = 3(37.5 - 1.5x)x = 112.5x - 4.5x^2$$
$$\frac{dA(x)}{dx} = 112.5 - 9x = 0$$

$x = 12.5$ y=18.75
$A = 703.125$ total of 3 garden
Calculate by using a numerical method

W7 NUMERICAL ANALYSIS 2019-2020 FALL

CURVE FITTING
zip file W7_NA.rar

CLASS EXERCISES
Class exercises will be completed and graded in class

EX 1 Following data is given by using excel file find least square curve fitting of quadratic polynomial equation $y = a_0 + a_1 x + a_2 x^2$
Required codes
Polynomial_least_square.xls
Data:

0	0
0.1	0.099833
0.2	0.198669
0.3	0.29552
0.4	0.389418
0.5	0.479426
0.6	0.564642
0.7	0.644218
0.8	0.717356
0.9	0.783327
1	0.841471
1.1	0.891207
1.2	0.932039
1.3	0.963558
1.4	0.98545
1.5	0.997495
1.6	0.999574
1.7	0.991665
1.8	0.973848
1.9	0.9463
2	0.909297
2.1	0.863209
2.2	0.808496

2.3	0.745705
2.4	0.675463
2.5	0.598472
2.6	0.515501
2.7	0.42738
2.8	0.334988
2.9	0.239249
3	0.14112
3.1	0.041581
3.2	-0.05837

EX 2 Take data given in EX1 using java file NA44.java find least square curve fitting of 6th degree polynomial equation $y = a_0 + a_1x + a_2x^2 + a_3x^3 + a_4x^4 + a_5x^5 + a_6x^6$

EX 3 Take data given in EX1 using java file NA44.py find least square curve fitting of 6th degree polynomial equation $y = a_0 + a_1x + a_2x^2 + a_3x^3 + a_4x^4 + a_5x^5 + a_6x^6$
Note: program has plot so use spyder editor to work with

EX 4 Take data given in EX1 using java file NA121.java find orthogonal least square curve fitting of 6th degree polynomial given by
Anthony Ralston,Philip Rabinowitz, "A First Course in Numerical Analysis", Mc Graw Hill ISBN 0-07-051158-6" [4]

$p_{j+1}(x) = (x - \alpha_{j+1})p_j(x) - \beta_j p_{j-1}(x) \quad j = 0,1,\ldots.$

$p_0(x) = 1 \quad p_{-1}(x) = 0$

α_{j+1} and β_j are polynomial constants to be calculated.

$$\beta_k = \frac{\sum_{i=1}^{n} w(x_i)x_i p_{k-1}(x_i)p_k(x_i)}{\sum_{i=1}^{n} w(x_i)x_i[p_{k-1}(x_i)]^2} = \frac{\sum_{i=1}^{n} w(x_i)[p_k(x_i)]^2}{\sum_{i=1}^{n} w(x_i)[p_{k-1}(x_i)]^2}$$

$$\alpha_k = \frac{\sum_{i=1}^{n} w(x_i)x_i[p_k(x_i)]^2}{\sum_{i=1}^{n} w(x_i)[p_{k-1}(x_i)]^2}$$

Least square algorithm can be given as follows:

$$y_m(x) = \sum_{i=1}^{n} b_j p_j(x)$$

$$b_j = \frac{\omega_j}{\gamma_j}$$

$$\omega_j = \sum_{i=1}^{n} w_i f_i p_j(x_i)$$

$$\gamma_j = \sum_{i=1}^{n} w_i [p_j(x_i)]^2$$

EX 5 Take the following surface data and find the following surface polynamial by using NA46.java
Data: b.txt

EX6 Take data given in EX1 using java file NA48.java(LAgrange polynomials) find orthogonal interpolation of the data
HOMEWORK EXERCISES

Homework exercises will be done at home and will bring to next friday class printed no late exercises will be excepted.

W7HW1: Following data is given: solve least square curve fitting of degree 2 (quadratic polynomial $y = a_0 + a_1x + a_2x^2$

a) By hand
b) By using excel file polynomial_least_square.xlsx
c) By using java file NA44.java
d) By using python file NA44.py

0.1	1.111
0.2	1.248
0.3	1.417
0.4	1.624
0.5	1.875
0.6	2.176
0.7	2.533
0.8	2.952
0.9	3.439
1	4
1.1	4.641
1.2	5.368
1.3	6.187
1.4	7.104
1.5	8.125
1.6	9.256
1.7	10.503
1.8	11.872
1.9	13.369
2	15

W7HW2: **data is given in W7HW1: solve least square curve fitting of degree 3(cubic polynomial $y = a_0 + a_1x + a_2x^2 + a_3x^3$**

a) By hand
b) By using excel file polynomial_least_square.xlsx
c) By using java file NA44.java
d) By using python file NA44.py

W7HW3 : **data is given in W7HW1: solve least square curve fitting of degree 4**

a) By using java file NA121java orthogonal polynomial curve fitting
b) By using Lagrange interpolation NA48.java

W7HW4: **Following surface formula is given make a surface fitting by using NA48.java d.txt**

0	1	1
0.1	1	1.34
0.2	1	1.76
0.3	1	2.26
0.4	1	2.84
0.5	1	3.5
0.6	1	4.24
0.7	1	5.06
0.8	1	5.96
0.9	1	6.94
1	1	8

1.1	1	9.14
1.2	1	10.36
1.3	1	11.66
1.4	1	13.04
1.5	1	14.5
1.6	1	16.04
1.7	1	17.66
1.8	1	19.36
1.9	1	21.14
2	1	23
0	2	2
0.1	2	2.61
0.2	2	3.44
0.3	2	4.49
0.4	2	5.76
0.5	2	7.25
0.6	2	8.96
0.7	2	10.89
0.8	2	13.04
0.9	2	15.41
	2	18
1.1	2	20.81
1.2	2	23.84
1.3	2	27.09
1.4	2	30.56
1.5	2	34.25
1.6	2	38.16
1.7	2	42.29
1.8	2	46.64
1.9	2	51.21
2	2	56
0	3	3
0.1	3	3.91
0.2	3	5.24
0.3	3	6.99
0.4	3	9.16
0.5	3	11.75
0.6	3	14.76
0.7	3	18.19
0.8	3	22.04
0.9	3	26.31
1	3	31
1.1	3	36.11
1.2	3	41.64
1.3	3	47.59
1.4	3	53.96
1.5	3	60.75
1.6	3	67.96
1.7	3	75.59
1.8	3	83.64
1.9	3	92.11
2	3	101
0	4	4
0.1	4	5.24
0.2	4	7.16
0.3	4	9.76
0.4	4	13.04
0.5	4	17
0.6	4	21.64
0.7	4	26.96

0.8	4	32.96
0.9	4	39.64
1	4	47
1.1	4	55.04
1.2	4	63.76
1.3	4	73.16
1.4	4	83.24
1.5	4	94
1.6	4	105.44
1.7	4	117.56
1.8	4	130.36
1.9	4	143.84
2	4	158
0	5	5
0.1	5	6.6
0.2	5	9.2
0.3	5	12.8
0.4	5	17.4
0.5	5	23
0.6	5	29.6
0.7	5	37.2
0.8	5	45.8
0.9	5	55.4
1	5	66
1.1	5	77.6
1.2	5	90.2
1.3	5	103.8
1.4	5	118.4
1.5	5	134
1.6	5	150.6
1.7	5	168.2
1.8	5	186.8
1.9	5	206.4
2	5	227
-0.1	1	0.74
-0.2	1	0.56
-0.3	1	0.46
-0.4	1	0.44
-0.5	1	0.5
-0.6	1	0.64
-0.7	1	0.86
-0.8	1	1.16
-0.9	1	1.54
-1	1	2
-1.1	1	2.54
-1.2	1	3.16
-1.3	1	3.86
-1.4	1	4.64
-1.5	1	5.5
-1.6	1	6.44
-1.7	1	7.46
-1.8	1	8.56
-1.9	1	9.74
-2	1	11
-0.1	2	1.61
-0.2	2	1.44
-0.3	2	1.49
-0.4	2	1.76
-0.5	2	2.25
-0.6	2	2.96

-0.7	2	3.89
-0.8	2	5.04
-0.9	2	6.41
-1	2	8
-1.1	2	9.81
-1.2	2	11.84
-1.3	2	14.09
-1.4	2	16.56
-1.5	2	19.25
-1.6	2	22.16
-1.7	2	25.29
-1.8	2	28.64
-1.9	2	32.21
-2	2	36
-0.1	3	2.51
-0.2	3	2.44
-0.3	3	2.79
-0.4	3	3.56
-0.5	3	4.75
-0.6	3	6.36
-0.7	3	8.39
-0.8	3	10.84
-0.9	3	13.71
-1	3	17
-1.1	3	20.71
-1.2	3	24.84
-1.3	3	29.39
-1.4	3	34.36
-1.5	3	39.75
-1.6	3	45.56
-1.7	3	51.79
-1.8	3	58.44
-1.9	3	65.51
-2	3	73
-0.1	4	3.44
-0.2	4	3.56
-0.3	4	4.36
-0.4	4	5.84
-0.5	4	8
-0.6	4	10.84
-0.7	4	14.36
-0.8	4	18.56
-0.9	4	23.44
-1	4	29
-1.1	4	35.24
-1.2	4	42.16
-1.3	4	49.76
-1.4	4	58.04
-1.5	4	67
-1.6	4	76.64
-1.7	4	86.96
-1.8	4	97.96
-1.9	4	109.64
-2	4	122
-0.1	5	4.4
-0.2	5	4.8
-0.3	5	6.2
-0.4	5	8.6
-0.5	5	12
-0.6	5	16.4

-0.7	5	21.8
-0.8	5	28.2
-0.9	5	35.6
-1	5	44
-1.1	5	53.4
-1.2	5	63.8
-1.3	5	75.2
-1.4	5	87.6
-1.5	5	101
-1.6	5	115.4
-1.7	5	130.8
-1.8	5	147.2
-1.9	5	164.6
-2	5	183
-0.1	6	5.39
-0.2	6	6.16
-0.3	6	8.31
-0.4	6	11.84
-0.5	6	16.75
-0.6	6	23.04
-0.7	6	30.71
-0.8	6	39.76
-0.9	6	50.19
-1	6	62
-1.1	6	75.19
-1.2	6	89.76
-1.3	6	105.71
-1.4	6	123.04
-1.5	6	141.75
-1.6	6	161.84
-1.7	6	183.31
-1.8	6	206.16
-1.9	6	230.39
-2	6	256

W8 NUMERICAL ANALYSIS 2019-2020 FALL

CURVE FITTING
zip file W8_NA.rar

CLASS EXERCISES
Class exercises will be completed and graded in class

EX 1 Following data is given. interpolate data by using cubic spline
Required codes
NA49.java (java version)
cubic_spline.py (python version-use spyder editor to run the program)
data: a1.txt

EX 2 Following data is interpolate data by using B spline
Required codes
B_spline.java (java version)
data: a1.txt

EX 3 Following data is interpolate data by using moving least square method
Required codes
NA129.java (java version)
data: a1.txt

EX 4 Take data given in EX1 (a1.txt) using java file NA121.java find orthogonal least square curve fitting of 10th degree polynomial given by
Anthony Ralston,Philip Rabinowitz, "A First Course in Numerical Analysis", Mc Graw Hill ISBN 0-07-051158-6" [4]
$p_{j+1}(x) = (x - \alpha_{j+1})p_j(x) - \beta_j p_{j-1}(x) \quad j = 0,1,\ldots.$
$p_0(x) = 1 \quad p_{-1}(x) = 0$
α_{j+1} and β_j are polynomial constants to be calculated.

$$\beta_k = \frac{\sum_{i=1}^{n} w(x_i)x_i p_{k-1}(x_i)p_k(x_i)}{\sum_{i=1}^{n} w(x_i)x_i[p_{k-1}(x_i)]^2} = \frac{\sum_{i=1}^{n} w(x_i)[p_k(x_i)]^2}{\sum_{i=1}^{n} w(x_i)[p_{k-1}(x_i)]^2}$$

$$\alpha_k = \frac{\sum_{i=1}^{n} w(x_i)x_i[p_k(x_i)]^2}{\sum_{i=1}^{n} w(x_i)[p_{k-1}(x_i)]^2}$$

Least square algorithm can be given as follows:

$$y_m(x) = \sum_{i=1}^{n} b_j p_j(x)$$

$$b_j = \frac{\omega_j}{\gamma_j}$$

$$\omega_j = \sum_{i=1}^{n} w_i f_i p_j(x_i)$$

$$\gamma_j = \sum_{i=1}^{n} w_i [p_j(x_i)]^2$$

And then try the same data (a1.txt) by using moving orthogonal polynomial least square method NA121A.java

EX5 Apply surface fitting by using moving least square method to the following data
Required codes:
NA129H.java
curve fitting input data : axy.txt
required curve fitting independent variable list data : axy1.txt

HOMEWORK EXERCISES

Homework exercises will be done at home and will bring to next friday class printed no late exercises will be excepted.

W7HW1: Following data is given: solve curve fitting

e) of degree 2 polynomial least square by hand
f) of degree 2 polynomial least square by using excel file polynomial_least_square.xlsx
g) of degree 5 polynomial least square by using java file NA44.java

h) of degree 5 polynomial least square by using java file ny using python file NA44.py

i) By using cubic spline NA49.java

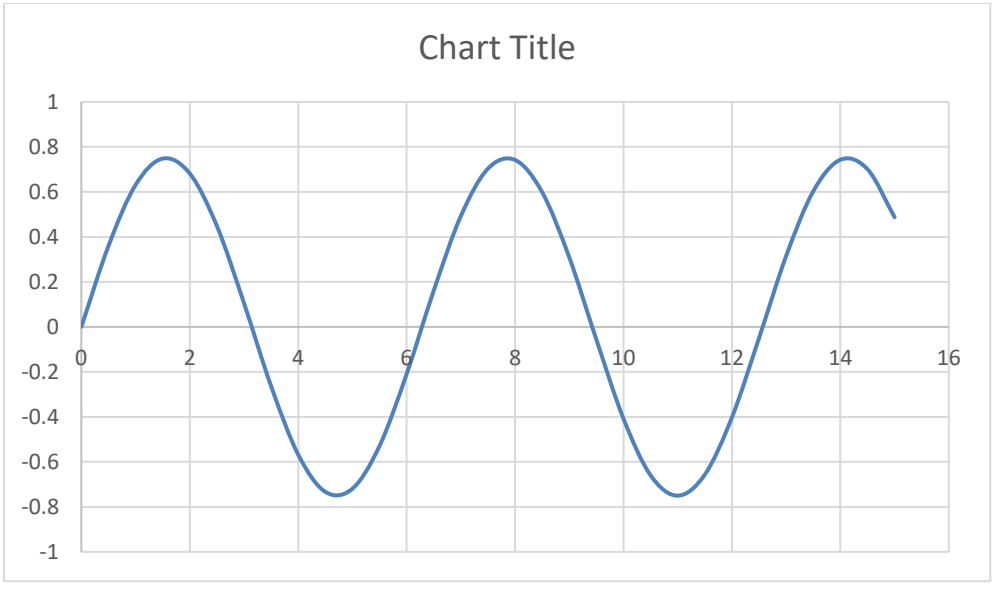
j) By using cubic spline cubic_spline.py

k) By using B spline B_spline.java

l) By using moving least square of degree 5

Data: q.txt

0	0.5
0.5	0.79836
1	0.901254
1.5	0.78349
2	0.4739
2.5	0.048282
3	-0.38916
3.5	-0.73132
4	-0.89442
4.5	-0.83855
5	-0.57736
5.5	-0.17482
6	0.270524
6.5	0.649634
7	0.869691
7.5	0.876818
8	0.669269
8.5	0.297859
9	-0.14648
9.5	-0.55495
10	-0.82755
10.5	-0.89754
11	-0.74778
11.5	-0.41494
12	0.019497
12.5	0.449158
13	0.768849
13.5	0.900299
14	0.811324
14.5	0.523709
15	0.107872



W7HW2: **Following data is given: solve curve fitting by using moving least square method to the following data**
Required codes:
NA129H.java
curve fitting input data : bxy.txt
required curve fitting independent variable list data : bxy1.txt

W9 NUMERICAL ANALYSIS 2019-2020 FALL

INTEGRAL
zip file W9_NA.rar

CLASS EXERCISES
Class exercises will be completed and graded in class
EX1
Blackbody radiation function F(0- λT), λ(μm) T(degree Kelvin) can be calculated as

$$x = \frac{C_2}{\lambda T} = \frac{14387.69\ \mu mK}{\lambda T}$$

$$F(0-\lambda T) = 1 - \frac{15}{\pi^4}\int_0^x \frac{x^3}{e^x - 1}dx$$

It can also be calculated from the series solution

$$F(0-\lambda T) = \frac{15}{\pi^4}\sum_{n=1}^{\infty}\left[\frac{e^{-nx}}{n}(x^3 + \frac{3x^2}{n} + \frac{6x}{n^2} + \frac{6}{n^3})\right]$$

By creating a program that calculates the result with using series solution and integration. Compare the results. Resulting tables are given as a reference

λ T μ mK	F(0-λT)	λ T μ mK	F(0-λT)	λ T μ mK	F(0-λT)	λ T μ mK	F(0-λT)	λ T μ mK	F(0-λT)	λ T μ mK	F(0-λT)
1000	0.00032	2900	0.25056	4800	0.60754	6900	0.8022	10700	0.92709	18000	0.98081
1050	0.00056	2950	0.26191	4850	0.61428	7000	0.80808	10800	0.92872	18200	0.98137
1100	0.00091	3000	0.27323	4900	0.62089	7100	0.81374	10900	0.93031	18400	0.98191
1150	0.00142	3050	0.28453	4950	0.62737	7200	0.81918	11000	0.93185	18600	0.98243
1200	0.00213	3100	0.29578	5000	0.63373	7300	0.82443	11200	0.9348	18800	0.98293
1250	0.00308	3150	0.30697	5050	0.63996	7400	0.82949	11400	0.93758	19000	0.98341
1300	0.00432	3200	0.3181	5100	0.64608	7500	0.83437	11600	0.94021	19200	0.98387
1350	0.00587	3250	0.32915	5150	0.65207	7600	0.83907	11800	0.9427	19400	0.98431
1400	0.00779	3300	0.34011	5200	0.65795	7700	0.8436	12000	0.94505	19600	0.98474
1450	0.01011	3350	0.35097	5250	0.66371	7800	0.84797	12200	0.94728	19800	0.98516
1500	0.01285	3400	0.36173	5300	0.66937	7900	0.85219	12400	0.94939	20000	0.98555
1550	0.01605	3450	0.37238	5350	0.67491	8000	0.85625	12600	0.95139	20500	0.98649
1600	0.01972	3500	0.38291	5400	0.68034	8100	0.86018	12800	0.95329	21000	0.98735
1650	0.02388	3550	0.39332	5450	0.68566	8200	0.86397	13000	0.95509	21500	0.98814
1700	0.02854	3600	0.4036	5500	0.69089	8300	0.86763	13200	0.95681	22000	0.98886
1750	0.03369	3650	0.41375	5550	0.696	8400	0.87116	13400	0.95843	22500	0.98952
1800	0.03934	3700	0.42377	5600	0.70102	8500	0.87457	13600	0.95998	23000	0.99014
1850	0.04549	3750	0.43364	5650	0.70594	8600	0.87787	13800	0.96145	23500	0.9907
1900	0.05211	3800	0.44338	5700	0.71077	8700	0.88105	14000	0.96285	24000	0.99123
1950	0.0592	3850	0.45297	5750	0.7155	8800	0.88413	14200	0.96419	24500	0.99172
2000	0.06673	3900	0.46241	5800	0.72013	8900	0.88711	14400	0.96546	25000	0.99217
2050	0.07469	3950	0.47172	5850	0.72468	9000	0.88999	14600	0.96667	26000	0.99297
2100	0.08306	4000	0.48087	5900	0.72914	9100	0.89278	14800	0.96783	27000	0.99368
2150	0.0918	4050	0.48987	5950	0.73351	9200	0.89547	15000	0.96893	28000	0.99429

2200	0.10089	4100	0.49873	6000	0.73779	9300	0.89808	15200	0.96999	29000	0.99482
2250	0.11031	4150	0.50744	6050	0.74199	9400	0.9006	15400	0.971	30000	0.99529
2300	0.12003	4200	0.516	6100	0.74611	9500	0.90305	15600	0.97196	32000	0.99607
2350	0.13002	4250	0.52442	6150	0.75015	9600	0.90541	15800	0.97289	34000	0.99669
2400	0.14026	4300	0.53269	6200	0.75411	9700	0.9077	16000	0.97377	36000	0.99719
2450	0.15071	4350	0.54081	6250	0.758	9800	0.90992	16200	0.97461	38000	0.99759
2500	0.16136	4400	0.54878	6300	0.76181	9900	0.91207	16400	0.97542	40000	0.99792
2550	0.17217	4450	0.55662	6350	0.76554	10000	0.91416	16600	0.9762	45000	0.99851
2600	0.18312	4500	0.56431	6400	0.76921	10100	0.91618	16800	0.97694	50000	0.9989
2650	0.19419	4550	0.57186	6450	0.7728	10200	0.91814	17000	0.97765	55000	0.99917
2700	0.20536	4600	0.57927	6500	0.77632	10300	0.92004	17200	0.97834	60000	0.99935
2750	0.2166	4650	0.58654	6600	0.78317	10400	0.92188	17400	0.97899		
2800	0.22789	4700	0.59367	6700	0.78976	10500	0.92367	17600	0.97962		
2850	0.23922	4750	0.60067	6800	0.7961	10600	0.9254	17800	0.98023		

EX 1a Use Newton_cotes integration method
newton_cotes5.java
if_x.java

EX1b Use Romberg integration with trapezodial rule
romberg.java
if_x.java

EX1c Use Gauss-Legendre integration (integration formulas is given inside if_x interface)
Gauss_Legendre.java
if_x.java

EX1d Use Gauss-Legendre integration with excel integral.xls
integral.xls

EX1e Use Gauss-Legendre integration (integration formulas is given inside if_x interface)
Use python (Script editor)
Gauss_Legendre.py
if_x.py

EX1f Use Gauss-Kronrod integration (an error correction method)
Gauss_Kronrod.java
if_x.java
EX 2 Calculate

$$I = \int_{-1}^{1}\int_{-1}^{1} [2xy + 2x - x^2 - 2y^2]\, dxdy$$

EX2a Monte-Carlo integral
monte_carlo.java
if_xj.java
EX2b Gauss-Legendre integral
Gauss_legendre_2.java
f_xj.java
f_xj2D.java

EX 3 Calculate

$$I = \int_{x=0}^{2} \left[\int_{y=x^2}^{x} (y^2 x) dy \right] dx$$

By using Gauss-Legendre integral
Analytical solution:

$$\int_{x=0}^{2} \left[\frac{y^3 x}{3} \right]_{y=x^2}^{x} dx = \int_{x=0}^{2} \left[\frac{x^4}{3} - \frac{x^7}{3} \right] dx = \frac{32}{15} - \frac{256}{15} = -\frac{128}{15} = -8.533333$$

Gauss_legendre_3.java
f_xj.java
f_xj2D.java

HOMEWORK EXERCISES

Homework exercises will be done at home and will bring to next friday class printed no late exercises will be excepted.

W9HW1: Error function is defined as:

$$erf(x) = \frac{1}{\sqrt{\pi}} \int_{0}^{\eta} e^{-x^2} dx$$

Serial opening of the function can be given as:

$$\text{erf(x)} = \frac{2}{\sqrt{\pi}} \sum_{n=0}^{\infty} (-1)^n \frac{x^{2n+1}}{n!\,(2n+1)}$$

error_func.java
Calculate the integral and compare it with the serial solution
W9HW1a Use Newton_cotes integration method
newton_cotes5.java
if_x.java

W9HW1b Use Romberg integration with trapezodial rule
romberg.java
if_x.java

W9HW1c Use Gauss-Legendre integration (integration formulas is given inside if_x interface)
Gauss_Legendre.java
if_x.java
W9HW1d Use Gauss-Legendre integration with excel
integral.xls

W9HW1e Use Gauss-Legendre integration (integration formulas is given inside if_x interface)
Use python (Script editor)
Gauss_Legendre.py
if_x.py

W9HW1f Use Gauss-Kronrod integration (an error correction method)
Gauss_Kronrod.java

if_x.java

W9HW2 Solve integral for r=1

$$I = 4 \int_{x=0}^{r} (r^2 - x^2)^{1/2} dx$$

a) Analytically
b) Newton_cotes integration method
c) Romberg integration with trapezodial rule
d) Gauss_Legendre integration with java
e) Gauss_Legendre integration with excel
f) Gauss_Legendre integration with python
g) Gauss-Kronrod integration

W9HW3 Solve

$$I = \int_{x=1.4}^{2} \left[\int_{y=1.0}^{1.5} ln(x + 2y) dy \right] dx$$

W9HW3a Monte-Carlo integral
monte_carlo.java
if_xj.java
W9HW3b Gauss-Legendre integral
Gauss_legendre_2.java
f_xj.java
f_xj2D.java

W10 NUMERICAL ANALYSIS 2019-2020 FALL

DERIVATIVE
zip file W10_NA.rar

CLASS EXERCISES
Class exercises will be completed and graded in class
EX1 : using derivative tables
f(x)=sin(x)
find function value, first,second and third derivatives at x=0
find function value, first,second and third derivatives at x= π/4
find function value, first,second and third derivatives at x= π/3
find function value, first,second and third derivatives at x= π/2
find function value, first,second and third derivatives at x= π

EX 1a Use excel file derivative.xlsx
Ex 1b Use java file if_x.java and if_xtest.java

EX2 : using derivative tables
f(x)=sin(x)
plot function, first,second and third derivatives in the region $0 \leq x \leq 2\pi$ by using

java file if_x.java and if_x1test.java

EX3 : using Richardson extrapolation formula

f(x)=sin(x)

find function value, first,second and third derivatives at x=0

find function value, first,second and third derivatives at x= π/4

find function value, first,second and third derivatives at x= π/3

find function value, first,second and third derivatives at x= π/2

find function value, first,second and third derivatives at x= π

EX4: using derivative tables (through if_xj.java)

Ex4a: plot function $f(x,y) = x^2y + xy^2$ use if_xjtest.java and if_xj.java

Ex4b: plot first derivative of function $\frac{\partial f(x,y)}{\partial x}$ use if_xjtest.java and if_xj.java

Ex4b: plot first derivative of function $\frac{\partial f(x,y)}{\partial y}$ use if_xjtest.java and if_xj.java

EX5 : using derivative tables

$f(x) = \frac{3x^3-5.2x^2+1.4x+2.0}{5x^2+4.0}$

find function value, first,second and third derivatives at x=0

find function value, first,second and third derivatives at x= 1

find function value, first,second and third derivatives at x= 2

find function value, first,second and third derivatives at x= 3

find function value, first,second and third derivatives at x=4

EX 5a Use excel file derivative.xlsx

Ex 5b Use java file if_x.java and if_xtest.java

EX26: using derivative tables

f(x)=sin(x)

plot function, first,second and third derivatives in the region $0 \leq x \leq 4$ by using

java file if_x.java and if_x1test.java

EX7 : using Richardson extrapolation formula

$$f(x) = \frac{3x^3 - 5.2x^2 + 1.4x + 2.0}{5x^2 + 4.0}$$

find function value, first,second and third derivatives at x=0

find function value, first,second and third derivatives at x= 1

find function value, first,second and third derivatives at x= 2

find function value, first,second and third derivatives at x= 3

find function value, first,second and third derivatives at x=4

EX 7a Use excel file derivative.xlsx

Ex 7b Use java file if_x.java and if_xtest.java

EX8: using derivative tables (through if_xj.java)

Ex4a: plot function $f(x,y) = f(x) = \frac{3x^3y - 5.2x^2y^2 + 1.4x + 2.0}{x^2 + 4.0}$ **use if_xjtest.java and if_xj.java**

Ex4b: plot first derivative of function $\frac{\partial f(x,y)}{\partial x}$ **use if_xjtest.java and if_xj.java**

Ex4b: plot first derivative of function $\frac{\partial f(x,y)}{\partial y}$ **use if_xjtest.java and if_xj.java**

HOMEWORK EXERCISES

Homework exercises will be done at home and will bring to next friday class printed no late exercises will be excepted.

W10HW1:

X	**0.99**	**2.10**	**3.22**	**4.40**	**5.70**	**7.12**	**8.01**	**8.37**	**9.32**	**9.98**
f(x)	**4.90**	**5.70**	**4.20**	**7.04**	**8.31**	**7.82**	**5.97**	**7.01**	**6.68**	**4.79**

Data given in the above table. Find the derivative by using cubic spline derivation method.

W10HW2:

A vehicle is at complete stop. When it is started velocity of the vehicle is recorded for the first 5 minutes. Calculate the acceleration of vehicle

t(minutes)	0	0,5	1,0	1,5	2,0	2,5	3,0	3,5	4,0	4,5	5,0
v (km/h)	0	4	7	11	15	21	25	30	32	35	40

Note : $v = \frac{dx}{dt}$ $a = \frac{d^2x}{dt} = \frac{dv}{dt}$

W10HW3:

Calculate velocity and acceleration by using below data with

j) cental difference method [with $O(h^4)$ error level],
k) forward difference method [with $O(h^2)$ error level],
l) backward difference method [with $O(h^2)$ error level],

Time, t (s)	0	1	2	3	4	5	6	7	8
distance, x (m)	0	0.7	1.8	3.4	5.1	6.5	7.3	8.0	8.4

Note : Velocity $v = \frac{dx}{dt}$Acceleration : $a = \frac{d^2x}{dt^2}$

W10HW4:

using derivative tables

f(x)=tan(x)

find function value, first,second and third derivatives at x=0

find function value, first,second and third derivatives at x= π/4

find function value, first,second and third derivatives at x= π/3

find function value, first,second and third derivatives at x= π/2

find function value, first,second and third derivatives at x= π

W10HW4 1a Use excel file derivative.xlsx

W10HW4 1b Use java file if_x.java and if_xtest.java

W10HW5:
: using derivative tables
f(x)=tan(x)
plot function, first,second and third derivatives in the region $0 \leq x \leq 2\pi$ by using
java file if_x.java and if_x1test.java

W10HW6: using Richardson extrapolation formula
f(x)=tan(x)
find function value, first,second and third derivatives at x=0
find function value, first,second and third derivatives at x= π/4
find function value, first,second and third derivatives at x= π/3
find function value, first,second and third derivatives at x= π/2
find function value, first,second and third derivatives at x= π

W10HW6: using derivative tables (through if_xj.java)
W10HW6a: plot function $f(x,y) = sin(x^2y) + xy^2$ **use if_xjtest.java and if_xj.java**
W10HW6b: plot first derivative of function $\frac{\partial f(x,y)}{\partial x}$ **use if_xjtest.java and if_xj.java**
W10HW6b: plot first derivative of function $\frac{\partial f(x,y)}{\partial y}$ **use if_xjtest.java and if_xj.java**

12.0 REFERENCES

1. Chapra, Steven C.; Canale, Raymod P., "Numerical Methods for Engineers", sixth Edition, 2010,McGraw-Hill, ISBN 978-007-126759-5
2. Pres, W. H.,Teulkolsky, S. A.,Vetterling, W. T., Flannery B. P.; "Numerical Recipes in C", 1992, Cambridge University Pres, ISBN 0-521-43108-5
3. Borse, G.J.;"Numerical Methods with MATLAB" 1997,PWS Publishing company, ISBN 0-534-93822-1
4. Ralston, A.;Rabinowitz, P;"A First Course in Numerical Analysis", second Edition, 1978, McGraw-Hill, ISBN 007-051158-6
5. Atkinson, K.E.;"An Introduction to Numerical Analysis", Second edition, 1989, John Wiley & Sons, ISBN 0-471-62489-6
6. Burden, R.L.,Faires,D.F.;"Numerical Analysis", Eight Edition, 2005,Thomson Brooks/Cole, ISBN 0-534-39200-8
7. Mathews,J.H.,Fink,K.D.;"Numerical Methods Using MATLAB",1999,Third Edition, Prentice Hall, ISBN 0-13-270042-5
8. Schilling,R.J.,Haris, S. L."Applied Numerical Methods for Engineers",2000, Brooks/Cole, ISBN 0-534-37014-4
9. Dahlquist, G.,Anderson N.;"Numerical Methods,1974,Prentice Hall, ISBN 0-13-627315-7
10. Mathews J. H.,"Numerical Methods for Mathematics, Science and Engineering", Second Edition, 1992, Prentice-Hall, ISBN 0-13-625047-5
11. Michalewicz, Z., "Genetic Algorithms+Data Structures = Evaluation Programs", second extended edition, 1992, Springer-Verlag, ISBN 3-540-58090-5
12. Karaoğlu, B.,"Sayisal Fizik",2004, Seyir Yayincilik, ISBN 975-92544-9-2
13. Bayram, M,"Nümerik Analiz", 2002, "Aktif Yayinevi", ISBN 975-6755-50-4
14. Tapramaz, R.,"Sayisal Çözümleme", 2002, Literatür Yayinlari, ISBN 975-8431-82-X
15. Scheid, F.,"SCHAUM'S OUTLINES Nümerik Analiz", 1988,Nobel Yayim Dağitim, ISBN 975-591-078-6
16. Fox, L., Mayers D. F., "Computing Methods for Scientist and Engineers", 1968, Oxford University Press
17. Johnson, R. L.,"Numerical Methods A Software Aproach", 1982, John Willey & Sons
18. Mitchell, M," An Introduction to Genetic Algorithms"1999, A Bradford Book The MIT Pres, ISBN 0-262-13316-4
19. Arnold, D. N. , "A Concise Introduction to Numerical Analysis", 1991
20. Nash, J.C. ,"A Compact Methods for Computers Linear Algebra and Function Minimisation"1990, Adam Hilger, ISBN 0-85274-318-1
21. Chenney, W.,Kinkaid, D.;"Numerical Mathematics and Computing", Third Edition, Brooks/Cole publishing
22. Çoban, M. Turhan, "Java Programlama Kılavuzu, Herkes için ", 2000, Alfa yayinevi, ISBN 975-316-631-1
23. Deitel & Deitel, Java How to Program, Second Edition, Prentice Hall, http://www.deitel.com, second edition,1998, ISBN 0-13-906249-1
22. Naughton, Patrick, t Schildt, Herbert ;"Java 1.1 The Complete Reference", Second Adition, 1998, Mc Graw Hill, ISBN 0-07-882436-2
23. Zukowski, John; "Mastering Java 1.2 (Renamed Java 2 by sun microsystems)",1998, Sybex publishing, ISBN 0-7821-2180-2
24. Learning Java, Covers Java 1.3, Patrick Niemeyer & Jonathan Knudsen, O'Reilly , 2000, ISBN 1-56592-718-4
25. Java Swing, Robert Eckstein, Marc Loy & Dave Wood, O'Reilly , 1998, ISBN 1-56592-455-X
26. JDBC Database Access with Java, The Java Series, Graham Hamilton, Rick Cattell, Maydene Fisher, Addison Wesley, 1997, ISBN 0-201-30995-5
27. Java Algorithms, Scott Robert Ladd, Mc Graw-Hill, 1998, ISBN 0-07-913696-6
28. Fundamentals of Database Systems, Elmasri & Navathe, Benjamin/Cummings, 1994, ISBN 0-8053-1753-8
29. Thinking in Java, Bruce Eckel, Prentice Hall, 2000, ISBN 013-659723-8
30. VisAD graphic package, http://www.ssec.wisc.edu/~billh/visad.html
31. Java Development kit ve 3D graphic package, http://java.sun.com

32. ALGLIN-NET internet site, (Gauss- Hermit integration) http://www.alglib.net/integral/gq/ghermite.php
33. ALGLIN-NET internet site, (Gauss-Laguerre integration) http://www.alglib.net/integral/gq/glaguerre.php
34. Gander, W. ,W. Gautschi, "Adaptive Quadrature - Revisited", BIT, Vol. 40, 2000, page. 84-101. http://www.inf.ethz.ch/personal/gander
35. H.J. Korsch H.-J. Jodl T. Hartmann,"Chaos, A Program Collection for the PC", Springer publishers, 2008, ISBN 978-3-540-74866-3
36. Forman S. Acton, "Numerical Methods that work", Mathematical Association of America, 1990, ISBN 0-88385-450-3
37. Alfio Quarteroni, Riccardo Sacco, Fausto Saleri, "Numerical Mathematics", Springer and Verlag, 1991, ISBN 0-387-98959-5
38. Perviz Moin, "Fundamentals of Enginering Numerical Analysis", Cambridge publications, 2010, ISBN 978-0-521-88432-7
39. J.H. Verner, Numerically optimal Runge--Kutta pairs with interpolants. Numerical Algorithms. **53**, (2010) pp. 383--396.
40. J.H. Verner, Improved Starting methods for two-step Runge--Kutta methods of stage-order p-3, Applied Numerical Mathematics, **10**, (2006) pp. 388--396.
41. J.H. Verner, Starting methods for two-step Runge--Kutta methods of stage-order 3 and order 6, J. Computational and Applied Mathematics, **185**, (2006) pp. 292--307.
42. T. Macdougall and J.H. Verner, Global error estimators for Order 7,8 Runge--Kutta pairs, *Numerical Algorithms* **31**, (2002) pp. 215--231.
43. Z. Jackiewicz and J.H. Verner, Derivation and implementation of two-step Runge-Kutta pairs. *Japan Journal of Industrial and Applied Mathematics* **19** (2002), pp. 227--248.
44. R.W. Lewis, P. Nithiarasu, K. N. Seetheramu, Fundamentals of the Element Method for Heat and Fluid Flow, John Willey & Sons, 2004, ISBN 0-470-84788-3
45. Computational Methods for Algebraic Spline Surfaces , Tor Dokken, Bert Jüttler, 2000, ISBN 3-540-23274-5, Springer Berlin Heidelberg New York
46. Peter Lancaster, Kestutis Salkauskas, Curve and Surface Fitting, an introduction, Academic Press, Harcourt Brace Jovanovich Publishers, 1986, ISNB 0-12-436060-2
47. Bength Fornberg, "Generation of Finite Difference Formulas on Arbitrary Spaced Grids", Mathematics of Computation, Volume 51, Number 184, October 1988, pages 699-706
48. S. Abbasbandy, Improving Newton-Raphson method for nonlinear equations by modified Adomian decomposition method, Appl. Math. Comput. 145 (2003), pp. 887-893.
49. Shenfeng Li, Rujing Wang, Two fourth order Iterative Methods Based on Continued Fraction for Root-finding Problems, World Academy of Science, Engineering and Technology 60 2011
50. Hasan, Mohammed A., On the Derivation of Higher Order Root-finding Methods, Proceedings of the 2007 American Control Conference, ThA07.3, New York City, USA, July 11-13 2007
51. Xin-She Yang.: A New Metaheuristic Bat-Inspired Algorithm. Nature Inspired Cooperative Strategies for Optimization (NICSO 2010). Studies in Computational Intelligence Vol. 284 (2010) 65-74.
52. Xin-She Yang, Engineering Optimisation, An introduction with Metaheuristic applications, John Willey & Sons , 2010, ISBN 978-0-470-58246-6
53. Z. W. Geem, J. H. Kim, and G. V. Loganathan, "A new heuristic optimization algorithm: Harmony search", *Simulation*, 76, 60-68 (2001).
54. Ashrafi, S.M., Dariane, A.B., Performance evaluation of an improved harmony search algorithm for numerical optimization: Melody Search (MS). Eng. Appl. Artif. Intel. (2012), http://dx.doi.org/10.1016/j.engappai.2012.08.005
55. Parikshit Yadav, Rajesh Kumar, S.K. Panda, C.S. Chang, "An Intelligent Tuned Harmony Search algorithm for optimization", Information Sciences 196(2012) 47-72 elsevier publ.
56. Yang, X. S. and Deb, S., 2009. 'Cuckoo search via Levy flights', *Proceeings of World Congress on Nature & Biologically Inspired Computing* (NaBIC 2009,India), IEEE Publications, USA, pp. 210-214.
57. [1] Civicioglu P., Transforming geocentric cartesian coordinates to geodetic coordinates by using differantial search algorithm, Computers & Geosciences 46, 229-247, 2012
58. [2] Vito., T., Elio, T., Kevin, M.P., et al., Swarm cognition: an interdisciplinary approach to the study of self-organising biological collectives, Swarm Intelligence 5, 3-18, 2011

59. [1] Wu, B., Qian, C., Ni, W., Fan, S., Hybrid harmony search and artificial bee colony for global optimization problems, Computers and Mathematics with Applications 64, 2621-2634, 2012
60. [2] Pan, Q., Suganhtan, P.H., Tasgetiren M.F., et al., A self – adaptive global best harmony search algorithm for continuous optimization problems, Appl. Math. Comput. 216, 830-848, 2010
61. [3] Kennedy, J., Eberhart, R.C., Shi, Y., Swarm Intelligence, Morgan Kaufmann Publishers, San Francisco, 2001
62. [1] J, Sun., B, Feng., W, Xu., "Particle Swarm Optimization with Particles Having Quantum Behavior", *IEEE Proc. Of Congress on Evolutionary Computation*, 2004
63. [2] J, Sun., W, Xu., B, Feng., "Global Search Strategy of quantum Behaved Particle Swarm Optimization", *IEEE Proc.of Conference on Cybernatics and Intelligent Systems*, Singapore, 2004
64. [3] van den Bergh, F., A, Engelbrecht., A new locally convergent particle swarm optimizer. In Proceedings of the IEEE Conference on systems, Man, and Cybernatics Hammamet, Tunisia, 1-6, 2002
65. [4] He, S., Q, Wu., J, Wen, J. Saunders., R, Baton., A particle swarm optimizer with passive congregation, Biosystems 78, 135-147, 2004
66. [5] Angeline, P.J., Evolutionary optimization versus particle swarm optimization: Phlosophy and performance differences. Proceedings of the 7th Annual Conference on Evoltionary Programming, San Diego, USA , 601-610, 1998a
67. [6] Xie, X.F., W.J. Zhang ve Z.L. Yang, Adaptive particle swarm optimization on individual level , 6th International Conference on Signal Processing, 1215-1218, 2002a
68. [7] He, Q., C, Han., An improved particle swarm optimization algorithm with disturbance term. Comput. Intell. Bioinfo. 4115, 100-108, 2006
69. [8] Padhye, N., Topology optimization of compliant mechanism using multi-objective particle swarm optimization. Proceedings of the Conference Companion on Genetic and Evolutionary Computation, 2008
70. [9] Jiao , B., Z. Lian and X. Gu, A dynamic inertia weight particle swarm optimization algorithm. Chaos, Solitons Fract. 37, 698-705
71. [10] Engelbracht A., Heteregenous particle swarm optimization. Proceedings of the 7th International conference on Swarm Intelligence, 191-202, 2010
72. [11] Chuang, L.Y., S.W. Tsai., C.H. Yang., Chaotic catfish particle warm optimization for solving global numerical optimization problems, Appl. Math. Comput. 217, 6900-6916, 2011
73. [12] Clerc, M., Kennedy, J., "The particle swarm: explosion, stability, and convergence in a multi-dimensional complex space." IEEE Transactions on Evolutionary Computation, vol. 6, no.1, pp.58-73, 2002
74. [13] Sun, J., Fang, W., Wu, X.J., Palade, V., Xu, W.B., Quantum-behaved particle swarm optimization: analysis of the individual particle's behavior and parameter selection, Evolutionary Computation, doi:10.1162/EVCO_a_00049.
75. [14] Su, J., Fang, W., Palade V., Wu, X., Xu, W., Quantum behaved particle swarm optimization with Gaussian distributed local attractor point, Applied Mathematics and Computation 218, 3763-3775, 2011
76. [15] Xi, M., Sun,J., Xu, W., An improved quantum behaved particle swarmoptimization algorithm with weighted mean best position, Applied Mathematics and Computation 205, 751-759, 2008
77. [16] Coelho, L.D.S., A quantum particle swarm optimizer with chaotic mutation operator, Chaos, Solitons & Fractals 37, 1409-1418, 2008
78. [17] Liu B., Wang, L., Jin Y.H., Tang, F., Huang D.X., Directing orbits of chaotic systems by particle swarm optimization, Chaos, Solitons & Fractals 29(2) , 454-461, 2006
79. [18] Zaslavski ,G.M., "The Simplest case of a strange attractor ",*Phys. Lett. A* 69 (3), 145-147, 1978
80. [19] Yang, X., Shi, P., Shen, W., Jiang, K., Pang, S., Multiobjective Quantum-behaved Particle Swarm Optimization with Entopy Based Density Assesment and Chaotic Mutation Operator, Journal of Computational Information Systems 9: 10 , 3873-3881, 2013
81. [20] Tavazoei, M.S., Haeri, M., Comparison of different one dimensional masps as chaotic search pattern in chaos optimization algorithms, Appl. Math. Comput. 187, 1076-1085, 2007
82. [21] Hilborn, R.C., Chaos and nonlinear dynamics: an introduction for scientists and engineers, 2nd ed. New York: Oxford Univ. Press; 2004
83. [22] May, R.M., Simple mathematical models with very complicated dynamics, Nature 261, 459-467, 1976
84. [23] Li, Y., Deng, S., Xiao, D., A ovel Hash algorithm construction based on chaotic neural network, Neural. Comput. Applic. 20, 133-141, 2011
85. [24] Ott, E., Chaos in dynamical systems, Cambridge, U.K. : Cambridge University Press; 2002

86. [25] Tomida, A.G., Matlab Toolbox and GUI for Analyzing One-Dimensional Chaotic Maps, In: International Conference on Computational Sciences and Its Applications ICCSA, IEEE Press, 321-330, 2008
87. Heat Conduction, M. Necati Özışık, Second Edition, John Wiley & Sons, Inc. ISBN 0-471-53256-8
88. Fundamentals of the Finite Element Method for Heat and Fluid Flow, Roland W. Lewis, Perumal Nithiarasu, Kankanhally N. Seetharamu, Wiley publishers, ISBN 0-470-847-88-3
89. Generation of Finite Difference Formulas on Arbitrary Spaced Grids, Bength Fornberg, Mathematics of Computation, Volume 51, Number 184, October 1988, pp 699-706
90. On the rapid computation on various polylog-arithmetic constants, David H. Bailey, Peter B. Borwein, and simon Plouffe, Mathematics of Computation 66, no 218 (1997) 903-913
91. https://en.wikipedia.org/wiki/Artificial_neural_network
92. A. Askerzadeh, A novel metaheuristic method for solving constrained engineering optimization problems: Crow search algorithm, Computers and Structures, 169, 2016, pp. 1-12.
93. A. A. Heidari, S. Mirjalili, H. Faris, I Aljarah, Harris hawks optimisation:Algorithm and applications, Elsevier Future Generation Computer Systems, 97(2019) p. 849-872
94. A. KAveh, A. Dadras, A novel meta-heuristic optimization algorithm: Thermal exchange optimization, Elsevier Advances in Engineering Software, 110(2017) 69-84