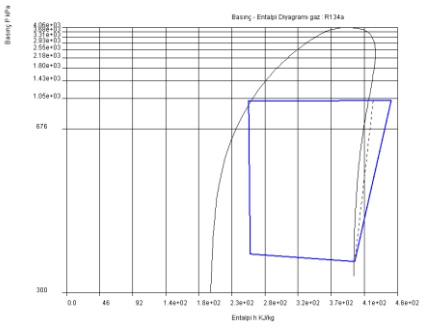
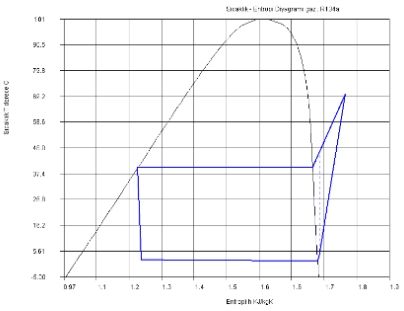
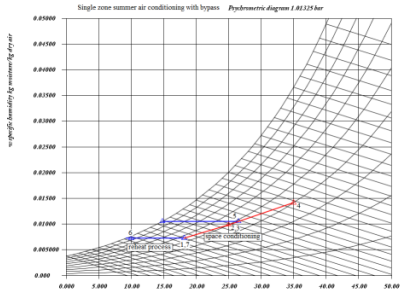
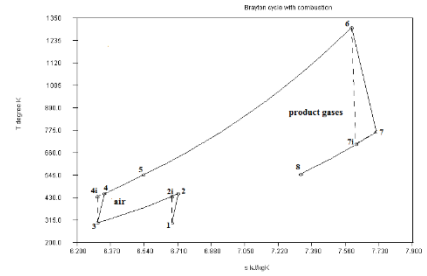
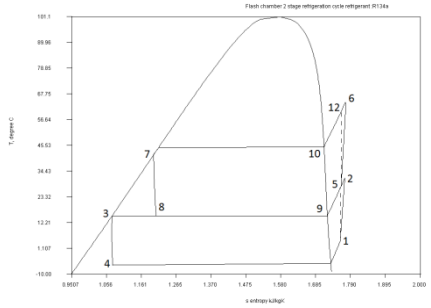
NUMERICAL THERMODYNAMICS

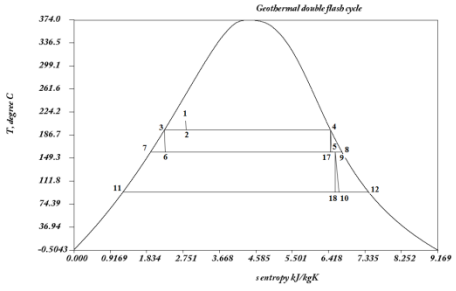












Dr. M. Turhan ÇOBAN
Ege University, School of Engineering,
Department of Mechanical Engineering

INDEX

1. Introductory Concepts
1.1 Introduction to thermodynamics
1.2 Root finding methods
2. Equations of States
2.1 Ideal gas equation of states
2.2 Cubic equation of states
2.3 Lee-Kesler equation of states
2.4 ISO 17584-2005 Equation of States
2.5 Helmholts Equation of States for Water and Ammonia
2.6 Martin-Hou equation of states
2.7 Benedict-Webb-Rubin Equation of States
2.8 Peng-RObinson-Stryjek-Vera Equation of States
2.9 Thermodynamic properties of amonia-water mixture
2.10 thermodynamic properties of lithiumbromide-water mixture
3. Power cycles
3.1 Carnot cycle
3.2 Air standart Brayto-Ericsson cycle
3.3 Rankine cycle
3.4 Otto cycle
4. Refrigeration and heat pumps
4.1 Carnot cycle
4.2 Refrigeration cycle
4.3 Brayton-Ericsson cycle
4.4 Liquifaction of gases
4.5 Absorption refrigeration
5. Air conditioning processes
5.1 Properties of moisturised air
5.2 Air conditioning (psychrometric) processes
5.3 Cooling towers
6. Combustion and Thermochemistry
6.1 basic of combustio processes
6.2 chemical equilibrium with stoichiometric approach
6.3 Chemical equilibrium with Gibbs free energy minimisation
6.3 Chemical kinetics of combustion
6.4 Brayton ericcson (gas turbine) cycle with combustion
6.4 Fuel cells
6.5 boiler efficiency analysis
7. Realtime thermodynamic applications
8. Appendix
9. 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 Üniversitesi, School of Engineering, Department of mechanical engineering(Turkey), Gebze Yüksek Teknoloji Enstitüsü, Computer science(Turkey), Ege Üniversitesi, School of Engineering, Department of mechanical engineering(Turkey). 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(Turkey), 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

1. INTRODUCTORY CONCEPTS

1.1 INTRODUCTION TO THERMODYNAMICS

Previous to working on thermodynamic systems, processes and cycles let us investigate the introduction to thermodynamics. We will start by using Callen's postulates. Herbert B. Callen in his classic thermodynamic book gives us three basic postulates

Postulate I : There exist particular states (called equilibrium states) of simple systems that, macroscopicaly, are characterized completely by the internal energy U, the volume V, and the mole numbers N_1, N_2,N_3,…,N_r of the chemical components.

Postulate II: There exist a function (called the entropy S) of the extensive parameters of any composite system, defined for all equlibrium states and having the following property: the values assumed by the extensive parameters in the absanse of an internal constraint are those that maximize the entropy over the manifold of constraint equilibrium states.

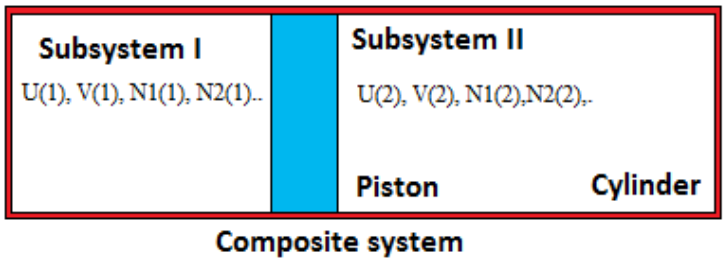


Figure 1.1 : 2 A composite system with two subsystems

In figure 1 a composite system with two subsystem is shown. This two subsystem are seperated by a mechanical manifold called piston. Which permits volume parameters to change in sub systems. A selective membrane can permit a specific matter to exchange between subsystems If the membrane is a selective membrane for certain materials mole number can be transfer from one subsystem to the other. For example a zirconiumoxide membrane is a selective membrane for oxygen ions. Copper or gold membrane can transfer internal energy between subsystems due to high thermal conductivity coefficient(manifold for internal energy). In this, composite system and subsystems are still in equilibrium states.

Postulate III: The entropy of a composite system is additive over the constituent subsystems. The entropy is continuous and differentiable and is a monotonically increasing function of the energy.

$$S = \sum_{\alpha} S^{(\alpha)} \quad (1.1)$$

S in the equation is entropy and α indicates composite subsystems. In the equilibrium states each subsystem entropy is the function of volume, internal energy and mole numbers of species.

$$S^{(\alpha)} = S^{(\alpha)}\left(U^{(\alpha)}, V^{(\alpha)}, N_1{}^{(\alpha)}, N_2{}^{(\alpha)}, \ldots\right) \quad (1.2)$$

As it is seen entropy of the simple system is simply function of expressed as internal energy U, volume V and mole number of the subspecies N_1, $N_{2,\ldots,}$ N_r. Postulate II also tells us changing of the entropy function between subsystems cause the derivative of the function with respect to changing parameter will be positive.

$$\left(\frac{\partial S}{\partial U}\right)_{V,N_1,N_2,\ldots..,N_r} > 0 \quad (1.3a)$$

$$\left(\frac{\partial S}{\partial V}\right)_{U,N_1,N_2,\ldots..,N_r} > 0 \quad (1.3b)$$

$\left(\frac{\partial S}{\partial N_i}\right)_{U,V,N_{j,j\neq i}} > 0$ (1.3c)

$S = S(U,V,N_1,N_2,..)$ (1.4a)
$V = S(U,S,N_1,N_2,..)$ (1.4b)
$N_i = S(U,V,S,N_{j,j=1..m,j\neq i})$ (1.4c)

Because of the continuous and derivable form of entropy function will allow us the change the function and express it as internal energy function as function of entropy, volume and moles $U = U(S,V,N_1,N_2,..)$ (1.5) . postulate III are also indicates that total entropy could be expressed as addition of entropy of the subsystems . Therefore

$S = S(U,V,N_1,N_2,..) = NS(U/N,V/N,N_1/N,N_2/N,..)$ (1.6)

$\frac{1}{\mathrm{N}} = \frac{1}{\sum_{\mathrm{k}} \mathrm{N_k}}$ (1.7)

Can be written , so properties of the system can be expressed with internal properties of the system. This properties called specific properties (specific internal energy, specific volume, specific mol numbers)

$u = \frac{U}{N}$ (1.8a)
$v = \frac{V}{N}$ (1.8b)
$s = \frac{S}{N}$ (1.8c)

Postulate IV: Entropy of a system will vanish when

$\left(\frac{\partial U}{\partial S}\right)_{V,N_1,\ldots,N_r} = 0$ (1.9)



Herbert B. Callen July 1, 1919 – May 22, 1993

Equation

$S = S(U,V,N_1,N_2,..)$ or $U = U(S,V,N_1,N_2,..)$ is called equation of state. We are already defined that equation of state is differentiable. If the equation of state is written in differential form:

$dU = \left(\frac{\partial U}{\partial S}\right)_{V,N_1,\ldots,N_r} dS + \left(\frac{\partial U}{\partial V}\right)_{S,N_1,\ldots,N_r} dV + \sum_{j=1}^{r}\left(\frac{\partial U}{\partial N_j}\right)_{S,N_1,\ldots,N_r} dN_j$ (1.10)

Properties defined by first and second postulates, S,U,V, $\mathrm{N_j}$, are called extensive parameters, while thermodynamic parameters defined by the derivative functions are called *intensive parameters*. Intensive can be called by different names such as:

$\left(\frac{\partial U}{\partial S}\right)_{V,N_1,\ldots,N_r} = T$ (1.11) This function is called as temperature

$-\left(\frac{\partial U}{\partial V}\right)_{S,N_1,\ldots,N_r} = P$ (1.12) This function is called as pressure

$\left(\frac{\partial U}{\partial N_j}\right)_{S,N_1,\ldots,N_r} = \mu_j$ (1.13) This function is called electrochemical potential of jth component.

Equation 1.10 can be re-written as a function of these new intensive parameters as:

$dU = TdS - PdV + \sum_{j=1}^{r} \mu_j dN_j$ (1.14) .

In this equation each term has a particular mean in classical thermodynamics –PdV term is called quasi-static mechanical work,

$dW_M = -PdV$ (1.15)

$\mu_j dN_j$indicates chemical work

$dW_c = \sum_{j=1}^{r} \mu_j dN_j$ (1.16)

and TdS term indicates heat transfer

$dQ = TdS$ (1.17) If these equavalent terms are substituted into the equation, equation becomes:

$dU=dQ+dW_M+dW_c$ (1.18) this form of the equation is called first law of thermodynamics of conservation of energy laws in classical thermodynamics.

In this differential form, three new thermodynamic properties are defined. These properties can also be used to define equation of states in the following forms:

$T=T(S,V,N_1,N_2,..)$ (1.19)

$P=P(S,V,N_1,N_2,..)$ (1.20)

$\mu_j= \mu_j (S,V,N_1,N_2,..)$ (1.21)

Intensive parameters can also be defined by using derivatives of the entropy equation of state:

$$dS = \left(\frac{\partial S}{\partial U}\right)_{V,N_1,\ldots,N_r} dU + \left(\frac{\partial S}{\partial V}\right)_{S,N_1,\ldots,N_r} dV + \sum_{j=1}^{r} \left(\frac{\partial S}{\partial N_j}\right)_{S,N_1,\ldots,N_r} dN_j \quad (1.22)$$

But In this case intensive parameters will be :

$$\left(\frac{\partial S}{\partial U}\right)_{V,N_1,\ldots,N_r} = \frac{1}{T} \quad (1.23)$$

$$\left(\frac{\partial S}{\partial V}\right)_{S,N_1,\ldots,N_r} = \frac{P}{T} \quad (1.24)$$

$$\left(\frac{\partial S}{\partial N_j}\right)_{U,V,N_i,i=1..m,i\neq j} = -\frac{\mu_j}{T} \quad (1.25)$$

A relation among intensive and extensive properties such as eqn 1.5,1.6,1.19,1.20,1.21 is called equation state. As an example let us introduce the fundamental equation of a perfect monatomic gas[77]

$$S = S(U,V,N) = \left(\frac{N}{N_0}\right) S_0 + NRln\left[\left(\frac{U}{U_0}\right)^{3/2} \left(\frac{V}{V_0}\right) \left(\frac{N}{N_0}\right)^{5/2}\right] \quad (1.26)$$

Where subscript 0 refers to a reference state and R=8314.5 J/kmolK is the universal gas constant.

$$\frac{1}{T} = \left(\frac{\partial S}{\partial U}\right)_{V,N} = \frac{3}{2}\frac{NR}{U} \quad (1.27)$$

$$\frac{P}{T} = \left(\frac{\partial S}{\partial V}\right)_{S,N} = \frac{NR}{V} \quad (1.28)$$

$$-\frac{\mu}{T} = \left(\frac{\partial S}{\partial N_j}\right)_{U,V} = \frac{S_0}{N_0} + Rln\left[\left(\frac{U}{U_0}\right)^{\frac{3}{2}} \left(\frac{V}{V_0}\right) \left(\frac{N}{N_0}\right)^{\frac{5}{2}}\right] - \frac{5}{2}R \quad (1.29)$$

This equations can be arranged as:

$$U = \frac{3}{2}NRT \quad (1.30)$$

$$PV = NRT \quad (1.31)$$

$$\mu = RT\left[\frac{5}{2} - \frac{S_0}{RN_0} + ln\left(\frac{NRT^{\frac{5}{2}}}{V_0 T_0^{\frac{3}{2}}}\right) + \ln(P)\right] \quad (1.32)$$

In here, let us repeat the difference between extensive and intensive parameters. Extensive parameters are independent parameters and intensive parameters are defined as a derivative function of intensive parameters. In practical measurements properties like temperature, pressure and volume can be measured more easily by compare to properties like internal energy and entropy. For this reason these properties can be easily converted to each other, or integration as well as derivation of the properties should be used continously. Integration of the function is more difficult compare to derivation because integration requires additional properties to give a unique solution. Consider mathematical function

$Y=Y(X_0,X_1,X_2,..,X_t)$ and its derivative $P_k = \left(\frac{\partial Y}{\partial X}\right)$

In order to express thermodynamic properties as a function of P_k, We will utilise Legendre transforms. To understand process more easily let us consider a simpler function of Y=Y(X). This equation can be shown in X-Y cartesian coordinate system



Figure 1.2 Y(x) function

If we consider derivative of this function $P = \left(\frac{\partial Y}{\partial X}\right)$, P is the tangent of the original function at the given function. Now, we would like to use P instead of X in the equation as independent variable so it will be in the form of that Y=Y(P). The mathematical process to achive this is the integration of the equation. But in the integration process we will obtain a family of equations, instead of a single equation. Additional knowledge is required to obtain a single equation. This problem can be overcome by defining a seconday variable Ψ to express the equation. In this case conversion of equation will be Ψ=Ψ(P) instead. In this equation Ψ is the crossing of tangent to Y axis (X=0) function Ψ is called Legendre transformation of Y.

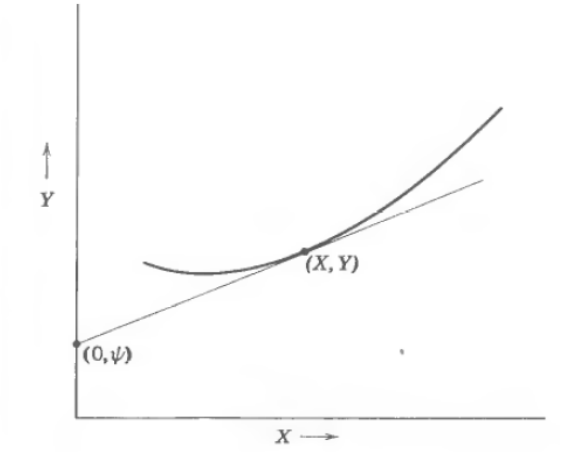


Figure 1.3 Ψ=Ψ(P) and y=Y(X) functions.

$P = \left(\frac{Y-\Psi}{X-0}\right)$ (1.33)

$\Psi = Y - PX$ (1.34) can also be written. Consider that Y=Y(X) equation is given. By using derivative of this equation, derivative function P can be obtained. By using equation 1.26 Ψ can be found and by eliminating X and Y Legendre transform can be completed as Ψ=Ψ(P) . Similarly inverse Legendre transform can also be performed by taking derivative of Ψ=Ψ(P)

$d\Psi = DY - PdX - XdP$ (1.35)

$-X = \left(\frac{d\Psi}{dP}\right)$ (1.36)

If Ψ and P elimined from this equation inverse equation can be obtained again.

Y=Y(X)	Ψ=Ψ(P)
$P = \left(\frac{\partial Y}{\partial X}\right)$	$-X = \left(\frac{d\Psi}{dP}\right)$
$\Psi = Y - PX$	Y=XP+Ψ
Elimination of X and Y gives Ψ=Ψ(P)	Elimination of P and Ψ gives Y=Y(X)

Now Let us apply Legendre transform to the thermodynamic equation of states. If $U = U(S,V,N)$ is taken as the main equation of state and Legendre transform with respect to S is carried out

$U = U(S,V,N)$	$A = A(T,V,N)$
$T = \left(\frac{\partial U}{\partial S}\right)$	$-S = \left(\frac{\partial A}{\partial T}\right)$
$A = U - TS$	U=A+TS
Eliminating U and S gives $A = A(T,V,N)$	Elimination of A gives $U = U(S,V,N)$

This new Legendre transform function is called Helmholtz free energy function



Hermann Ludwig Ferdinand von Helmholtz, 1821-1894

Now the main equation of state is again $U = U(S,V,N)$ and now Legendre transform with respect to V is taken

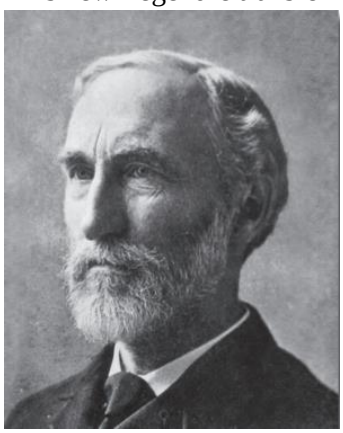
$U = U(S,V,N)$	$H = H(S,P,N)$
$-P = \left(\frac{\partial U}{\partial V}\right)$	$V = \left(\frac{\partial H}{\partial P}\right)$
$H = U + PV$	U=H-PV
If U and V is eliminated	If H ve P is eliminated
$H = H(S,P,N)$	$U = U(S,V,N)$

This new Legendre transform function is called enthalpy function

Again the main equation of state is $U = U(S,V,N)$, but in this case Legendre transform with respect to both T and V is taken

$U = U(S,V,N)$	$G = G(T,P,N)$
$T = \left(\frac{\partial U}{\partial S}\right)$	$-S = \left(\frac{\partial G}{\partial T}\right)$
$-P = \left(\frac{\partial U}{\partial V}\right)$	$V = \left(\frac{\partial G}{\partial P}\right)$
$G = U - TS + PV$	U=G+TS-PV
U , T and V is eliminated	G, T and P is eliminated
$G = G(T,P,N)$	$U = U(S,V,N)$

This new Legendre transform function is called Gibbs free energy function



Josiah Willard Gibbs, 1839-1903

This transfers can also be carried out for chemical potentials. Legendre transform for T and μ becomes

$U = U(S,V,N)$ $T = \left(\frac{\partial U}{\partial S}\right)$ $\mu = \left(\frac{\partial U}{\partial N}\right)$ $U[T,\mu] = U - TS - \mu N$ U , S and N eliminated $U[T,\mu] = U[T,\mu](T,V,\mu)$	$U[T,\mu] = U[T,\mu](T,V,\mu)$ $-S = \left(\frac{\partial U[T,\mu]}{\partial T}\right)$ $-N = \left(\frac{\partial U[T,\mu]}{\partial \mu}\right)$ U=U[T,μ]+TS+μN U[T,μ], T and μ is eliminated $U = U(S,V,N)$

This Legendre transformation function does not have any specific name so it is remained as U[T,μ]. If Entropy equation of state is taken and Legendre transforms are taken, these set of functions are called Mathieu functions.

$S = S(U,V,N)$ $\frac{1}{T} = \left(\frac{\partial S}{\partial U}\right)$ $S[1/T] = S - \frac{1}{T}U = -\frac{A}{T}$ U and S eliminated $S[1/T] = S[1/T](T,V,N)$	$S[1/T] = S[1/T](T,V,N)$ $-\frac{1}{T}U = \left(\frac{\partial S[1/T]}{\partial T}\right)$ S=S[T,μ]+$\frac{1}{T}U$ S[T,μ], T is eliminated $S = S(U,V,N)$

Before going any further in thermodynamic definitions some additional definitions of some partial derivatives are given below:

$\alpha = \frac{1}{V}\left(\frac{\partial V}{\partial T}\right)_P$ (1.37)

This function is called as thermal expansion coefficient:

$\kappa_T = -\frac{1}{V}\left(\frac{\partial V}{\partial P}\right)_T$ (1.38)

This function is called as isothermal compressibility:

$C_P = \frac{T}{N}\left(\frac{\partial S}{\partial T}\right)_P$ (1.39) This function is called specific heat at constant pressure definition at eqn 1.8c is utilized then equation becomes:

$C_P = T\left(\frac{\partial s}{\partial T}\right)_P$ (1.39a) If the equation is written for specific enthapy

$C_v = \frac{T}{N}\left(\frac{\partial S}{\partial T}\right)_V$ (1.40) This function is called specific heat at constant volume If specific enthapy definition at eqn 1.8c is utilized then equation becomes:

$C_v = T\left(\frac{\partial s}{\partial T}\right)_V$ (1.40a)

Now let us went back to the differential form of the internal energy equation of state

$dU = \left(\frac{\partial U}{\partial S}\right)_{V,N} dS + \left(\frac{\partial U}{\partial V}\right)_{S,N} dV + \left(\frac{\partial U}{\partial N}\right)_{V,S} dN$ (1.10)

$dU = TdS - PdV + \mu dN$ (1.12)

If the second derivatives of the dU equation of state is taken , the following relations are formed

$$\left(\frac{\partial^2 U}{\partial S \partial V}\right)_{S,N} = \left(\frac{\partial^2 U}{\partial V \partial S}\right)_{V,N} \quad \rightarrow \quad \left(\frac{\partial T}{\partial V}\right)_{S,N} = -\left(\frac{\partial P}{\partial S}\right)_{V,N} \quad (1.34)$$

As it is seen from the equations second derivatives will be equal to each other. The same rule of second derivatives can also apply to any Legendre transformation functions.

For Helmholtz free energy functions : $A = A(T, V, N)$

$$dA = \left(\frac{\partial A}{\partial T}\right)_{V,N} dT + \left(\frac{\partial A}{\partial V}\right)_{T,N} dV + \left(\frac{\partial U}{\partial N}\right)_{V,S} dN \quad (1.41)$$

$$dA = -SdT - PdV + \mu dN \quad (1.42)$$

$$\left(\frac{\partial^2 A}{\partial T \partial V}\right)_{T,N} = \left(\frac{\partial^2 A}{\partial V \partial T}\right)_{V,N} \quad \rightarrow \quad \left(\frac{\partial S}{\partial V}\right)_{T,N} = \left(\frac{\partial P}{\partial T}\right)_{V,N} \quad (1.43)$$

Entalpy equation : H=H(S,P,N)

$$dH = \left(\frac{\partial H}{\partial S}\right)_{P,N} dS + \left(\frac{\partial H}{\partial P}\right)_{S,N} dP + \left(\frac{\partial H}{\partial N}\right)_{P,S} dN \quad (1.44)$$

$$dH = TdS + VdP + \mu dN \quad (1.45)$$

$$\left(\frac{\partial^2 H}{\partial S \partial P}\right)_{S,N} = \left(\frac{\partial^2 H}{\partial P \partial S}\right)_{P,N} \quad \rightarrow \quad \left(\frac{\partial T}{\partial P}\right)_{S,N} = \left(\frac{\partial V}{\partial S}\right)_{P,N} \quad (1.46)$$

Gibbs free energy equation : G=G(T,P,N)

$$dG = \left(\frac{\partial G}{\partial T}\right)_{P,N} dT + \left(\frac{\partial G}{\partial P}\right)_{T,N} dP + \left(\frac{\partial G}{\partial N}\right)_{P,T} dN \quad (1.47)$$

$$dG = -SdT + VdP + \mu dN \quad (1.48)$$

$$\left(\frac{\partial^2 G}{\partial T \partial P}\right)_{T,N} = \left(\frac{\partial^2 G}{\partial P \partial T}\right)_{P,N} \quad \rightarrow \quad -\left(\frac{\partial S}{\partial P}\right)_{T,N} = \left(\frac{\partial V}{\partial T}\right)_{P,N} \quad (1.49)$$

This set of equations of second derivatives of Equation of states are called **Maxwell relations**, and by using these relations variables of equations of states can be converted into equations of known variables such as **T,P and V**. Addition to Maxwell relations, mathematical relations can be applied as well $\left(\frac{\partial X}{\partial Y}\right)_Z = \frac{1}{\left(\frac{\partial Y}{\partial X}\right)_Z}$ (1.50) and

$$\left(\frac{\partial X}{\partial Y}\right)_Z \left(\frac{\partial Y}{\partial Z}\right)_X \left(\frac{\partial Z}{\partial X}\right)_Y = -1 \quad (1.51)$$

If an equation of state is given then all thermodynamic properties can be calculated by Consider an equation of state in the form of P(T,V)

$ds = \left(\frac{\partial s}{\partial T}\right)_v dT + \left(\frac{\partial s}{\partial v}\right)_T dv$ (1.52) where $v = \frac{V}{N}$ $s = \frac{S}{N}$ can be written. If equation $C_v = C_v = T\left(\frac{\partial s}{\partial T}\right)_V$ (1.40a) and Maxwel relation eqn $\left(\frac{\partial S}{\partial V}\right)_{T,N} = \left(\frac{\partial P}{\partial T}\right)_{V,N}$ (1.43) is used, equation becomes

$$ds = \frac{C_v}{T} dT + \left(\frac{\partial P}{\partial T}\right)_v dv \quad (1.53)$$

$$s = s_0 + \int_{T_0}^{T} \frac{C_v}{T} dT + \int_{v_0}^{v} \left(\frac{\partial P}{\partial T}\right)_v dv \quad (1.53a)$$

u equation of state 1.14 rewritten as $du = Tds - Pdv$ and above equation is substituted for ds

$$du = T\left(\frac{C_v}{T} dT + \left(\frac{\partial P}{\partial T}\right)_v dv\right) - Pdv$$

$$du = C_v dT + \left(T\left(\frac{\partial P}{\partial T}\right)_v - P\right) dv \quad (1.54) \text{ integration of the equation gives}$$

$$u = u_0 + \int_{T_0}^{T} C_v dT + \int_{v_0}^{v} \left(T\left(\frac{\partial P}{\partial T}\right)_v - P\right) dv \quad (1.54a)$$

As it is seen, an equation of state of P(T,V) is suitable to calculate internal energy and entropy. Of course function like enthalpy can easily be evaluated from these equations by using h=u+pv equation. As a common equation of state Helmholts free energy equation is also used quite commonly. Advantages of Helmholts equation is that you can express all the thermodynamic functions in derivative forms.

$dA = -SdT - PdV$ (1.36a)

$S = \left(\frac{\partial A}{\partial T}\right)_V$ (1.55)

$P = -\left(\frac{\partial A}{\partial V}\right)_T$ (1.56)

$dU = TdS - PdV$

When thermodynamic properties are considered, phase change region is always considered as additional important region. The same equations are developed so far can be adopted to phase change region. Consider the equation

$dh = Tds + vdP$ (1.39 a). In the phase change of pure substances pressure remains constant for a given temperature. So the equation becomes: $dh = Tds$ (1.39b) . This equation can be written as

$(h_g - h_f) = T(s_g - s_f)$ (1.39c) In this equation h_g entalphy of saturated vapour and h_f is the enthalpy of saturated liquid. Similarly, s_g entropy of saturated vapour and s_f is the entropy of saturated liquid. If equation 1.37 is applied to this region

$\left(\frac{\partial S}{\partial V}\right)_T = \left(\frac{\partial P}{\partial T}\right)_v$

$\left(\frac{s_g - s_f}{v_g - v_f}\right)_T = \left(\frac{\partial P}{\partial T}\right)_v$ (1.57a) If this equation merge with equation 1.39c it becomes

$\left(\frac{h_g - h_f}{T(v_g - v_f)}\right)_T = \left(\frac{\partial P}{\partial T}\right)_v$ (1.57b). This equation is called **Classius –Clapeyron** equation.



Benoit Paul Emile Clapeyron, 1799-1864



Rudolf Julius Emmanuael Classius, 1822-1888

As it is seen from here if P(T) relation in the region is known enthapy and entropy knowledge can be derived from enthalpy information, but additional saturated vapor and liquid specific volume knowledge is required. This values can be obtain from equation of states or be given as separate relations.

1.2 ROOT FINDING METHODS

Basic equation of states are usually given as P(T,v) or A(v,T) or G(v,T) form. Then other properties such as enthalphy, entropy etc. can also be calculated from these properties as

h(T,v), s(T,v). If any other property set is required, such as v(T,P), P(h,v) .. a root finding method should be employed. For example to find v(T,P) following root function equation should be solved

$$P(T,v) - P = 0 \quad (1.2-1)$$

If a given thermodynamic set is (P,s) or (P,h) , then basic independent parameters of T and v(or ρ) both is not existed in the equation. In this case A system of non-linear equation should be solved such as:

$$P(T,v) - P = 0 \quad (1.2-2a)$$
$$s(T,v) - s = 0 \quad (1.2.2b)$$

Or

$$P(T,v) - P = 0 \ (1.2-3a)$$
$$h(T,v) - h = 0 \ (1.2.3b)$$

Root finding is such a main part of thermodynamic equation of state calculations, as a primary information we would like to give more information about non-linear root finding methods of equations.

1.2.1 BISECTION METHOD

In the bisection method root of the functions is searched in a given region **a <= x <= b** If a single root is existed in the given region equation

$f(a)f(b) < 0$ (1.2.4)

will be satisfied. If this condition is not satisfied, it can be assumed that there is no root in the given region. If the root existed region is divide to two equal parts as p=(a+b)/2 and function is evaluated in the middle. If

f(a)*f(p) < 0 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 can happen, that will indicate the root. The new root region is used in further iterations. In order to use iterative process an exceptible criteria for error limit can be assumed

$$\left|\frac{(b-a)}{(b+a)}\right| < \varepsilon \quad (1.2.5)$$

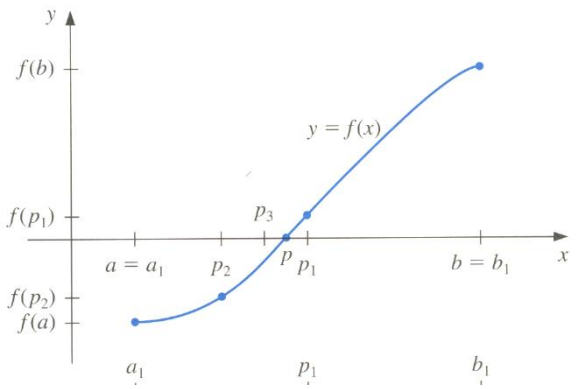


Figure 1.2.1 Convergence of bisection Method

An empty function shell called function interface is given below and as submethod of this interface all the root finding functions are embedded inside.

Program 1.2.1-1A function interface if_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);
}
//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-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();}
}
```

An example code is created for the Bisection method.

Program 1.2.1-1 Bisection method NA1

```
public class bisectiontest
{ //Bisection root finding method
  public static void main(String arg[])
  { if_x f1=x->x*x-2.0;
   double a=0;double b=3;
   double x=f1.bisection(a,b);
   System.out.println("x="+x);
   f1.plot(a,b);
  }
}
```

```
---------- Capture Output ----------
> "E:\co\java\bin\javaw.exe" bisectiontest
x=1.4142136573791504
> Terminated with exit code 0.
```

1.2.2 NEWTON-RAPHSON METHOD

Taylor Formula can be written as

$$f(x_{n+1}) = f(x_n) + f'(x_n)(x_{n+1} - x_n) + f^{"(x_n)}\frac{(x_{n+1} - x_n)^2}{2!} + f^{(3)(x_n)}\frac{(x_{n+1} - x_n)^3}{3!} + \cdots (1.2.6)$$

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

$$0 = f(x_n) + f'(x_n)(x_{n+1} - x_n) \quad (1.2.7)$$

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

$$x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)} \quad (1.2.8)$$

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

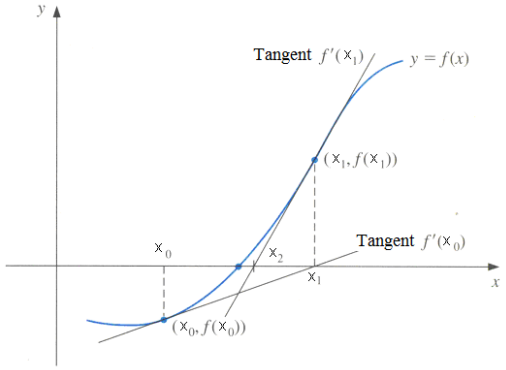


Figure 1.2.1-1 Convergence of the Newton-Raphson Method

1.2.3 SECANT METHOD

One of the difficulty of the Newton-Raphson method is the requirement to find the second 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

$$f'(x_n) \cong \frac{f(x_n)-f(x_{n-1})}{(x_n-x_{n-1})} \quad (1.2.9)$$

Newton –Raphson Formula is converted to

$$x_{n+1} = x_n - \frac{f(x_n)(x_n - x_{n-1})}{f(x_n) - f(x_{n-1})} \quad (1.2.10)$$

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)

$$f'(x_n) \cong \frac{f(x_n + \Delta x) - f(x_n - \Delta x)}{2\Delta x} \quad (1.2.11)$$

Second equation becomes:

$$x_{n+1} = x_n - \frac{2f(x_n)}{f(x_n + \Delta x) - f(x_n - \Delta x)} \quad (1.2.12)$$

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

$$f'(x_n) \cong \frac{-f(x_n + 2\Delta x) + 8f(x_n + \Delta x) - 8f(x_n - \Delta x) + f(x_n - 2\Delta x)}{12\Delta x} \quad (1.2.13)$$

$$f'(x_n) \cong \frac{f(x_n + 3\Delta x) - 9f(x_n + 2\Delta x) + 45f(x_n + \Delta x) - 45f(x_n - \Delta x) + 9f(x_n - 2\Delta x) - f(x_n - 2\Delta x)}{60\Delta x} \quad (1.2.14)$$

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

(1.2.15)

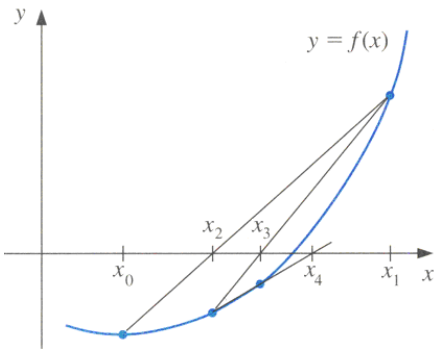


Figure 1.2.3-1 : Second method approximation with linear difference

PROBLEM 1.2.3-1 NA8 : Secant method with one initial root estimation, lineer difference Formula

```
 public class bisectiontest
{ //Bisection root finding method
  public static void main(String arg[])
  { if_x f1=x->x*x-2.0;
   double a=1;
   double x=f1.secant(a);
   System.out.println("x="+x);
   f1.plot(0.0,3.0);
  }
}
```

```
---------- Capture Output ----------

> "E:\co\java\bin\javaw.exe" bisectiontest

x=1.4142135623730951
```

1.2.4 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 achive 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 1.2.4-1 NA11 : Newton-Raphson (Secant)-Bisection combined method

```
 public class bisectiontest
{ //Bisection root finding method
  public static void main(String arg[])
  { if_x f1=x->x*x-2.0;
   double a=0;double b=3;
   double x=f1.newton_bisection(a,b);
   System.out.println("x="+x);
   f1.plot(a,b);
  }}
```

When two equations are given a system of equation solution method should be used.

1.2.5 SECANT METHOD FOR SYSTEM OF NONLINEAR EQUATIONS

One dimensional Newton-Raphson Formula was:

$$x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)} \qquad (1.216)$$

If the equation format changed a little, it can be written in the form of

$$\Delta x_n = x_{n+1} - x_n = -\frac{f(x_n)}{f'(x_n)} \qquad (1.217)$$

$$f'(x_n)\Delta x_n = -f(x_n) \qquad (1.2.18)$$

Now the same equation can be considered for a system of non-linear equation

$$[\nabla f] = \begin{bmatrix} \frac{\partial f_1}{\partial x_1} & \frac{\partial f_1}{\partial x_2} & \cdots & \frac{\partial f_1}{\partial x_n} \\ \frac{\partial f_2}{\partial x_1} & \frac{\partial f_2}{\partial x_2} & \cdots & \frac{\partial f_2}{\partial x_n} \\ \cdots & \cdots & \cdots & \cdots \\ \frac{\partial f_n}{\partial x_1} & \frac{\partial f_n}{\partial x_2} & \cdots & \frac{\partial f_n}{\partial x_n} \end{bmatrix} \quad (1.2.19) \qquad \{\Delta x\} = \begin{Bmatrix} \Delta x_1 \\ \Delta x_2 \\ \cdots \\ \Delta x_n \end{Bmatrix} \quad \{f\} = \begin{Bmatrix} f_1 \\ f_2 \\ \cdots \\ f_n \end{Bmatrix} \quad (1.2.20)$$

So multidimensional Newton-Raphson equation becomes

$$[\nabla f]\{\Delta x\} = -\{f\} \qquad (1.2.21)$$

If derivatives taken numercally (by using difference formulations defined above secant method section), it becomes secant method.

Program 1.2.5-1 Interface ifi_xj.java function interface for system of equations

```
@FunctionalInterface
interface  ifi_xj
{
  // 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
  public double[] func(double x[]);

default  double[] xp(double x[],double j,double m,double h)
{
//h = dx
//j=-4..4 multiplication factor
//m derivative taken value
int k=x.length;
double xx[]=new double[k];
for(int i=0;i<k;i++)
{               if(i==m) xx[i]=x[i]+j*h;
   else     xx[i]=x[i];
}
return xx;
}
default double[][] dfunc(double x[])
{
double h=1.0e-12;
return dfunc(x,1,h);
}
default double[][] dfunc(double x[],int n)
{
double h=1.0e-12;
return dfunc(x,n,h);
}

default double[][] dfunc(double x[],int n,double h)
{
int k=x.length; //number of x values (dimension)
double df[][]=new double[k][k];
for(int i=0;i<k;i++)
{ for(int j=0;j<k;j++)
  {df[i][j]=dfunc(x,n,i,j,h);}
}
return df;
```

```
}

default double dfunc(double x[],int n,int j,int m)
{
double h=1.0e-12;
return dfunc(x,n,j,m,h);
}

default double dfunc(double x[],int n,int j,int m,double h)
{ // derivative of x[m]
  //m derivative taken value
  //jth equation
  //nth derivative of the equation (n=0) function value
  //n=1 first derivative n=2 second derivative..
int k=x.length; //number of x values (dimension)
double hh=1/h;
double ff[][]=new double[9][3];
double xp[][]=new double[9][3];
double df;
int j1;
double x1=0;
x1=x[m];
    for(int jj=-4;jj<=4;jj++)
    {  j1=jj+4;
                xp[j1]=xp(x,jj,m,h);
       ff[j1]=func(xp[j1]);
    }
if(n==0)
{df=ff[4][j];}
else if(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;}
else if(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;}
else if(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;}
else if(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;}
else if(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;}
else if(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;}
else if(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;}
else if(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;
}
default double[] secant2(double x[])
{
// Roots of nonlinear system of equations secant method
//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=20;
```

```
double tolerance=1.0e-5;
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(func(x),dfunc(x)),-ti);
  x=Matrix.add(x,b);
}
if(i >= nmax) System.out.println("Warning : Maximum number of iterations are exceeded \nResults may not be valid");
return x;
}
}
```

Program 1.2.5-2 Secant method example for system of equations

```
public class secant2test
{ //Bisection root finding method
  public static void main(String arg[])
  {  ifi_xj f2=(double x[])->
                {  double ff[]=new double[2];
        ff[0]=x[0]*x[0]+x[1]*x[1]-8;
        ff[1]=x[0]-x[1];
        return ff;
      };
   double a=0;double b=3;
   double c[]={a,b};
   double x[]=f2.secant2(c);
   System.out.println("x="+x[0]+"y="+x[1]);
  }}
```

```
---------- Capture Output ----------
> "E:\co\java\bin\javaw.exe" secant2test
x=2.000000000192567y=2.000000000192567
> Terminated with exit code 0.
```

2.0 EQUATION OF STATES

In this section, several equation of states will be investigated and approaches to prepare mathematical models of the equation of state will be introduced, a full working model of equation of states will be given in java programming language codes. We will start from a relatively easier model of equation of state called “ideal gas equation of state”

2.1 IDEAL GAS EQUATION OF STATE

Ideal gas equation of state is given as

$$P(T,V) = \frac{NR_UT}{V} \quad (2.1.1)$$

In this equation N is mole number of the chemical components. In this equation a single component is considered and eqution will be used only for gases. R_u term in the equation is called universal gas constant and the value R_u=8.3145 kJ/(kmol K) can be taken as constant. It is a common practice to use m , mass in kg, in the equation instead of N, in that case it should be converted by using basic relation between mass and mole numbers as:

$$N = \frac{m}{M} \quad (2.1.2)$$

In this equation m is mass in kg, N is molar mass in kmol, and M is called molar mass in kg/kmol. Molar mass is a constant property for each pure gas. Therefore ideal gas equation can also be used as:

$$P(T,V) = \frac{m\left(\frac{R_U}{M}\right)T}{V} = \frac{mRT}{V} \quad (2.1.3)$$

in this form of the equation $R=R_u/M$ is simply called gas constant. In practice equation can will be used with the specific volume form as well

$$P(T,v) = \frac{R_U T}{v} (2.1.4)$$

in this form v is in m^3/kmol unit. Now that basic form of ideal gas equation of state is defined, the remaining thermodynamic properties can be formed from the basic equations defined in previous chapter

$$ds = \frac{C_v}{T}dT + \left(\frac{\partial P}{\partial T}\right)_v dv \quad (1.47)$$

$$du = C_v dT + \left(T\left(\frac{\partial P}{\partial T}\right)_v - P\right)dv \quad (1.48)$$

was given before. As an additional relations

$$C_p - C_v = R_U \quad (2.1.5)$$

can be defined. It is a common practice to use C_p in the equations in most of the equation of states, but as it is seen from eqn (2.1.5) they can be converted as $C_p = C_v + R_U$ (2.1.4) so we will write out equations as a function of C_p

$$ds = \frac{C_p - R_U}{T}dT + \left(\frac{\partial P}{\partial T}\right)_v dv \quad (1.47b)$$

$$du = (C_p - R_U)dT + \left(T\left(\frac{\partial P}{\partial T}\right)_v - P\right)dv \quad (1.48b)$$

If ideal gas equation of states are substituted into the equations:

$$\left(\frac{\partial P}{\partial T}\right)_v = \frac{R_U}{v} \quad (2.1.5)$$

$$ds = \frac{C_p - R_U}{T}dT + \left(\frac{\partial P}{\partial T}\right)_v dv = \frac{C_p - R_U}{T}dT + \frac{R_U}{v}dv \quad (2.1.6)$$

$$du = (C_p - R_U)dT + \left(T\left(\frac{\partial P}{\partial T}\right)_v - P\right)dv = (C_p - R_U)dT + \left(T\left(\frac{R_U}{v}\right) - \frac{R_U T}{v}\right)dv$$

$$du = (C_p - R_U)dT = C_v dT \quad (2.17)$$

As it is seen from equations for the ideal gas equation of state internal energy is not function of volume change and only function of temperature specific heat C_p is also only function of temperature and can easily be calculated from experimental data obtained for the gases. Entalpy from its definition

$$h = u + Pv = u + R_U T \quad (2.1.8)$$

$$dh = du + R_U dT = \left(C_p - R_U + R_U\right)dT = C_p dT \quad (2.18a)$$

can also be obtained. Integration of equations gives us

$$s = s_0 + \int_{T_0}^{T} \frac{C_p - R_U}{T} dT + R_U ln\left(\frac{v}{v_0}\right) \quad (2.1.9)$$

$$u = u_0 + \int_{T_0}^{T} \left(C_p - R_U\right)dT \quad (2.1.10)$$

$$h = h_0 + \int_{T_0}^{T} C_p dT \quad (2.1.11)$$

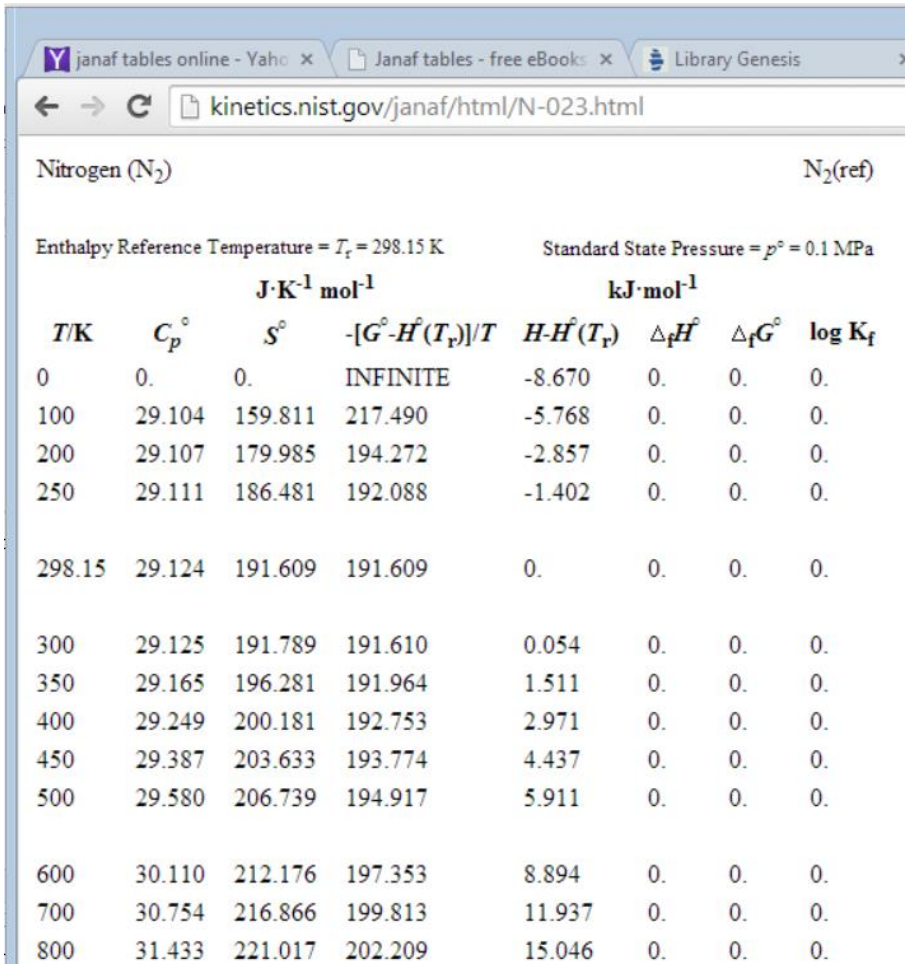
In here 0 is the reference point taken to calculate the equations. Absulute values of the thermodynamic properties are depends on the reference values. Common referance temperatures taken in the ideal gas tables are 0K, 273.15 K, 233.15 K and 293.15 K. Thermodynamic is and old science therefore a common standart for the references is not established, but the difference will remain the same. 233.15 K reference is chosen due to simplicity of converting degree Rankine to degree Kelvin. 293.15 K is usually used in chemical thermodynamics as a close approximation to room temperature.
Now let us give our attention to calculation of specific heat , C_p, value. It is a key component in not only ideal gas equation of state but also most of the other equation of states that we will be defining. Data on specific heat of gases can be found in two excellent boks, JANAF Thermochemical Tables[1], and Ihsan Barin Thermochemical Data of Pure Substances[2]. Nowadays JANAF tables can be accessed through internet as well. JANAF table list through internet adress https://janaf.nist.gov/. It is shown In figure 2.1.1.
Another good data source for Cp values is **http://www.chem.msu.su/rus/handbook/ivtan/**
Another source for Cp data is “Thermochemical data of Elements and Compounds”[83],

Knowing the table values Cp(T) data can be created by using curve fitting methods. Some of the common equations used can be summerized as follows:
Polynomial curve fitting equations:

$$C_p(T) = \sum_{i=0}^{n} a_i T^i \quad (2.1.12)$$

This type of coefficient of equations are listed for approximately 600 species in “The properties of Gases and Liquids”[4]. It should be note that taking C_p value constant (only first term of the polynomial) are also used commonly in thermodynamics. Polynomial coefficients and table values of C_p of most common gases are usually given in most of the thermodynamic text books. We can work an example. If a polynomial least square curve fitting equation for nitrogen data is required to be created

Figure 2.1.1 JANAF Tables at internet adress kinetics.nist.gov/janaf



janaf tables online - Yaho | Janaf tables - free eBooks | Library Genesis

kinetics.nist.gov/janaf/html/N-023.html

Nitrogen (N_2) N_2(ref)

Enthalpy Reference Temperature = T_r = 298.15 K Standard State Pressure = p° = 0.1 MPa

		J·K^{-1} mol^{-1}		kJ·mol^{-1}			
T/K	C_p°	S°	$-[G^\circ-H^\circ(T_r)]/T$	$H-H^\circ(T_r)$	$\Delta_f H^\circ$	$\Delta_f G^\circ$	log K_f
0	0.	0.	INFINITE	-8.670	0.	0.	0.
100	29.104	159.811	217.490	-5.768	0.	0.	0.
200	29.107	179.985	194.272	-2.857	0.	0.	0.
250	29.111	186.481	192.088	-1.402	0.	0.	0.
298.15	29.124	191.609	191.609	0.	0.	0.	0.
300	29.125	191.789	191.610	0.054	0.	0.	0.
350	29.165	196.281	191.964	1.511	0.	0.	0.
400	29.249	200.181	192.753	2.971	0.	0.	0.
450	29.387	203.633	193.774	4.437	0.	0.	0.
500	29.580	206.739	194.917	5.911	0.	0.	0.
600	30.110	212.176	197.353	8.894	0.	0.	0.
700	30.754	216.866	199.813	11.937	0.	0.	0.
800	31.433	221.017	202.209	15.046	0.	0.	0.

Input data 2.1 Least square curve fittinginput data , Nitrogen Cp(T), source : NIST Janaf Tables

T	Cp	T	Cp	T	Cp	T	Cp	T	Cp
100	29.104	1100	33.241	2500	36.616	3900	37.508	5300	38.013
200	29.107	1200	33.723	2600	36.713	4000	37.55	5400	38.046
250	29.111	1300	34.147	2700	36.801	4100	37.59	5500	38.08
298.15	29.124	1400	34.518	2800	36.883	4200	37.629	5600	38.116
300	29.125	1500	34.843	2900	36.959	4300	37.666	5700	38.154
350	29.165	1600	35.128	3000	37.03	4400	37.702	5800	38.193
400	29.249	1700	35.378	3100	37.096	4500	37.738	5900	38.234
450	29.387	1800	35.6	3200	37.158	4600	37.773	6000	38.276
500	29.58	1900	35.796	3300	37.216	4700	37.808		
600	30.11	2000	35.971	3400	37.271	4800	37.843		
700	30.754	2100	36.126	3500	37.323	4900	37.878		
800	31.433	2200	36.268	3600	37.373	5000	37.912		
900	32.09	2300	36.395	3700	37.42	5100	37.947		
1000	32.697	2400	36.511	3800	37.465	5200	37.981		

Now by using a least suare curve fitting program we can create a curve fitting equation.

Program 2.1.1 Least square curve fitting program

```
import java.io.*;
import java.util.*;
import javax.swing.*;
import java.awt.Color;

class SCO11B4
{
```

```
//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[] EKK(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;
Text.printT(X);
 return 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 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]-funcEKK(e,x[k]))*(y[k]-funcEKK(e,x[k]));
 }
total=Math.sqrt(total);
return total;
}

public static double[][]  funcEKK(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
```

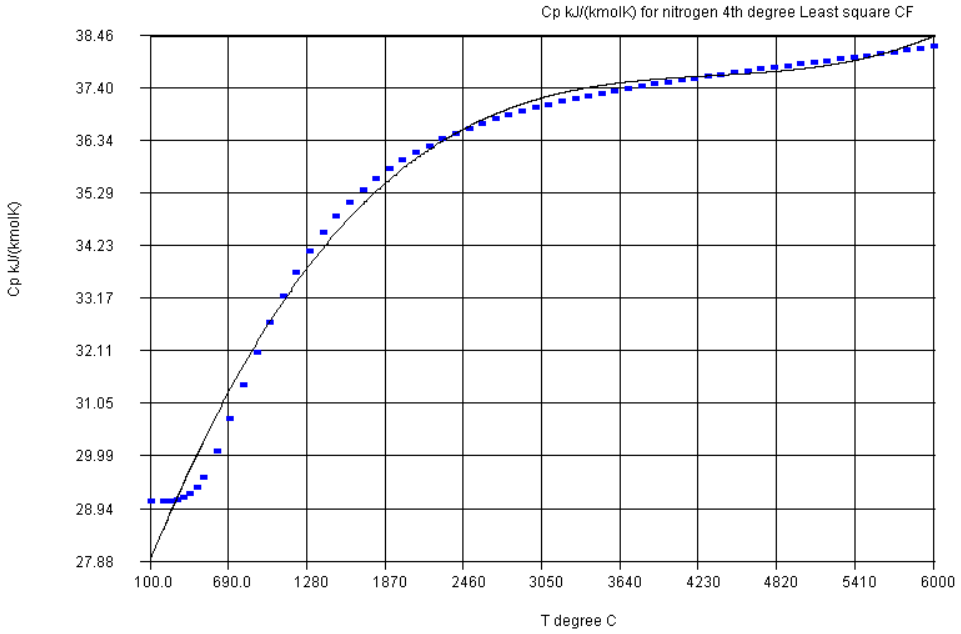
```
            int n=xi.length;
            int nn=(n-1)*(aradegersayisi+1)+1;
            double z[][]=new double[2][nn];
   double E[]=EKK(xi,yi,polinomkatsayisi);
   double hata=hata(xi,yi,E);
   //System.out.println("hata="+hata+"\nkatsayılar :\n"+Matrix.toStringT(E));
   double dx=0;
   int k=0;
   int i;
   for(i=0;i<(n-1);i++)
     {z[0][k]=xi[i];z[1][k]=funcEKK(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]=funcEKK(E,z[0][k]);k++;}
     }
     z[0][k]=xi[i];z[1][k]=funcEKK(E,z[0][k]);
   return z;
 }
 public static double yavg(double y[])
 {
 int n=y.length;
 double total=0;
 for(int i=0;i<n;i++) total+=y[i];
 return total/n;
 }
 public static double R2(double y[],double f[])
 {
 double yavg=yavg(y);
 int n=y.length;
 double SStot=0;
 double SSerr=0;
 for(int i=0;i<n;i++)
 {SStot=(y[i]-yavg)*(y[i]-yavg);
  SSerr=(y[i]-f[i])*(y[i]-f[i]);
 }
 return (1-SSerr/SStot);
 }

 public static double R2(double x[],double y[],double e[])
 { int n=y.length;
 double f[]=new double[n];
   for(int k=0;k<n;k++)
    {f[k]=funcEKK(e,x[k]);}
   return R2(y,f);
 }

  public static void main(String args[]) throws IOException
  {
  double x[];
  double y[];
  String s1=JOptionPane.showInputDialog("dosya adı (R134a_Cp.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[][]=funcEKK(x,y,n,10);
  Text.printT(z);
  System.out.println("E.K.K.\n"+Matrix.toStringT(z));
  Plot pp=new Plot(a[0],a[1]);
  pp.setPlabel("Cp kJ/(kmolK) for nitrogen 4th degree Least square CF");
  pp.setXlabel("T degree C");
  pp.setYlabel("Cp kJ/(kmolK)");
  pp.setPlotType(0,23);
  pp.addData(z[0],z[1]);
  pp.setGrid(1,1);
  pp.setColor(0,Color.BLUE);
  pp.plot();
  }
}
```

The program output:



$C_p(T)$= 27.19872924529033+0.006943211110342824*T-1.568881107785314E-6*T^2+1.211840254124337E-10*T^3-7.162042075128584E-17*T^4

Fitted data into the curve:

100	27.8775	750	31.5747	1800	35.3193	2850	37.0442	3900	37.5865	4950	37.7812
150	28.2053	800	31.8112	1850	35.4406	2900	37.0902	3950	37.5971	5000	37.796
200	28.5256	850	32.0413	1900	35.5574	2950	37.1337	4000	37.6069	5050	37.812
225	28.6829	900	32.2651	1950	35.6698	3000	37.1746	4050	37.6162	5100	37.8292
250	28.8384	950	32.4827	2000	35.778	3050	37.2131	4100	37.6249	5150	37.8479
274.1	28.9863	1000	32.6942	2050	35.8818	3100	37.2493	4150	37.6332	5200	37.868
298.2	29.1326	1050	32.8996	2100	35.9816	3150	37.2833	4200	37.6411	5250	37.8896
299.1	29.1382	1100	33.0991	2150	36.0773	3200	37.3151	4250	37.6489	5300	37.9129
300	29.1438	1150	33.2928	2200	36.1691	3250	37.3449	4300	37.6564	5350	37.9379
325	29.2937	1200	33.4807	2250	36.257	3300	37.3727	4350	37.6639	5400	37.9647
350	29.4419	1250	33.6629	2300	36.3412	3350	37.3987	4400	37.6714	5450	37.9935
375	29.5882	1300	33.8395	2350	36.4217	3400	37.4228	4450	37.679	5500	38.0242
400	29.7327	1350	34.0107	2400	36.4986	3450	37.4453	4500	37.6869	5550	38.057
425	29.8755	1400	34.1765	2450	36.572	3500	37.4662	4550	37.695	5600	38.092
450	30.0165	1450	34.3369	2500	36.642	3550	37.4856	4600	37.7035	5650	38.1293
475	30.1558	1500	34.4922	2550	36.7086	3600	37.5035	4650	37.7124	5700	38.1689
500	30.2933	1550	34.6423	2600	36.7721	3650	37.5202	4700	37.722	5750	38.211
550	30.5631	1600	34.7874	2650	36.8324	3700	37.5355	4750	37.7322	5800	38.2556
600	30.826	1650	34.9276	2700	36.8897	3750	37.5498	4800	37.7431	5850	38.3028
650	31.0822	1700	35.0629	2750	36.944	3800	37.563	4850	37.7548	5900	38.3528
700	31.3318	1750	35.1934	2800	36.9955	3850	37.5752	4900	37.7675	5950	38.4056
										6000	38.4612

As it is seen from the least square cure fitting results, using a single equation is always cause errors, and error grows even further close to the critical point of gases. In this context, a more accurate partially continious curve fitting method will be utilized in our modelling of ideal gas C_p

$Cp_i(T) = A_i + B_i*10^{-3}*T + C_i*10^5/T^2 + D_i*10^{-6}*T^2$ $T_{Li} >= T > T_{Hi}$ KJ/kmol K (2.1.13)

By using this type of partially continuous equation gives us oppurtunity to adjust error range of the equations. In order to find coefficients of equation Least square curve fitting method will be applied. Let us define the method of Least square It is assumed that T_i, C_{pi} i=0...(n-1) data is divided into L data region and data for each set becomes T_i,C_{pi} i=(n-1)/L*k...(n-1)/L*(k+1) k=0…(L-1). If a curve fitting function

$$C_{pk}(T) = \sum_{j=0}^{m} a_{jk}^{(m)} \phi_j(T), \qquad T_{Li} \le T \le T_{Hi} \quad k = 0 \dots (L-1) \quad (2.1.14)$$

general linear function is defined with multiplication functions $\phi_j(T)$ for each linear coefficient of equation. For the equation 2.1.13 this coefficients will be: $\phi_0=1$, $\phi_1=10^{-3}*T$, $\phi_2=10^5/T^2$, $\phi_3=10^{-6}*T^2$

In order to apply least square curve fitting minimisation of the following function will be carried out.

$$H(a_{0k}^{(m)}, .., a_{mk}^{(m)}) = \sum_{i=\frac{n-1}{L}*k}^{\frac{n-1}{L}*(k+1)} w_k(T_i) \left[\frac{C_{pi} - \sum_{j=0}^{m} a_{jk}^{(m)} \phi_j(T_i)}{(L-1)} \right]^2 \qquad (2.1-15)$$

$w_k(T)$ is called the weight function. The value of the weight function can be assumed to be equal to unity to simplify the process. The minimum of the function is the roots of its derivatives with respect to coefficients of the equation..

$$\frac{\partial H_k(a_{0k}^{(m)}, .., a_{mk}^{(m)})}{\partial a_p^{(m)}} = \frac{2}{(L-1)} \sum_{i=1}^{n} w(T_i) \left[\frac{C_{pi} - \sum_{j=0}^{m} a_j^{(m)} \phi_j(T_i)}{(L-1)} \right] \phi_p(T_i) = 0 \quad p = 0, \dots, m \qquad (2.1.16)$$

This equation is m+1 linear system of equation and can be solved by using any system of equation solving methods, such as Gauss elimination process. For weight function $w(x_i)$=1, general least square equation becomes:

$$\begin{bmatrix} \sum_{i=1}^{n} w(T_i)\phi_0(T_i)\phi_0(T_i) & \sum_{i=1}^{n} w(T_i)\phi_0(T_i)\phi_1(T_i) & \sum_{i=1}^{n} w(T_i)\phi_0(T_i)\phi_2(T_i) & \cdots & \sum_{i=1}^{n} w(T_i)\phi_0(T_i)\phi_n(T_i) \\ \sum_{i=1}^{n} w(T_i)\phi_1(T_i)\phi_0(T_i) & \sum_{i=1}^{n} w(T_i)\phi_1(T_i)\phi_1(T_i) & \sum_{i=1}^{n} w(T_i)\phi_1(T_i)\phi_2(T_i) & \cdots & \sum_{i=1}^{n} w(T_i)\phi_2(T_i)\phi_n(T_i) \\ \sum_{i=1}^{n} w(T_i)\phi_2(T_i)\phi_0(T_i) & \sum_{i=1}^{n} w(T_i)\phi_2(T_i)\phi_1(T_i) & \sum_{i=1}^{n} w(T_i)\phi_2(T_i)\phi_2(T_i) & \cdots & \sum_{i=1}^{n} w(T_i)\phi_2(T_i)\phi_n(T_i) \\ \cdots & \cdots & \cdots & \cdots & \cdots \\ \sum_{i=1}^{n} w(T_i)\phi_n(T_i)\phi_0(T_i) & \sum_{i=1}^{n} w(T_i)\phi_n(T_i)\phi_1(T_i) & \sum_{i=1}^{n} w(T_i)\phi_n(T_i)\phi_2(T_i) & \cdots & \sum_{i=1}^{n} w(T_i)\phi_n(T_i)\phi_n(T_i) \end{bmatrix} \begin{Bmatrix} a_0 \\ a_1 \\ a_2 \\ \cdots \\ a_n \end{Bmatrix} = \begin{Bmatrix} \sum_{i=1}^{n} w(T_i)\phi_0(T_i)C_p(T_i) \\ \sum_{i=1}^{n} w(T_i)\phi_1(T_i)C_p(T_i) \\ \sum_{i=1}^{n} w(T_i)\phi_2(T_i)C_p(T_i) \\ \cdots \\ \sum_{i=1}^{n} w(T_i)\phi_n(T_i)C_p(T_i) \end{Bmatrix} \quad (2.1.17)$$

A curve fitting program is developed in Java language to carry out this procedure. The list of the program is given below.

Program 2.1.2 Partially continuous curve fitting program

```
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 error(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[][]  errorEKKPlot(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);}

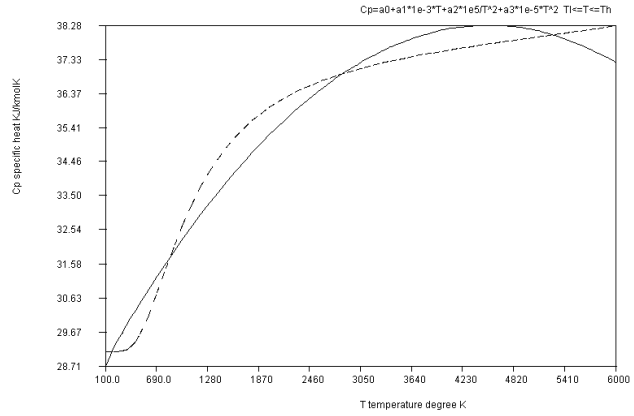
  public double k_l(double t)
```

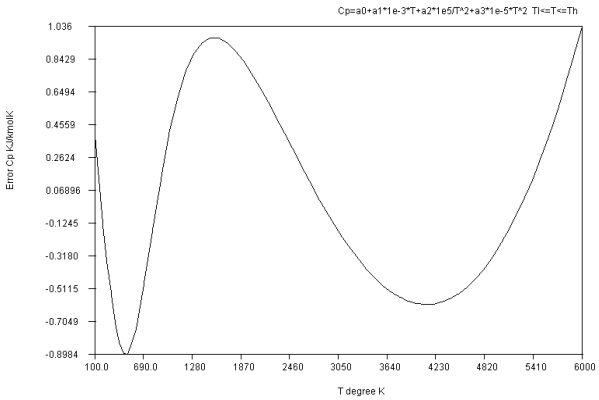
```
 {return funcEKKgeneral(f2,kl,t);}

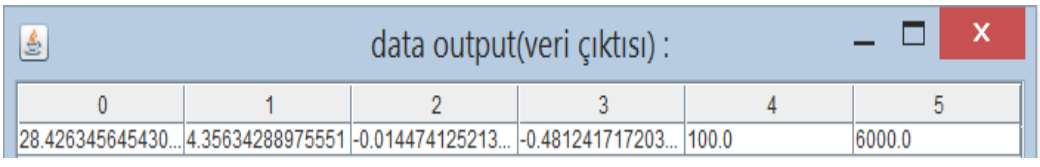

 public static void main(String args[]) throws IOException
 {
 String s1=JOptionPane.showInputDialog("name of the input file : ");
 double c[][]=Text.readDoubleT(s1);
 int n=c[0].length;
 Text.print(Text.T(c));
 double d[][]=EKKgeneral(new fcp(), c[0],c[1],n);
 funcEKKPlot(new fcp(), c[0],c[1],n);
 Text.print(d);
 errorEKKPlot(new fcp(), c[0],c[1],n);
 System.out.println(Matrix.toString(d));
 }

}
```

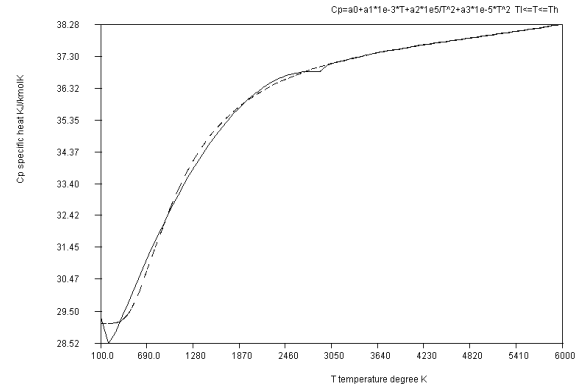
Output 2.1-1 Least square curve fitting, Nitrogen Cp(T) one equation for the full range (n=1)

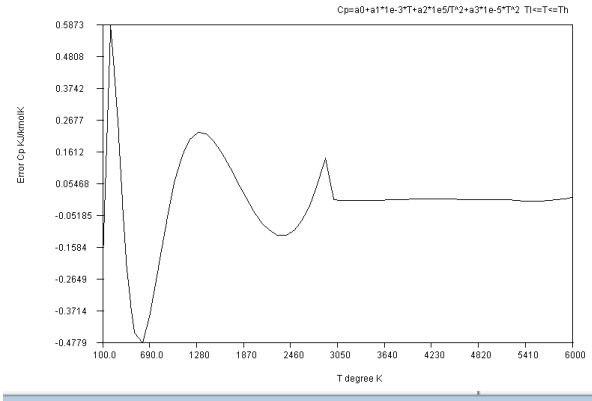


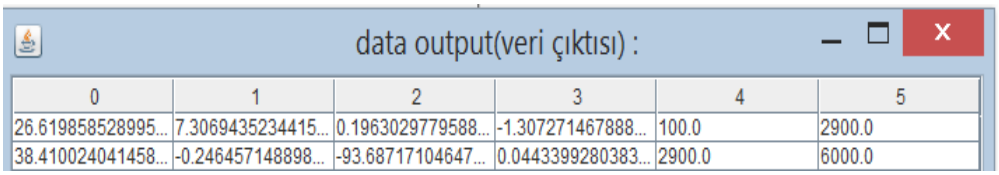




Output 2. 1-2 Least square curve fitting, Nitrogen Cp(T) one equation for the full range (n=2)

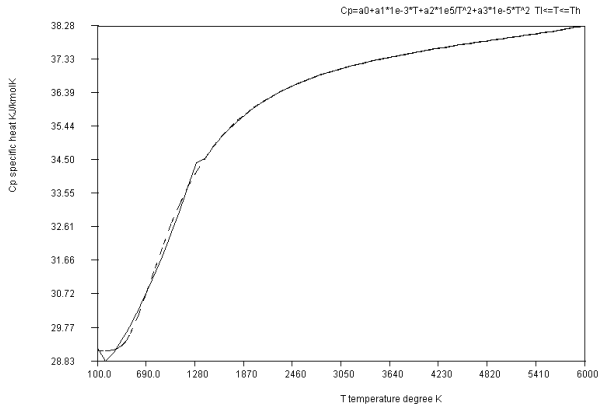


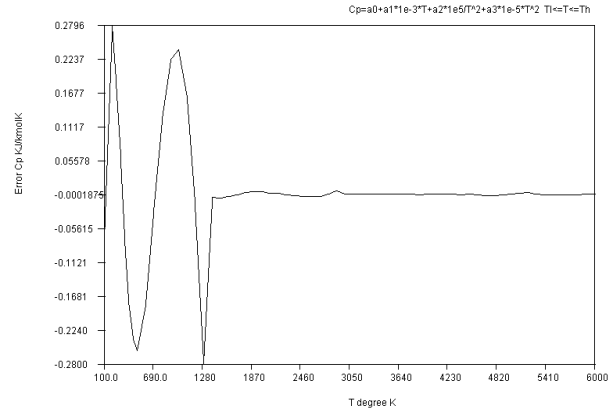




0	1	2	3	4	5
26.619858528995...	7.3069435234415...	0.1963029779588...	-1.307271467888...	100.0	2900.0
38.410024041458...	-0.246457148898...	-93.68717104647...	0.0443399280383...	2900.0	6000.0

Output 2. 1-2 Least square curve fitting, Nitrogen Cp(T) one equation for the full range (n=4)

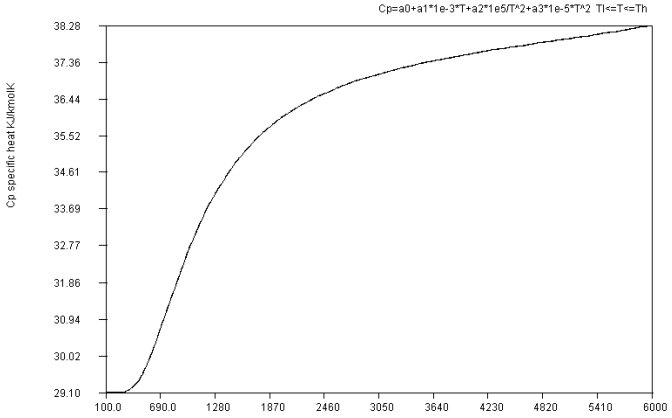


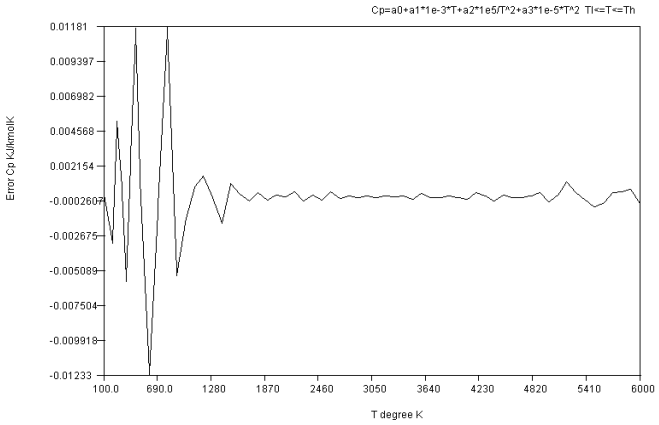


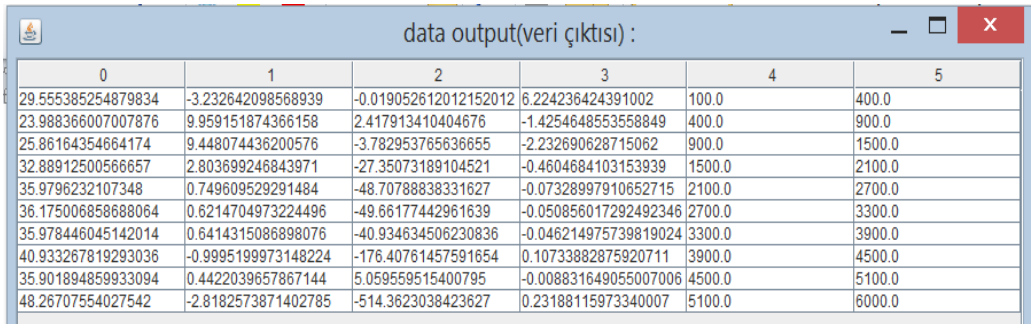
data output(veri çıktısı) :

0	1	2	3	4	5
27.95585931143741	2.890045926186791	0.09175734463285376	1.6027541706504944	100.0	1300.0
34.129733722662024	1.815381004555621	-32.73483985828752	-0.243935897854228...	1300.0	2900.0
37.10276237021983	0.2150440463728742	-63.93913586682272	-8.392331032663524...	2900.0	4500.0
45.43488884308771	-2.0861200239998876	-392.3837574823669	0.17908400732262333	4500.0	6000.0

Output 2. 1-2 Least square curve fitting, Nitrogen Cp(T) one equation for the full range (n=10)







0	1	2	3	4	5
29.555385254879834	-3.232642098568939	-0.019052612012152012	6.224236424391002	100.0	400.0
23.988366007007876	9.959151874366158	2.417913410404676	-1.4254648553558849	400.0	900.0
25.86164354664174	9.448074436200576	-3.782953765636655	-2.232690628715062	900.0	1500.0
32.88912500566657	2.803699246843971	-27.35073189104521	-0.4604684103153939	1500.0	2100.0
35.9796232107348	0.749609529291484	-48.70788838331627	-0.07328997910652715	2100.0	2700.0
36.175006858688064	0.6214704973224496	-49.66177442961639	-0.050856017292492346	2700.0	3300.0
35.978446045142014	0.6414315086898076	-40.934634506230836	-0.046214975739819024	3300.0	3900.0
40.933267819293036	-0.9995199973148224	-176.40761457591654	0.10733882875920711	3900.0	4500.0
35.901894859933094	0.4422039657867144	5.059559515400795	-0.008831649055007006	4500.0	5100.0
48.26707554027542	-2.8182573871402785	-514.3623038423627	0.23188115973340007	5100.0	6000.0

Program outputs clearly indicates how the error level of the equations can be improved with further division of the curve fitting data ranges. In table 2.1 coefficients calculates for most common gases are given.

Table 2. 1 Cp specific energy at constant pressure coefficients for partially continuous equation 2.23 for some selected gases (KJ/kmol K)

O2(OXYGEN)

A_i	B_i	C_i	D_i	T_{Li}	T_{Hi}
22.2733042	20.09559	1.5767767	-7.443438	298	600
26.2330328	13.48713	-2.0281281	-4.647975	600	1000
35.7634821	0.5428001	-17.710825	0.3348063	1000	1400
34.5568094	1.5534385	-12.681988	0.097656	1400	1800
32.682339	2.7805744	-0.2328397	-0.124124	1800	2200
34.1893024	2.0032457	-16.591274	-0.012322	2200	2600
23.1341719	7.7683414	105.06076	-0.860499	2600	3000
35.3620341	2.1744887	-71.654038	-0.136367	3000	3400
37.7138695	1.3732901	-128.8807	-0.061344	3400	3800
49.4863253	-2.649491	-428.22144	0.3255739	3800	4200
57.8026664	-5.225491	-684.17715	0.5497138	4200	4600
67.3497053	-7.837178	-1057.8244	0.7497392	4600	5000

O(OXYGEN)

A_i	B_i	C_i	D_i	T_{Li}	T_{Hi}
21.671447	-2.000487	0.6392959	1.3201975	298.2	600
20.9035553	-0.139446	1.0331238	0.0475465	600	1000
20.4553758	0.3849957	1.9839752	-0.123782	1000	1400
21.6687811	-0.766509	-2.1607524	0.1875427	1400	1800
17.6656043	1.9100293	24.481016	-0.317612	1800	2200
24.9178388	-2.537163	-32.696431	0.4494757	2200	2600
21.487001	-0.625	-1.33E-05	0.1500001	2600	3000
17.0117402	1.3203316	70.107043	-0.087746	3000	3400
8.95294307	4.2042415	252.79664	-0.375537	3400	3800
20.0977517	0.1079969	7.5767905	0.0482227	3800	4200
-1.3420264	6.7095788	678.62469	-0.523824	4200	4600
724.70512	-195.5212	-26984.851	15.305527	4600	5000

CH4 (METHANE)

A_i	B_i	C_i	D_i	T_{Li}	T_{Hi}
0.26362115	103.39063	6.665204	-33.10402	298	600
6.78991069	89.603901	2.2331944	-24.83552	600	1000
27.7451351	61.806127	-33.708994	-14.39854	1000	1400
80.8115508	12.886412	-220.78185	-1.66196	1400	2000

CO2 (CARBONDIOXIDE)

A_i	B_i	C_i	D_i	T_{Li}	T_{Hi}
25.4671973	51.023411	-1.3120972	-23.31607	298.2	600
34.6006273	30.277362	-6.7786053	-9.892535	600	1000

48.1290242	11.835579	-28.908503	-2.765834	1000	1400
54.2079272	5.7117773	-46.594799	-1.032728	1400	1800
59.0590366	2.0250122	-71.9214	-0.240514	1800	2200
62.0435167	0.2785119	-98.089295	0.0484271	2200	2600
65.5225099	-1.394249	-142.79412	0.2749848	2600	3000

C2H6 (ETHANE)

A_i	B_i	C_i	D_i	T_{Li}	T_{Hi}
7.22082458	169.49999	-0.310051	-54.76307	298.1	600
15.8323038	151.00556	-6.0042494	-43.46719	600	1000

C4H10(BUTHANE)

A_i	B_i	C_i	D_i	T_{Li}	T_{Hi}
8.84111125	333.98478	0.1153296	-115.5945	298.1	600
6.01218614	339.97468	2.0345276	-119.2002	600	1000
-3.8865264	352.83922	19.725427	-123.9349	1000	1500

C6H6(BENZENE)

A_i	B_i	C_i	D_i	T_{Li}	T_{Hi}
-18.481746	402.83801	-2.687926	-180.804	298.1	600
53.4042651	249.95242	-51.078552	-88.3468	600	1000
302.997008	-74.83644	-496.35487	31.372196	1000	1500

C3H8(PROPANE)

A_i	B_i	C_i	D_i	T_{Li}	T_{Hi}
2.7092291	274.00712	-1.2065294	-104.1566	298	600
35.0742571	205.13062	-22.964953	-62.47967	600	1000
147.689338	58.572903	-223.8476	-8.450814	1000	1500

C5H12(PENTANE)

A_i	B_i	C_i	D_i	T_{Li}	T_{Hi}
7.82987398	434.08356	-1.9963999	-168.1041	298	600
61.2821127	320.36074	-37.953456	-99.30518	600	1000

NO(NITROUX OXIDE)

A_i	B_i	C_i	D_i	T_{Li}	T_{Hi}
26.0647325	6.3213237	1.472044	2.6988662	298	600
22.8517346	16.355181	1.876206	-5.412385	600	1000
30.8676094	5.6277872	-11.7589	-1.33654	1000	1400
35.035619	1.6318768	-25.363977	-0.254688	1400	1800
36.4013997	0.6632486	-33.453572	-0.061003	1800	2200
34.6513029	1.6332455	-16.591279	-0.212322	2200	2600
34.0190004	1.7499998	-5.73E-06	-0.2	2600	3000

NO2(NITROUSOXIDE)

A_i	B_i	C_i	D_i	T_{Li}	T_{Hi}
35.7317582	22.766346	-4.715792	-6.218409	298	600
35.771235	22.757046	-4.7810842	-6.262163	600	1000
35.7679926	22.813891	-4.8874622	-6.305161	1000	1400
177.159804	-96.36465	-581.69215	21.69875	1400	1800
53.8006783	1.2561487	-0.4656812	0.0017521	1800	2200
55.3567682	0.3959388	-15.619017	0.1359161	2200	2600
53.8428231	1.2484724	-1.8087021	0.0017687	2600	3000

He(HELIUM)

A_i	B_i	C_i	D_i	T_{Li}	T_{Hi}
20.786	0	0	0	298	3000

CO(KARBONMONOXIDE)

A_i	B_i	C_i	D_i	T_{Li}	T_{Hi}
28.6560206	-2.085128	0.337893	8.1811747	298.2	600

20.3829755	18.160934	4.52649	-5.81348	600	1000
26.7303654	9.2194438	-4.8584718	-2.281008	1000	1400
33.941026	2.1828711	-27.618905	-0.341279	1400	1800
34.8280427	1.5459515	-32.522207	-0.214507	1800	2200
33.3217675	2.2909392	-15.619011	-0.314084	2200	2600
32.8390896	2.2832633	0.9043359	-0.275884	2600	3000

H2O(STEAM)

A_i	B_i	C_i	D_i	T_{Li}	T_{Hi}
28.6877265	12.039738	1.1463393	0.25555	298.2	600
27.3108165	14.029766	2.5637497	-0.329693	600	1000
19.4614944	25.197732	14.032779	-4.795708	1000	1400
26.780184	18.605935	-12.587814	-3.128211	1400	1800
37.0771036	10.778625	-65.742941	-1.451386	1800	2200
44.5695869	6.3876967	-130.29958	-0.727978	2200	2600
46.3095925	5.3240178	-141.88969	-0.5509	2600	3000
51.008483	3.1345329	-205.6806	-0.26442	3000	3400
41.4946818	6.6320807	2.4818852	-0.625887	3400	3800
59.8614036	0.0025212	-416.85494	0.0479064	3800	4200
68.5382723	-2.651393	-689.72852	0.2756027	4200	4600
50.0986401	2.6143779	-15.736572	-0.148221	4600	5000

H(HIDROGEN)

A_i	B_i	C_i	D_i	T_{Li}	T_{Hi}
20.786	0.00E+00	0.00E+00	0.00E+00	298.2	5000

NH3(AMMONIA)

A_i	B_i	C_i	D_i	T_{Li}	T_{Hi}
20.3177799	48.018943	1.8656171	-12.18994	298.2	600
22.1735304	44.053981	0.6126955	-9.769835	600	1000
26.5786168	38.208896	-6.8714374	-7.581379	1000	1400
36.5302097	28.798806	-40.206718	-5.069489	1400	1800
50.6669644	18.375459	-117.49906	-2.90565	1800	2200
72.7680843	4.9328427	-295.24014	-0.602956	2200	2600
108.872152	-13.48839	-707.63993	2.0437324	2600	3000

Ar(ARGON)

A_i	B_i	C_i	D_i	T_{Li}	T_{Hi}
20.786	0	0	0	298.2	5000

Another way for the interpolation process is to fit a different polynomial in between each two point. If a third degree polinomial is considered:

$$r_k(x) = a_k(x - x_k)^3 + b_k(x - x_k)^2 + c(x - x_k)^3 + y_k \quad 1 \le k \le n \quad (2.1.18)$$

In the interpolation proses polinoms should be passing through all data points

$$r_k(x_{k+1}) = y_{k+1} \quad 1 \le k \le n \quad (2.1.19)$$

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

$$r'_{k-1}(x_k) = r'_k(x_k) \quad 1 \le k \le n \quad (2.1.20)$$

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

$$r''_{k-1}(x_k) = r''_k(x_k) \quad 1 \le k \le n \quad (2.1.21)$$

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

$$r''_1(x_1) = A \qquad r''_{n-1}(x_n) = B \quad (2.1.22)$$

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 $h_k = x_{k+1} - x_k \quad 1 \le k \le n \quad (2.1.23)$
System of equations become:

$$a_k h_k^3 + b_k h_k^2 + b_k h_k = y_{k+1} - y_k \quad 1 \le k \le n \quad (2.1.24)$$
$$3a_{k-1} h_{k-1}^2 + 2b_{k-1} h_{k-1} + c_{k-1} - c_k = 0$$
$$6a_{k-1} h_{k-1} + 2b_{k-1} + 2b_k = 0$$
$$3b_0 = 0$$
$$6a_{n-1} h_{n-1} + 2b_{n-1} = 0$$

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:

$$s_k(x) = a_k(x - x_k) + b_k(x_{k+1} - x) + [(x - x_k)^3 c_{k+1} + (x_{k+1} - x)^3 c_k] / (6h_k) \quad 1 \le k \le n \quad (2.1.25)$$

then derivative equations becomes

$$s'_k(x) = a_k - b_k + [(x - x_k)^2 c_{k+1} - (x_{k+1} - x)^2 c_k] / h_k \quad 1 \le k \le n \quad (2.1.26)$$
$$s''_k(x) = [(x - x_k) c_{k+1} - (x_{k+1} - x) c_k] / h_k \quad 1 \le k \le n$$

a_k ve b_k coefficients can be expressed as a function of c_k

$$b_k = \frac{[6y_k - h_k c_k]}{6h_k} \quad 1 \le k \le n \quad (2.1.27)$$

$$a_k = \frac{[6y_{k+1} - h_k^2 c_{k+1}]}{6h_k} \quad 1 \le k \le n \quad (2.1.28)$$

In this case only c_k terms left in the system of equations to be solved.

$$h_{k-1} c_{k-1} + 2(h_{k-1} - h_k) c_{k-1} + h_k c_{k+1} c_{k+1} = 6\left[\frac{y_{k+1} - y_k}{h_k} - \frac{y_k - y_{k-1}}{h_{k-1}}\right] \quad 1 \le k \le n \quad (2.1.29)$$

This system of equation has only n-2 terms to be solved. By making definition

$w_k = \frac{y_{k+1} - y_k}{h_k}, \quad 1 \le k \le n \quad (2.1.31)$

System of equation becomes

$$\begin{bmatrix} 1 & & & & & & \\ h_1 & 2(h_1 + h_2) & h_2 & & & & \\ & h_2 & 2(h_2 + h_3) & \cdots & & & \\ & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots \\ & & & \cdots & 2(h_{n-3} + h_{n-2}) & h_{n-2} & \\ & & & & h_{n-2} & 2(h_{n-2} + h_{n-1}) & h_{n-1} \\ & & & & & 1 & \end{bmatrix} \begin{Bmatrix} c_0 \\ c_1 \\ c_2 \\ \cdots \\ c_{n-2} \\ c_{n-1} \\ c_n \end{Bmatrix} = \begin{Bmatrix} A \\ 6(w_2 - w_1) \\ 6(w_3 - w_2) \\ \cdots \\ 6(w_{n-2} - w_{n-3}) \\ 6(w_{n-1} - w_{n-2}) \\ B \end{Bmatrix} \quad (2.1.30)$$

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). Now by using cubic spline, the same Nitrogen Cp data will be used for the curve fitting.

Program 2.1.3 Cubic spline curve fitting program

```
//Cubic spline curve fitting
import java.io.*;
import javax.swing.*;

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;
 }
  // Türev formülleri
// =================
  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)
  {
    //kübik şerit türev formülü
  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;
 }

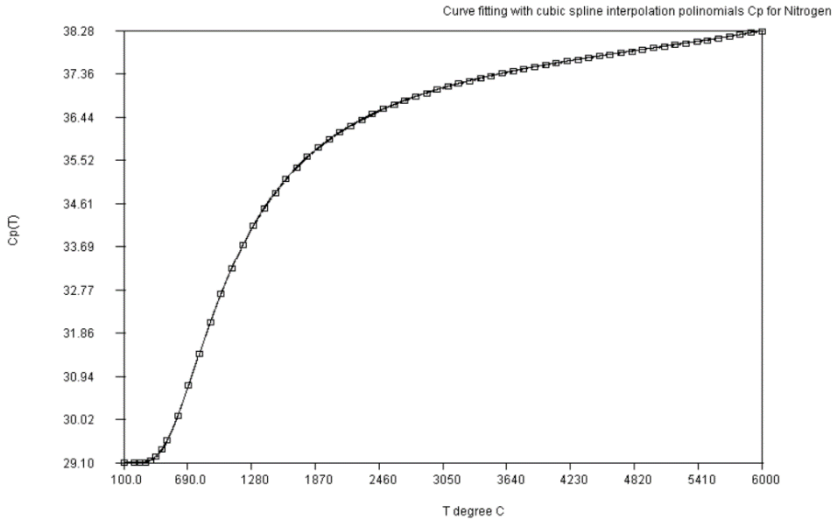
 public static double  dfSpline(double xi[],double yi[],double x)
 {
   //doğal kübik şerit türev formülü
              double S[][]=cubic_spline(xi,yi,0,0);
   return dfSpline(S,x);
 }

 public static double intSpline(double S[][],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 static double golden(double S[][],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 = -funcSpline(S,a);
    double yb = -funcSpline(S,b);
    double c = a + r2*h;
    double d = a + r1*h;
    double yc = -funcSpline(S,c);
    double yd = -funcSpline(S,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 = -funcSpline(S,c);
        }
      else
       {
         a = c;
         ya = yc;
         c = d;
         yc = yd;
         h = b - a;
         d = a + r1 * h;
         yd = -funcSpline(S,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[]) throws IOException
  {
  String s1=JOptionPane.showInputDialog("file name : ");
  double c[][]=Text.readDoubleT(s1);
  double z3[][]=funcSpline(c[0],c[1],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(c[0],c[1]);
  pp.setPlabel("Curve fitting with cubic spline interpolation polinomials");
  pp.setPlotType(0,20);
  pp.addData(z3[0],z3[1]);
  pp.plot();
  }
}
```



As it is seen from the graphic, cubic spline will give us almost perfect fitting. The only problem with this kind of fitting is that there are too many eqautions and coeficients for the curve fitting. But since computer programs can keep the coefficients in memory, it will not cause any problem. This format is not very suitable for hand calculations.

We would like to investigate also how good single equations given by the reseachers would behave with respect to actual behaviors. Aly-Lee[41] equation, which is based on statistical thermodynamics has the form:

$$C_{p(T)} = a_0 + a_1\left(\frac{a_2/T}{\sinh\,(a_2/T)}\right) + a_3\left(\frac{a_4/T}{\sinh\,(a_4/T)}\right) \quad (2.1.31)$$

This equation is non-linear, so in order to curve-fit data into this equation non-linear equation fitting method is required. The following program utilises Nelder-Mead Curve fitting Method.

Program 2.1.4 Non linear Nelder-Mead optimisation curve fitting program for Aly-Lee equation

```
 import java.io.*;
import javax.swing.*;

    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,double ia[])
    {
     //read the data to curvefit
     //get the data file and initial fit coefficient when class is defined
              double[] Xi=new double[500];
     double[] 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 bulunamadı"); }
    nn=i;
    xi=new double[nn];
    yi=new double[nn];
    for(int j=0;j<nn;j++){xi[j]=Xi[j];yi[j]=Yi[j];}
    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 Cp(double T,double ai[])
    {a=ai;
    double A1=ai[2]/T/Math.sinh(ai[2]/T);
    double A2=a[3]/T/Math.cosh(ai[4]/T);
    double cp=ai[0]+ai[1]*A1*A1+ai[3]*A2*A2;
    return cp; //kJ/kgK
    }

    public double func(double ai[])
    {
    double ff=0;
    double w;
    double yy;
     for(int i=0;i<nn;i++)
              {w= Cp(xi[i],ai)-yi[i];
     ff+=w*w;
              }
    return ff;
            }
}
public class nelderi
{//2. Nelder ve Mead method(simpleks
method)_____
public static double[] nelder(if_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 minimisation method
  //  Nelder & Mead 1965 Computer J, v.7, 308-313.

  // Giriş değişkenleri tanımlaması
  // fnelder : abstract çok boyutlu fonksiyon f(x)
  // x : for n boyutlu n+1 simlex noktasını içerir bağımsız değişken seti
  // maxiteration : maximum iterasyon sayısı
  // tolerance :
      int NDIMS = x.length-1;
      int NPTS = x.length;
      int FUNC = NDIMS;
      int ncalls = 0;
        ////// başlangıç simplexini oluştur ///////////////////
        double p[][]=new double[NPTS][NPTS]; // [row][col] = [whichvx][coord,FUNC]
        double z[]=new double[NDIMS];
        double best = 1E99;
        ///////////////// iilk fonksiyon değerlerini hesapla /////////////////
        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++)
        {
          ///////////  lo, nhi, hi noktalarını tanımla ///////////////
          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;}
          ////////// çıkış kriteri ///////////////
          if ((iter % 4*NDIMS) == 0)
          {
            //yi nin standart hata kriteri set değerinden (tolerance)
            // küçük olmalı
            // ortalama değeri hesapla (en büyük değer de dahil olmak üzere)
            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];
}

///// ave[] vektorünü en büyük değeri hariç tutarak hesapla //////

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);


///////// yansıt ////////////////

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)  //// genişlet, else kabul et
{
    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;
    }
}

////////////// daralt:

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   /////// daralt
                {
                   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(if_xj fnelder,double a[],double da[],double tolerance)
     {return nelder(fnelder,a,da,300,tolerance,0);}

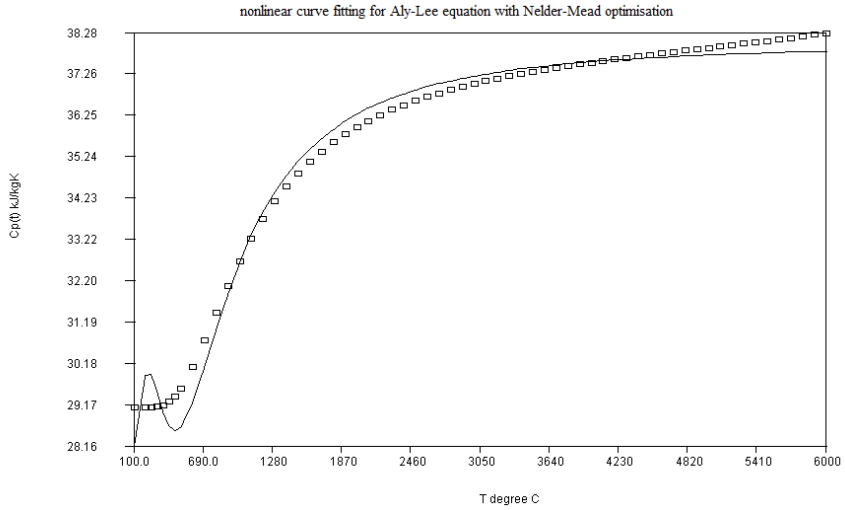
     public static double[] nelder(if_xj fnelder,double a[],double da[])
     {return nelder(fnelder,a,da,500,1.0e-20,0);}

    public static void main(String args[]) throws IOException
    {
    String in_name=JOptionPane.showInputDialog(" veri dosyasının ismini giriniz : ");
    //String in_name="D1.txt";
    double a[];
    a=new double[5];
    a[0] = 1.0;
    a[1] = 1.0;
    a[2] = 1.0;
    a[3] = 1.0;
```

```
    a[4] = 1.0;
    double da[];
    da=new double[5];
    da[0] =0.3;
    da[1] =0.3;
    da[2] =0.3;
    da[3] =0.3;
    da[4] =0.3;
    yy f=new yy(in_name,a);
    int i=0;
    double p[]=nelder(f,a,da,1000,1e-15,0);
    String s1=" optimisation value  : \n"+Matrix.toStringT(p)+"\n";
    s1+="original function value = "+f.func(a)+"\n";
    s1+="function value = "+f.func(p)+"\n";
    s1+="standard division= "+Math.sqrt(f.func(p)/(f.nn-1));
    System.out.println(s1);
    Plot pp=new Plot(f.xi,f.yi);
    pp.setPlotType(0,20);
    double y[]=new double[f.xi.length];
    for(i=0;i<f.xi.length;i++)
    {y[i]=f.Cp(f.xi[i],f.a);}
    pp.addData(f.xi,y);
    pp.plot();
    }
}
```

```
---------- Capture Output ----------
> "C:\java\bin\javaw.exe" nelder
end of file
 optimisation value  :
       27.950645606873834
       10.078840538226359
     -400.883987151156600
     -193.009818119868000
      645.817054212756800

original function value = 72086.49376191813
function value = 8.585551215087992
standard division= 0.3691593026264981
> Terminated with exit code 0.
```



PPDS equation is also widely used to define Cp values for ideal gas specific heat data for gases[40].

$$C_{p(T)} = a_1 + (a_2 - a_1)\left(\frac{T}{a_0 + T}\right)^2 \left[a_3 + a_4\left(\frac{T}{a_0 + T}\right) + a_5\left(\frac{T}{a_0 + T}\right)^2 + a_6\left(\frac{T}{a_0 + T}\right)^3\right] \quad (2.1.32)$$

This equation is also non-linear with respect to coefficients so the same Nelder-Mead nonlinear least square program is used.

Program 2.1.5 Non linear Nelder-Mead optimisation curve fitting program for PPDS equation

```
 import java.io.*;
import javax.swing.*;
    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,double ia[])
    {
     //read the data to curvefit
     //get the data file and initial fit coefficient when class is defined
              double[] Xi=new double[500];
     double[] 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 bulunamadı"); }
    nn=i;
    xi=new double[nn];
    yi=new double[nn];
    for(int j=0;j<nn;j++){xi[j]=Xi[j];yi[j]=Yi[j];}
    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 Cp(double T,double ai[])
    {a=ai;
```

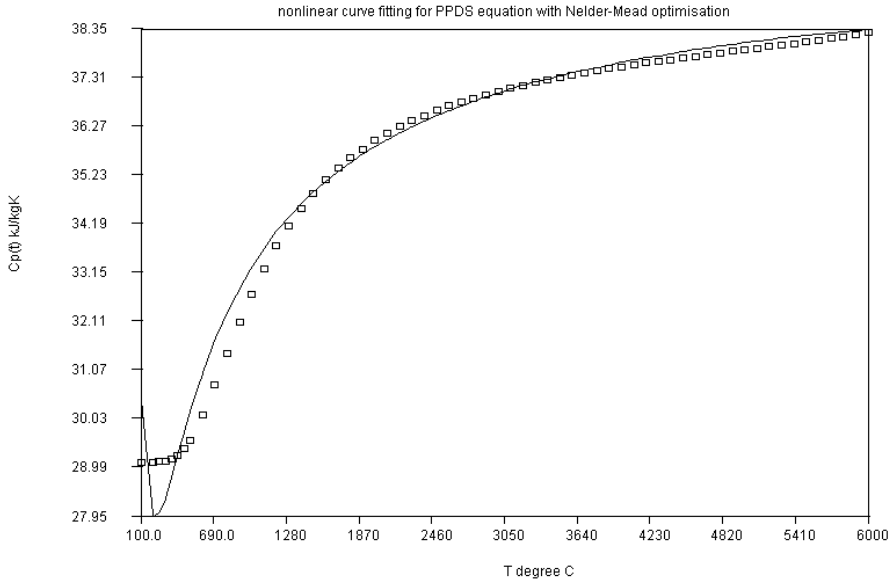
```
    double R=8.3145;
    double A1=T/(ai[0]+T);
    double A2=ai[0]/(ai[0]+T);
    double cp=R*(ai[1]+(ai[2]-ai[1])*A1*A1*(1.0-A2*(ai[3]+ai[4]*A1+ai[5]*A1*A1+ai[6]*A1*A1*A1)));
    return cp; //kJ/kgK
    }

    public double func(double ai[])
    {
    double ff=0;
    double w;
    double yy;
     for(int i=0;i<nn;i++)
              {w= Cp(xi[i],ai)-yi[i];
      ff+=w*w;
              }
    return ff;
             }
}
public class nelder1i
{
   public static void main(String args[]) throws IOException
   {
   String in_name=JOptionPane.showInputDialog(" veri dosyasının ismini giriniz : ");
   //String in_name="D1.txt";
   double a[];
   a=new double[7];
   a[0] = 1.0;
   a[1] = 1.0;
   a[2] = 1.0;
   a[3] = 1.0;
   a[4] = 1.0;
   a[5] = 1.0;
   a[6] = 1.0;
   double da[];
   da=new double[7];
   da[0] =0.3;
   da[1] =0.3;
   da[2] =0.3;
   da[3] =0.3;
   da[4] =0.3;
   da[5] =0.3;
   da[6] =0.3;
   yy f=new yy(in_name,a);
   int i=0;
   double p[]=nelderi.nelder(f,a,da,1000,1e-15,0);
   String s1=" optimisation value  : \n"+Matrix.toStringT(p)+"\n";
   s1+="original function value = "+f.func(a)+"\n";
   s1+="function value = "+f.func(p)+"\n";
   s1+="standard division= "+Math.sqrt(f.func(p)/(f.nn-1));
   System.out.println(s1);
   Plot pp=new Plot(f.xi,f.yi);
   pp.setPlotType(0,20);
   double y[]=new double[f.xi.length];
   for(i=0;i<f.xi.length;i++)
   {y[i]=f.Cp(f.xi[i],f.a);}
   pp.addData(f.xi,y);
   pp.plot();
```

```
    }
}
```

```
---------- Capture Output ----------
> "C:\java\bin\javaw.exe" nelder1
end of file
 optimisation value  :
     218.014481532370040
       4.734899316748212
       4.813940589168753
     345.654306606462340
    -534.657797593246000
     239.168194745319400
      26.384727444067180

original function value = 47629.457639999986
function value = 10.34871648952423
standard division= 0.40529660879849366
```



Now we can back to ideal gas equation of state. If we use a simple polynomial, Ideal gas equation of state can be modelled by using a simple excel program

Consider equation $ds = \frac{C_p - R_u}{T} dT + \frac{R_u}{v} dv$ (2.1.31)

Also consider that for ideal gas equation of state:

$T = \frac{Pv}{R}$ and $dT = \frac{Pdv + vdP}{R}$ (2.1.32)

substituting dT and T above equation gives:

$$ds = \frac{C_p}{T} dT - \frac{R_u}{T} dT + \frac{R_u}{v} dv = \frac{C_p}{T} dT - R_u \left[\frac{Pdv + vdP}{Pv} - \frac{dv}{v}\right] \quad (2.1.33)$$

$$ds = \frac{C_p}{T} dT - \frac{R_u}{P} dP \quad (2.1.34)$$

$$dh = Tds + vdP = T\left[\frac{C_p}{T} dT - \frac{R_u}{P} dP\right] + vdP$$

$$dh = C_p dT - \frac{R_u T}{P} dP + vdP = C_p dT - vdP + vdP$$

$$dh = C_p dT \quad (2.1.35)$$

Furthermore, if a polynomial equation

$$C_p(T) = a_0 + a_1T + a_2T^2 + a_3T^3 + a_4T^4 \quad \frac{kJ}{kmol} \qquad (2.1.36)$$

is taken into the account:

$$h(T) = h_0 + a_0(T - T_0) + \frac{a_1}{2}(T^2 - T_0^2) + \frac{a_2}{3}(T^3 - T_0^3) + \frac{a_3}{4}(T^4 - T_0^4) + \frac{a_4}{5}(T^5 - T_0^5) \quad (2.1.37)$$

$$s(T) = s_0 + a_0 ln\left(\frac{T}{T_0}\right) + a_1(T - T_0) + \frac{a_2}{2}(T^2 - T_0^2) + \frac{a_2}{3}(T^3 - T_0^3) + \frac{4}{4}(T^4 - T_0^4) \quad (2.1.38)$$

Let us apply these equations in excel environment to calculate ideal gas properties:

Ideal gas properties

N2 Cp=a0+a1T+a2*T^2+a3*T^3+a4*T^4

a0	2.7198729E+01	
a1	6.9432111E-03	
a2	-1.5688811E-06	
a3	1.2118403E-10	
a4	-7.1620421E-17	
R	8.3144720E+00	kJ/kmolK
T0	2.9815000E+02	deg K
h0	8.6700000E+03	
s0	1.9160900E+02	
P0	1.0000000E+02	kPa
M	2.8013480E+01	kg/kmol

T	3.0000000E+02	degree K
P	100	kPa
Cp	2.9143765E+01	kJ/kmolK
h	8.7239056E+03	kJ/kmol
s	1.9178924E+02	kJ/kmolK
s0	1.9178924E+02	kJ/kmolK
Cp	2.9143765E+01	kJ/kgK
h	8.7239056E+03	kJ/kg
s	1.9178924E+02	kJ/kgK
s0	1.9178924E+02	kJ/kgK

if partial continuous form of C_p is taken into the account, our basic thermodynamic equations for partially continious Cp equation

$$C_{pi}(T) = a_{0i} + a_{1i}10^{-3}T + \frac{a_{2i}10^5}{T} + a_{3i}10^{-3}T^2 \quad \frac{kJ}{kmolK} \quad T_{Li} \le T \le T_{Hi} \qquad (2.1.39)$$

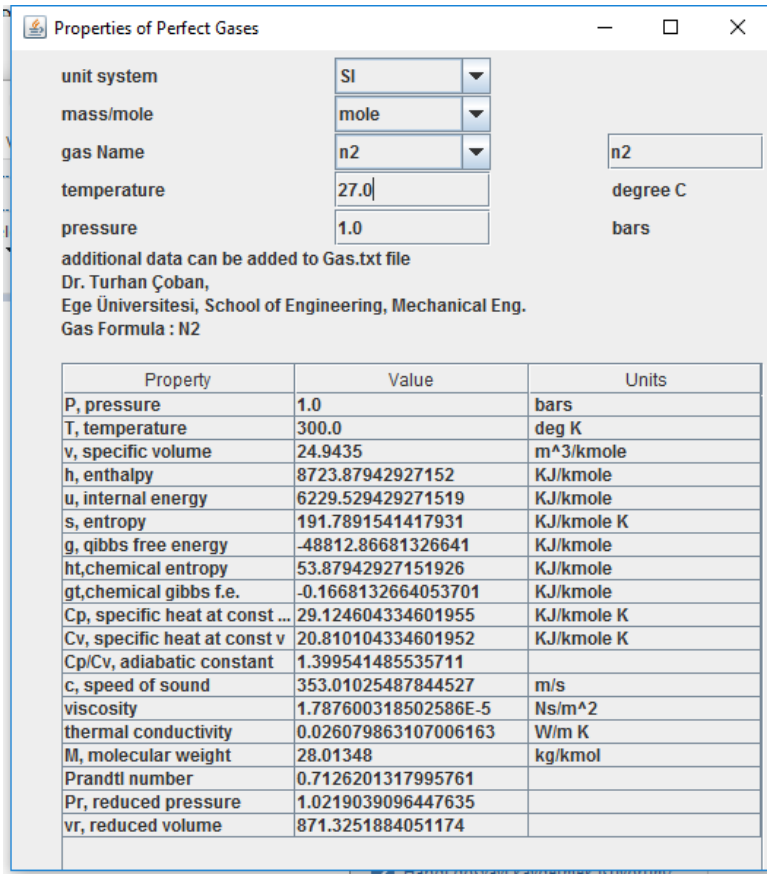
becomes :

$$h(T) = h_0 + \int_{T_0}^{T_{Hi}} C_{pi}(T)dT + \sum_{i=1}^{N-1} \int_{T_{Li}}^{T_{Hi}} C_{pi}(T)dT + \int_{T_{Li}}^{T} C_{pi}(T)dT \quad (2.1.40)$$

$$s(T) = s_0 + \int_{T_0}^{T_{Hi}} \frac{C_{pi}(T)}{T}dT + \sum_{i=1}^{N-1} \int_{T_{Li}}^{T_{Hi}} \frac{C_{pi}(T)}{T}dT + \int_{T_{Li}}^{T} \frac{C_{pi}(T)}{T}dT - Rln\frac{P}{P_0} \quad (2.1.41)$$

Now we will convert this equations into a computer program. Program is written to calculate a varity of gases. The gas data is inserted into the program can be obtained by calling the name of the gas. Note that as an example case Nitrogen in the previous excel polynomial curve fitting example is given so that reasults can be compared. The copy of the program and a running version can also be found in www.turhancoban.com adress (SCO1.jar included all the programs in this book)

User interface GasTable using Gas.java (Partially continious polynomial equations)



Property	Value	Units
P, pressure	1.0	bars
T, temperature	300.0	deg K
v, specific volume	24.9435	m^3/kmole
h, enthalpy	8723.87942927152	KJ/kmole
u, internal energy	6229.529429271519	KJ/kmole
s, entropy	191.7891541417931	KJ/kmole K
g, qibbs free energy	-48812.86681326641	KJ/kmole
ht,chemical entropy	53.87942927151926	KJ/kmole
gt,chemical gibbs f.e.	-0.1668132664053701	KJ/kmole
Cp, specific heat at const ...	29.124604334601955	KJ/kmole K
Cv, specific heat at const v	20.810104334601952	KJ/kmole K
Cp/Cv, adiabatic constant	1.399541485535711	
c, speed of sound	353.01025487844527	m/s
viscosity	1.787600318502586E-5	Ns/m^2
thermal conductivity	0.026079863107006163	W/m K
M, molecular weight	28.01348	kg/kmol
Prandtl number	0.7126201317995761	
Pr, reduced pressure	1.0219039096447635	
vr, reduced volume	871.3251884051174	

.

When the gases are not pure but a mixture of gases are given, ideal gas mixtures can be found by using ideal gas mixture rules

$$\frac{v_i}{v} = \frac{V_i}{V} = \frac{N_i}{N} = x_i \quad (2.1.42)$$

V_i is the volume of each gas in the mixture, N_i is the mole number of each gas, x_i is the volumetric ratio

$$N = \sum_{i=1}^{m} N_i \quad (2.1.43)$$

$$h = \sum_{i=1}^{m} h_i\, x_i \quad (2.1.44)$$

$$s = \sum_{i=1}^{m} s_i\, x_i \quad (2.1.45)$$

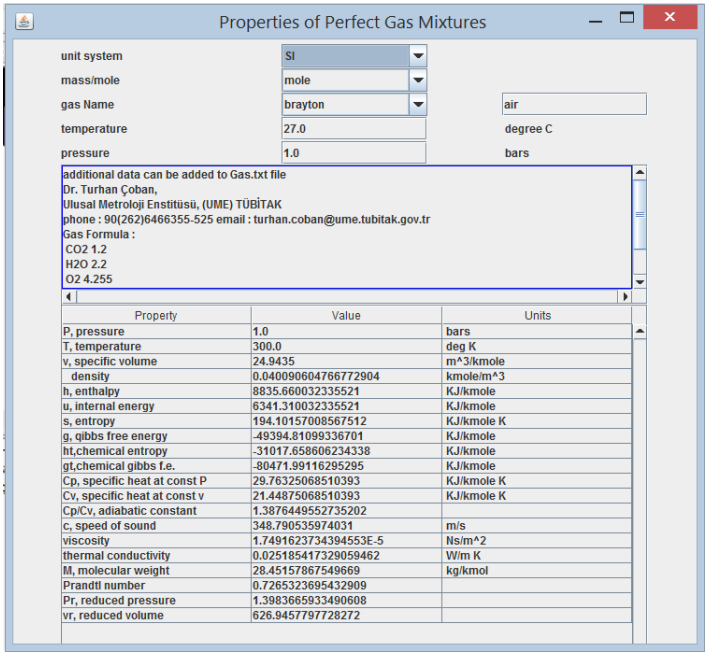
For mixture programming Gmix.java program is given as a human interface graphic program GmixTable is prepared. Program lists are given in internet site www.turhancoban.com adress (SCO1.jar included all the programs in this book)

A mixture can be created for the program by adding gas mixing data into the Gmix.txt as:

```
brayton
4
co2 1.2
h2o 2.2
o2 4.255
n2 24.659
```

wetair
3
n2 0.79
o2 0.21
h2o 0.002

User interface GmixTable using Mix.java (Partially continious polynomial equations)



In order to create perfect gas equation of state, cubic spline curve fitting can also be used as well. As an interpolation formula cubic splines are partial continious equations. Their first and second derivatives are also continious at the end of each segment. In this case enthalpy and entropy calculations will be carried out as follows: remembering cubic spline curve fitting equation given in the form:

$$s_k(x) = a_k(x - x_k) + b_k(x_{k+1} - x) + \left[(x - x_k)^3 c_{k+1} + (x_{k+1} - x)^3 c_k\right]/(6h_k) \quad 1 \le k \le n \quad (2.1.46)$$

If the cubic terms opened and equation written as Cp form

$$C_{pk}(T) = a_k(T - T_k) + b_k(T_{k+1} - T) + \left[\left(T^3 - 3T^2T_k + 3TT_k^2 - T_k^3\right)c_{k+1} + \left(T_{k+1}^3 - 3T_{k+1}^2T + 3T_{k+1}T^2 + T^3\right)c_k\right] /(6h_k) \quad (2.1.47)$$

Integration of this term gives:

$$\int C_{pk}(T)dT = a_k\left(\frac{T^2}{2} - TT_k\right) + b_k\left(TT_{k+1} - \frac{T^2}{2}\right) + \left[\left(\frac{T^4}{4} - T^3T_k + \frac{3}{2}T^2T_k^2 - TT_k^3\right)c_{k+1} + \left(TT_{k+1}^3 - \frac{3}{2}T_{k+1}^2T^2 + T_{k+1}T^3 + \frac{T^4}{4}\right)c_k\right] /(6h_k) \quad (2.1.48)$$

The other integration needed for ideal gas equation of state will be:

$$\int \frac{C_{pk}(T)}{T}dT = a_k(T - T_k\ln(T)) + b_k(T_{k+1}\ln(T) - T) + \left[\left(\frac{T^3}{3} - \frac{3}{2}T^2T_k + 3TT_k^2 - T_k^3\ln(T)\right)c_{k+1} + \left(T_{k+1}^3\ln(T) - 3T_{k+1}^2T + \frac{3}{2}T_{k+1}T^2 + \frac{T^3}{3}\right)c_k\right] /(6h_k) \quad (2.1.49)$$

These eqautions can be apply with summation processes of partially continuous cubic spline equations to calculate enthaly, entropy and other properties

$$h(T) = h_0 + \int_{T_0}^{T_{kN}>T_0} C_{pi}(T)dT + \sum_{k=kN}^{kM} \int_{T_{kN}}^{T_{kM}} C_{pi}(T)dT + \int_{T_{kN}<T}^{T} C_{pi}(T)dT \quad (2.1.50)$$

$$s(T) = s_0 + h_0 + \int_{T_0}^{T_{kN}>T_0} \frac{C_{pi}(T)}{T}dT + \sum_{k=kN}^{kM} \int_{T_{kN}}^{T_{kM}} \frac{C_{pi}(T)dT}{T} + \int_{T_{kN}<T}^{T} \frac{C_{pi}(T)}{T}dT - Rln\frac{P}{P_0} \quad (2.1.51)$$

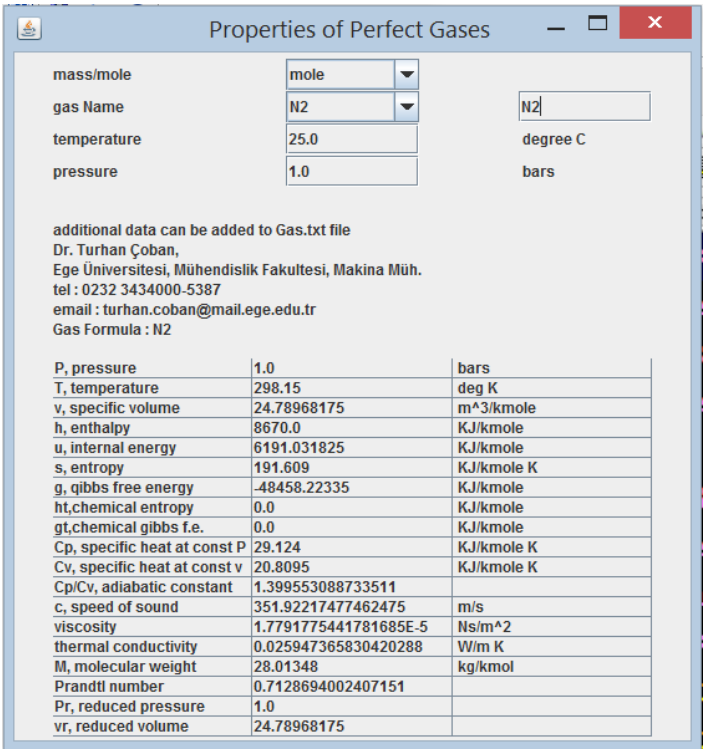
Integration terms are preformed inside cubic spline class (NA49) and carried out into the GasCS program. The list of the GasCP program is given below:

Program 2.1.3 GasCStest output program

```
public class GasCStest
{
public static void main(String arg[])
{GasCS g=new GasCS("N2");
double T=300;
 System.out.println("h0="+g.h0(T)+"\n"+"s0="+g.s0(T)+"\n"+g.toString(T,1.0));
}}
```

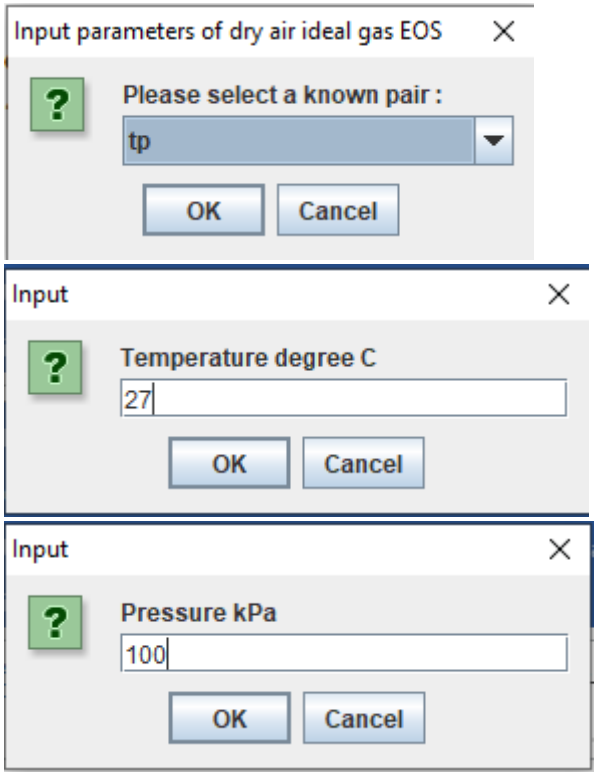
```
---------- Capture Output ----------
> "C:\java\bin\java.exe" GasCS
P, pressure                1.01325 bars
T, temperature             300.0 deg K
v, specific volume         24.617320503330866 m^3/kmole
h, enthalpy                54.35389737985001 KJ/kmole
u, internal energy         -2439.9961026201504 KJ/kmole
s, entropy                 258.6841691304673 KJ/kmole K
g, qibbs free energy       -77550.89684176033 KJ/kmole
ht,chemical entropy        54.35389737985001 KJ/kmole
gt,chemical gibbs f.e.     -101619.13877052117 KJ/kmole
Cp, specific heat at const P 29.385 KJ/kmole K
Cv, specific heat at const v21.070500000000003 KJ/kmole K
Cp/Cv, adiabatic constant  1.394603829999288
c, speed of sound          329.7139156184356 m/s
viscosity                  2.0707616451899093E-5 Ns/m^2
thermal conductivity       0.02637152631378943 W/m K
M, molecular weight        31.9988 kg/kmol
Prandtl number             0.7210856436043875
Pr, reduced pressure       1.0416020232991565
vr, reduced volume         23.94724610940586
> Terminated with exit code 0.
```

User interface GasCSTable



P, pressure	1.0	bars
T, temperature	298.15	deg K
v, specific volume	24.78968175	m^3/kmole
h, enthalpy	8670.0	KJ/kmole
u, internal energy	6191.031825	KJ/kmole
s, entropy	191.609	KJ/kmole K
g, qibbs free energy	-48458.22335	KJ/kmole
ht,chemical entropy	0.0	KJ/kmole
gt,chemical gibbs f.e.	0.0	KJ/kmole
Cp, specific heat at const P	29.124	KJ/kmole K
Cv, specific heat at const v	20.8095	KJ/kmole K
Cp/Cv, adiabatic constant	1.399553088733511	
c, speed of sound	351.92217477462475	m/s
viscosity	1.7791775441781685E-5	Ns/m^2
thermal conductivity	0.025947365830420288	W/m K
M, molecular weight	28.01348	kg/kmol
Prandtl number	0.7128694002407151	
Pr, reduced pressure	1.0	
vr, reduced volume	24.78968175	

Dry air is one of the most important gas that can be treated as an ideal gas. A recent code is developed to process only for dry air. Program input-output of air_PG is shown below.



Properties of dry air ideal gas EOS

array no	property name	dry air property	unit
0	P, pressure	100.0	kPA
1	T, temperature	300.15	deg K
2	v, specific volume	0.861577954052...	m³/kg
3	h, enthalpy	27.16103226066...	KJ/kg
4	u, internal energy	144.7895171411...	KJ/kg
5	s, entropy	0.098601359074...	KJ/kgK
6	x, quality	1.0	kg vap/kg mix
7	Cp, specific heat...	1.006359551191...	KJ/kgK
8	Cv, specific heat...	0.719310424403...	KJ/kgK
9	α isobaric therm...	0.0	1/K
10	κ isothermal co...	0.0	1/Pa
11	η Dynamic visco...	1.855104504597...	Pa.s
12	k Thermal condu...	0.026222311846...	W/mK
13	σ Surface tension	0.0	N/m
14	Prandtl number	0.713455532780...	
15	ρ density	1.160661081560...	kg/m³
16	speed of sound	347.1887930980...	m/s

Another program defined as perfect gas EoS is Gas_PG. In this program approximately 500 gases are defined as partial continious curve fitting or polynomial curve fitting specific heat based equations. A sample usage of this class:

```
public static void main(String arg[])
{Gas_PR1 pr=new Gas_PR1("N2");
  double T=300.0; //degree K
  double P=100.0; //kPA
   pr.print("tp",T,P);
}
```

Properties of dry air ideal gas EOS

array no	property name	dry air property	unit
0	P, pressure	100.0	kPA
1	T, temperature	300.0	deg K
2	v, specific volume	24.94338785445...	m³/kmol
3	h, enthalpy	781.8775745665...	KJ/kmol
4	u, internal energy	558.6342532691...	KJ/kmol
5	s, entropy	2.839787587724...	KJ/kmolK
6	x, quality	2.0	kg vap/kg mix
7	Cp, specific heat...	29.12505435085...	KJ/kmolK
8	Cv, specific heat...	20.81059173269...	KJkmolK
9	α isobaric therm...	0.003333333333...	1/K
10	κ isothermal co...	2.8013E-4	1/Pa
11	η Dynamic visco...	1.787600318502...	Pa.s
12	k Thermal condu...	0.027666480768...	W/mK
13	σ Surface tension	0.0	N/m
14	Prandtl number	6.717746743016...	
15	ρ density	0.040090785014...	kmol/m³
16	speed of sound	353.0116734039...	m/s

You can obtain all properties as an array by using property method. Property method accepts two known thermodynamic property set. Known property sets are as :

Known set	meaning	Known set	meaning
tp	Temperature Pressure	pt	Pressure temperature
vt	Specific volume	tv	Temperature

	Temperature		Specific volume
rot	Density temperature	tro	Temperature density
pv	Pressure Specific volume	vp	Specific volume pressure
pro	Pressure density	rop	Density pressure
ph	Pressure enthalpy	Hp	Enthalpy Pressure
ps	Pressure entropy	sp	Entropy pressure

In the following example property pair “tp” is used.

```
public static void main(String arg[])
{Gas_PR1 pr=new Gas_PR1("N2");
  double T=300.0; //degree K
  double P=100.0; //kPA
  double a[]=pr.property("tp",T,P);
  for( double x :a) System.out.println(x);
}
```

```
---------- Capture Output ----------
> "E:\co\java\bin\javaw.exe" Gas_PG
100.0
300.0
24.93274483602078
783.5833903494341
559.6638294989782
2.844780548049795
2.0
29.12505435085314
20.810591732699898
2.7999016699420404E-4
2.7999016699420404E-4
1.787600318502586E-5
0.027666480768509534
0.0
6.717746743016922E-4
0.04010789853170447
353.0244735640423

> Terminated with exit code 0.
```

Program List of ideal gases :

Gas	Ideal gas EOS for multiple gases, using partial continious least square curve fitting
GasCS	Ideal gas EOS fr multiple gases, using cubic spline curve fitting
air_PG	Air as a perfect gas using partial continious least square curve fitting
Gas_PG	Approximately 500 gases defined in subclass Gas_Data as using partial continious least square curve fitting and polynomial curve fitting
GasTable	User graphical interface for Gas
GasCSTable	User graphical interface for GasCS
GasModel	Subprogram of User graphical interface for Gas
GasCSModel	Subprogram of User graphical interface for GasCS
Gas_GEN_Data	partial continious least square curve fitting program for class Gas, air_PG

2.2 CUBIC EQUATION OF STATES

A group of gas equation of states that has one certain advantage is called cubic equation of states. In an equation of states given P(T,v) form, solving v is usually required root finding methods. But cubic equation of states can be written an inverse form of cubic polynomials, therefore by solving roots of cubic polynomial, v can be solved.

Cubic Equation of State has a general form of equation

$$P = \frac{RT}{V-b} - \frac{a}{V^2+ubV+wb^2} \quad (2.2.1)$$

The equation can also be written in the following form:

$$Z^3 - (1 + B^* - uB^*)Z^2 + (A^* + wB^{*2} - uB^* - uB^{*2})Z - A^*B^* - B^{*2} - wB^{*2} - wB^{*3} = 0 \quad (2.2.2)$$

where $A^* = \frac{aP}{R^2T^2}$ (2.2.3) and $B^* = \frac{bP}{RT}$ (2.2.4) $Z = \frac{PV}{RT}$ (2.2.5)

Some of the most common equation of states are: van der Waals , Redlich-Kwong (RK) ,Soave (SRK) , ve Peng-Robinson (PR). All these equations are the same form, but structures of the coefficients *u*, *w*, *a*, *b* are different

Tablo (2.2.1). Coefficients of cubic equation of states

Denklem	u	w	b	a	fω
van der Waals	**0**	**0**	$\frac{RT_{crit}}{8P_{crit}}$	$\frac{27}{64}\frac{R^2T_{crit}^2}{P_{crit}}$	
Redlich Kwong	**1**	**0**	$\frac{0.08664RT_{crit}}{P_{crit}}$	$0.42748\frac{R^2T_{crit}^{2.5}}{P_{crit}T^{1/2}}$	
Soawe	**1**	**0**	$\frac{0.08664RT_{crit}}{P_{crit}}$	$\frac{0.42748R^2T_{crit}^2}{P_{crit}}[1 + f\omega(1 - T_r^{0.5})]^2$	$0.48 + 1.574\omega - 0.176\omega^2$
Peng Robinson	**2**	**-1**	$\frac{0.0780RT_{crit}}{P_{crit}}$	$\frac{0.45724R^2T_{crit}^2}{P_{crit}}[1 + f\omega(1 - T_r^{0.5})]^2$	$0.37464 + 1.54226\omega - 0.269992\omega^2$

ω in Peng-Robinson and Soawe equation of states coefficient is called **Pitzer's accentric factor**. This factor is calculated as

$$\omega = -log_{10}P_{saturated\ vapor}(at\ T_r = 0.7) - 1$$

To obtain values of ω, the reduced vapor pressure ($P_r = P/P_{crit}$) at $T_r = T/T_{crit} = 0.7$ is required. For example for water T_{crit}=647.3 K T=0.7*647.3=453.11 K=179.96 C so for this temperature P_v=10.01168 bar and P_{crit}=221.2 bar

$$P_r = P/P_{crit} = \frac{10.01168}{221.2} = 0.045261$$

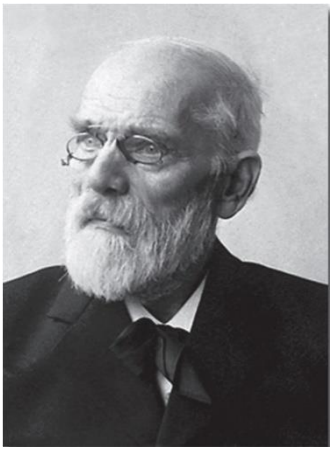
$$\omega = -log_{10}P_{saturated\ vapor}(0.045261) - 1 = 1.344278 - 1 = 0.344278$$

A list of Pitzer's acentric factor, critical properites etc. for some substances are given in the table below:

		molar mass	Boiling T	Critical T	Critical P	Critical comp.	Pitzer's acentric factor	specific heat	Thermal conduc.	Dyn vis
		M	Tb	Tc	Pc	Zc	ω	Cp	k	μ10⁶
Substance	**Formula**	**kg/kmol**	**K**	**K**	**Mpa**			**J/kgK**	**W/(mK)**	**Pa.s**
Acetone	C3H6O	0.058	329.2	508	4.7	0.233	0.309	1300		
Acetylene	C2H2	0.026	189.5	309	6.2	0.271	0.184	1580	0.019	9.3
Air		0.029	82e	132g	3.75g	0.28g	0.035	1004	0.024	18.1
Ammonia	NH3	0.017	239.8	406	11.3	0.242	0.25	2200	0.022	9.3
Argon	Ar	0.04	87.4	151	4.86	0.291	0	523	0.018	21
Benzene	C6H6	0.078	353.3	563	4.92	0.271	0.212	1300	0.007	7
1,3-Butadiene	C4H6	0.054	268.5	425	4.33	0.27	0.193	1510		
n-Butane	C4H10	0.058	272.6	425	3.8	0.274	0.193	1580	0.015	7
iso-Butane	C4H10	0.058	261.5	408	3.64	0.28	0.176	1580	0.015	9
Carbon dioxide	CO2	0.044	194.7f	304	7.38	0.274	0.225	840h	0.016	14.4
Carbon monoxide	CO	0.028	81.7	133	3.5	0.295	0.049	1100	0.023	17
Carbon tetracholoride	CCl4	0.154	349.7	556	4.56	0.272	0.194	862	0.017	16

Cyclohexane	C6H6	0.084	353.9	554	4.07	0.273	0.212			
n-Decane	C10H22	0.142	447.3	619	2.12	0.247	0.49	1680		
n-Dodecane	C12H26	0.17	489.4	659	1.8	0.24	0.562	1690		
DME(dimethylether)	C2H6O	0.046	250.6	400	5.37	0.271	0.274	1430		
Ethane	C2H6	0.03	184.6	305	4.88	0.285	0.1	1700	0.02	11
Ethanol	C2H6O	0.046	351.5	516	6.39	0.248	0.635	1520	0.013	14.2
Ether(diethyl ether)	C4H10O	0.074	307.6	467	3.61	0.26	0.281	1600	0.015	7.5
ETBE(ethyl tert-butyl ether)	C6H14O	0.102	345	517	3.11	0.274	0.298	1550		
Ethylene	C2H4	0.028	169.5	283	5.12	0.276	0.085	1470	0.018	9.6
Ethylene glycol	C2H6O2	0.062	471	645	7.53	0.268	1.137	1410		
Helium(4 He)	He	0.004	4.2	5.3	0.23	0.301	-0.387	5190	0.142	19
Helium(3 He)	He	0.003	3.2	3.3	0.11	0.301	-0.46			
n-Heptane	C7H16	0.1	371	540	2.77	0.263	0.35	1650	0.013	6.5
n-Hexane	C6H14	0.086	342	508	3.03	0.263	0.296	1700	0.014	6.5
Hydrazine	N2H4	0.032	387	653	14.7	0.376	0.325			
Hydrogen	H2	0.002	20.1	33	1.32	0.305	-0.22	14200	0.168	8.4
(Hydrogen) Deuterium	D2	0.004	23.6	38	1.66	0.249	-0.16	14200	0.131	12
Mercuryi	Hg	0.201	630	736	104					
Methane	CH4	0.016	112	191	4.6	0.288	0.01	2180	0.031	10.3
Methanol	CH4O	0.032	338.1	513	8.08	0.224	0.559	1350	0.015	9.8
MTBE(methyl tert-butyl ether)	C5H12O	0.088	328	497	3.43	0.273	0.267	1500		
Neon	Ne	0.02	26.2	44	2.7	0.301	0	1030	0.046	30
Nitrogen	N2	0.028	77.4	126	3.39	0.29	0.038	1040	0.024	16.6
Nitrogen dioxide	NO2	0.046	294.4	431	10.1	0.233		800	0.017	130
Nitrogen monooxide	NO	0.03	121.2	180	6.55	0.25	0.607	996	0.024	29.4
di-Nitrogen oxde	N2O	0.044	184.7	310	7.26	0.272	0.141	864	0.015	13.6
n-Octane	C8H18	0.114	399	569	2.49	0.259	0.394	1700	0.02	7.5
iso-Octane	C8H18	0.114	372	544	2.59	0.267	1650			
Ozone	O3	0.048	161.4	268	6.78	0.272				
Oxygen	O2	0.032	90.2	155	5.08	0.288	0.021	913	0.024	19.1
iso-Pentane	C5H12	0.072	301.3	461	3.33	0.268	0.227	1680	0.015	11.7
n-Pentane	C5H12	0.072	309.2	470	3.38	0.262	0.251	1680	0.015	11.7
Phenol	C6H6O	0.094	455	694	6.13	0.243	0.426			
Propane	C3H8	0.044	231.1	370	4.26	0.281	0.152	1570	0.015	7.4
iso-Propanol	C3H8O	0.06	355.4	508	4.76	0.248	0.669	1540		
Propylene(propene)	C3H6	0.042	225.4	365	4.62	0.275	0.148	1460	0.014	8.1
Propylene glycol	C3H8O2	0.076	461.3	626	6.1	0.28	1.107			
R12(dichlorodifluoromethane)	CCl2F2	0.121	243	385	4.14	0.28	0.179	573	0.008	12.5
R134a(tetrafluoroethane)	CF3CH2F	0.102	246.6	374	4.07	0.258	0.33	840	0.014	12.2
R410Al	n.a.	0.073	221.8	345	4.9	0.271	0.296	820		
Sulfurdioxide	SO2	0.064	263.2	430	7.87	0.264	0.251	607	0.009	11.6
Sulfurhexafluoride	SF6	0.146	204.9f	319	3.76	0.36	0.21	598	0.12	16
Toluene	C6F14	0.092	383.7	592	4.13	0.284	0.266			

Tetradecafluorohexane	hexafluoriden	0.338	329	449	1.83					
Uranium hexafluoride	UF6	0.352	329	503	4.6	0.282	0.092	370	0.009	20
Waterp	H2O	0.018	372.8	647.3	22.12	0.229	0.344	2050	0.025	12.1
Xenon	Xe	0.131	165	289.8	5.84	0.291	0	158	0.006	22.5



Johannes Diderik Van der Waals, 1837-1923

As an example, let us calculate enthapy and entropy for van der Waals equation of state. This equation has the form

$P = \frac{R_u T}{v-b} - \frac{a}{v^2}$ (2.2.6) where a and b are constant (is not function of temperature). Therefore

$$ds = \frac{C_v}{T}dT + \left(\frac{\partial P}{\partial T}\right)_v dv \qquad (1.47)$$

$$du = C_v dT + \left(T\left(\frac{\partial P}{\partial T}\right)_v - P\right)dv \qquad (1.48)$$

$$\left(\frac{\partial P}{\partial T}\right)_v = \frac{R_u}{v-b} \quad (2.2.7)$$

$$du = C_v dT + \left[T\frac{R_u}{v-b} - \left(\frac{R_u T}{v-b} - \frac{a}{v^2}\right)\right]dv \qquad (2.2.8)$$

$$du = C_v dT + \left(\frac{a}{v^2}\right)dv \qquad (2.2.8a)$$

$$ds = \frac{C_v}{T}dT + \frac{R_u}{v-b}dv \qquad (2.2.9)$$

$$u = u_0 + \int_{T_0}^{T} C_v dT + a\left(\frac{1}{v} - \frac{1}{v_0}\right) \qquad (2.2.10)$$

$$s = s_0 + \int_{T_0}^{T} \frac{C_v}{T}dT + R_u ln\frac{v-b}{v_0-b} \qquad (2.2.11)$$

Now let us look at a little bit more complex equation of state. If we consider Redlich-Kwong equation of state, we should notice that coefficient b is actualy function of temperature, therefore;



Otto Redlich



Joseph Neng Shun Kwong

$$P = \frac{RT}{v-b} - \frac{a}{v^2+bv} \quad (2.2.12)$$

$$b = \frac{0.0780RT_{crit}}{P_{crit}} = const.$$

$$a(T) = 0.42748\frac{R^2T_{crit}^{2.5}}{P_{crit}T^{1/2}}$$

$$\left(\frac{\partial P}{\partial T}\right)_v = \frac{R_u}{v-b} - \frac{\partial}{\partial a}\left(\frac{a}{v^2+bv}\right)\frac{\partial a}{\partial T} \quad (2.2.13)$$

$$\frac{\partial a(T)}{\partial T} = -\frac{1}{2}0.42748\frac{R^2T_{crit}^{2.5}}{P_{crit}T^{3/2}} \quad (2.2.14)$$

$$\left(\frac{\partial P}{\partial T}\right)_v = \frac{R_u}{v-b} + 0.21374\left(\frac{1}{v^2+bv}\right)\frac{R^2T_{crit}^{2.5}}{P_{crit}T^{3/2}} \quad (2.2.15)$$

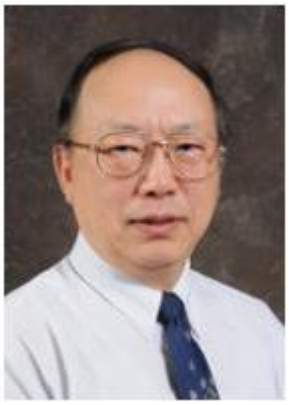
Of course derivatives can be taken numerically. With formulas such as:

$$\left(\frac{\partial P}{\partial T}\right)_v \cong \frac{-P(T+2\Delta T) + 8P(T+\Delta T) - 8P(T-\Delta T) + P(T-2\Delta T)}{12\Delta T} \quad (2.2.16a)$$

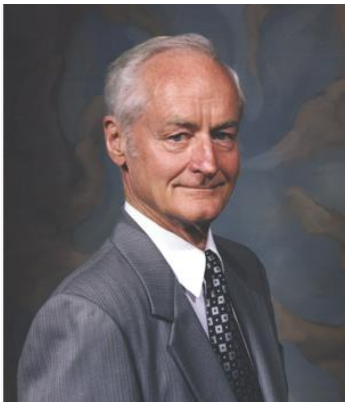
$$\left(\frac{\partial P}{\partial T}\right)_v \cong \frac{-P(T+3\Delta T) - 9P(T+2\Delta T) + 45P(T+\Delta T) - 45P(T-\Delta T) + 9P(T-2\Delta T) - P(T-3\Delta T)}{60\Delta T} \quad (2.2.16b)$$

then the result can be substite in original equation to solve the thermodynamic properties. Then the value can be substitude back into the main equations to solve them.

Let us look at Peng-Robinson Equation of States[78] now



Ding-Yu Peng



Donald B. Robinson

$$ds = \frac{C_v}{T}dT + \left(\frac{\partial P}{\partial T}\right)_v dv \quad (1.47)$$

$$du = C_v dT + \left(T\left(\frac{\partial P}{\partial T}\right)_v - P\right)dv \quad (1.48)$$

$$P = \frac{RT}{v-b} - \frac{a}{v^2+2bv-b^2} \quad (2.2.17)$$

$b = \frac{0.08664RT_c}{P_c} = const.$

$f\omega = (0.37464 + 1.54226\omega - 0.269992\omega^2)$

$$a(T) = \frac{0.45724R^2T_c^2}{P_c}\left[1 + f\omega\left[1 - \left(\frac{T}{T_c}\right)^{0.5}\right]\right]^2$$

$$\left(\frac{\partial P}{\partial T}\right)_v = \frac{R_u}{v-b} - \frac{0.91448R^2T_c^2}{P_c}\left(\frac{1}{v^2+2bv-b^2}\right)\left[1 + f\omega\left[1 - \left(\frac{T}{T_c}\right)^{0.5}\right]\right] f\omega\left(\frac{1}{TT_c}\right)^{0.5} \quad (2.2.18)$$

In order to solve thermodynamic properties Helmholtz energy based departure functions will be used. Helmholz energy in derivative form:

$$dA = -Pdv \quad (2.2.19)$$

$$A - A_0 = -\int_{v_0}^{v} Pdv \quad (2.2.20)$$

The evaluation of this integral is inconvenient since one limit of integration refers to the real state but the other to the reference, the perfect gas state, so integral will be break into two

$$A - A_0 = -\int_{\infty}^{v} Pdv - \int_{v_0}^{\infty} Pdv \quad (2.2.21)$$

Second term in the integral is ideal gas state so $P = \frac{RT}{v}$ (2.2.22)

Lets add and subtract integral $\int_{\infty}^{v}\frac{RT}{v}dv$ and $\int_{v_0}^{\infty}\frac{RT}{v}dv$ into the equation

$$A - A_0 = -\int_{\infty}^{v} Pdv - \int_{v_0}^{\infty}\frac{RT}{v}dv + \int_{\infty}^{v}\frac{RT}{v}dv - \int_{\infty}^{v}\frac{RT}{v}dv + \int_{v_0}^{\infty}\frac{RT}{v}dv - \int_{v_0}^{\infty}\frac{RT}{v}dv \quad (2.2.23)$$

$$A - A_0 = -\int_{\infty}^{v}\left(P - \frac{RT}{v}\right)dv - \int_{v_0}^{v}\frac{RT}{v}dv \quad (2.2.24)$$

Substituting Peng-Robinson equation of state into the equation

$$A - A_0 = -\int_{\infty}^{v}\left(\frac{RT}{v-b} - \frac{a}{v^2+2bv-b^2} - \frac{RT}{v}\right)dv - \int_{v_0}^{v}\frac{RT}{v}dv \quad (2.2.25)$$

$$A - A_0 = -\int_{\infty}^{v}\left(\frac{RT}{v-b}\right)dv + \int_{\infty}^{v}\left(\frac{a}{v^2+2bv-b^2}\right)dv + \int_{\infty}^{v}\left(\frac{RT}{v}\right)dv - \int_{v_0}^{v}\frac{RT}{v}dv \quad (2.2.26)$$

$\int_{\infty}^{v}\left(\frac{1}{v^2+2bv-b^2}\right)dv = \int_{\infty}^{v}\left(\frac{1}{(v+b)^2-2b^2}\right)dv \quad X = (v+b) \quad C = \sqrt{2}b \quad \int\frac{1}{X^2-C^2}dX =$

$\int\frac{1}{(X-C)(X+C)}dX = \frac{1}{2C}\left[\int\frac{1}{X-C}dX + \int\frac{1}{X+C}dX\right]$ (2.2.27)

$$\int_{\infty}^{v}\left(\frac{a}{v^2+2bv-b^2}\right)dv = \frac{a}{2\sqrt{2}b}ln\frac{\left(v+b-\sqrt{2}b\right)}{\left(v+b+\sqrt{2}b\right)} \quad (2.2.28)$$

So $A - A_0 = -RTln(v-b) + \frac{a}{2\sqrt{2}b}ln\frac{(v+b-\sqrt{2}b)}{(v+b+\sqrt{2}b)} + RTln(v) - RTln\frac{v}{v_0}$ (2.2.29)

Entropy departure function:

$$S - S_0 = -\left.\frac{\partial(A-A_0)}{\partial T}\right|_v$$

$$= -\frac{\partial}{\partial T}\left[-RTln(v-b) + \frac{a}{2\sqrt{2}b}ln\frac{\left(v+b-\sqrt{2}b\right)}{\left(v+b+\sqrt{2}b\right)} + RTln(v) - RTln\frac{v}{v_0}\right] \quad (2..30)$$

$$S - S_0 = Rln(v-b) - \frac{da}{dT}\frac{1}{2\sqrt{2}b} ln\frac{\left(v+b-\sqrt{2}b\right)}{\left(v+b+\sqrt{2}b\right)} - Rln(v) + Rln\frac{v}{v_0} \quad (2.2.31)$$

Where $a = \frac{0.45724R^2T_{crit}^2}{P_{crit}}\left(1 + f\omega\left[1 - \left(\frac{T}{T_c}\right)^{1/2}\right]\right)^2$ (2.2.32)

$$\frac{da}{dT} = \frac{0.45724R^2T_{crit}^2}{P_{crit}} 2\left(1 + f\omega\left[1 - \left(\frac{T}{T_c}\right)^{1/2}\right]\right)\left(\frac{1}{2}\left(\frac{1}{TT_c}\right)^{1/2}\right) \quad (2.2.33)$$

Enthalpy departure function:

$$H - H_0 = (A - A_0) + T(S - S_0) + RT(Z-1) \quad (2.2.34)$$

Where $Z = \frac{Pv}{RT} = \frac{v}{RT}\left[\frac{RT}{v-b} - \frac{a}{v^2+2bv-b^2}\right]$ (2.2.35)

Internal energy departure function:

$$U - U_0 = (A - A_0) + T(S - S_0) \quad (2.2.36)$$

Gibbs energy departure function:

$$G - G_0 = (A - A_0) + RT(Z-1) \quad (2.2.37)$$

The fugacity-pressure ratio:

$$ln\frac{f}{P} = \frac{(A - A_0)}{RT} + ln\frac{v}{v_0} + (Z-1) - \ln(Z) \quad (2.2.38)$$

Now let us also look at solution of the inverse form of the cubic equation of states. If equations 2.2.2 to 2.2.5 are investigated, it should be noticed that Coefficients A^* and B^* are function of P and T, therefore solving Z means solving the specific volume, v.

$Z^3 - (1 + B^* - uB^*)Z^2 + (A^* + wB^{*2} - uB^* - uB^{*2})Z - A^*B^* - B^{*2} - wB^{*2} - wB^{*3} = 0$ (2.2.39)

where $A^* = \frac{aP}{R^2T^2}$ (2.2.3) and $B^* = \frac{bP}{RT}$ (2.2.4) $Z = \frac{PV}{RT}$ (2.2.40)

Now to solve the cubic equation. In order to solve cubic equation cubic root solving formulas can be used.

Calculation of the roots of quadratic polynomials are well known. Similar to quadratic polynomials, third degree polynomials can also be solved by using analytical formulations. Roots should be calculated as complex variables.

The basic formulas used to calculate cubic roots are as follows:

$y = a_0 + a_1x + a_2x^2 + a_3x^3$ (2.2.41)

$a = a_2/a_3 \quad b = a_1/a_3 \quad c = a_0/a_3$

$y = c + bx + ax^2 + x^3$

$Q = \frac{a^2-3b}{9} \quad z = 2a^3 - 9ab + 27c \qquad R = z/54$

$if(R^2 < Q^3)$

$$\left\{\begin{array}{c} \theta = cos^{-1}\left(\frac{R}{\sqrt{Q^3}}\right) \\ x_0 = -2\sqrt{Q}cos[\theta/3] - a/3 \\ x_1 = 2\sqrt{Q}cos[(\theta - 2\pi)/3] - a/3 \\ x_2 = 2\sqrt{Q}cos[(\theta - 2\pi)/3] - a/3 \end{array}\right\}$$

$else$

$$\left\{\begin{array}{c} A = -\left(R + \sqrt{R^2 - Q^3}\right)^{1/3} \\ if(a == 0)B = 0 \\ else\ B = Q/A \\ x_0 = (A + B - a/3) \\ x_1 = \left[\left(-\frac{A+B}{2}\right) - a/3\right] + \left[\frac{\sqrt{3(A-B)}}{2}\right]i \\ x_2 = \left[\left(-\frac{A+B}{2}\right) - a/3\right] - \left[\frac{\sqrt{3(A-B)}}{2}\right]i \end{array}\right\} \quad (2.2.42)$$

Now cubic equations can be programmed for utilization

Program 2.2.2 cubicTest.java

```
 import java.io.*;
import javax.swing.*;

class cubicTest
{
 public static void main(String s[]) throws IOException
 {
  //Text cin=new Text();
  String isim;
  //System.out.print("isim = ");
  isim=JOptionPane.showInputDialog("isim = ");
  //cin.readString();
  cubicEquationofState x=new cubicEquationofState(isim,1,"Peng Robinson");
  System.out.println("equation of state name   = "+x.GasEquation);
  System.out.println("gas name   = "+x.Name);
  System.out.println("formula = "+x.Formula);
  System.out.println("Mole Weight  = "+x.MolWt);
  System.out.println("M      = "+x.M);
  System.out.println("Tfp    = "+x.Tfp);
  System.out.println("Tb     = "+x.Tb);
  System.out.println("Tc     = "+x.Tc);
  System.out.println("Pc     = "+x.Pc);
  System.out.println("Vc     = "+x.Vc);
  System.out.println("Zc     = "+x.Zc);
  System.out.println("Omega  = "+x.Omega);
  System.out.println("Dipm   = "+x.Dipm);
  System.out.println("CPVAPA = "+x.CPVAPA);
  System.out.println("CPVAPB = "+x.CPVAPB);
  System.out.println("CPVAPC = "+x.CPVAPC);
  System.out.println("CPVAPD = "+x.CPVAPD);
  System.out.println("DELHF  = "+x.DELHF);
  System.out.println("DELGF  = "+x.DELGF);
  System.out.println("VPEq   = "+x.VPEq);
  System.out.println("VPA    = "+x.VPA);
  System.out.println("VPB    = "+x.VPB);
  System.out.println("VPC    = "+x.VPC);
  System.out.println("VPD    = "+x.VPD);
  System.out.println("Tmin   = "+x.Tmin);
  System.out.println("Tmax   = "+x.Tmax);
  System.out.println("LDEN   = "+x.LDEN);
  System.out.println("TDEN   = "+x.TDEN);
  double T,P;
  //System.out.print("P = ");
  //P=cin.readDouble();
  P=Double.parseDouble(JOptionPane.showInputDialog("P = "));
  //System.out.print("T = ");
  T=Double.parseDouble(JOptionPane.showInputDialog("T = "));
  //T=cin.readDouble();
  System.out.println("V   = "+x.V(T,P));
  System.out.println("Z   = "+x.Z(T,P));
  System.out.println("P   = "+x.P(T,x.V(T,P)));
  //System.out.println("Pvc   = "+x.Pvp(T));
 }
}
```

```
 ---------- Capture Output ----------
> "C:\java\bin\java.exe" cubicTest
equation of state name   = Peng Robinson
gas name   = nitrogene
formula = N2
Mole Weight  = 28.013
M      = 28.01348
Tfp    = 63.3
Tb     = 77.4
```

```
Tc     = 126.2
Pc     = 33.9
Vc     = 89.8
Zc     = 0.29
Omega  = 0.039
Dipm   = 0.0
CPVAPA = 31.15
CPVAPB = -0.01357
CPVAPC = 2.68E-5
CPVAPD = -1.168E-8
DELHF  = 0.0
DELGF  = 0.0
VPEq   = 1
VPA    = -6.09676
VPB    = 1.1367
VPC    = -1.04072
VPD    = -1.93306
Tmin   = 63.0
Tmax   = 126.2
LDEN   = 0.804
TDEN   = 78.0
V   = 3.980219370045505
Z   = 0.957416409897289
P   = 2.0
> Terminated with exit code 0.
```

Let us look at for different cubic equation of states isoterms in P-v diagram. A small test program is prepared for this purpose:

Program 2.2.3 cubicTest1.java: a program to plot isoterms for N2 with van der Walls equation of state

```
import java.io.*;
import javax.swing.*;

class cubicTest1
{
 public static void main(String s[]) throws IOException
 {
   //Text cin=new Text();
   String gasname;
   //System.out.print("gasname = ");
   gasname=JOptionPane.showInputDialog("gasname = ");
   cubicEquationofState x=new cubicEquationofState(gasname,1,"van der Waals");
   double T[]=new double[20];
   double P[]=new double[1000];
   double V[]=new double[1000];
    double tr[]={0.1,0.2,0.3,0.4,0.5,0.6,0.7,0.8,0.9,1.0,1.5,2.0};
    int m=tr.length;
    int i=0;
    int n=1000;
    T[i]=x.Tc*tr[i];
    for(int j=0;j<n;j++)
    {V[j]=0.3/n*j+0.001;;
     P[j]=x.P(T[i],V[j]);
    }
    Plot pp=new Plot(V,P);
    for(i=1;i<m;i++)
      {T[i]=x.Tc*tr[i];
      for(int j=0;j<n;j++)
      {V[j]=0.3/n*j+0.001;
       P[j]=x.P(T[i],V[j]);
      }
    pp.addData(V,P);
      }
```

```
   pp.plot();
  }
 }
```

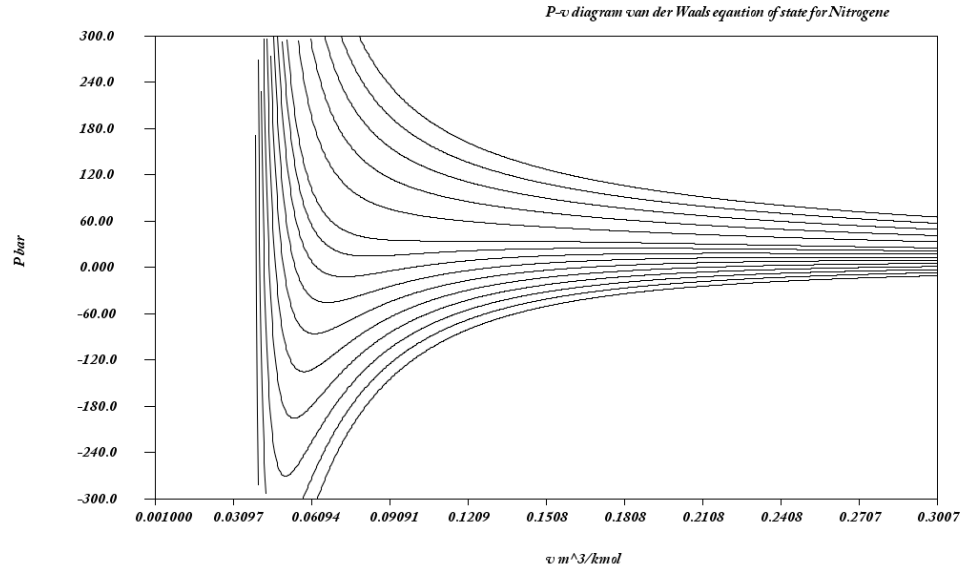


Figure 2.2.1 P-v diagram of van der Waals equation of states, constant temperature lines

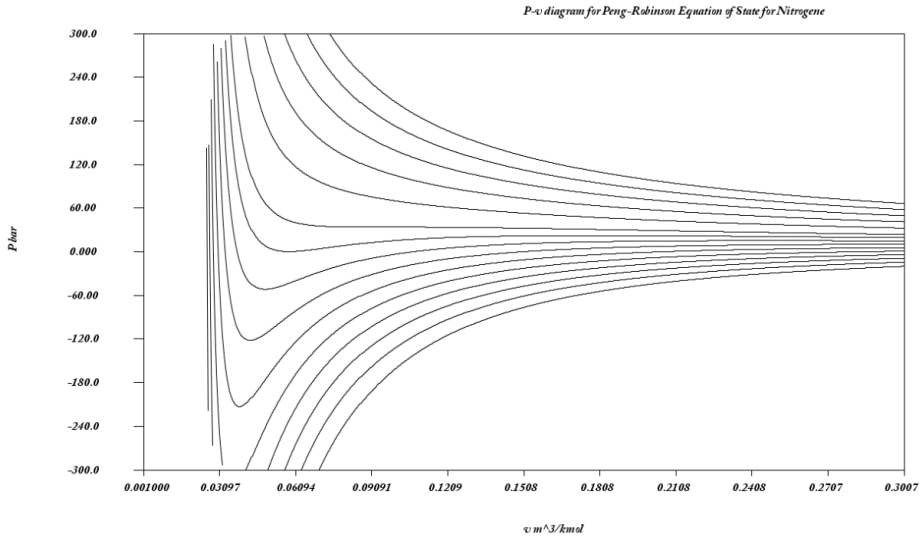


Figure 2.2.2 P-v diagram of van der Waals equation of states, constant temperature lines

One of the most common application of the cubic equation of states, is using them as the first estimate for the liquid and vapor specific volume. If the gas pressure is under the critical pressure, equations will give three real roots. The biggest root is given the specific volume of the gas state, the smallest root is specific volume of the liquid states, even if the result might not be very close to the actual liquid specific volume. And the middle solution does not have any physical meaning. If the pressure is above the critical pressure, equation will be given one real root and two complex roots. Real root is the specific volume of the supercritical fluid.

A gas equation of state is developed by using Peng Robinson equation of state. Equation listed above is used to create this EoS Model(Gas_PR1). Detailed codes are given in appendix. Program utilisation codes are as follows:

```
public static void main(String arg[])
{Gas_PR1 pr=new Gas_PR1("N2");
```

```
  double T=300.0; //degree K
  double P=100.0; //kPA
   pr.print("tp",T,P);
}
```

Properties of N2 nitrogene Peng-Robinson EoS

array no	property name	property	unit
0	P, pressure	100.0	kPA
1	T, temperature	300.0	deg K
2	v, specific volume	24.93274483602...	m³/kmol
3	h, enthalpy	783.5833903494...	KJ/kmol
4	u, internal energy	559.6638294989...	KJ/kmol
5	s, entropy	2.844780548049...	KJ/kmolK
6	x, quality	2.0	kg vap/kg mix
7	Cp, specific heat...	29.12505435085...	KJ/kmolK
8	Cv, specific heat...	20.81059173269...	KJkmolK
9	α isobaric therm...	2.799901669942...	1/K
10	κ isothermal co...	2.799901669942...	1/Pa
11	η Dynamic visco...	1.787600318502...	Pa.s
12	k Thermal condu...	0.027666480768...	W/mK
13	σ Surface tension	0.0	N/m
14	Prandtl number	6.717746743016...	
15	ρ density	0.040107898531...	kmol/m³
16	speed of sound	353.0244735640...	m/s

Figure 2.2.3 A GUI output of a sample gas N_2

You can obtain all properties as an array by using property method. Property method accepts two known thermodynamic property set. Known property sets are as :

Known set	meaning	Known set	meaning
tp	Temperature Pressure	pt	Pressure temperature
vt	Specific volume Temperature	tv	Temperature Specific volume
rot	Density temperature	tro	Temperature density
pv	Pressure Specific volume	vp	Specific volume pressure
pro	Pressure density	rop	Density pressure
ph	Pressure enthalpy	Hp	Enthalpy Pressure
ps	Pressure entropy	sp	Entropy pressure

In the following example property pair “tp” is used.

```
public static void main(String arg[])
{Gas_PR1 pr=new Gas_PR1("N2");
  double T=300.0; //degree K
  double P=100.0; //kPA
  double a[]=pr.property("tp",T,P);
  for( double x :a) System.out.println(x);
}
```

```
---------- Capture Output ----------
> "E:\co\java\bin\javaw.exe" Gas_PR1
100.0
300.0
24.93274483602078
783.5833903494341
559.6638294989782
2.844780548049795
2.0
29.12505435085314
20.810591732699898
2.7999016699420404E-4
2.7999016699420404E-4
1.787600318502586E-5
0.027666480768509534
```

0.0 6.717746743016922E-4 0.04010789853170447 353.0244735640423

2.3 LEE-KESLER & SCHREİBER- PİTZER EQUATION OF STATES

Benedict-Webb-Rubin (BWR) equation of state is one of the most successful equation of state. We will look at utilisation of this equation of state in later stages. The success of the original BWR equation of state has led to a number of studies wherin the modified form of the equation has been generalised to apply to many types of compounds.



Byung IK Lee



Michael G. Kesler

Lee and Kesler developed a modified form of BWR[75]. This equation of state is developed based on two reference fluids, the first one is called simple fluid (accentricity factor ω =0) and those of n-octane as a reference fluid. Asume that $Z = \frac{PV}{T}$ compressibility factor is to be calculated for a fluid at some temperature and pressure. First using critical properties of this fluid, determine $T_r = \frac{T}{T_{crit}}$ and $P_r = \frac{P}{P_{crit}}$ reduced temperature and pressure. Then determine an ideal reduced volume of a simple fluid with

$$Z^{(0)} = \frac{P_r V_r^{(0)}}{T_r} = 1 + \frac{B}{V_r^{(0)}} + \frac{C}{\left(V_r^{(0)}\right)^2} + \frac{D}{\left(V_r^{(0)}\right)^5} + \frac{c_4}{T_r^3\left(V_r^{(0)}\right)^2}\left[\beta + \frac{\gamma}{\left(V_r^{(0)}\right)^2}\right] exp\left[-\frac{\gamma}{\left(V_r^{(0)}\right)^2}\right] \quad (2.3.1)$$

Where

$$B = b_1 - \frac{b_2}{T_r} - \frac{b_3}{T_r^2} - \frac{b_4}{T_r^3} \quad (2.3.2)$$

$$C = c_1 - \frac{c_2}{T_r} + \frac{c_3}{T_r^3} \quad (2.3.3)$$

$$D = d_1 - \frac{d_2}{T_r} \quad (2.3.4)$$

$$V_r^{(0)} = \frac{P_c V^{(0)}}{R T_c} \quad (2.35)$$

Properties of simple and reference fluid(n-octane) is given in table 2.3.1

Tablo 2.3.1 Lee Kesler coefficients for simple and reference fluids

Constants	Simple fluid	Referance fluid(n-octane)
b_1	0.1181193	0.2026579
b_2	0.265728	0.331511
b_3	0.154790	0.027655
b_4	0.030323	0.203488
c_1	0.0236744	0.0313385
c_2	0.0186984	0.0503618
c_3	0.0	0.016901

c_4	0.042724	0.041577
d_1 x 10^4	0.155488	0.48736
d_2 x 10^4	0.623689	0.0740336
β	0.65392	1.226
γ	0.060167	0.03754

Next, using the same reduced temperature and pressure

$$V_r^{(R)} = \frac{P_c V^{(R)}}{RT_c} \quad (2.3.6)$$

for reference fluid is solved by using the same basic Lee-Kesler equation

$$Z^{(R)} = \frac{P_r V_r^{(R)}}{T_r} = 1 + \frac{B}{V_r^{(R)}} + \frac{C}{\left(V_r^{(R)}\right)^2} + \frac{D}{\left(V_r^{(R)}\right)^5} + \frac{c_4}{T_r^3\left(V_r^{(R)}\right)^2}\left[\beta + \frac{\gamma}{\left(V_r^{(R)}\right)^2}\right] exp\left[-\frac{\gamma}{\left(V_r^{(R)}\right)^2}\right] \quad (2.3.7)$$

The compressibility factor Z for the fluid of interest is then calculated by

$$Z = \frac{PV}{RT} = Z^{(0)} + \left(\frac{\omega}{\omega_R}\right)\left(Z^{(R)} - Z^{(0)}\right) \quad (2.3.8)$$

$$Z^{(1)} = \frac{1}{\omega_R}\left(Z^{(R)} - Z^{(0)}\right) \quad (2.3.8a)$$

$$Z = \frac{PV}{RT} = Z^{(0)} + \omega Z^{(1)} \quad (2.3.8b)$$

Where accentricity factor $\omega_R = 0.3978$ and $\omega_0 = 0.0$. In hydrocarbon calculations primary error of the equation is in the range of less than 2% for both vapor and liquid phases.

If T and V values are given, finding of P is as defined above, but if T and P is given and finding of V is required, then a numerical root solving methods should apply to get a solution of the equation. This equation has a capacity to predict both liquid and gas phases, but in root solving of V it causes a problem of initial guess estimation due to big differences between liquid and gas values of specific volumes. Additional search methods combining with simpler equation of states volume estimation such as Peng-Robinson equation of states should be involve for an accurate estimation process. It should be note that while one real root of cubic equation of states given liquid volume, the second one will give the volume of the gas state.

Prediction of the rest of the thermodynamic properties by Lee-Kesler method

Equation of states gives us prediction capability of unknown variable when two of the three T , v , P thermodynamic properties are given. But in thermal design further thermodynamic properties may also be required. Enthapy of a real gas is function of pressure and temperature and can be expressed as follows:

$$H = f(T, P) \quad (2.9)$$

$$dH = \left(\frac{\partial H}{\partial P}\right)_T dP + \left(\frac{\partial H}{\partial T}\right)_P dT \quad (2.3.9)$$

$$H_2 - H_1 = \int_{P_1}^{P_2} \left(\frac{\partial H}{\partial P}\right)_T dP + \int_{T_1}^{T_2} \left(\frac{\partial H}{\partial T}\right)_P dT \quad (2.3.10)$$

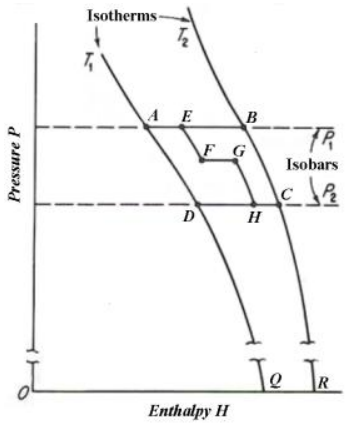


Figure 2.3.1 Enthalpy change calculation process

In here, temperature kept constant at T_1 and enthalpy change for pressure difference calculated between some very low reference pressure p^0 and P_1, then entalpy difference at p^0 pressure is calculated. Since the reference pressure is low, ideal gas enthalpy difference can be applied between temperatures T_1 and T_2 to calculate entalpy change, and then temperature is kept constant at T_2 and enthalpy change for pressure difference is calculated. Process is shown in the figure 2.3.1

$$\Delta H = \int_{P_1}^{P_0} \left(\frac{\partial H(T,P)}{\partial P}\right)_{T_1} dP + \int_{T_1}^{T_2} C_p(T)\, dT + \int_{P_0}^{P_1} \left(\frac{\partial H(T,P)}{\partial P}\right)_{T_2} dP \quad (2.3.11)$$

$$\Delta H = (H^0 - H_{P_1})_{T_1} + \int_{T_1}^{T_2} C_p(T)\, dT - \left(H^0 - H_{P_2}\right)_{T_2} \quad (2.3.12)$$

In equation (2.3.12) the terms $(H^0 - H_{P_1})_{T_1}$ and $\left(H^0 - H_{P_2}\right)_{T_2}$ functions indicated enthalpy changes at constant temperatures T_1 and T_2 between reference and working pressures. These type of functions are called departure functions, and since they are only function of pressure, they can be listed as tables as well. They indicated departure of the enthalpy from ideal gas state.

Other thermodynamic departure functions

Similar to entalpy functions, departure functions can be calculated for other thermodynamic functions as well for a defined low pressure reference state.

$$G - G^0 = \int_{P^0}^{P} V dP + \int_0^P V dP = \int_0^P \left(V - \frac{RT}{P}\right) dP + RTln\left(\frac{P}{P_0}\right) = RT\int_0^P (Z-1) dln(P) + +RTln\left(\frac{P}{P_0}\right)$$

(2.3.13)

$$S - S^0 = -\frac{\partial (G - G^0)_P}{\partial T} = R\int_0^P \left[1 - Z - T\left(\frac{\partial Z}{\partial T}\right)_P\right] dlnP - Rln\left(\frac{P}{P_0}\right) \quad (2.3.14)$$

In this equations Z, T and P functions are defined by Lee-Kesler equation, after calculating Gibbs free energy and entropy, the remaining thermodynamic properties can be easily defined from these relations

$$H - H^0 = (A - A^0) + T(S - S^0) + RT(Z-1) \quad (2.3.15)$$

$$U - U^0 = (A - A^0) + T(S - S^0) \quad (2.3.18)$$

And finaly departure function of Helmholts free energy will be :

$$(A - A^0) = G - G^0 - RT(Z-1) \quad (2.3.19)$$

If Lee-Kesler equation of state is substituted into the above equations, the equations for simple fluid will take the following forms:

$$\left[\frac{H^0 - H}{RT_c}\right]^{(0)} = -T_r\left[Z^{(0)} - 1 - \frac{b_2 + \frac{2b_3}{T_r} + \frac{3b_4}{T_r^2}}{T_r\left(V_r^{(0)}\right)} - \frac{c_2 - \frac{3c_3}{T_r^2}}{2T_r\left(V_r^{(0)}\right)^2} + \frac{d_2}{5T_r\left(V_r^{(0)}\right)^5} + 3E\right]$$

$$E = \frac{c_4}{2T_r^3\gamma}\left\{\beta + 1 - \left[\beta + 1 + \frac{\gamma}{\left(V_r^{(0)}\right)^2}\right] exp\left[-\frac{\gamma}{\left(V_r^{(0)}\right)^2}\right]\right\} \quad (2.3.20)$$

$$\left[\frac{S^0 - S}{R}\right]^{(0)} = -ln\frac{P_0}{P} - lnS^{(0)} + \frac{b_1 + \frac{b_3}{T_r^2} + \frac{32}{T_r^3}}{\left(V_r^{(0)}\right)} + \frac{c_1 - \frac{2c_3}{T_r^3}}{2\left(V_r^{(0)}\right)^2} + \frac{d_1}{5\left(V_r^{(0)}\right)^5} - 2E$$

When reference state and simple fluid state concept of Lee-Kesler equation is applied, the actual departure function is found as:

$$\left[\frac{H^0-H}{RT_c}\right] = \left[\frac{H^0-H}{RT_c}\right]^{(0)} + \frac{\omega}{\omega_R}\left[\left[\frac{H^0-H}{RT_c}\right]^{(R)} - \left[\frac{H^0-H}{RT_c}\right]^{(0)}\right] \quad (2.3.21)$$

Or by defining

$$\left[\frac{H^0 - H}{RT_c}\right]^{(1)} = \frac{1}{\omega_R}\left[\left[\frac{H^0 - H}{RT_c}\right]^{(R)} - \left[\frac{H^0 - H}{RT_c}\right]^{(0)}\right] \quad (2.3.22)$$

$$\left[\frac{H^0 - H}{RT_c}\right] = \left[\frac{H^0 - H}{RT_c}\right]^{(0)} + \omega\left[\frac{H^0 - H}{RT_c}\right]^{(1)} \quad (2.3.23)$$

Similarly for entropy function;

$$\left[\frac{S^0 - S}{RT_c}\right] = \left[\frac{S^0 - S}{RT_c}\right]^{(0)} + \frac{\omega}{\omega_R}\left[\left[\frac{S^0 - S}{RT_c}\right]^{(R)} - \left[\frac{S^0 - S}{RT_c}\right]^{(0)}\right] \quad (2.3.24)$$

$$\left[\frac{S^0 - S}{RT_c}\right]^{(1)} = \frac{1}{\omega_R}\left[\left[\frac{S^0 - S}{RT_c}\right]^{(R)} - \left[\frac{S^0 - S}{RT_c}\right]^{(0)}\right] \quad (2.3.25)$$

$$\left[\frac{S^0 - S}{RT_c}\right] = \left[\frac{S^0 - S}{RT_c}\right]^{(0)} + \omega\left[\frac{S^0 - S}{RT_c}\right]^{(1)} \quad (2.3.26)$$

Program listing of Lee Kesler equation is given below. In an example program Z compressibility factor is calculated and plotted by using this program.

Program 2.3.1 A test program to run LeeKesler.java: a program for the general real gas equation of state, creates compressibility factor charts

```
 import java.io.*;
import javax.swing.*;

class LeeKeslerTest5
{
 public static void main(String s[]) throws IOException
 {
  //Text cin=new Text();
  String gasname;
  //System.out.print("gasname = ");
  //gasname=JOptionPane.showInputDialog("gasname = ");
  LeeKesler x=new LeeKesler("H2",1);
  //cubicEquationofState x=new cubicEquationofState(gasname,1,"Peng Robinson");
   System.out.println("equation of state name   = "+x.GasEquation);
  System.out.println("gas name   = "+x.Name);
  double T[]=new double[20];
  double P[]=new double[1000];
  double V[]=new double[1000];
  double Tr[]=new double[20];
  double Pr[]=new double[1000];
  double Vr[]=new double[1000];
  double Z[]=new double[1000];
```

```
    double pp1=0;
    double R=8314.5e-5;
     double tr[]={1.0,1.05,1.10,1.15,1.2,1.3,1.4,1.5,1.6,1.8,2.0,2.5,3.0,3.5};
     int m=tr.length;
     int i=0;
     int n=1000;
     T[i]=x.Tc*tr[0];
     Tr[i]=T[i]/x.Tc;
     //System.out.println( T[i]);
       for(int j=0;j<n;j++)
       {  pp1=1.0/100*j+0.01;
                  P[j]=pp1*x.Pc;
                  Pr[j]=pp1;
                  //System.out.println( P[j]);
        V[j]=x.V_TP_LK_V(T[i],P[j]);
        Z[j]=P[j]*V[j]/(R*T[i]);
        Vr[j]=V[j]/x.Vc;
        System.out.println( Z[j]+" "+Tr[i]+" "+Pr[j]+" "+V[j]);
        }
     Plot pp=new Plot(Pr,Z);
       //pp.addData(Vr,Z,1,0,0,255);
     for(i=1;i<m;i++)
       {T[i]=x.Tc*tr[i];
        Tr[i]=T[i]/x.Tc;
       for(int j=0;j<n;j++)
       {  pp1=1.0/100*j+0.01;
                  P[j]=pp1*x.Pc;
                  Pr[j]=P[j]/x.Pc;
                  //System.out.println( P[j]);
        V[j]=x.V_TP_LK_V(T[i],P[j]);
        Vr[j]=V[j]/x.Vc;
        Z[j]=P[j]*V[j]/(R*T[i]);
        System.out.println( Z[j]+" "+Tr[i]+" "+Pr[j]+" "+Vr[j]);
        }
       pp.addData(Pr,Z);
       //pp.addData(Vr,Z,1,0,0,255);
       }
    pp.plot();
  }
 }
```

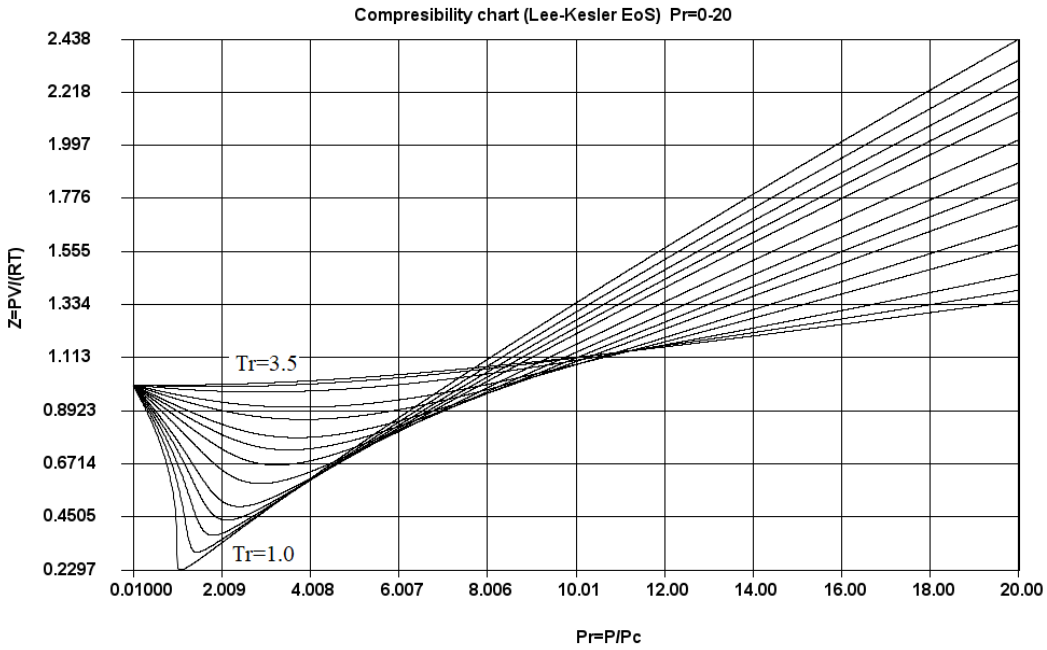


Figure 2.3.2 Compressibilty chart created by Lee Kesler equation of state Pr=0-20

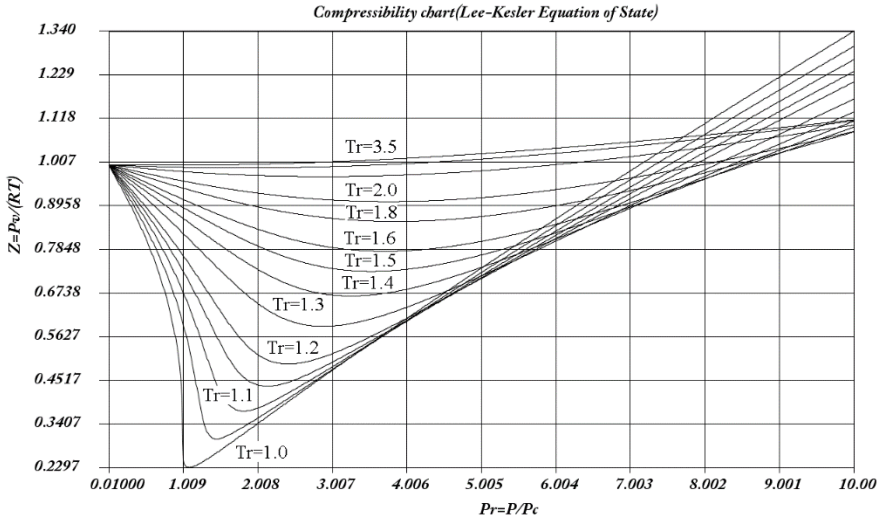


Figure 2.3.3 Compressibilty chart created by Lee Kesler equation of state Pr=0-10

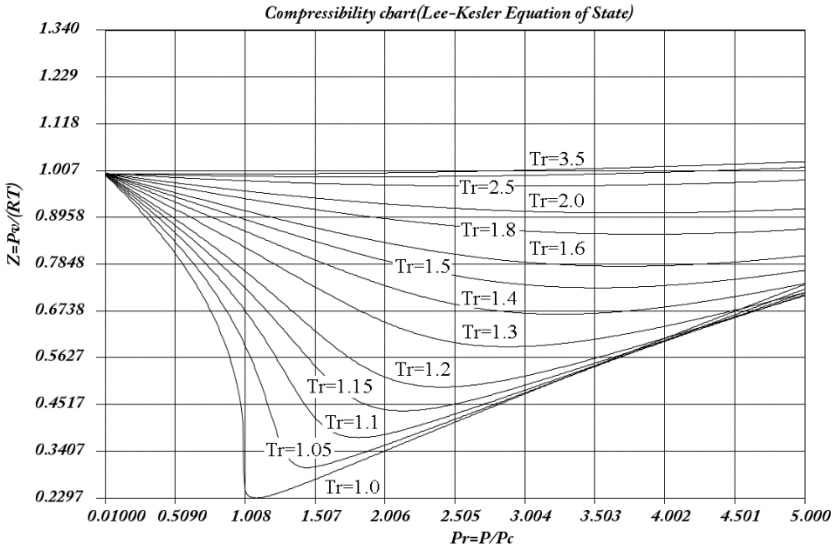


Figure 2.3.4 Compressibilty chart created by Lee Kesler equation of state Pr=0-5

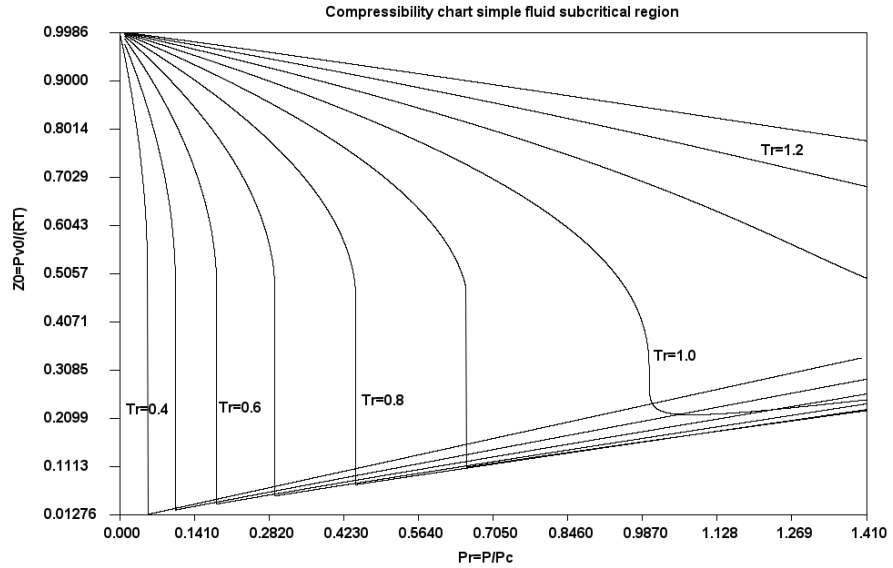


Figure 2.3.5 Compressibilty chart, simple fluid (Lee Kesler EoS) with subcritical region

We should note the phase change behaivor in Z0 when it is in subcritical region. The prediction of liquid densities (specific volumes) in this region could be a valuble tool, and can be utilised as first estimation in more accurate EoS.

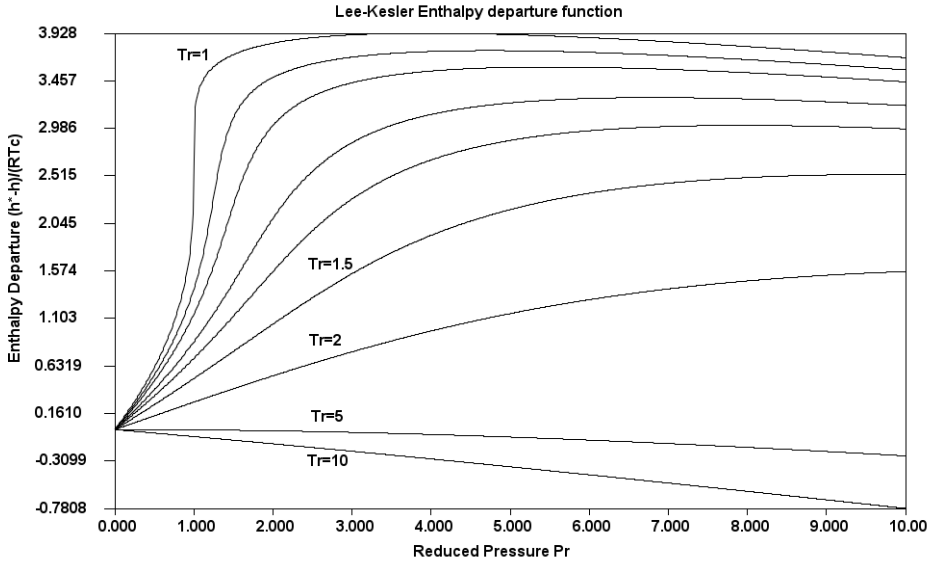


Figure 2.3.6 Enthalpy departure function chart created by Lee Kesler equation of state, supercritical region

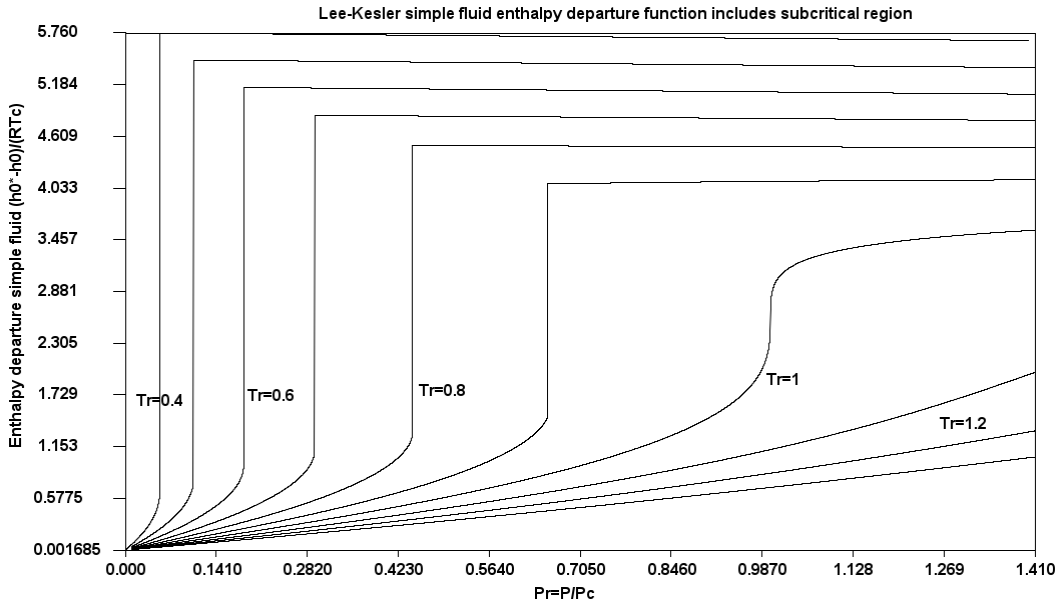


Figure 2.3.7 Enthalpy departure function of simple fluid (Lee-Kesler EoS) includes subcritical phase-change region and supercritical region Pr=0-1.4

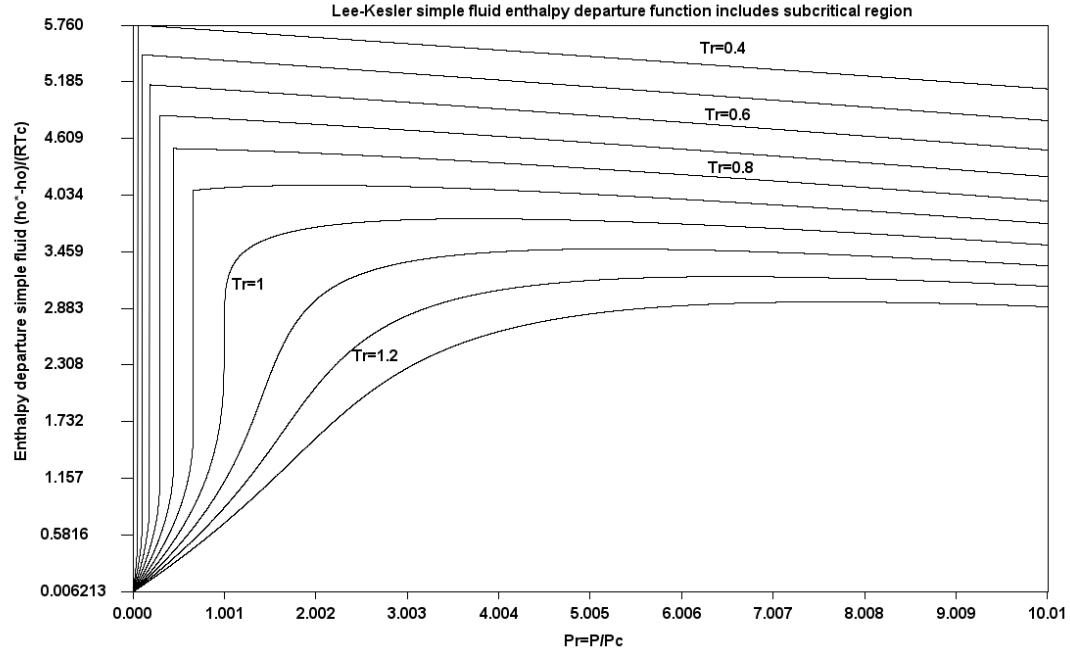


Figure 2.3.8 Enthalpy departure function of simple fluid (Lee-Kesler EoS) includes subcritical phase-change region and supercritical region Pr=0-10

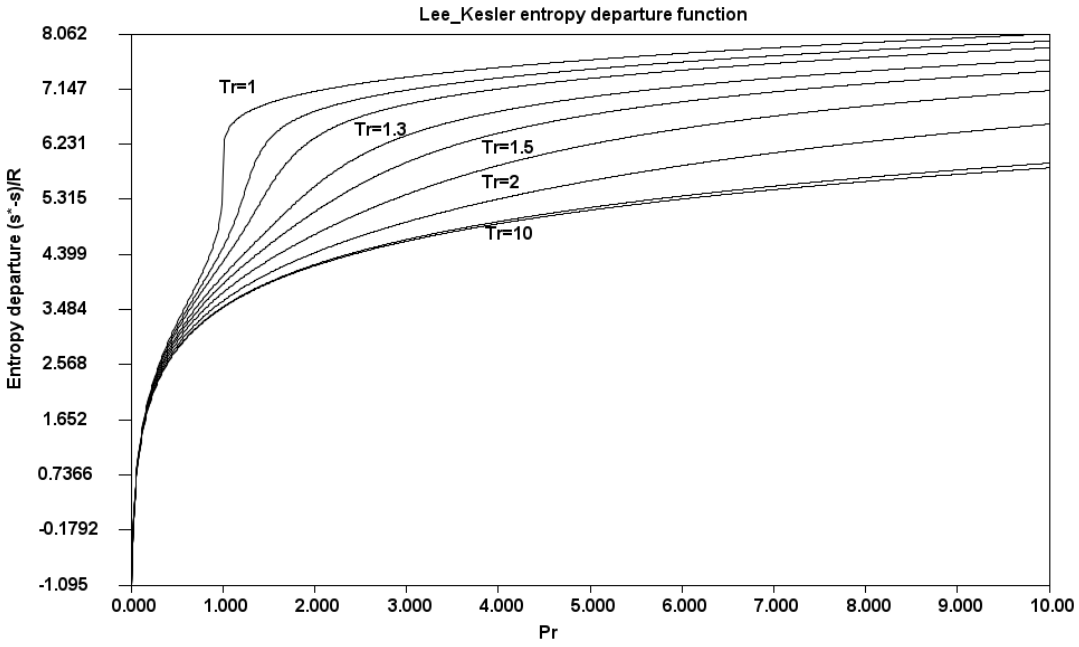


Figure 2.3.9 Entropy departure function chart created by Lee Kesler equation of state supercritical region

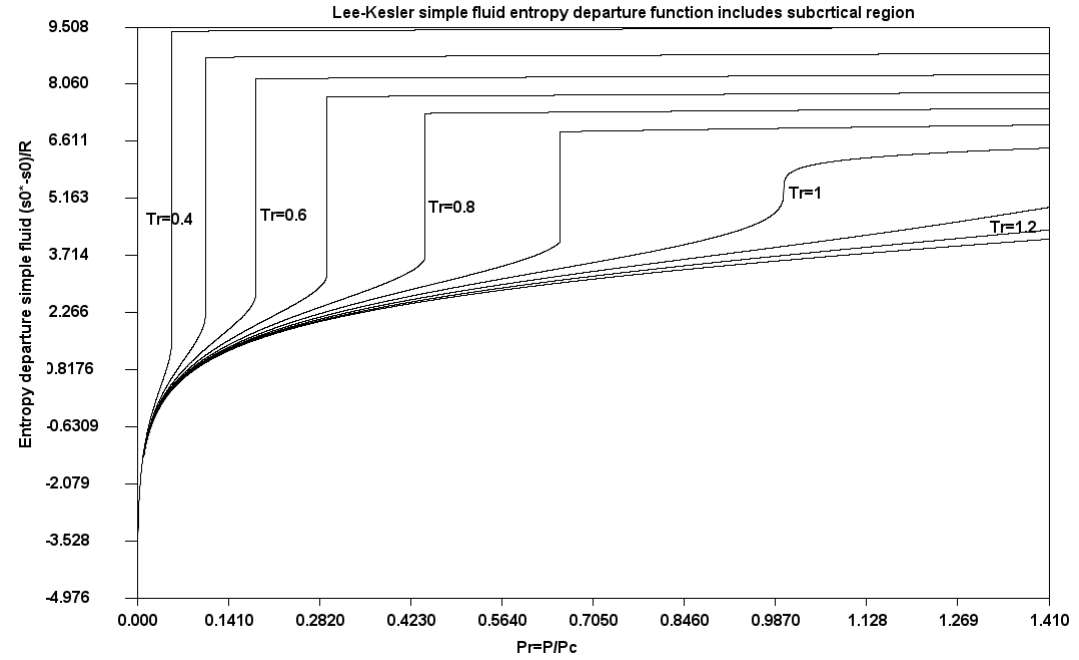


Figure 2.3.10 Entropy departure function chart created by Lee Kesler equation of state includes subcritical phase-change region and supercritical region Pr=0-1.4

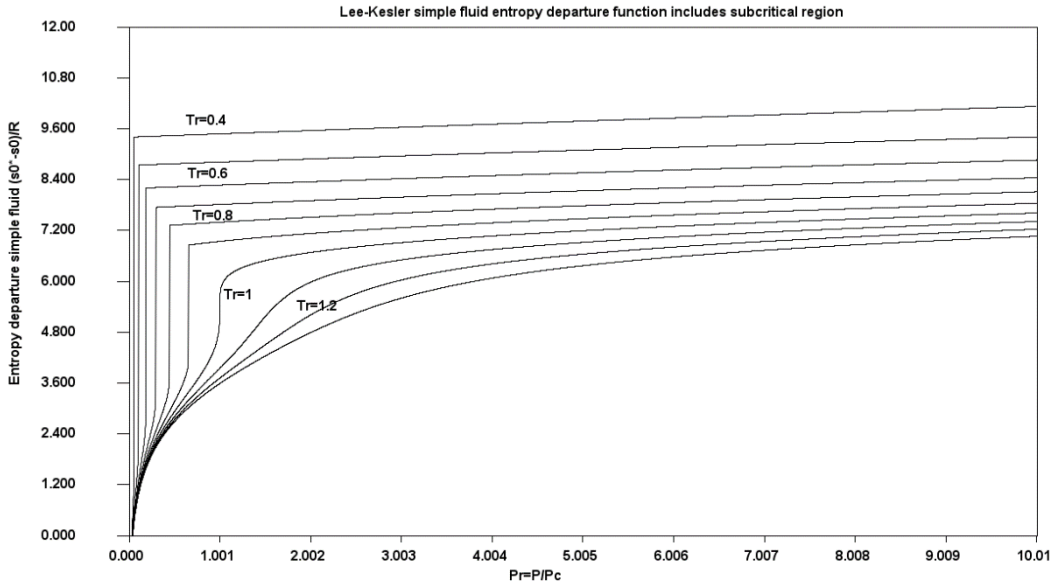


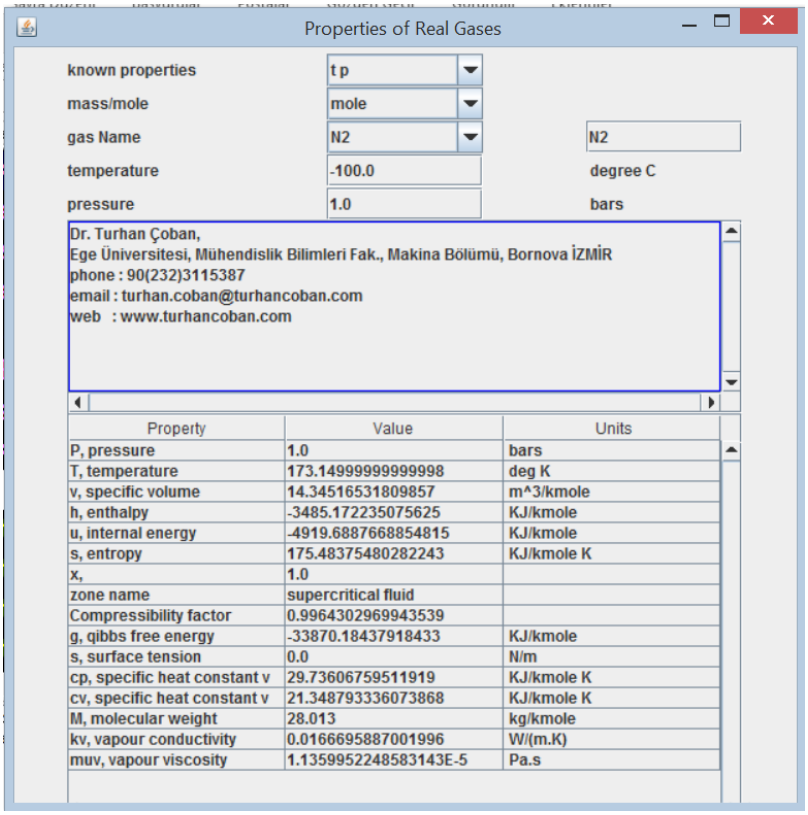
Figure 2.3.11 Entropy departure function chart created by Lee Kesler equation of state includes subcritical phase-change region and supercritical region Pr=0-10

Some 618 different species are entered as data to Lee Kesler equation of state, so as a general form of equation of state, it can handle quite a few species. A table form of user interface program is also created for general utilisation of the package. Program code is given in appendix A3 and www.turhancoban.com.

Program List of Lee-Kesler EOS:

LeeKesler	Basic Equation of state
LeeKeslerMix	Basic equation of state for mixtures
LKTable	Graphic user Interface for LeeKesler EOS
LKMixTable	Graphic user Interface for LeeKesler Mix EOS
LKModel	GUI subprogram
LKMixModel	GUI subprogram

Program graphic user interface (GUI) output looks like



Property	Value	Units
P, pressure	1.0	bars
T, temperature	173.14999999999998	deg K
v, specific volume	14.34516531809857	m^3/kmole
h, enthalpy	-3485.172235075625	KJ/kmole
u, internal energy	-4919.6887668854815	KJ/kmole
s, entropy	175.48375480282243	KJ/kmole K
x,	1.0	
zone name	supercritical fluid	
Compressibility factor	0.9964302969943539	
g, qibbs free energy	-33870.18437918433	KJ/kmole
s, surface tension	0.0	N/m
cp, specific heat constant v	29.73606759511919	KJ/kmole K
cv, specific heat constant v	21.348793336073868	KJ/kmole K
M, molecular weight	28.013	kg/kmole
kv, vapour conductivity	0.0166695887001996	W/(m.K)
muv, vapour viscosity	1.1359952248583143E-5	Pa.s

Figure 2.3.12 LeeKeslerTable user interface output

A Gas equation by using Lee-Kesler EoS is developed to calculate Gas phase of several chemicals. Full program codes are given in appendix and www.turhancoban.com internet site. A small sample code is given below to show calling of thermodynamic and thermophysical properties for a given state.

```
public static void main(String arg[])
{ Gas_LK lk=new Gas_LK("N2");
lk.mole=false;
double T=300;  //Degree K
double P=100.0; //kPa
//double b[]=lk.property("tp",T,P);
 //Text.printT(b);
 lk.print("tp",T,P);
}}
```

Properties of pure gas Lee-Kesler EOS GasName: nitrogene N2

array no	property name	gasproperty	unit
0	P, pressure	100.0	kPA
1	T, temperature	300.0	deg K
2	v, specific volume	0.89023861655846...	m³/kg
3	h, enthalpy	27.96186719687929	KJ/kg
4	u, internal energy	19.9726630400349...	KJ/kg
5	s, entropy	0.10154977141673...	KJ/kgK
6	x, quality	2.0	kg vap/kg mix
7	Cp, specific heat at...	1.03969779569675...	KJ/kgK
8	Cv, specific heat at...	0.74289050557597...	KJ/kgK
9	α isobaric thermal ...	9.97864279047862...	1/K
10	κ isothermal compr...	9.97864279047862...	1/Pa
11	η Dynamic viscosity	1.78760031850258...	Pa.s
12	k Thermal conduct...	0.02766648076850...	W/mK
13	σ Surface tension	0.0	N/m
14	Prandtl number	6.71774674301692...	
15	ρ density	1.12329434086543...	kg/m³
16	speed of sound	353.3528789034441	m/s

Figure 2.3.13 A GUI output of a sample gas N_2

You can obtain all properties as an array by using property method. Property method accepts two known thermodynamic property set. Known property sets are as :

Known set	meaning	Known set	meaning
tp	Temperature Pressure	pt	Pressure temperature
vt	Specific volume Temperature	tv	Temperature Specific volume
rot	Density temperature	tro	Temperature density
pv	Pressure Specific volume	vp	Specific volume pressure
pro	Pressure density	rop	Density pressure
ph	Pressure enthalpy	Hp	Enthalpy Pressure
ps	Pressure entropy	sp	Entropy pressure

In the following example property pair “tp” is used.

```
  public static void main(String arg[])
{Gas_LK lk=new Gas_LK("N2");
  double T=300.0; //degree K
  double P=100.0; //kPA
  double a[]=lk.property("tp",T,P);
  for( double x :a) System.out.println(x);
}
```

```
 ---------- Capture Output ----------
> "E:\co\java\bin\javaw.exe" Gas_LK
100.0
300.0
24.938254365652387
783.2957857861796
559.4942097404995
2.844713746696977
2.0
29.12505435085314
20.810591732699898
2.800530395352282E-4
2.800530395352282E-4
1.787600318502586E-5
0.027666480768509534
0.0
6.717746743016922E-4
0.04009903762058447
353.0238423174664

> Terminated with exit code 0.
```

Comparison of Gas EoS are shown in the table below.

Table 4.2 Comparisons of 3 different EOS for Nitrogen.

Class	P	T K	v m3/kg	h kJ/kg	u kJ/kg	s kJ/kgK
Peng-Robinson_EoS	100	300	0.8900419	27.972134	19.978718	0.1015522
Lee-Kesler_EoS	100	300	0.8902386	27.961867	19.972663	0.1015498
Perfect_Gas_EoS	100	300	0.8904219	27.91124	19.941965	0.1013739
Peng-Robinson_EoS	500	300	0.1777167	26.896945	19.049385	-0.3780739
Lee-Kesler_EoS	500	300	0.1779105	27.11667	19.196101	-0.3787223
Perfect_Gas_EoS	500	300	0.1780844	27.91124	19.941965	-0.376319
Peng-Robinson_EoS	1000	300	0.0886897	25.572702	17.893751	-0.5861674
Lee-Kesler_EoS	1000	300	0.08888	26.068717	18.223375	-0.5876666
Perfect_Gas_EoS	1000	300	0.0890422	27.91124	19.941965	-0.5820501
Peng-Robinson_EoS	3000	300	0.0293885	20.500848	13.346164	-0.921081
Lee-Kesler_EoS	3000	300	0.029566	21.980869	14.317412	-0.9264455
Perfect_Gas_EoS	3000	300	0.0296807	27.91124	19.941965	-0.9081263

Peng-Robinson_EoS	5000	300	0.0175748	15.799574	8.9361282	-1.0805435
Lee-Kesler_EoS	5000	300	0.0177418	18.080752	10.406455	-1.090458
Perfect_Gas_EoS	5000	300	0.0178084	27.91124	19.941965	-1.059743
Peng-Robinson_EoS	100	500	1.484483	237.25911	169.82159	0.6356745
Lee-Kesler_EoS	100	500	1.4845669	237.20416	169.78213	0.6356683
Perfect_Gas_EoS	100	500	1.4840365	236.98671	169.65598	0.6350433
Peng-Robinson_EoS	500	500	0.2972572	237.02473	169.40693	0.157842
Lee-Kesler_EoS	500	500	0.2973432	237.02567	169.38874	0.1571919
Perfect_Gas_EoS	500	500	0.2968073	236.98671	169.65598	0.1573504
Peng-Robinson_EoS	1000	500	0.1488576	236.73955	168.89269	-0.048052
Lee-Kesler_EoS	1000	500	0.1489463	236.80956	168.89795	-0.0495107
Perfect_Gas_EoS	1000	500	0.1484036	236.98671	169.65598	-0.0483808
Peng-Robinson_EoS	3000	500	0.0499378	235.68321	166.88054	-0.3746543
Lee-Kesler_EoS	3000	500	0.0500367	236.02183	166.94628	-0.3793857
Perfect_Gas_EoS	3000	500	0.0494679	236.98671	169.65598	-0.3744569
Peng-Robinson_EoS	5000	500	0.0301657	234.7561	164.93829	-0.5266081
Lee-Kesler_EoS	5000	500	0.0302746	235.354	165.0158	-0.534664
Perfect_Gas_EoS	5000	500	0.0296807	236.98671	169.65598	-0.5260737

Comparisons of thermodynamic properties in graphic form for gas nitrogen is listed in below figures. By using equations given above a class called Gas_LK is developed. The result of this class is compared with results of Perfect gas EoS and Peng-Robinson EoS.

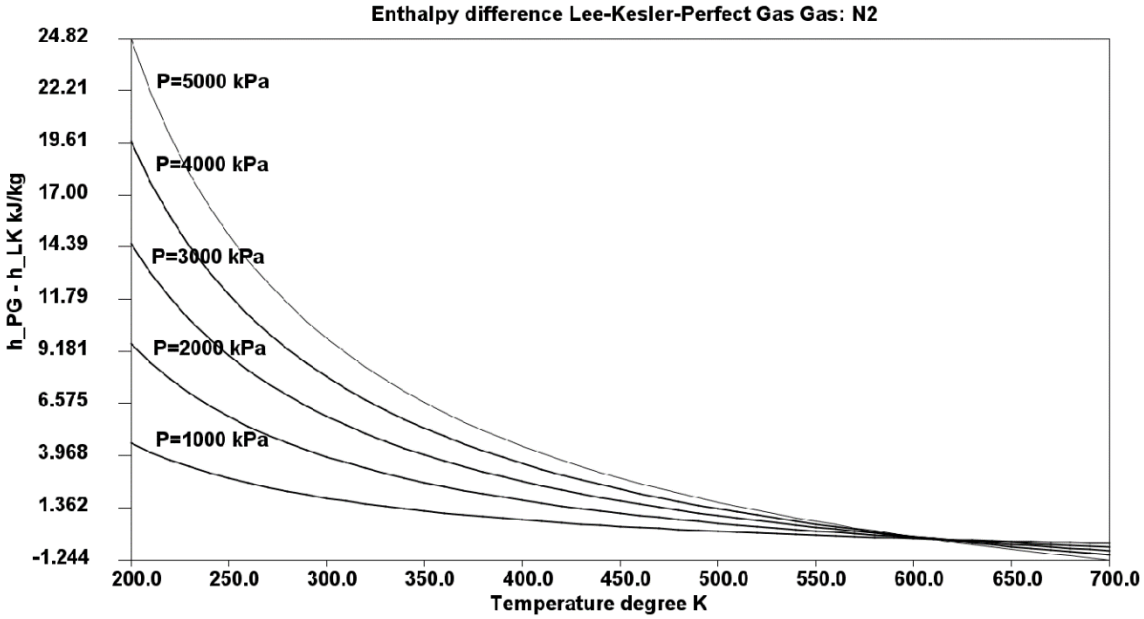


Figure 2.3.14 Enthalpy difference of Lee-Kesler and perfect gas EOS

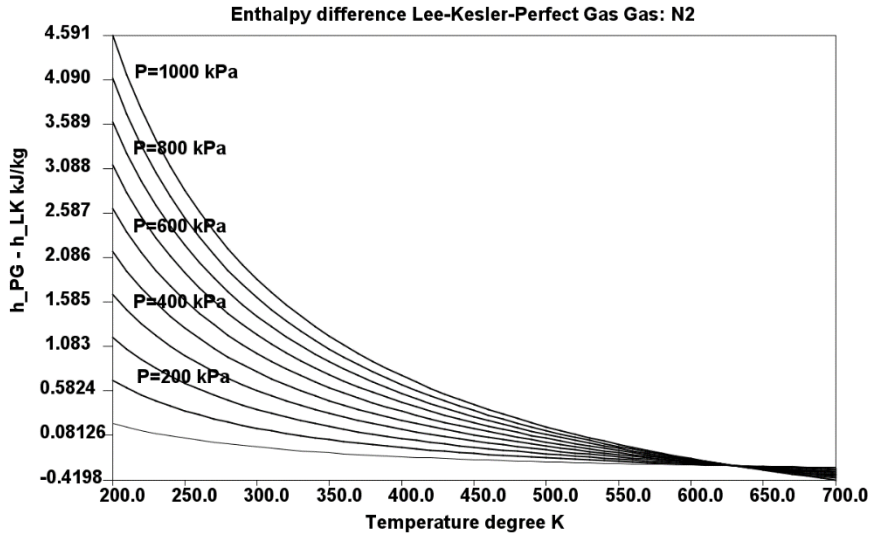


Figure 2.3.15 Enthalpy difference of Lee-Kesler and perfect gas EOS

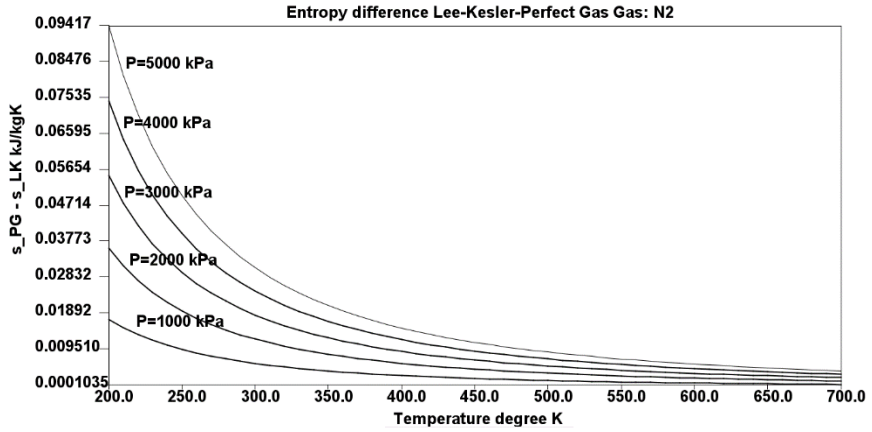


Figure 2.3.16 Entropy difference of Lee-Kesler and perfect gas EOS

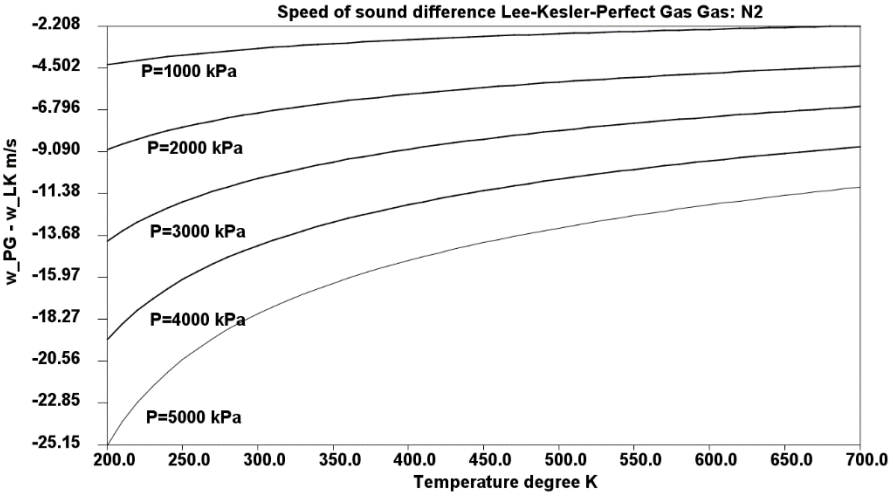


Figure 2.3.17 Speed of sound difference of Lee-Kesler and perfect gas EOS

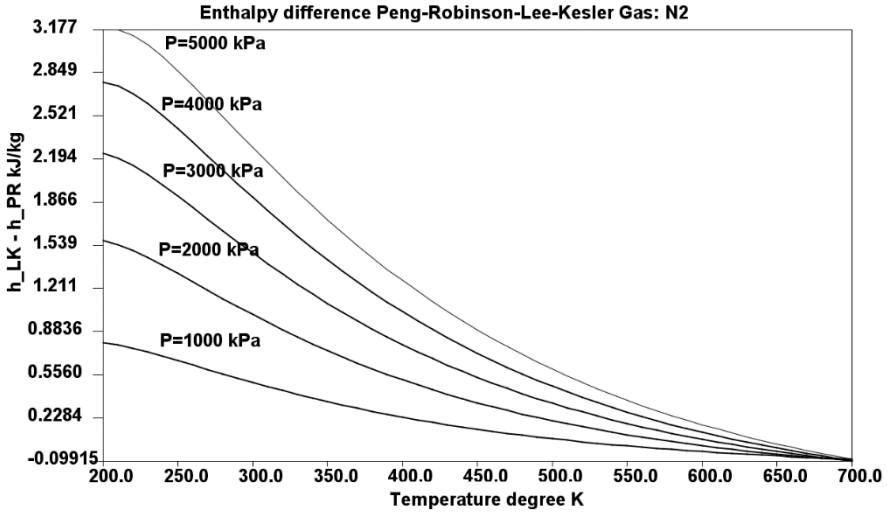


Figure 2.3.18 Enthalpy difference of Peng Robinson and Lee-Kesler EOS

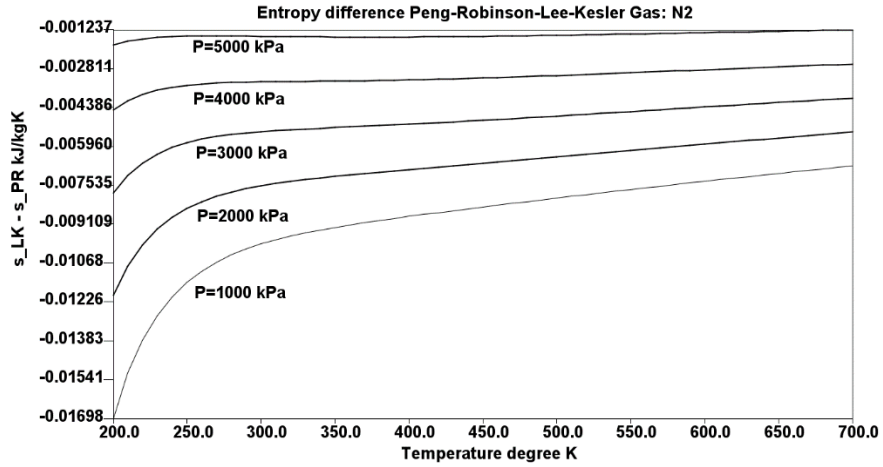


Figure 2.3.19 Entropy difference of Peng Robinson and Lee-Kesler EOS

Another derivation based on Benedict-Webb-Rubin (BWR) equation of state is Schreiber-Pitzer equation of state[82]. Details of the Scheiber-Pitzer EoS is given below.



Donald R. Schreiber



Kenneth Sanborn Pitzer

$$Z = \frac{P}{\rho RT} = 1 + B(T_r, \rho_r)\rho_r + C(T_r, \rho_r)\rho_r^2 + D(T_r, \rho_r)\rho_r^3 + E(T_r, \rho_r)\rho_r^5 + F(T_r, \rho_r)\rho_r^7 + G(T_r, \rho_r)\rho_r^8 + H(T_r, \rho_r)\rho_r^{10} + I(T_r, \rho_r)\rho_r^{12} \quad (2.3.27a)$$

$$B(T_r, \rho_r) = c_1 + \frac{c_2}{T_r} + \frac{c_3}{T_r^2} + \frac{c_4}{T_r^6} \quad (2.3.27b)$$

$$C(T_r,\rho_r) = c_5 + \frac{c_5}{T_r} + \frac{c_7}{T_r^3} + \frac{c_8}{T_r^4}\exp(-\rho_r^2) = C_1(T_r) + C_2(T_r)\exp(-\rho_r^2) \quad (2.3.27b)$$

$$D(T_r,\rho_r) = c_9 + \frac{c_{10}}{T_r} + \frac{c_{11}}{T_r^2} \quad (2.3.27d)$$

$$E(T_r,\rho_r) = \frac{c_{12}}{T_r^2} + \frac{c_{13}}{T_r^3} \quad (2.3.27e)$$

$$F(T_r,\rho_r) = \frac{c_{14}}{T_r^2} + \frac{c_{15}}{T_r^3} \quad (2.3.27d)$$

$$G(T_r,\rho_r) = \frac{c_{16}}{T_r^3} + \left(\frac{c_{17}}{T_r^3} + \frac{c_{18}}{T_r^4}\right)\exp(-\rho_r^2) = G_1(T_r) + G_2(T_r)\exp(-\rho_r^2)$$

$$H(T_r,\rho_r) = \left(\frac{c_{19}}{T_r^3} + \frac{c_{20}}{T_r^4}\right)\exp(-\rho_r^2) = H_2(T_r)\exp(-\rho_r^2)$$

$$I(T_r,\rho_r) = \left(\frac{c_{21}}{T_r^3} + \frac{c_{22}}{T_r^4}\right)\exp(-\rho_r^2) = I_2(T_r)\exp(-\rho_r^2)$$

$$c_i = C_{i,0} + C_{i,1}\omega + C_{i,2}\omega^2$$

Where ω in Scheiber-Pitzer equation of states coefficient is called **Pitzer's accentric factor**. This factor is calculated as

$\omega = -log_{10}P_{saturated\ vapor}(at\ T_r = 0.7) - 1$

Table 2.1 C[i][j] Coefficients of Scheiber-Pitzer EoS

	j=1	j=2	j=3
C1,j	0.4422590000	0.7256500000	0.0000000000
C2,j	-0.9809700000	0.2187140000	0.0000000000
C3,j	-0.6111420000	-1.2497600000	0.0000000000
C4,j	-0.0051562400	-0.1891870000	0.0000000000
C5,j	0.1513654000	2.3067060000	-10.4117400000
C6,j	-0.0438262500	4.6960680000	15.1414600000
C7,j	1.1026990000	3.1293840000	-9.5214090000
C8,j	-0.6361056000	0.3266766000	2.9046220000
C9,j	0.0087596260	-3.2040990000	8.0023380000
C10,j	0.3412103000	8.8721690000	-14.4038600000
C11,j	-0.8842722000	-6.6874710000	11.7685400000
C12,j	0.1375109000	0.2432806000	-0.5515101000
C13,j	-0.1443457000	1.2869320000	-2.1809880000
C14,j	-0.0059695540	0.0454196100	0.0000000000
C15,j	0.0245053700	-0.4158241000	0.7914067000
C16,j	-0.0041995900	0.0910596000	-0.1786378000
C17,j	0.0004665477	-1.2620280000	-2.8267720000
C18,j	-0.0194510100	0.7812220000	4.1900460000
C19,j	0.0408364300	1.3988440000	0.0000000000
C20,j	-0.0354691700	-1.4560410000	0.0000000000
C21,j	-0.0028779550	0.2104505000	0.0000000000
C22	0.0058962650	0.2191255000	0.0000000000

Derivatives of the equations:

$$\frac{\partial B(T_r,\rho_r)}{\partial T} = \frac{1}{T_c}\left[-\frac{c_2}{T_r^2} - 2\frac{c_3}{T_r^3} - 6\frac{c_4}{T_r^7}\right]$$

$$\frac{\partial C(T_r,\rho_r)}{\partial T} = \frac{1}{T_c}\left[-\frac{c_5}{T_r^2} - 3\frac{c_7}{T_r^4} - 4\frac{c_8}{T_r^5}\exp(-\rho_r^2)\right]$$

$$\frac{\partial D(T_r,\rho_r)}{\partial T}=\frac{1}{T_c}\left[-\frac{c_{10}}{T_r^2}-3\frac{c_{11}}{T_r^4}\right]$$
$$\frac{\partial E(T_r,\rho_r)}{\partial T}=\frac{1}{T_c}\left[-\frac{c_{12}}{T_r^2}-3\frac{c_{13}}{T_r^4}\right]$$

$$\frac{\partial F(T_r,\rho_r)}{\partial T}=\frac{1}{T_c}\left[-2\frac{c_{14}}{T_r^3}-3\frac{c_{13}}{T_r^4}\right]$$
$$\frac{\partial G(T_r,\rho_r)}{\partial T}=\frac{1}{T_c}\left[-3\frac{c_{16}}{T_r^4}+\left(-3\frac{c_{17}}{T_r^4}-4\frac{c_{18}}{T_r^5}\right)\exp\left(-\rho_r^2\right)\right]$$
$$\frac{\partial H(T_r,\rho_r)}{\partial T}=\frac{1}{T_c}\left[\left(-3\frac{c_{19}}{T_r^4}-4\frac{c_{18}}{T_r^5}\right)\exp\left(-\rho_r^2\right)\right]$$
$$\frac{\partial I(T_r,\rho_r)}{\partial T}=\frac{1}{T_c}\left[\left(-3\frac{c_{21}}{T_r^4}-4\frac{c_{22}}{T_r^5}\right)\exp\left(-\rho_r^2\right)\right]$$

Helmholtz energy equation will be used to predict other thermodynamic properties

$$dA=-SdT-Pdv$$
$$\left.\frac{\partial A}{\partial v}\right|_T=-P$$
$$dA=-Pdv=\frac{P}{\rho^2}d\rho$$
$$A-A_0=\int_{\rho_0}^{\rho}\frac{P}{\rho^2}d\rho=\int_{0}^{\rho}\frac{P}{\rho^2}d\rho+\int_{\rho_0}^{0}\frac{P}{\rho^2}d\rho=\int_{0}^{\rho}\frac{P}{\rho^2}d\rho+\int_{\rho_0}^{0}\frac{\rho RT}{\rho^2}d\rho$$

The second term (limits between low density ρ_0 and $density$ 0) can be defined as an ideal state case where $P=\rho RT$. Now we can add and substract ideal gas density term of the equation

$$A-A_0=\int_{\rho_0}^{\rho}\frac{P}{\rho^2}d\rho=\int_{0}^{\rho}\frac{P}{\rho^2}d\rho+\int_{\rho_0}^{0}\frac{\rho RT}{\rho^2}d\rho+\int_{0}^{\rho}\frac{\rho RT}{\rho^2}d\rho-\int_{0}^{\rho}\frac{\rho RT}{\rho^2}d\rho$$

Considering for real EoS $P=Z\rho RT$ equation becomes

$$A-A_0=\int_{\rho_0}^{\rho}\frac{P}{\rho^2}d\rho=\int_{0}^{\rho}\frac{P}{\rho^2}d\rho+\int_{\rho_0}^{0}\frac{\rho RT}{\rho^2}d\rho+\int_{0}^{\rho}\frac{\rho RT}{\rho^2}d\rho-\int_{0}^{\rho}\frac{\rho RT}{\rho^2}d\rho$$
$$A-A_0=\int_{0}^{\rho}\frac{Z\rho RT-\rho RT}{\rho^2}d\rho+\int_{\rho_0}^{\rho}\frac{\rho RT}{\rho^2}d\rho$$
$$A-A_0=\int_{0}^{\rho}\frac{RT(Z-1)}{\rho}d\rho+\int_{\rho_0}^{\rho}\frac{\rho RT}{\rho^2}d\rho$$

$$A - A_0$$
$$= \int_0^{\rho_r} \frac{RT[B(T_r,\rho_r)\rho_r + C(T_r,\rho_r)\rho_r^2 + D(T_r,\rho_r)\rho_r^3 + E(T_r,\rho_r)\rho_r^5 + F(T_r,\rho_r)\rho_r^7 + G(T_r,\rho_r)\rho_r^8 + H(T_r,\rho_r)\rho_r^1}{\rho_r}$$
$$+ \int_{\rho_{r0}}^{\rho_r} \frac{RT}{\rho_r} d\rho_r$$

$$A - A_0 = RT \int_0^{\rho_r} [B(T_r,\rho_r) + C(T_r,\rho_r)\rho_r + D(T_r,\rho_r)\rho_r^2 + E(T_r,\rho_r)\rho_r^4 + F(T_r,\rho_r)\rho_r^6 + G(T_r,\rho_r)\rho_r^7 + H(T_r,\rho_r)\rho_r^9 + I(T_r,\rho_r)\rho_r^{11}]d\rho_r + RTln\frac{\rho_r}{\rho_{r0}}$$

Some of the terms in this integration includes terms of exponential and power multiplications. This integrations are carried out as follows:

$K(m,\rho_r) = \int_0^{\rho_r} \rho_r^m \exp(-\rho_r^2) d\rho_r = \int_0^{\rho_r} \sum_{n=0}^{\infty} (-1)^n \frac{\rho_r^{2n+m}}{n!} d\rho_r = \sum_{n=0}^{\infty} (-1)^n \frac{\rho_r^{2n+m+1}}{(2n+m+1)n!}$

Handling these terms such as above Taylor series are much easier than taken numerical integration, then Helmholtz departure function becomes:

$$A - A_0 = RT\left(B\rho_r + C_1\frac{\rho_r^2}{2} + C_2K(1,\rho_r) + D\frac{\rho_r^3}{3} + E\frac{\rho_r^5}{5} + F\frac{\rho_r^7}{7} + G_1\frac{\rho_r^8}{8} + G_2,K(8,\rho_r) + H_2K(10,\rho_r) + I_2K(12,\rho_r)\right) + RTln\left(\frac{\rho_r}{\rho_{r0}}\right)$$

Entropy departure function:

$$S - S_0 = -\left.\frac{\partial(A - A_0)}{\partial T}\right|_{\rho}$$

$$S - S_0 = R\left(B\rho_r + C_1\frac{\rho_r^2}{2} + C_2K(1,\rho_r) + D\frac{\rho_r^3}{3} + E\frac{\rho_r^5}{5} + F\frac{\rho_r^8}{8} + G_1\frac{\rho_r^9}{9} + G_2,K(8,\rho_r) + H_2K(10,\rho_r) + I_2K(12,\rho_r)\right)$$
$$+ RT\left(\frac{dB}{dT}\rho_r + \frac{dC_1}{dT}\frac{\rho_r^2}{2} + \frac{dC_2}{dT}K(1,\rho_r) + \frac{dD}{dT}\frac{\rho_r^3}{3} + \frac{dE}{dT}\frac{\rho_r^5}{5} + \frac{dF}{dT}\frac{\rho_r^8}{8} + \frac{dG_1}{dT}\frac{\rho_r^9}{9} + \frac{dG_2}{dT}K(8,\rho_r) + \frac{dH_2}{dT}K(10,\rho_r) + \frac{dI_2}{dT}K(12,\rho_r)\right) + Rln\left(\frac{\rho_r}{\rho_{r0}}\right)$$

Enthalpy departure function:

$$H - H_0 = (A - A_0) + T(S - S_0) + RT(Z - 1) \quad 2.26$$

Internal energy departure function:

$$U - U_0 = (A - A_0) + T(S - S_0) \quad 2.28$$

Gibbs energy departure function:

$$G - G_0 = (A - A_0) + RT(Z - 1) \quad 2.29$$

The fugacity-pressure ratio:

$$ln\frac{f}{P} = \frac{(A - A_0)}{RT} + ln\frac{v}{v_0} + (Z - 1) - \ln(Z) \quad 2.30$$

A Graphic user interface programs are also given for non-researchers to utilise these programs. For utilisation of researchers a small sample code is given to show calling of thermodynamic and thermophysical properties for a given state.

```
public static void main(String arg[])
{ Gas_SP sp=new Gas_SP("N2");
sp.mole=false;
double T=300;  //Degree K
double P=100.0; //kPa
//double b[]=lk.property("tp",T,P);
 //Text.printT(b);
 sp.print("tp",T,P);
}
```

Properties of N2 nitrogene Schreiber and Pitzer Eos

array no	property name	property	unit
0	P, pressure	100.0	kPA
1	T, temperature	300.0	deg K
2	v, specific volume	0.8904155410028988	m³/kg
3	h, enthalpy	27.963873781544986	KJ/kg
4	u, internal energy	19.93938568155416	KJ/kg
5	s, entropy	0.1016123679115501	KJ/kgK
6	x, quality	2.0	kg vap/kg mix
7	Cp, specific heat at const...	1.0396977956967528	KJ/kgK
8	Cv, specific heat at const...	0.7428905055759789	KJ/kgK
9	α isobaric thermal expansi...	1.0000071023699687E-5	1/K
10	κ isothermal compressibility	1.0000071023699687E-5	1/Pa
11	η Dynamic viscosity	1.787600318502586E-5	Pa.s
12	k Thermal conductivity	0.027666480768509534	W/mK
13	σ Surface tension	0.0	N/m
14	Prandtl number	6.717746743016922E-4	
15	ρ density	1.1230711436973273	kg/m³
16	speed of sound	353.0091651518848	m/s

Figure3.1 A GUI output of a sample gas N_2

You can obtain all properties as an array by using property method. Property method accepts two known thermodynamic property set. Known property sets are as :

Known set	meaning	Known set	meaning
tp	Temperature Pressure	pt	Pressure temperature
vt	Specific volume Temperature	tv	Temperature Specific volume
rot	Density temperature	tro	Temperature density
pv	Pressure Specific volume	vp	Specific volume pressure
pro	Pressure density	rop	Density pressure
ph	Pressure enthalpy	hp	Enthalpy Pressure
ps	Pressure entropy	sp	Entropy pressure

In the following example property pair “tp” is used.

```
  public static void main(String arg[])
{Gas_SP lk=new Gas_SP("N2");
  double T=300.0; //degree K
  double P=100.0; //kPA
  double a[]=lk.property("tp",T,P);
  for( double x :a) System.out.println(x);
}
```

```
 ---------- Capture Output ----------
> "E:\co\java\bin\javaw.exe" Gas_SP
100.0
300.0
24.93853374087881
783.3508278861523
558.0997487204611
```

```
2.844904873520239
2.0
29.12505435085314
20.810591732699898
1.000189954343909E-5
1.000189954343909E-5
1.787600318502586E-5
0.027666480768509534
0.0
6.717746743016922E-4
0.04009858840902172
352.9438027622723
> Terminated with exit code 0.
```

In Table 4.2 different EOS results compared for several thermodynamic states. A Graphic user interface programs are also given for non-researchers to utilise these programs. For utilisation of researchers a small sample code is given to show calling of thermodynamic and thermophysical properties for a given state.

Table 4.2 Comparisons of 4 different EOS for dry Nitrogen

Class	P kPa	T degree	v m^3/kg	h kJ/kg	u kJ/kg	s kJ/kgK
Schreiber-Pitzer_Eos	100	300	0.890416	27.96387	19.93939	0.101612
Peng-Robinson_EoS	100	300	0.890042	27.97213	19.97872	0.101552
Lee-Kesler_EoS	100	300	0.890239	27.96187	19.97266	0.10155
Perfect_Gas_EoS	100	300	0.890422	27.91124	19.94196	0.101374
Schreiber-Pitzer_Eos	500	300	0.178078	27.08606	19.94962	-0.37859
Peng-Robinson_EoS	500	300	0.177717	26.89694	19.04938	-0.37807
Lee-Kesler_EoS	500	300	0.17791	27.11667	19.1961	-0.37872
Perfect_Gas_EoS	500	300	0.178084	27.91124	19.94196	-0.37632
Schreiber-Pitzer_Eos	1000	300	0.089036	26.02189	19.95403	-0.58745
Peng-Robinson_EoS	1000	300	0.08869	25.5727	17.89375	-0.58617
Lee-Kesler_EoS	1000	300	0.08888	26.06872	18.22338	-0.58767
Perfect_Gas_EoS	1000	300	0.089042	27.91124	19.94196	-0.58205
Schreiber-Pitzer_Eos	3000	300	0.029675	22.12961	19.96102	-0.92584
Peng-Robinson_EoS	3000	300	0.029388	20.50085	13.34616	-0.92108
Lee-Kesler_EoS	3000	300	0.029566	21.98087	14.31741	-0.92645
Perfect_Gas_EoS	3000	300	0.029681	27.91124	19.94196	-0.90813
Schreiber-Pitzer_Eos	5000	300	0.017802	18.81358	19.96427	-1.08952
Peng-Robinson_EoS	5000	300	0.017575	15.79957	8.936128	-1.08054
Lee-Kesler_EoS	5000	300	0.017742	18.08075	10.40646	-1.09046
Perfect_Gas_EoS	5000	300	0.017808	27.91124	19.94196	-1.05974

By using equations given above a class called Gas_SP is developed. The result of this class is compared with Lee-Kesler EoS and Perfect results as a mixing of the gases inside air and as a single gas with pseudocritical properties plus ideal gas equation of state. Results are given in table 4.2. It should be note that enthalpies of the perfect gas case (air_PG and Gas) is the same regardless of pressure. In the following figures Peng-Robinson Equation of states is compared with perfect gas equation of state and IAPWS for dry air EOS. IAPWS equation based on the experimental data therefore it can be assumed given carrect values for the range of experimentation. Peng-Robinson equation results are falling between Perfect gas and IAPWS results. This is the expected kind of behevior for such a general EOS based on **accentric factor**.

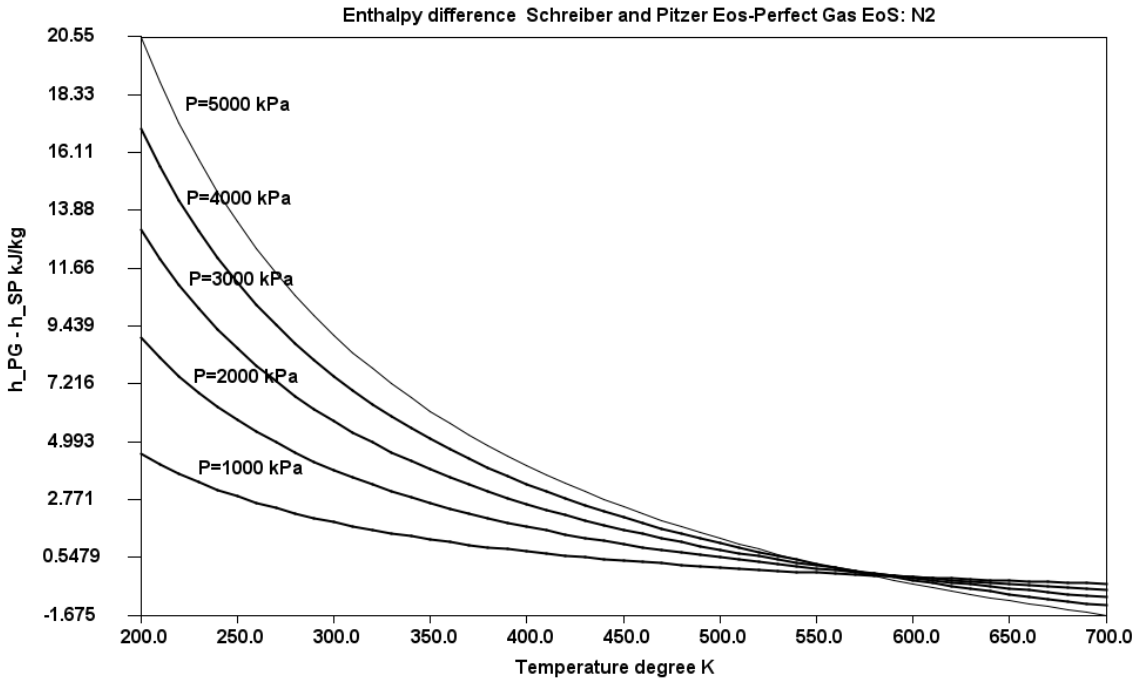


Figure 2.3.20 Enthalpy difference of Schereiber-Pitzer EoS and Perfect Gas EoS 1000-5000 kPa

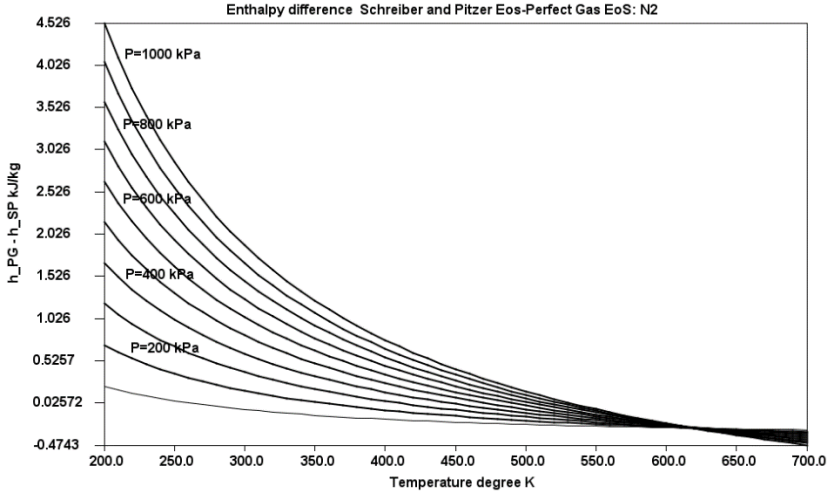


Figure 2.3.21 Enthalpy difference of Schereiber-Pitzer EoS and Perfect Gas EoS 100-1000 kPa

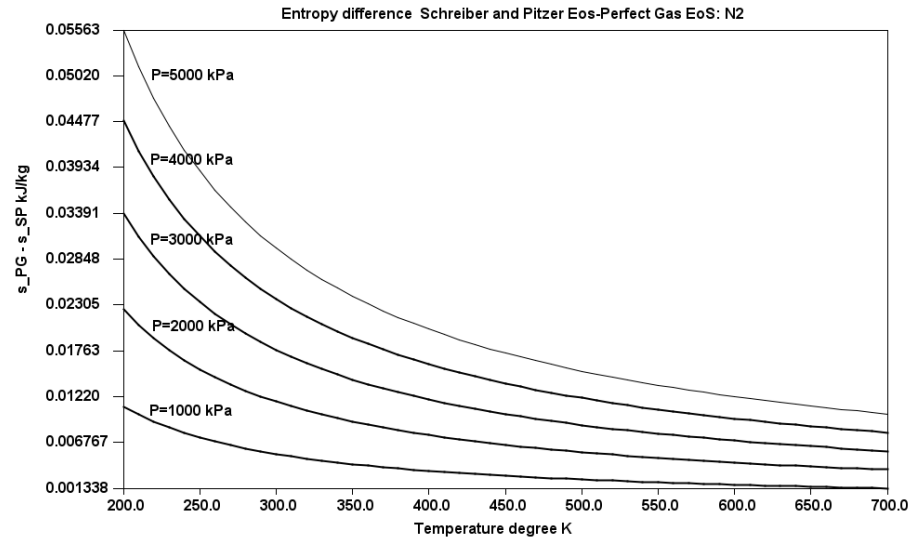


Figure 2.3.22 Entropy difference of Schereiber-Pitzer EoS and perfect gas EOS

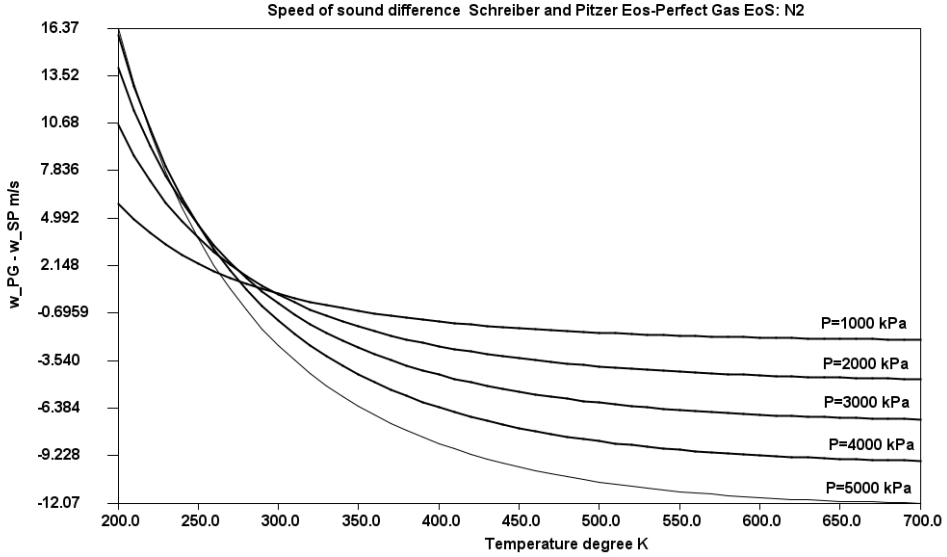


Figure 2.3.23 Speed of sound difference of Schereiber-Pitzer EoS and perfect gas EOS

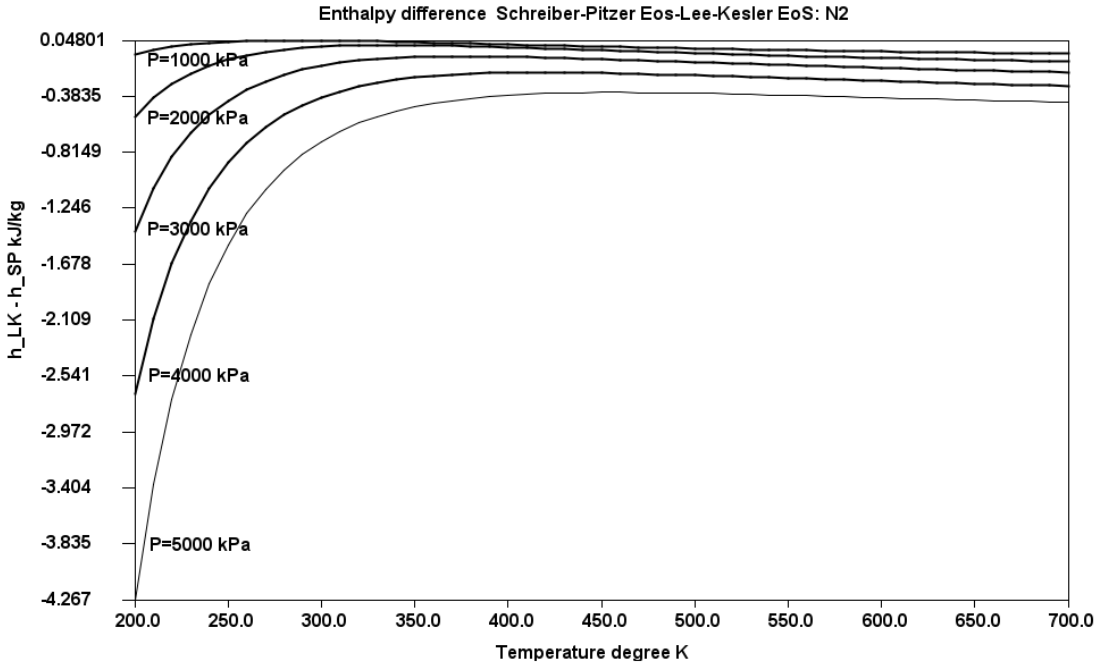


Figure 2.3.24 Enthalpy difference of Schereiber-Pitzer EoS and Lee-Kesler EOS

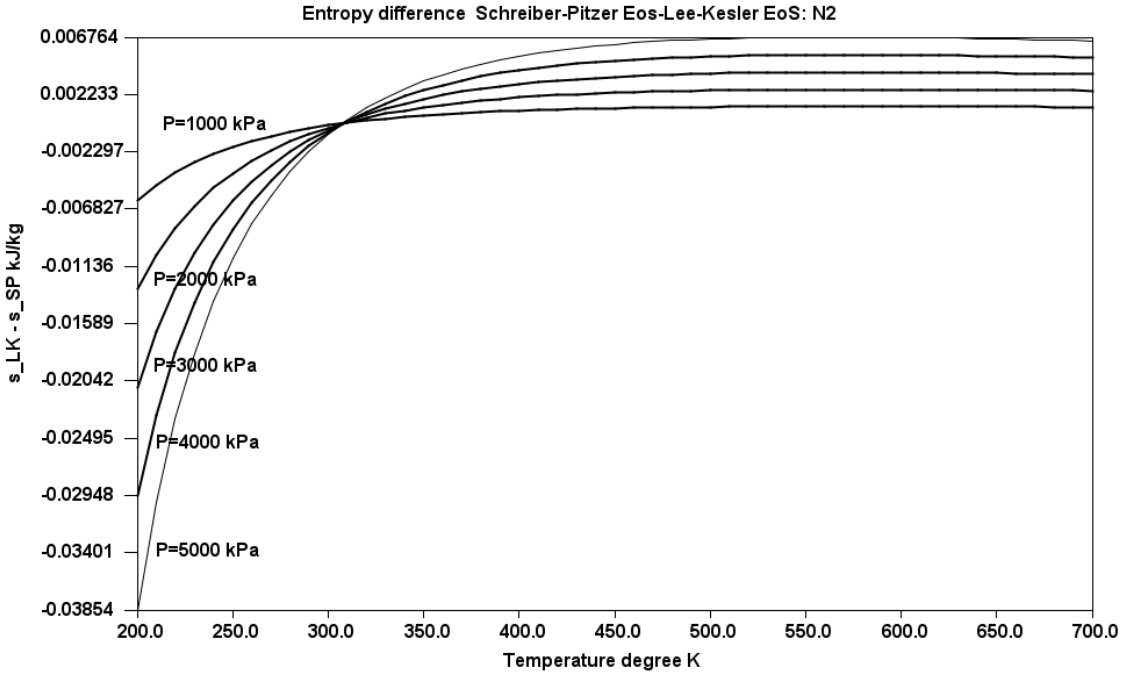


Figure 2.3.25 Entropy difference of Schereiber-Pitzer EoS and Lee-Kesler EOS

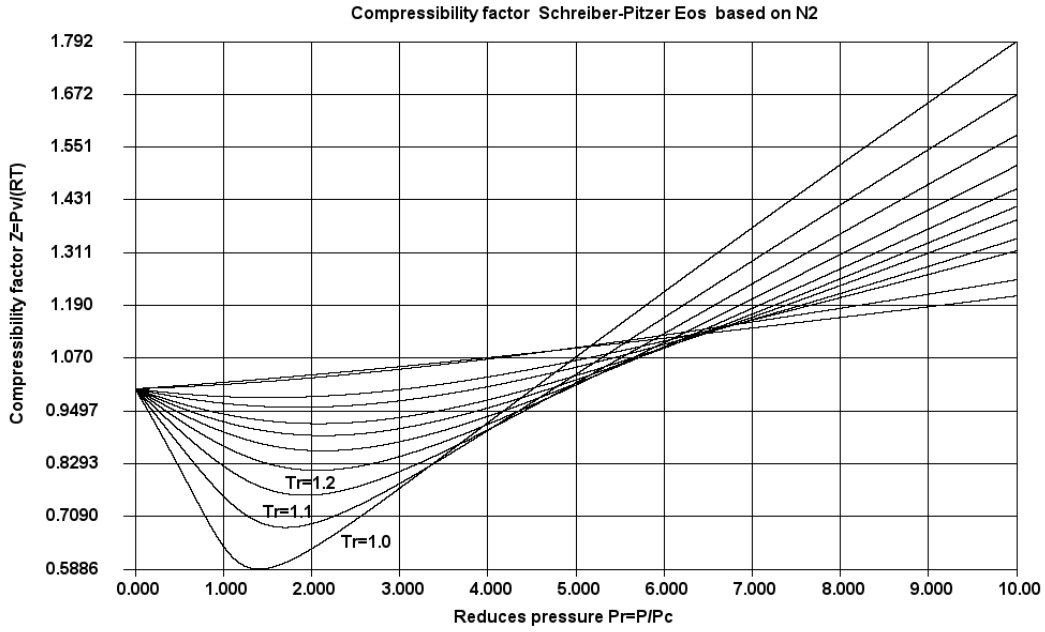


Figure 2.3.26 Compressibility factor Schreiber-Pitzer EoS

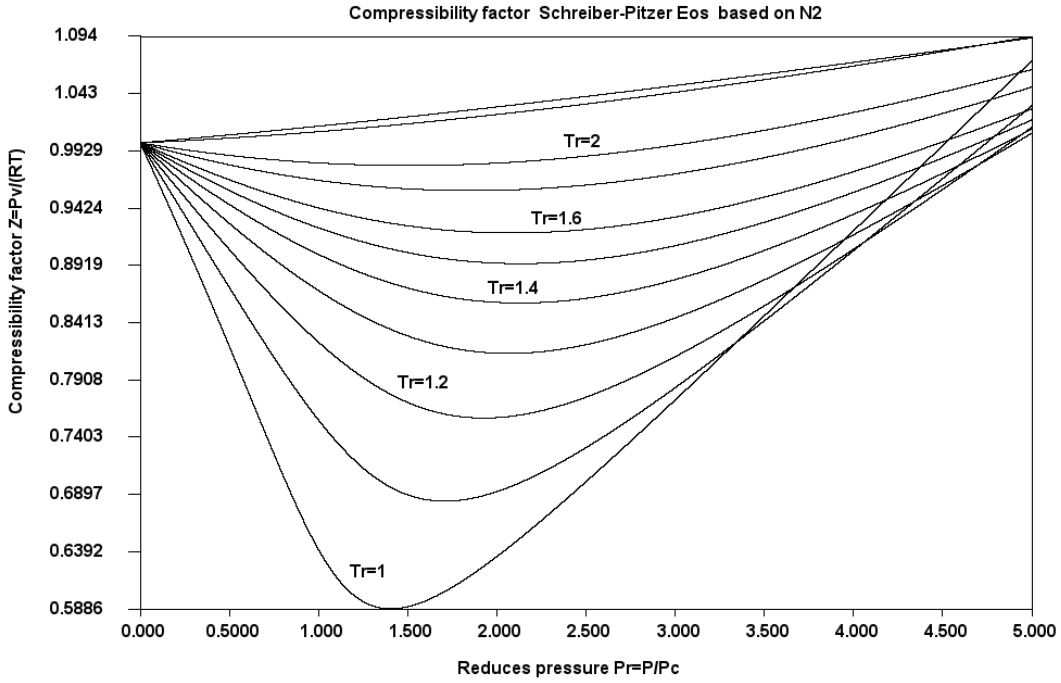


Figure 2.3.27 Compressibility factor Schereiber-Pitzer EoS

```
public class gas_SP_test1
{
public static void main(String arg[])
{  Gas_SP sp=new Gas_SP("N2");
   double Tr[]={1,1.1,1.2,1.3,1.4,1.5,1.6,1.8,2.0,3.0,4.0};
   int n=100000;
    double Z[][]=new double[Tr.length][n];
    double P=0;
    double ro=0;
    double Pr[]=new double[n];
    double T=0;
    for(int i=0;i<Tr.length;i++)
```

```
    {int j=0;
     for(j=0;j<n;j++)
      {  Pr[j]=0.0001*j;
         P=Pr[j]*sp.Pc;
         T=Tr[i]*sp.Tc;
         ro=sp.ro(T,P);
         Z[i][j]=sp.Z(T,ro);
      }
    }
    Plot pp=new Plot(Pr,Z[0]);
    pp.setPlabel("Compressibility factor  Schreiber-Pitzer Eos  based on N2");
    pp.setXlabel("Reduces pressure Pr=P/Pc");
    pp.setYlabel("Compressibility factor Z=Pv/(RT)");
    for(int i=1;i<Tr.length;i++)
    {pp.addData(Pr,Z[i]);
     pp.setPlotType(i,0);
    }
    pp.plot();
}}
```

Mixing of the gases: Pseudocritical property approach will be used. According to this concept a pseudocritical properties of the gas mixtures are calculated and then gas mixture can be evaluated similar to a single gas equation. Mixing rules for a Lee-Kesler mix is given as follows:

$$N = \sum_i N_i$$

Where N_i is mole number of each substance in the mixture.

$$y_i = \frac{N_i}{N}$$

Where $y_i$is mole percentages of eachsubstance.

$$T_c = \frac{1}{v_c^{1/4}} \sum_i \sum_j y_i y_j v_{cij}^{1/4} T_{cij}$$

Where $T_c$is pseudo-critical temperature of mixture. Due to charecteristic of mixture there is no one critical temperature but critical temperature of each species in the mixture, therefore this value does not have a physical significance and only for calculation purposes.

$$v_c = \sum_i \sum_j y_i y_j v_{cij}$$

Where v_c is pseudocritical specific volume

$$\omega = \sum_i y_i \omega_i$$

Where ω is Pitzer's acentric factor of the mixture

$$T_{cij} = \left(T_{ci} T_{cj}\right)^{1/2} k'_{ij}$$

$$k'_{ij} = \frac{8\left(v_{ci} v_{cj}\right)^{1/2}}{\left(v_{ci}^{1/3} + v_{cj}^{1/3}\right)^3}$$

k'_{ij} is binary interaction coefficients. Due to difference of molecules in the mixture and intermolecular forces effecting because of the difference. This equation will give only approximation. The exact value can only be obtain by experimental measurements of mixture of two gases. As an example of experimental results : if it is a binary mixture of hydrocarbons, the following value can be taken as k'_{ij} value. It can be applied to calculate properties of natural gas or iiquid petroleum gas(LPG), or gasoline if you can obtain a formulation for the mixture.

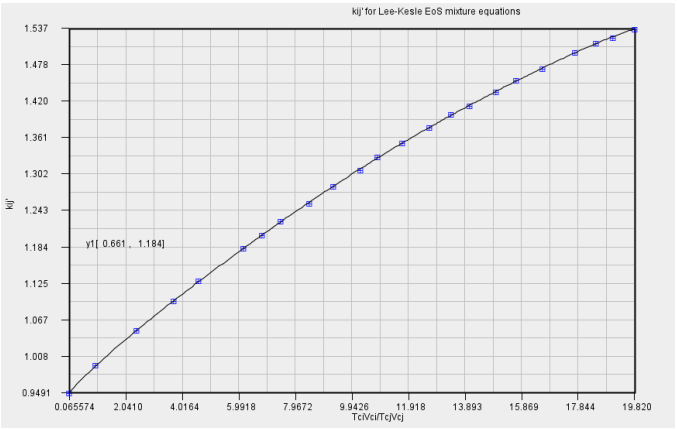


Figure 2.3.29 interaction coefficient k'$_{ij}$ curve-fitting plot

$k'_{ij}(\frac{T_{ci}v_{ci}}{T_{cj}v_{cj}}) = \sum_{i=0}^{6} a_i \left(\frac{T_{ci}v_{ci}}{T_{cj}v_{cj}}\right)^i$ coefficients for hydrocarbon pairs

a0	9.4637932E-01
a1	4.8752931E-02
a2	-3.5797272E-03
a3	5.0398694E-04
a4	-4.2033220E-05
a5	1.7107314E-06
a6	-2.7048029E-08

Hydrocarbon pairs k'_{ij} values are alo listed in table format below:

	first hyd.car.	2nd hyd.car.	k'_{ij}		first hyd.car.	2nd hyd.car.	k'_{ij}
1	Methane	Ethane	1.052	30	Ethylene	n-butane	0.998
2	Methane	Ethylene	1.014	31	Ethylene	benzene	1.094
3	Methane	propane	1.113	32	Ethylene	n-heptane	1.163
4	Methane	propylene	1.089	33	Aceltylene	ethylene	0.948
5	Methane	n-butane	1.171	34	Propane	propylene	0.948
6	Methane	isobutane	1.155	35	Propane	n-butane	1.003
7	Methane	n-pantane	1.24	36	Propane	isobutane	1.003
8	Methane	isopentane	1.228	37	Propane	n-pantane	1.006
9	Methane	n-hexane	1.304	38	Propane	isopentane	1.009
10	Methane	cyclehexane	1.269	39	Propane	n-hexane	1.047
11	Methane	benzene	1.234	40	Propane	cycohexane	1.037
12	Methane	n-heptane	1.367	41	Propane	benzene	1.011
13	Methane	n-octane	1.423	42	Propane	n-heptane	1.067
14	Methane	n-nonane	1.484	43	Propane	n-octane	1.09
15	Methane	n-decane	1.533	44	Propane	n-nonane	1.115
16	Ethane	Ethylene	0.991	45	Propane	n-decane	1.139
17	Ethane	propane	1.01	46	Propylene	n-butane	1.01
18	Ethane	propylene	1.002	47	Propylene	isobutane	1.009
19	Ethane	n-butane	1.029	48	Propylene	isobutane	1.006
20	Ethane	isobutane	1.036	49	n-butane	isobutane	1.001
21	Ethane	n-pantane	1.064	50	n-butane	n-pantane	0.994
22	Ethane	isopentane	1.07	51	n-butane	isopentane	0.998
23	Ethane	n-hexane	1.106	52	n-butane	n-hexane	1.018
24	Ethane	cyclehexane	1.081	53	n-butane	cyclohexane	1.008
25	Ethane	benzene	1.066	54	n-butane	benzene	0.999
26	Ethane	n-heptane	1.143	55	n-butane	n-heptane	1.027
27	Ethane	n-octane	1.165	56	n-butane	n-octane	1.046
28	Ethane	n-nonane	1.214	57	n-butane	n-nonane	1.064
29	Ethane	n-decane	1.237	58	n-butane	n-decane	1.078

$$v_{cij} = \frac{1}{8}\left(v_{ci}^{1/3} + v_{cj}^{1/3}\right)^3$$

$$Z_c = (0.2905 - 0.085\omega$$

Where Z_c is pseudo-critical compressibility factor. As it is seen from the equation, it is given only as a function of Pitzer's acentric factor.

$$P_c = Z_c R T_c / v_c$$

Where Z_c is pseudo-critical pressure.

Class Gmix_LK (based on Gas_LK) is created to calculate gas mixtures by using pseudo-critical properties. UTilisation of this program is same as Gas class with exception of mixture definition requirement at the inlet of the program.

```
public static void main(String arg[])
{  String name[]={"N2","O2"};
   double n[]={0.79,0.21};
            Gmix_LK pr=new Gmix_LK("Gmix_LK",name,n);
            double T=300.0;
            double P=100.0;
            pr.print("tp",T,P);
}
```

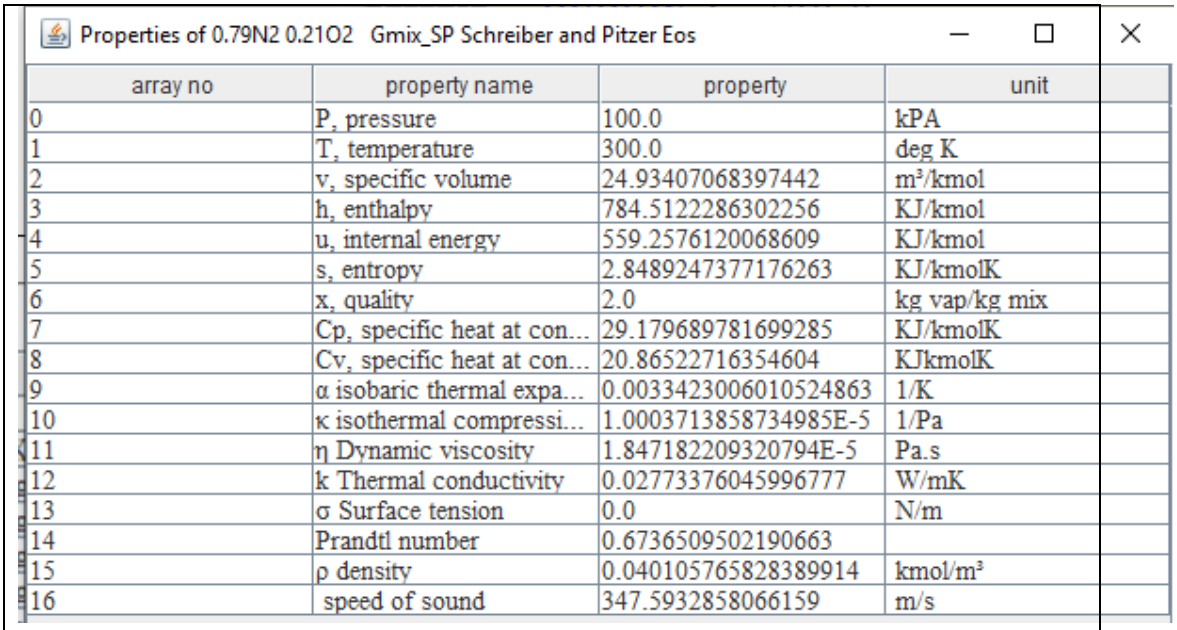
Properties of gas mixture Lee-Kesler EOS GasName: Gmix_LK 0.79N2 0.21O2

array no	property name	gasproperty	unit
0	P, pressure	100.0	kPA
1	T, temperature	300.0	deg K
2	v, specific volume	24.934158625269163	m³/kmol
3	h, enthalpy	784.4617107140538	KJ/kmol
4	u, internal energy	560.6535846981727	KJ/kmol
5	s, entropy	2.848750163959173	KJ/kmolK
6	x, quality	2.0	kg vap/kg mix
7	Cp, specific heat at co...	29.179689781699285	KJ/kmolK
8	Cv, specific heat at co...	20.86522716354604	KJkmolK
9	α isobaric thermal expa...	2.883752582719838E-4	1/K
10	κ isothermal compressi...	2.883752582719838E-4	1/Pa
11	η Dynamic viscosity	1.847182209320794E-5	Pa.s
12	k Thermal conductivity	0.02773376045996777	W/mK
13	σ Surface tension	0.0	N/m
14	Prandtl number	6.736509502190662E-4	
15	ρ density	0.040105624377738756	kmol/m³
16	speed of sound	347.73398483241436	m/s

Figure 2.3.30 Lee-Kesler gas mixture GUI output

Gas mixtures can be calculated by using Schreiber-Pitzer EoS in a simalar manner as Lee-Kesler EoS. A class Gmix_SP is developed fort his calculations. In Table 4.2 different EOS results compared for several thermodynamic states. A Graphic user interface programs are also given for non-researchers to utilise these programs. For utilisation of researchers a small sample code is given to show calling of thermodynamic and thermophysical properties for a given state.

```
public static void main(String arg[])
{  String name[]={"N2","O2"};
   double n[]={0.79,0.21};
            Gmix_SP pr=new Gmix_SP("Gmix_SP",name,n);
            double T=500.0;
            double P=100.0;
            pr.print("tp",T,P);
}
```



array no	property name	property	unit
0	P, pressure	100.0	kPA
1	T, temperature	300.0	deg K
2	v, specific volume	24.93407068397442	m³/kmol
3	h, enthalpy	784.5122286302256	KJ/kmol
4	u, internal energy	559.2576120068609	KJ/kmol
5	s, entropy	2.8489247377176263	KJ/kmolK
6	x, quality	2.0	kg vap/kg mix
7	Cp, specific heat at con...	29.179689781699285	KJ/kmolK
8	Cv, specific heat at con...	20.86522716354604	KJkmolK
9	α isobaric thermal expa...	0.0033423006010524863	1/K
10	κ isothermal compressi...	1.0003713858734985E-5	1/Pa
11	η Dynamic viscosity	1.847182209320794E-5	Pa.s
12	k Thermal conductivity	0.02773376045996777	W/mK
13	σ Surface tension	0.0	N/m
14	Prandtl number	0.6736509502190663	
15	ρ density	0.040105765828389914	kmol/m³
16	speed of sound	347.5932858066159	m/s

Figure 2.3.31A GUI output of a sample gas N_2

You can obtain all properties as an array by using property method. Property method accepts two known thermodynamic property set. Known property sets are as :

Known set	meaning	Known set	meaning
tp	Temperature Pressure	pt	Pressure temperature
vt	Specific volume Temperature	tv	Temperature Specific volume
rot	Density temperature	tro	Temperature density
pv	Pressure Specific volume	vp	Specific volume pressure
pro	Pressure density	rop	Density pressure
ph	Pressure enthalpy	Hp	Enthalpy Pressure
ps	Pressure entropy	sp	Entropy pressure

In the following example property pair “tp” is used.

```
  public static void main(String arg[])
{Gas_LK lk=new Gas_LK("N2");
  double T=300.0; //degree K
  double P=100.0; //kPA
  double a[]=lk.property("tp",T,P);
  for( double x :a) System.out.println(x);
}
```

```
 ---------- Capture Output ----------
> "E:\co\java\bin\javaw.exe" Gas_LK
100.0
300.0
24.938254365652387
783.2957857861796
559.4942097404995
2.844713746696977
2.0
29.12505435085314
```

```
20.810591732699898
2.800530395352282E-4
2.800530395352282E-4
1.787600318502586E-5
0.027666480768509534
0.0
6.717746743016922E-4
0.04009903762058447
353.0238423174664

> Terminated with exit code 0.
```

Table 4.2 Comparisons of 3 different EOS for079% Nitrogen %21%Oxygen mixture.

Class	P kPa	T K	v m3/kg	h kJ/kg	u kJ/kg	s kJ/kgK
Schreiber-Pitzer_EoS	100	300	0.864576	27.19283	19.40105	0.098853
Lee-Kesler_EoS	100	300	0.864267	27.19099	19.43336	0.098743
Perfect_Gas_EoS	100	300	0.864587	27.14156	19.4035	0.098572
Schreiber-Pitzer_EoS	500	300	0.172906	26.26978	19.4107	-0.36723
Lee-Kesler_EoS	500	300	0.172606	26.31127	18.67753	-0.3676
Perfect_Gas_EoS	500	300	0.172917	27.14156	19.4035	-0.36526
Schreiber-Pitzer_EoS	1000	300	0.086447	25.13207	19.41486	-0.5698
Lee-Kesler_EoS	1000	300	0.086157	25.21774	17.72959	-0.5705
Perfect_Gas_EoS	1000	300	0.086459	27.14156	19.4035	-0.56502
Schreiber-Pitzer_EoS	3000	300	0.028808	20.75674	19.42145	-0.89757
Lee-Kesler_EoS	3000	300	0.02856	20.92097	13.90944	-0.89956
Perfect_Gas_EoS	3000	300	0.02882	27.14156	19.4035	-0.88164
Schreiber-Pitzer_EoS	5000	300	0.017281	16.65477	19.42451	-1.05587
Lee-Kesler_EoS	5000	300	0.017076	16.7713	10.06135	-1.05903
Perfect_Gas_EoS	5000	300	0.017292	27.14156	19.4035	-1.02886
Schreiber-Pitzer_EoS	100	500	1.440993	231.6332	166.0118	0.620328
Lee-Kesler_EoS	100	500	1.441352	231.6224	166.1563	0.62038
Perfect_Gas_EoS	100	500	1.440978	231.4104	166.0333	0.619771
Schreiber-Pitzer_EoS	500	500	0.28821	231.3921	166.0279	0.155346
Lee-Kesler_EoS	500	500	0.288574	231.3932	165.7752	0.155787
Perfect_Gas_EoS	500	500	0.288196	231.4104	166.0333	0.155938
Schreiber-Pitzer_EoS	1000	500	0.144112	231.0977	166.0348	-0.04585
Lee-Kesler_EoS	1000	500	0.144483	231.1131	165.2994	-0.04492
Perfect_Gas_EoS	1000	500	0.144098	231.4104	166.0333	-0.04382
Schreiber-Pitzer_EoS	3000	500	0.048047	229.9971	166.0458	-0.36821
Lee-Kesler_EoS	3000	500	0.048443	230.0627	163.4042	-0.36522
Perfect_Gas_EoS	3000	500	0.048033	231.4104	166.0333	-0.36044
Schreiber-Pitzer_EoS	5000	500	0.028834	229.0215	166.0509	-0.52115
Lee-Kesler_EoS	5000	500	0.029254	229.1233	161.5246	-0.51601
Perfect_Gas_EoS	5000	500	0.02882	231.4104	166.0333	-0.50766
Schreiber-Pitzer_EoS	100	800	2.305593	552.8368	400.6438	1.12224
Lee-Kesler_EoS	100	800	2.306278	552.82	400.8613	1.122296
Perfect_Gas_EoS	100	800	2.305565	552.5325	400.6966	1.121564
Schreiber-Pitzer_EoS	500	800	0.46114	552.9204	400.6696	0.657734
Lee-Kesler_EoS	500	800	0.461829	552.8917	400.6464	0.658196
Perfect_Gas_EoS	500	800	0.461113	552.5325	400.6966	0.657731
Schreiber-Pitzer_EoS	1000	800	0.230584	553.0289	400.6806	0.457131
Lee-Kesler_EoS	1000	800	0.231276	552.985	400.3783	0.458102
Perfect_Gas_EoS	1000	800	0.230557	552.5325	400.6966	0.457969
Schreiber-Pitzer_EoS	3000	800	0.07688	553.5064	400.6982	0.137151
Lee-Kesler_EoS	3000	800	0.077585	553.3975	399.3115	0.140187
Perfect_Gas_EoS	3000	800	0.076852	552.5325	400.6966	0.141354
Schreiber-Pitzer_EoS	5000	800	0.046139	554.0546	400.7064	-0.01343
Lee-Kesler_EoS	5000	800	0.046857	553.8714	398.254	-0.00829
Perfect_Gas_EoS	5000	800	0.046111	552.5325	400.6966	-0.00586

By using equations given above a class called Gmix_SP is developed. The result of this class is compared with results of Perfect gas EoS, Gmix_PG, and and Lee-Kesler EoS,Gmix_LK.

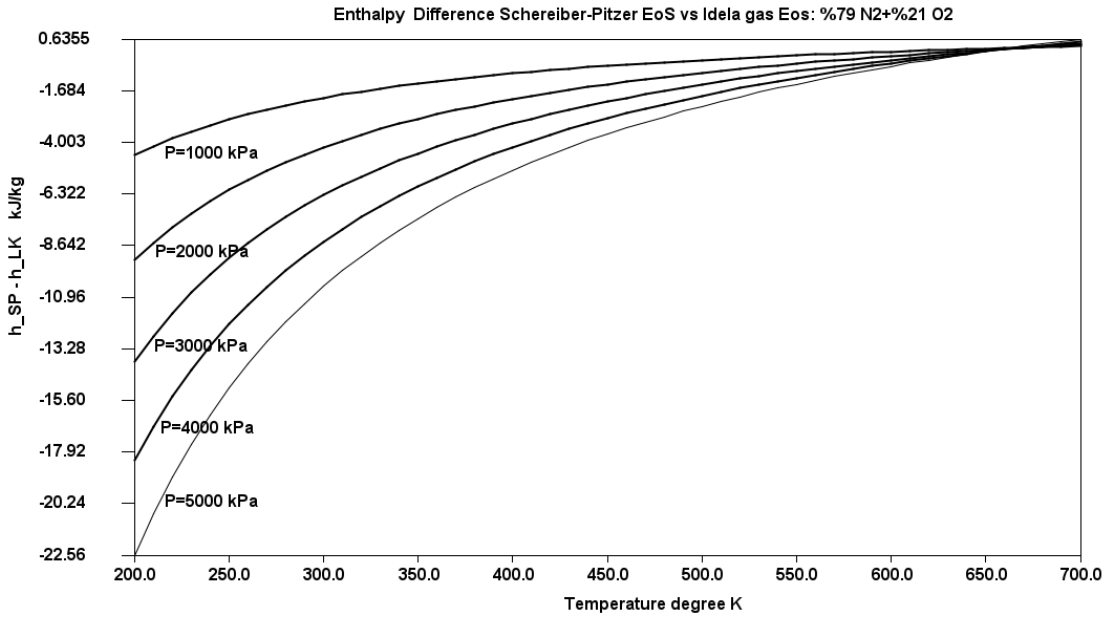


Figure 3.1 Enthalpy difference of Schereiber-Pitzer and perfect gas EOS

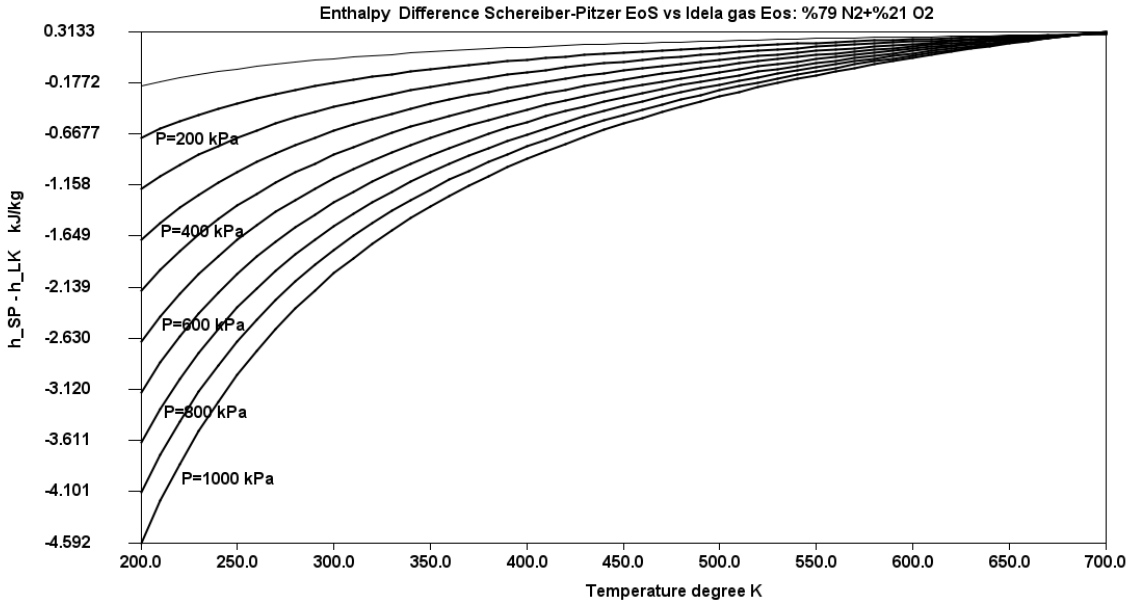


Figure 4.2 Enthalpy difference of Schereiber-Pitzer and perfect gas EOS

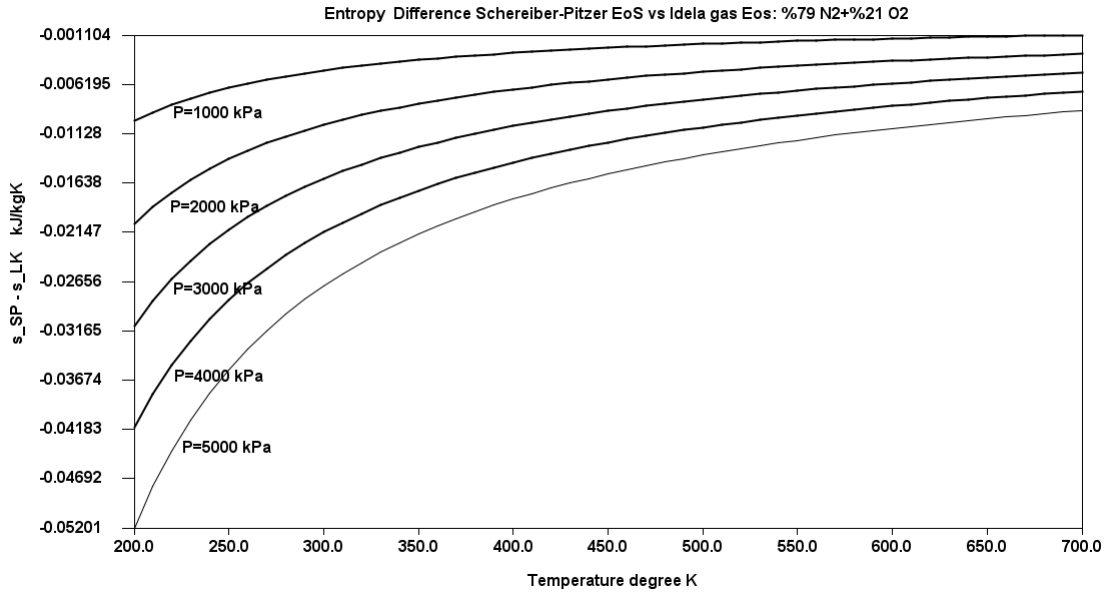


Figure 4.3 Entropy difference of Schereiber-Pitzer and perfect gas EOS

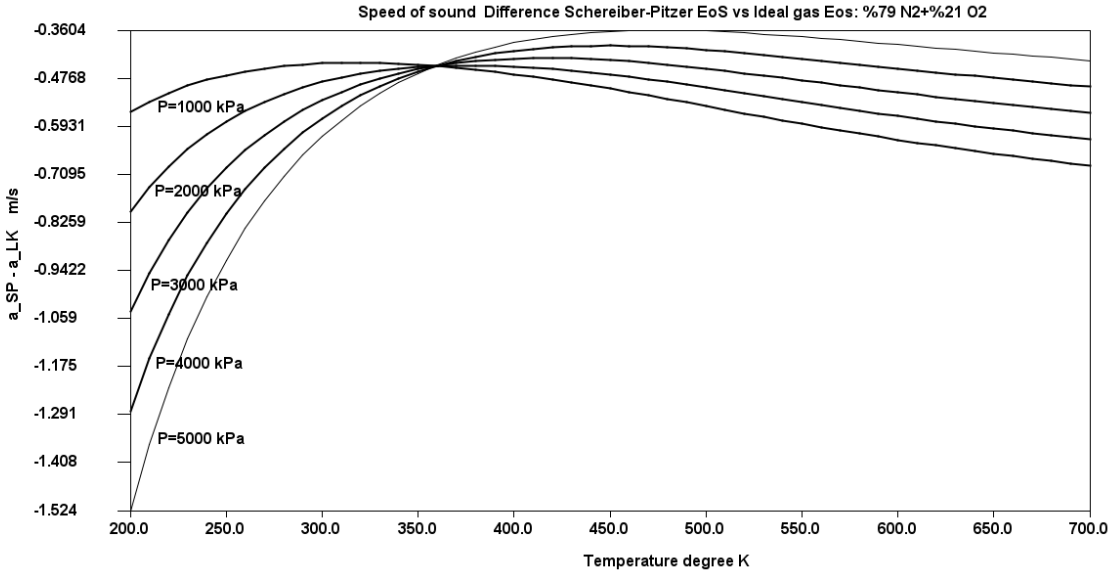


Figure 4.4 Speed of sound difference of Schereiber-Pitzer and perfect gas EOS

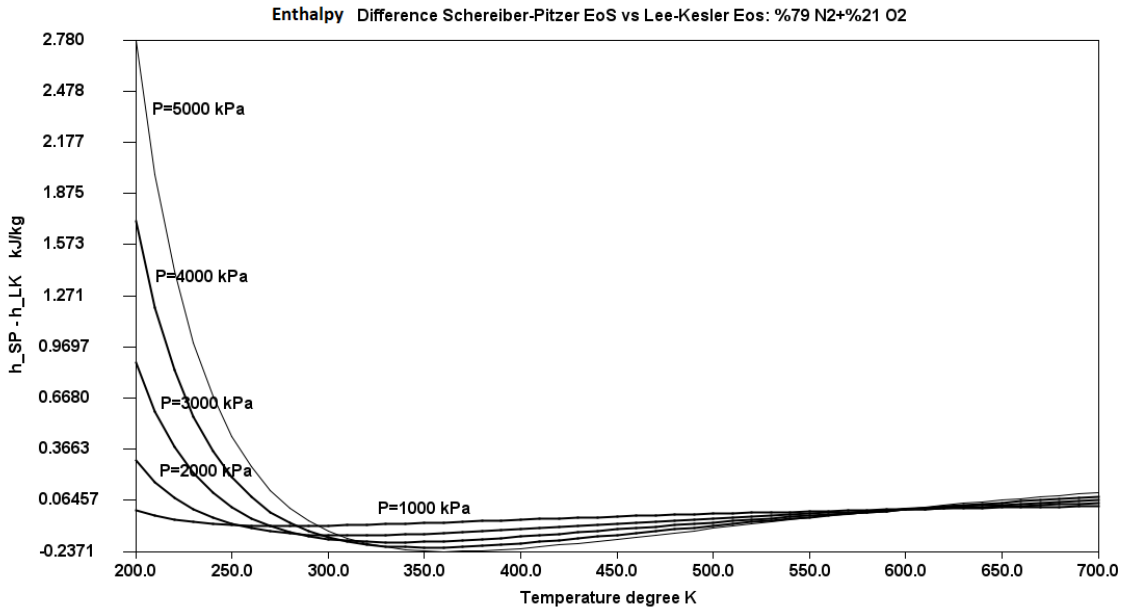


Figure 4.5 Enthalpy difference of Schereiber-Pitzer and Lee-Kesler EOS

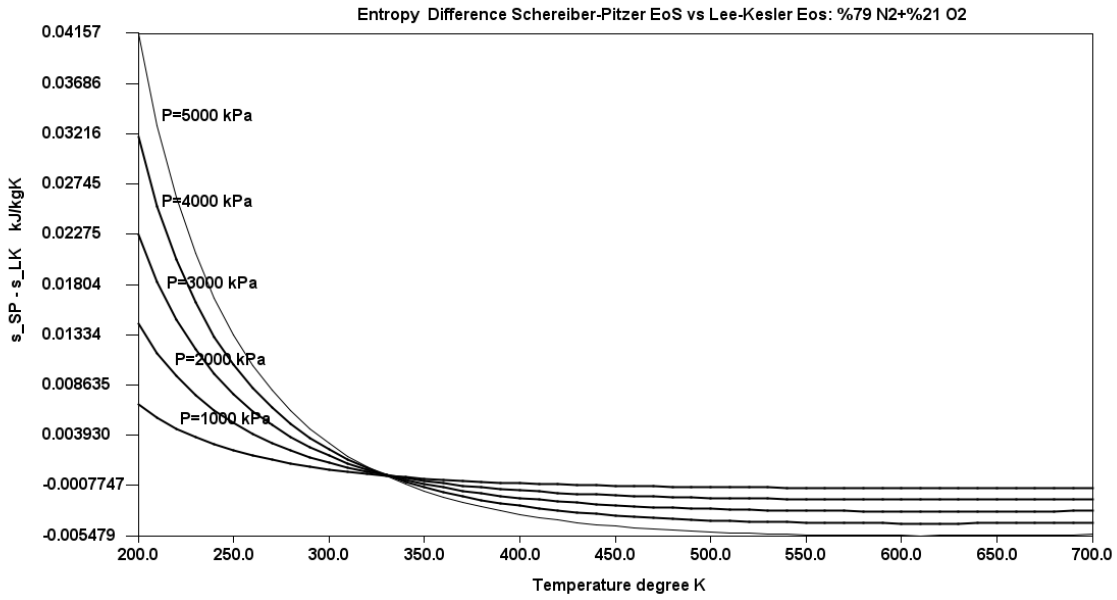


Figure 4.5 Entropy difference of Schereiber-Pitzer and Lee-Kesler EOS

2.4 ISO 17584-2005 EQUATION OF STATE

International Standard organisation defined a standard equation of state for refrigerants. So far the standard is unfinished and does not include properties of all refrigerants but still the best resource to define their properties. The standart includes the following refrigerants: R744(Carbondiokside), R717(Ammmonia), R12(dichlorodiflorometane), R22(chlorodiflorometane), R32(Diflorometane), R123(2,2dichloro,1,1,1trifloroetane), R125(Pentafloroetane), R134a(1,1,1,2tetrafloroethane), R143a(1,1,1,trifloroetane), R152a(1,1 difloroetane), R404A-R125/142a/134a(23/25/52), R407C-R32/125/134a(23/25/52), R410A-R32/125(50/50), R507a-R125/143A(50/50) The standart based on Helmholts free enegy type equation of state for pure refrigerants:

$$\emptyset = \frac{A}{RT} = \emptyset_{id} + \emptyset_r \qquad (2.4.1)$$

"id" in the equation indicates ideal gas and r indicates real gas part of the equation of state.

$$\emptyset_r = \sum_k N_k/\tau^{t_k}\,\delta^{d_k} exp[-\alpha_k(\delta-\varepsilon_k)^{l_k}]exp[-\beta_k(\tau-\gamma_k)^{m_k}] \quad (2.4.2)$$

In this equation

τ	Non-dimensional temperature parameter T^*/T
T^*	Normalisation factor, usually equal to critical temperature
δ	Non-dimensional density ρ/ρ^* $(1/(v\,\rho^*)$
ρ^*	Normalisation factor, usually equal to critical density
N_k	Coefficients of the equations
$\alpha_k,\beta_k,\varepsilon_k,\gamma_k$	Coefficients of the equations
t_k,d_k,l_k,m_k	Power coefficients of the equation

Ideal gas term of the equation :

$$\emptyset_{id} = \frac{h_{ref}}{RT} - \frac{s_{ref}}{R} - 1 + ln\left(\frac{RT\rho}{P_{ref}}\right) + \frac{1}{RT}\int_{T_{ref}}^{T} C_{p,id}\,dT\,\frac{1}{R} - \int_{T_{ref}}^{T}\frac{C_{p,id}}{T}\,dT \quad (2.4.3)$$

In this equation

h_{ref} : ideal gas enthalpy referance state ()

s_{ref} : İdeal gas entropy reference state (Usually taken as 1 kJ/kg K at 0 C liquid state entropy)

h_{ref} is usually selected as 200 kJ/kg at 0 C liquid . This equation also requires ideal gas specific heat values. These values can be computed as:

$$\frac{C_{p,id}}{R} = c_0 + \sum_{i=0}^{n} c_k T^{t_k} + \sum_k \frac{u_k^2 \exp(u_k)}{2} \quad (2.4.4)$$

In this term

$u_k = \frac{b_k}{T}$, c_k, a_k and t_k are coefficients

For some fluids additional components for the ideal gas part of the equation of state is defined as:

$$\emptyset_{id} = d_1 + d_2\tau + \ln(\delta) + d_3\ln(\tau) + \sum_k d_k\tau^{t_k} + \sum_k a_k ln[1-\exp(-\tau\gamma_k)] \quad (2.4.5)$$

In the equation d_1, d_2 ,d_3, d_k ,λ_k, t_k are coefficients. Equation (2.4.5) is same as equation (2.4.3) . An equivalent Cp value can also be written as:

$$\frac{C_{p,id}}{R} = d_0 + 1 - \sum_{i=0}^{n} d_k t_k (t_k - 1)\left(\frac{T^*}{T}\right) + \sum_k a_k \frac{u_k^2 \exp(u_k)}{[\exp(u_k)-1]^2} \quad (2.4.6)$$

For some refrigerants a third term is added to the Helmholts equation of state as

$$\emptyset = \frac{A}{RT} = \emptyset_{id} + \emptyset_r + \emptyset_{crt} \quad (2.4.1a)$$

$$\emptyset_{crt} = \sum_k N_k \delta \Delta^{b_k}\psi \quad (2.4.\ 7)$$

In here

$$\Delta = \theta^2 + B_k[(\delta-1)^2]^{a_k} \quad (2.4.8)$$

$$\theta = (1-\tau) + A_k[(\delta-1)^2]^{1/(2\beta_k)} \quad (2.4.9)$$

$$\psi = exp[-C_k(\delta-1)^2 - D_k(\tau-1)^2] \quad (2.4.10)$$

In equation N_k, A_k, B_k, C_k, D_k, α_k, β_k terms are coefficients determined by curve fittings data. As an alternative, if Benedict-Webb-Rubin eqution of state is given, this equation can be converted into Helmholts free energy form as:

$$P = \sum_{k=1}^{9} a_k\rho^k + \exp\left(-\frac{\rho^2}{\rho_{crit}^2}\right)\sum_{k=10}^{15} a_k\rho^{2k-1} \quad (2.4.11)$$

In this equation a_k are coefficients. Conversion factors are:

$$P = -\left(\frac{\partial A}{\partial V}\right)_T \quad (2.4.12)$$

$$\phi_r = \frac{A_r(T,\rho)}{RT} = -\int_V^\infty \left(\frac{P}{RT} - \rho\right) dV \quad (2.4.13)$$

This equation can be used together with the ideal gas part of the equation to get the complete equation of state. When equation of state is obtained, the remaining thermodynamic properties can be obtained by deriving Helmholts equation of state as:

$$P = RT\rho\left(1 + \delta\frac{\partial \phi_r}{\partial \delta}\right) \quad (2.4.14)$$

$$u = RT\left(\tau\frac{\partial \phi_{id}}{\partial \tau} + \tau\frac{\partial \phi_r}{\partial \tau}\right) \quad (2.4.15)$$

$$u = RT\left(1 + \tau\frac{\partial \phi_{id}}{\partial \tau} + \tau\frac{\partial \phi_r}{\partial \tau} + \delta\frac{\partial \phi_r}{\partial \delta}\right) \quad (2.4.16)$$

$$s = R\left(-(\phi_{id} + \phi_r) + \tau\frac{\partial \phi_{id}}{\partial \tau} + \tau\frac{\partial \phi_r}{\partial \tau}\right) \quad (2.4.17)$$

$$g = RT\left(1 + \phi_{id} + \phi_r + \delta\frac{\partial \phi_r}{\partial \delta}\right) \quad (2.4.18)$$

$$C_v = R\left(-\tau^2\frac{\partial^2 \phi_{id}}{\partial \tau^2} - \tau^2\frac{\partial^2 \phi_r}{\partial \tau^2}\right) \quad (2.4.19)$$

$$C_p = C_v + R\frac{\left(1 + \delta\frac{\partial \phi_r}{\partial \delta} - \delta\tau^2\frac{\partial^2 \phi_r}{\partial \tau \partial \delta}\right)}{\left(1 + 2\delta\frac{\partial \phi_r}{\partial \delta} + \delta^2\frac{\partial^2 \phi_r}{\partial \delta^2}\right)} \quad (2.4.20)$$

In the **saturation region** the following equations can be used to obtain saturation thermodynamic properties.

$$P\left(\tau, \delta_{saturated\ liquid}\right) = P\left(\tau, \delta_{saturated\ vapor}\right) \quad (2.4.21)$$

$$g\left(\tau, \delta_{saturated\ liquid}\right) = g\left(\tau, \delta_{saturated\ vapor}\right) \quad (2.4.22)$$

These saturation properties can also be obtain by using cubic spline interpolation from the original data. Cubic spline interpolation was given previously as a method to define specific heat data for ideal gas.

If our refrigerant is not a pure fluid, but a mixture of the fluids, by using mixing rules, mixture thermodynamic properties can be obtained from the pure fluid properties .

$$\varphi_{mix} = \frac{A}{RT} = \varphi_{mix,id} + \varphi_{mix,r} \quad (2.4.23)$$

$$\varphi_{mix,id} = \sum_{i=1}^{n}\left[x_i\varphi_{i,id} + x_i\, ln\, x_i\right] + f_3 + \frac{f_4}{T} \quad (2.4.24)$$

x_i component molar ratio

$$x_i = \frac{N_i}{\sum_{i=1}^{N} N_i} \quad (2.4.24a)$$

$x_i \ln x_i$ mixing term coming from entropy term.

f_3 and f_4 f_4 are used to shift the thermodynamic surface such that the reference state for enthalpy is 200 kJ/kg and entropy is 1 kJ/(kg·K) at the saturated liquid at 0 °C, similar to that done for the pure
fluids. Setting the parameters f_3 and f_4 to zero corresponds to a reference state based solely on the constituents of the mixture.

The residual part is given by Equation

$$\varphi_{mix,r} = \sum_{i=1}^{n} x_i \varphi_{i,r} + \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} x_i x_j \varphi_{ij} \quad (2.4.25)$$

$$T^* = \sum_{i=1}^{n} x_i T_i^* + \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} x_i x_j \zeta_{ij} \quad (2.4.26)$$

$$\delta = \frac{\rho_{mix}}{\rho^*} \quad (2.4.27)$$

$$\frac{1}{\rho^*} = \sum_{i=1}^{n} \frac{x_i}{\rho_i^*} + \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} x_i x_j \xi_{ij} \quad (2.4.28)$$

The parameters ξ_{ij} and ζ_{ij} are interaction parameters due to coexistance of two fluids together. T_i^* and ρ_i^* are coefficients of pure fluids, and taken usually as critical values . ϕ_{ij} function calculated as:

$$\varphi_{ij} = F_{ij} \sum_k N_k \delta^{d_k} \tau^{t_k} exp(-\delta^{l_k}) \quad (2.4.29)$$

As an example of coefficients of the equations, coefficients R744, carbondioxide is given in the tables below. For further coefficients, please refer to ISO 17584:2005(E) standard document.

Table 2.4.1 R744(carbondioxide) coefficients

k	a_k	b_k	c_k
0	—	—	3,5
1	1.99427042	958.49956	—
2	0.621052475	1858.80115	—
3	0.411952928	2061.10114	—
4	1.04028922	3443.89908	—
5	0.083276775	8238.20035	—

Table 2.4.2 R744(carbondioxide) coefficients

k	N_k	t_k	d_k	l_k	α_k	m_k	β_k	γ_k	ε_k
1	3.885682320320E-01	0	1	0	0	.	.		
2	2.938547594270E+00	0.75	1	0	0	.	.		
3	5.586718853490E+00	1	1	0	0	.	.		
4	7.675319959250E-01	2	1	0	0	.	.		
5	3.172900558040E-01	0.75	2	0	0	.	.		
6	5.480331589780E-01	2	2	0	0	.	.		
7	1.227941122030E-01	0.75	3	0	0	.	.		
8	2.165896154320E+00	1.5	1	1	1	.	.		
9	1.584173510970E+00	1.5	2	1	1	.	.		
10	2.313270540550E-01	2.5	4	1	1	.	.		
11	5.811691643140E-02	0	5	1	1	.	.		
12	5.536913720540E-01	1.5	5	1	1	.	.		
13	4.894661590940E-01	2	5	1	1	.	.		
14	2.427573984350E-02	0	6	1	1	.	.		
15	6.249479050170E-02	1	6	1	1	.	.		
16	1.217586022520E-01	2	6	1	1	.	.		
17	3.705568527010E-01	3	1	2	1	.	.		
18	1.677587970040E-02	6	1	2	1	.	.		
19	1.196073663800E-01	3	4	2	1	.	.		
20	4.561936250880E-02	6	4	2	1	.	.		
21	3.561278927030E-02	8	4	2	1	.	.		
22	7.442772713210E-03	6	7	2	1	.	.		
23	1.739570490240E-03	0	8	2	1	.	.		
24	2.181012128950E-02	7	2	3	1	.	.		
25	2.433216655920E-02	12	3	3	1	.	.		
26	3.744013342350E-02	16	3	3	1	.	.		
27	1.433871575690E-01	22	5	4	1	.	.		
28	1.349196908330E-01	24	5	4	1	.	.		

29	2.315122505350E-02	16	6	4	1	.	.		
30	1.236312549290E-02	24	7	4	1	.	.		
31	2.105832197290E-03	8	8	4	1	.	.		
32	3.395851902640E-04	2	10	4	1	.	.		
33	5.599365177160E-03	28	4	5	1	.	.		
34	3.033511805560E-04	14	8	6	1	.	.		
35	2.136548868830E+02	1	2	2	25	2	325	1.16	1
36	2.664156914930E+04	0	2	2	25	2	300	1.19	1
37	2.402721220460E+04	1	2	2	25	2	300	1.19	1
38	2.834160342400E+02	3	3	2	15	2	275	1.25	1
39	2.124728440020E+02	3	3	2	20	2	275	1.22	1
k	N_k	a_k	b_k	β_k	A_k	B_k	C_k	D_k	
40	6.664227654080E-01	3.5	0.875	0.3	0.7	0.3	10	275	
41	7.260863234990E-01	3.5	0.925	0.3	0.7	0.3	10	275	
42	5.506866861280E-02	3	0.875	0.3	0.7	1	12.5	275	

R744 Normalization factors:

T^* = 304.128 2 K, ρ* = 10.6249063 mol/l, M = 44.009 8 g/mol, R = 8.31451 J/(mol·K)

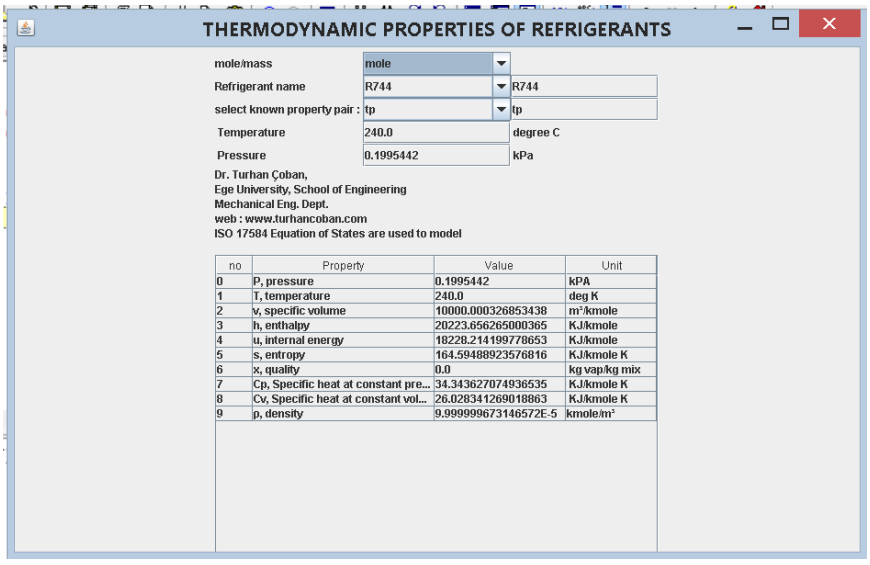
R744 Referance parameters

T_{ref} = 273.15 K, p_{ref} = 1.0 kPa, h_{ref} = 21389.328 J/mol, s_{ref} = 155.741 4 J/(mol·K), f_1 = 5.805 551 35, f_2 = 1555.79710

This EOS is used as a part of package refrigerant.java and induvidual program refISO.java which is listed in A5. A user interface of refISO.java is prepared as refTableISO, which has the followng output:

Program List of refISO EOS:

refISO	Basic Equation of state
refISOTable	Graphic user interface program
refISOModel	A subprogram for refrigerants
refrigerant.java	Multi-equation of state based refrigerant properties including refISO
RefrigerantData	Data reading and processing for refrigerant
ref_CS3	Saturatioon properties of refrigerants by using cubic spline curve fitting
refISO17584Data	Data reading and processing for refISO17584
refISO17584	Basic Equation of state
refISO17584Mix	Basic Equation of state for mixtures
refISO17584Model	A subprogram for refISO17584
refISO17584Table	Graphic user interface program



2.5 HELMHOLTZ EQUATION OF STATE FOR AMMONIA

As a third example of Helmholtz type equation of states Ammonia equation will be investigated.

$A(\rho,T) = A_{id}(T) + A_r(\rho,T)$ (2.5.11)

In the equation, "id" indicates ideal gas part, and "r" indicates real gas part of the equation of state. A =U- TS (2.5.2) is Helmholz free energy

U-Internal energy (kJ/kg) S-Entropy(kj/kg K) T-Temperature (K)

$A_{id}(T)$ –Ideal gas helmholtz free energy

$$A(\rho,T) = A_{id}(T) + A_r(\rho,T) = A_{id}(T) + RT[ln(\rho) + \rho Q(\rho,\tau)] \quad (2.5.2)$$

$$A_{id}(T) = RTC_1 \ln(T) + RT\sum_{i=2}^{11} C_i T^{i-3} - RT(1-\ln(RT)) \quad (2.5.3)$$

$$Q(\rho,\tau) = \sum_{i=1}^{9}\sum_{j=1}^{6} B_{ij}\rho^{i-1}(\tau-\tau_{crit})^{j-1} \quad (2.5.4)$$

In these τ=500/T, T K denotes temperature in Kelvin scale, Gas constant has the value of R=1.2333498 kJ/kgK. B_{ij} and C_i constants in the equation. ρ is density. B_{ij} and C_i values are given as tables.

Tablo 2.5.3 : Coefficient of Helmholtz free energy equation for ammonia

B_{ij}	1	2	3	4	5	6	7	8	9
1	-6.453022	8.0800944	1.0329949	-8.948265	-66.92286	247.34175	-306.5579	161.791	-27.82169
2	-13.71993	14.35692	55.843956	-169.7777	-1.753944	299.98392	24.116551	-507.4781	298.81292
3	-8.10062	-45.05298	492.01665	-1236.532	208.55337	4509.0806	-9323.357	8139.4704	-2772.597
4	-4.880096	-166.189	1737.836	-7812.161	21348.947	-37980.85	42724.099	-27458.71	7668.9287
5	-12.02878	37.90895	-30.87492	1.7795483	0	0	0	0	0
6	6.8063459	-40.73021	71.48353	-38.97461	0	0	0	0	0

Coeeficients of ideal gas ,A_{id}(T, helmholtz free energy is given in the next table.

Tablo 2.5.4: Coefficient of ideal gas Helmholtz free energy equation for steam

i	C_i	i	C_i	i	
1	-0.38727270E+01	5	0.86890833E-05	9	0.20563027E-16
2	0.64463724	6	-0.24085149E-07	10	-0.68534200E-20
3	0.32238759E+01	7	0.36893175E-10	11	0.99392427E-24
4	-0.21376925E-02	8	-0.35034664E-13		

The remaining thermodynamic properties are defined as derivatives of Helmholtz free energy equation of state.

$$P = \rho RT\left[1+\rho Q(\rho,T)+\rho^2\frac{\partial Q(\rho,T)}{\partial\rho}\right] \quad (2.5.15)$$

$$S = S_{id}(T) - R\left[\ln(\rho)+\rho Q(\rho,T)+\rho\tau\frac{\partial Q(\rho,T)}{\partial\tau}\right] \quad (2.5.16)$$

$$U = U_{id}(T) + RT\rho\tau\left[\frac{\partial Q(\rho,T)}{\partial\tau}\right] \quad (2.5.17)$$

$$H = U + \frac{P}{\rho} \quad (2.5.18)$$

The terms $\frac{\partial Q(\rho,T)}{\partial \rho}$ and $\frac{\partial Q(\rho,T)}{\partial \tau}$ is calculated by using below formulations:

$$\frac{\partial Q(\rho,T)}{\partial \tau} = \sum_{i=1}^{9}\sum_{j=1}^{6} B_{ij}(j-1)\rho^{i-1}(\tau-\tau_{crit})^{j-2} \qquad (2.5.19)$$

$$\frac{\partial Q(\rho,T)}{\partial \rho} = \sum_{i=1}^{9}\sum_{j=1}^{6} B_{ij}(i-1)\rho^{i-2}(\tau-\tau_{crit})^{j-1} \qquad (2.5.20)$$

Internal energy(U),entalphy(H) ve and (S) eqaution references can be calculated as

$S^0(T) = -dA^0/dT$ (2.5.21)

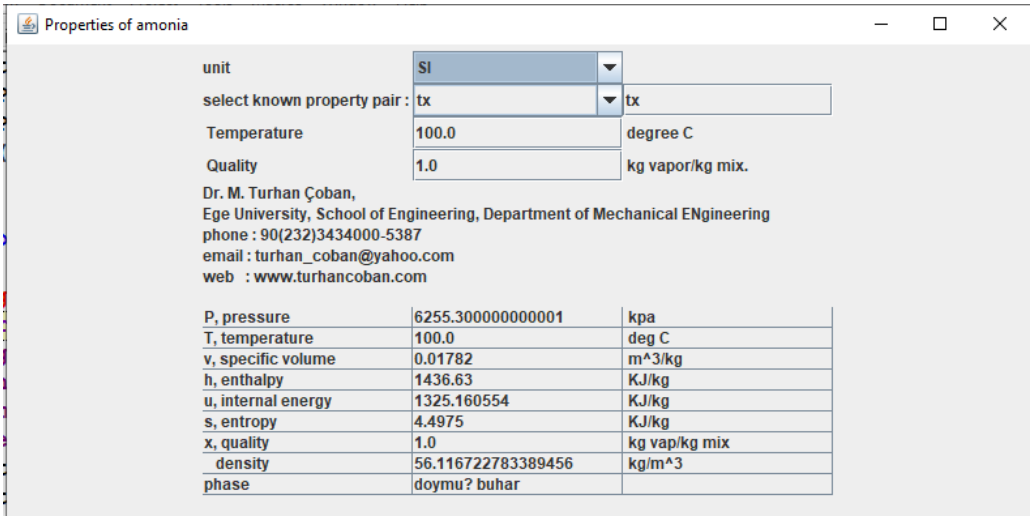
$$S^0(T) = -R\,C_1(\ln(T)+1) - R\sum_{i=2}^{11}(i-2)Ci\,T^{i-3} - R(\ln(RT)) \qquad (2.5.21)$$

$U^0(T) = A^0(T) + T\,S^0(T)$ (2.5.22)

Ammonia Helmholtz EOS is given in refrigerant.java and ammonia.java programs.

Program List of Ammonia Helmholtz EOS:

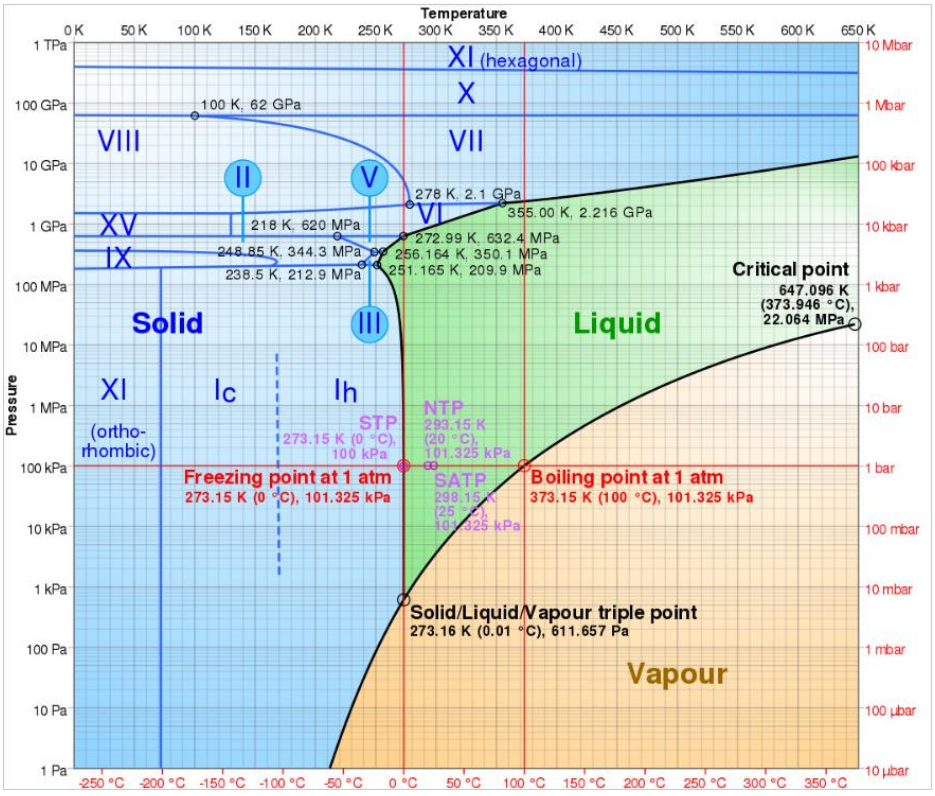
ammonia	Basic Equation of state
ammoniaTable	Graphic user interface program
amoniaModel	A subprogram for ammonia



2.6 STEAM(WATER-ICE) EQUATION OF STATES

Steam is one of the most important and most used working fluid in thermodynamic systems. More than one equation of state will be introduced in this chapter. One reason is historical development of steam EoS and related developments by author. Industrial revolution started by using steam as a thermodynamic fluid, therefore need of measurement of thermodynamic properties and creating EoS is established. The First version will be given here is known as Joseph H. Keenan, Frederick G. Keyes, Philip G. Hill, Joan G. Moore[70] version. Authors published their eqution of state and results in book format. Author of this book is used this EoS as a part of his PhD work in university of Utah(1983) . EoS codes were developed in Fortran language in that time. Later while author of this book teaching thermodynamics in Universirty of Nebraska Lincoln a new version in java language is created(2000). In that time US schools were teaching thermodynamic by using both english and US system, so this program developed accordingly. In relatively recent years International Association of water

and steam developed new equation of states. The property measurement range and accuracy of measurements are extended compare to previous tables, therefore set of these EoS are also written as programs in java language at Ege University , Turkey(2008,2015,2020). IAPWS equations are also cover ice regions. In thermodynamics we usually utilised liquids and gases, but for some problems ineraction zones of solid and liquid (or vapor) is important. Ice regions are also important ice growing phase change problems in heat transfer. Water ice has different crystal structures, some of them existed in high temperatures and very high pressures. A T-P diagram of water-steam systems are given below to Show phases of vapor , liquid and ice states of water.



Source: https://en.wikipedia.org/wiki/Phase_diagram#Binary_phase_diagrams
Figure 2.6.1 T-P diagram of water-steam-ice with several different crystal structures

A helmholtz free energy equation of state is investigated in the previous sections. The general form of Helmholtz equation can be written as

$A(\rho,T) = A_{id}(T) + A_r(\rho,T)$ (2.6.1)

In the equation, "id" indicates ideal gas part, and "r" indicates deviation of EoS from ideal gas states. A =U- TS (2.5.2) is Helmholz free energy

U-Internal energy (kJ/kg) S-Entropy(kj/kg K) T-Temperature (K)

$A_{id}(T)$ –Ideal gas helmholtz free energy. For steam tables equation of state used by J.H. Keenan, F.G. Keyes , P.G. Hill and J.G. Moore is utilized[70].

$$A(\rho,T) = A_{id}(T) + A_r(\rho,T) = A_{id}(T) + RT[\ln(\rho) + \rho Q(\rho,\tau)] \quad (2.6.2)$$

$$A_{id}(T) = \sum_{i=1}^{6} \frac{C_i}{\tau^{i-1}} + C_7 \ln(T) + C_8 \frac{\ln(T)}{\tau} \quad (2.6.3)$$

$$Q(\rho,\tau) = (\tau - \tau_c) \sum_{j=1}^{7} (\tau - \tau_{aj})^{j-2} \left[\sum_{i=1}^{8} B_{ij} (\rho - \rho_{aj})^{i-1} + \exp(-E\rho) \sum_{i=1}^{8} B_{ij}\, \rho^{i-9} \right] \quad (2.6.4)$$

In these τ =1000/T, T K denotes temperature in Kelvin scale, R=4.6151 bar cm^3/gram K or 0.46151 kJ/kgK. B_{ij} and C_i and E = 4.8 constants in yhe equation. ρ is density. B_{ij} and C_i values are given as tables.

$$\tau_{aj} = \tau_c \ (j = 1) \qquad \rho_{aj} = 0.634 \ (j = 1)$$
$$\tau_{aj} = 2.5 \ \ (j > 1) \qquad \rho_{aj} = 1 \ (j > 1)$$

$\tau_c = 1000/T_c$ (T_c critical temperature)

Tablo 2.6.1 : Coefficient of Helmholtz free energy equation for steam

B_{ij}	1	2	3	4	5	6	7
1	29.492937	-5.198586	6.833535	-0.15641	-6.397241	-3.96614	-0.690486
2	-132.1392	7.777918	-26.14975	-0.725461	26.409282	15.453061	2.740742
3	274.64632	-33.3019	65.326396	-9.273429	-47.74037	-29.14247	-5.102807
4	-360.9383	-16.25462	-26.18198	4.312584	56.32313	29.568796	3.963609
5	342.18431	-177.3107	0	0	0	0	0
6	-244.5004	127.48742	0	0	0	0	0
7	155.18535	137.46153	0	0	0	0	0
8	5.972849	155.97836	0	0	0	0	0
9	-410.3085	337.3118	-137.4662	6.787498	136.87317	79.84797	13.041253
10	-416.0586	-209.8887	-733.9685	10.401717	645.8188	399.1757	71.531353

Coefficients of ideal gas ,A_{id}(T, helmholtz free energy is given in the next table.

Tablo 2.6.2: Coefficient of ideal gas Helmholtz free energy equation for steam

i	C_i	i	C_i
1	1857.065	5	-20.5516
2	3229.12	6	4.85233
3	-419.465	7	46.0
4	36.6649	8	-1011.249

The remaining thermodynamic properties are defined as derivatives of Helmholtz free energy equation of state.

$$P = \rho^2 \left(\frac{\partial A}{\partial \rho}\right)_\tau \quad (2.6.5)$$
$$u = \left(\frac{\partial A\tau}{\partial \tau}\right)_\rho \quad (2.6.6)$$
$$s = -\left(\frac{\partial A}{\partial \tau}\right)_\rho \quad (2.6.7)$$

Values of specific enthalpy and specific Gibbs free energy are found in turn from their definition in the form of

$$h = u + Pv \quad (2.6.8)$$
$$g = A + Pv = u - Ts + Pv \quad (2.6.9)$$

Properties of saturated steam can be calculated as:

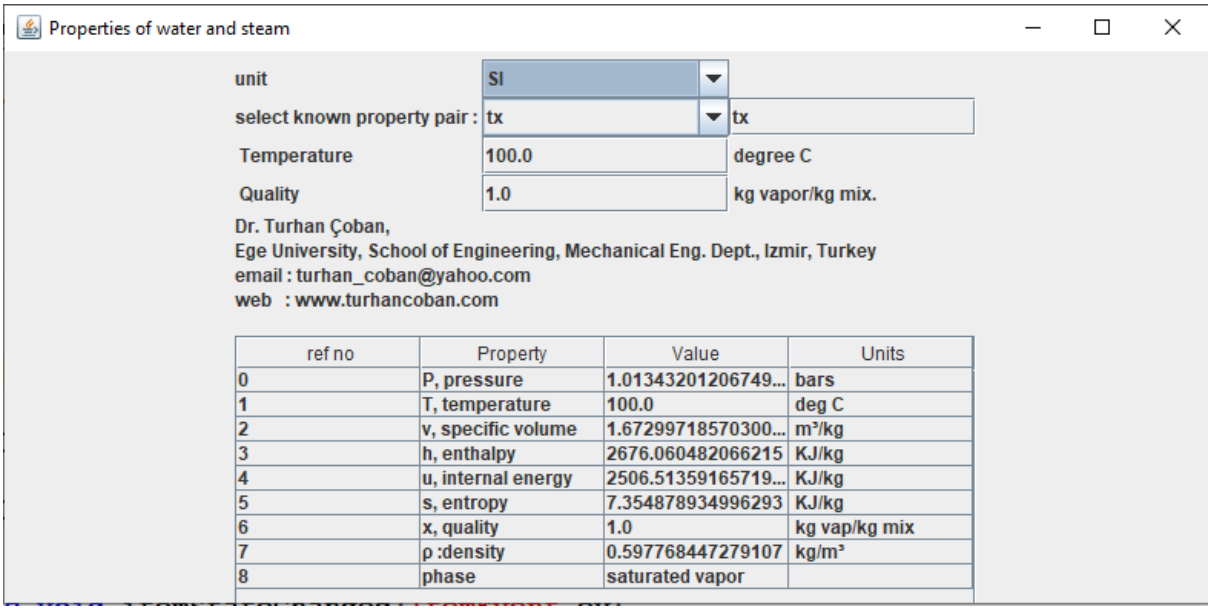
$$P_s = P_c exp\left[\tau 10^{-5}(T_c - T)\sum_{i=1}^{8} F_i(0.65 - 0.01T)^{i-1}\right] \quad (2.6.10)$$

F_i equation constants are given in the table below, T_c and P_c are critical properties :

Table 2.6.3 saturated steam equation constants.

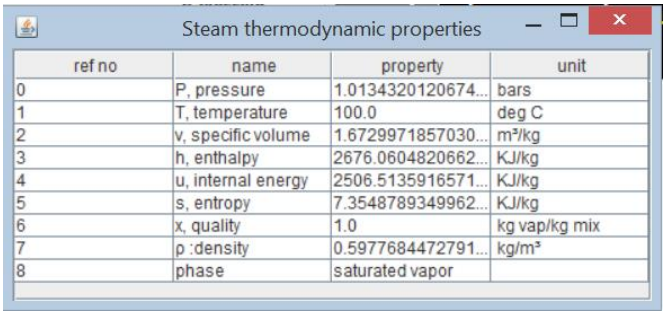
i	F_i	i	F_i	i	F_i	i	F_i
1	-741.9242	3	-11.55286	5	0.1094098	7	0.2520658
2	-29.721	4	-0.8685635	6	0.439993	8	0.05218684

A program based on this equation of state is developed to calculate properties of steam (Steam tables). Program code is given in A4. The program graphic user interface is given below:



Program 2.6.1 steamtest.java

```
public class steamtest
{
public static void main(String arg[])
{steam st=new steam();
 st.print("tx",100.0,1.0);
}}
```



Program 2.6.2 steamtest2.java plotting a T-s diagram constant pressure lines

```
 public class steamtest2
{
public static void main(String arg[])
{double pp[]={0.001,0.002,0.003,0.004,0.006,0.01,0.015,0.02,0.03,0.04,0.06,0.1,0.2,0.4,0.6,1,2,3,4,6,10,15,20};
double pp1[]={220.88,300,400,500,600,700,800,900,1000};
steam st=new steam();
double a[];
double dt=10.0;
double dt1=1.0;
int n=(int)((st.Tc-273.15-0.1)/dt1+0.000000000001);
```

```
double T[]=new double[n+2];
double s[]=new double[n+2];
double T1[];
double s1[];
double T2[];
double s2[];
double t=0.1;
int nn=st.ats.length;
double ts[]=new double[nn];
for(int i=0;i<nn;i++)
{ts[i]=st.ats[i]-273.15;}
Plot p=new Plot(st.ass,ts);
double t1=0;
int n1=0;
for(int j=0;j<pp.length;j++)
{ a=st.property("px",pp[j]*10,0.0);
  t1=a[1];
  n1=(int)((500-t1)/dt+0.00001);
  T1=new double[n1+2];
  s1=new double[n1+2];
  T1[0]=t1;
  s1[0]=a[5];
  a=st.property("px",pp[j]*10,1.0);
  T1[1]=t1;
  s1[1]=a[5];
for(int i=0;i<n1;i++)
{   t=t1+i*dt;
                a=st.property("tp",t,pp[j]*10);
                T1[i+2]=t;
                s1[i+2]=a[5];
}
p.addData(s1,T1);
}
for(int j=0;j<pp1.length;j++)
{ t1=(st.Tc-273.15-50);
  n1=(int)((500-t1)/dt1+0.00001);
  T2=new double[n1];
  s2=new double[n1];
for(int i=0;i<n1;i++)
{   t=t1+i*dt1;
                a=st.property("tp",t,pp1[j]);
                T2[i]=t;
                s2[i]=a[5];
}
p.addData(s2,T2);
}
p.plot();
}}
```

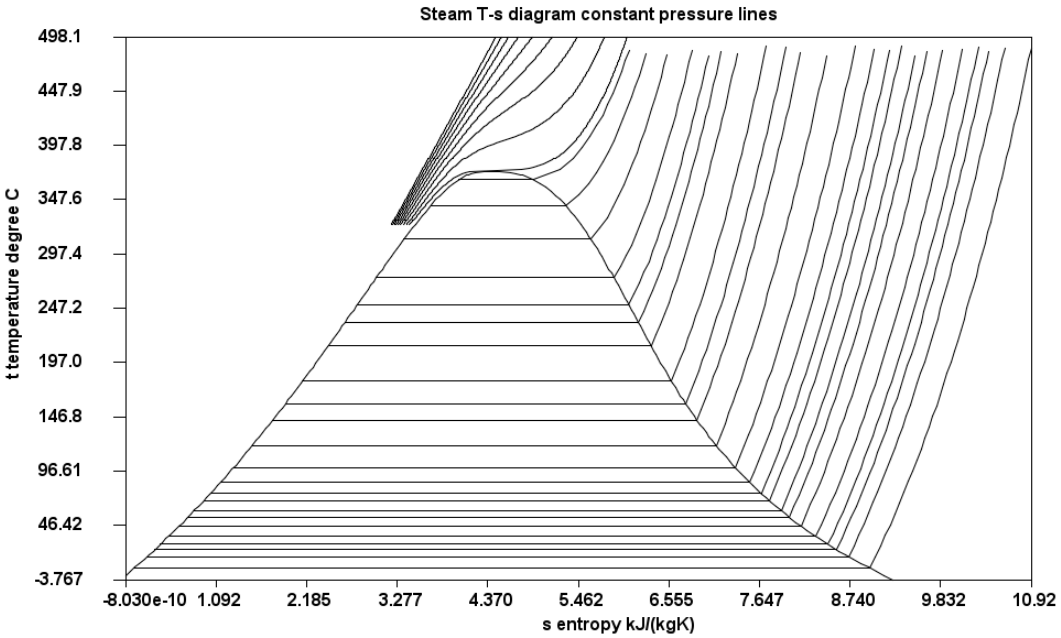


Figure 2.6.2 T-S diagram of steam created by class steam

Program List of steam EOS:

steam	Basic Equation of state
steamTable	Graphic user interface program
steamModel	A subprogram for steamTable

In previous section An EoS for steam is given. In recent years maximum operating temperatures and pressures of Rankine cycle power plants has increased. International Association for the Properties of Water and Steam(IAPWS) is developed a new set of equation of states which are more accurate and covers larger range of data. This new set of equations are developed in 1997[59]. Steam properties are given by 5 sets of equation of states, as shown in the Figure 2.6.1

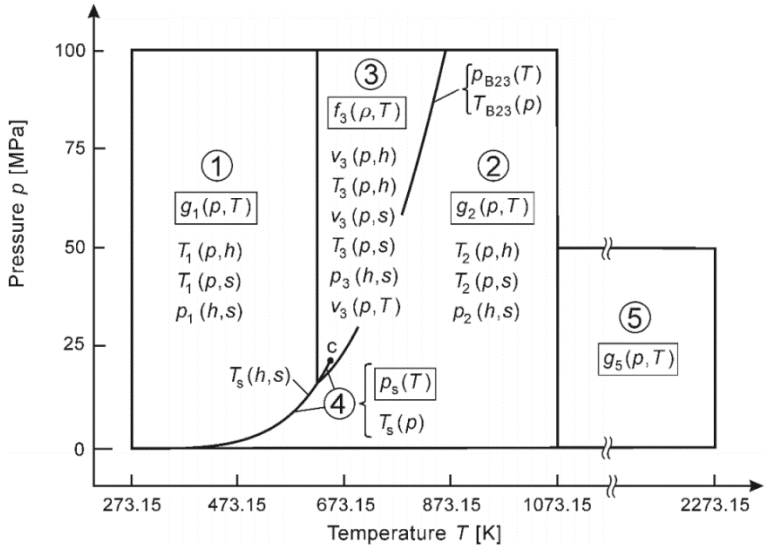


Figure 2.6.3 IAPWS 97 Equation of state regions for steam

The first equation, which covers basically liquid region has the following gibbs free energy form:

$$\frac{g_1(P,T)}{RT} = \gamma(\pi,\tau) = \sum_{i=1}^{34} n_i(71-\pi)^{I_i}(\tau-1222)^{J_i} \quad (2.6.11)$$

Where $\pi = \frac{P}{P^*}$ $\tau = \frac{T^*}{T}$ $p^{*=}$16.62 MPa and T^*=1386 K R=0461526 kJ/(kgK)

Table 2.6.4 coefficients of eqn. 2.6.11

i	I_i	J_i	n_i	i	I_i	J_i	n_i
1	0	-2	0.14632971213167	18	2	3	-4.4141845331E-06
2	0	-1	-0.84548187169114	19	2	17	-7.2694996298E-16
3	0	0	-3.75636036720400	20	3	-4	-3.1679644845E-05
4	0	1	3.38551691683850	21	3	0	-2.8270797985E-06
5	0	2	-0.95791963387872	22	3	6	-8.5205128120E-10
6	0	3	0.15772038513228	23	4	-5	-2.2425281908E-06
7	0	4	-0.01661641719950	24	4	-2	-6.5171222896E-07
8	0	5	0.00081214629984	25	4	10	-1.4341729938E-13
9	1	-9	0.00028319080124	26	5	-8	-4.0516996860E-07
10	1	-7	-0.00060706301566	27	8	-11	-1.2734301742E-09
11	1	-1	-0.01899006821842	28	8	-6	-1.7424871231E-10
12	1	0	-0.03252974877051	29	21	-29	-6.8762131296E-19
13	1	1	-0.02184171717541	30	23	-31	1.4478307829E-20
14	1	3	-0.00005283835797	31	29	-38	2.6335781663E-23
15	2	-3	-0.00047184321073	32	30	-39	-1.1947622640E-23
16	2	0	-0.00030001780793	33	31	-40	1.8228094581E-24
17	2	1	0.00004766139391	34	32	-41	-9.3537087292E-26

Thermodynamic relations can be calculated from these thermodynamic relations

Specific volume: $v = \left(\frac{\partial g}{\partial P}\right)_T$ (2.6.12)

Specific enthalpy: $h = g - T\left(\frac{\partial g}{\partial T}\right)_P$ (2.6.13)

Specific internal energy: $u = g - T\left(\frac{\partial g}{\partial T}\right)_P - P\left(\frac{\partial g}{\partial P}\right)_T$ (2.6.14)

Specific entropy: $s = \left(\frac{\partial g}{\partial T}\right)_P$ (2.6.15)

Specific isobaric heat capacity: $C_p = \left(\frac{\partial h}{\partial T}\right)_P$ (2.6.16)

Specific isochoric heat capacity: $C_v = \left(\frac{\partial u}{\partial T}\right)_v$ (2.6.17)

The second equation equation, which covers vapor region has the following gibbs free energy form:

$\frac{g_2(P,T)}{RT} = \gamma(\pi,\tau) = \gamma^0(\pi,\tau) + \gamma^r(\pi,\tau)$ (2.6.18)

Where $\pi = \frac{P}{P^*}$ $\tau = \frac{T^*}{T}$ R=0.461526 kJ/(kgK), $\gamma^0(\pi,\tau)$is the ideal gas part of EOS, and $\gamma^r(\pi,\tau)$is the real gas departure the EOS. İdeal gas part equation:

$$\gamma^0(\pi,\tau) = \ln(\pi) + \sum_{i=1}^{9} n_i^0 \tau^{J_i} \quad (2.6.19)$$

Where P^*=1MPa and T^*=540 K

Table 2.6.5 coefficients of eqn. 2.6.19

i	J_i	n_i^0	i	J_i	n_i^0
1	0	-9.692768650E+00	6	-2	1.4240819171E+00
2	1	1.008665597E+01	7	-1	-4.3839511319E+00
3	-5	-5.608791128E-03	8	2	-2.8408632461E-01
4	-4	7.145273808E-02	9	3	2.1268463753E-02
5	-3	-4.071049822E-01			

dimensionless residual part of the basic equation $g2(\ p,T\ )$ is as follows:

$$\gamma^r(\pi,\tau) = \sum_{i=1}^{43} n_i \pi^{I_i}(\tau-0.5)^{J_i} \quad (2.6.20)$$

Where P^*=1MPa and T^*=540 K

Table 2.6.6 coefficients of eqn. 2.6.20

i	I_i	J_i	n_i	i	I_i	J_i	n_i
1	1	0			7	0	-5.9059564324270E-18
2	1	1	-1.7834862292358E-02	24	7	11	-1.2621808899101E-06
3	1	2	-4.5996013696365E-02	25	7	25	-3.8946842435739E-02
4	1	3	-5.7581259083432E-02	26	8	8	1.1256211360459E-11
5	1	6	-5.0325278727930E-02	27	8	36	-8.2311340897998E+00
6	2	1	-3.3032641670203E-05	28	9	13	1.9809712802088E-08
7	2	2	-1.8948987516315E-04	29	10	4	1.0406965210174E-19
8	2	4	-3.9392777243355E-03	30	10	10	-1.0234747095929E-13
9	2	7	-4.3797295650573E-02	31	10	14	-1.0018179379511E-09
10	2	36	-2.6674547914087E-05	32	16	29	-8.0882908646985E-11
11	3	0	2.0481737692309E-08	33	16	50	1.0693031879409E-01
12	3	1	4.3870667284435E-07	34	18	57	-3.3662250574171E-01
13	3	3	-3.2277677238570E-05	35	20	20	8.9185845355421E-25
14	3	6	-1.5033924542148E-03	36	20	35	3.0629316876232E-13
15	3	35	-4.0668253562649E-02	37	20	48	-4.2002467698208E-06
16	4	1	-7.8847309559367E-10	38	21	21	-5.9056029685639E-26
17	4	2	1.2790717852285E-08	39	22	53	3.7826947613457E-06
18	4	3	4.8225372718507E-07	40	23	39	-1.2768608934681E-15
19	5	7	2.2922076337661E-06	41	24	26	7.3087610595061E-29
20	6	3	-1.6714766451061E-11	42	24	40	5.5414715350778E-17
21	6	16	-2.1171472321355E-03	43	24	58	-9.4369707241210E-07
22	6	35	-2.3895741934104E+01				

Region 3 equation is given as Helmholts free energy form:

$$\frac{f_3(\rho,T)}{RT} = \phi(\delta,\tau) = n_1 \ln(\delta) + \sum_{i=2}^{40} n_i \delta^{I_i} \tau^{J_i} \quad (2.6.21)$$

Where $\delta = \frac{\rho}{\rho^*}$ $\tau = \frac{T^*}{T}$, T^*=T_c=647.096 and R=0461526 kJ/(kgK)

Table 2.6.4 coefficients of eqn. 2.6.11

i	I_i	J_i	n_i	i	I_i	J_i	n_i
1	0	0	1.065807002851E+00	21	3	4	-2.0189915023570E+00
2	0	0	-1.573284529024E+01	22	3	16	-8.2147637173963E-03
3	0	1	2.094439697431E+01	23	3	26	-4.7596035734923E-01
4	0	2	-7.686770787872E+00	24	4	0	4.3984074473500E-02
5	0	7	2.618594778795E+00	25	4	2	-4.4476435428739E-01
6	0	10	-2.808078114862E+00	26	4	4	9.0572070719733E-01
7	0	12	1.205336969652E+00	27	4	26	7.0522450087967E-01
8	0	23	-8.456681281250E-03	28	5	1	1.0770512626332E-01
9	1	2	-1.265431547771E+00	29	5	3	-3.2913623258954E-01
10	1	6	-1.152440780668E+00	30	5	26	-5.0871062041158E-01
11	1	15	8.852104398432E-01	31	6	0	-2.2175400873096E-02
12	1	17	-6.420776518161E-01	32	6	2	9.4260751665092E-02
13	2	0	3.849346018667E-01	33	6	26	1.6436278447961E-01
14	2	2	-8.521470882421E-01	34	7	2	-1.3503372241348E-02
15	2	6	4.897228154188E+00	35	8	26	-1.4834345352472E-02
16	2	7	-3.050261725697E+00	36	9	2	5.7922953628084E-04
17	2	22	3.942053687915E-02	37	9	26	3.2308904703711E-03
18	2	26	1.255840842431E-01	38	10	0	8.0964802996215E-05
19	3	0	-2.799932969871E-01	39	10	1	-1.6557679795037E-04
20	3	2	1.389979956946E+00	40	11	26	-4.4923899061815E-05

It should be noted that this set of equation is function of density and temperature, and basic equation is helmholts equation so, let us list definition of other thermodynamic properties

Pressure: $P = \rho^2 \left(\frac{\partial f}{\partial \rho}\right)_T$ (2.6.12)

Specific enthalpy: $h = f - T\left(\frac{\partial f}{\partial T}\right)_p + \rho\left(\frac{\partial f}{\partial \rho}\right)_T$ (2.6.13)

Specific internal energy: $u = f - T\left(\frac{\partial f}{\partial T}\right)_p$ (2.6.14)

Specific entropy: $s = \left(\frac{\partial f}{\partial T}\right)_\rho$ (2.6.15)

Specific isobaric heat capacity: $C_p = \left(\frac{\partial h}{\partial T}\right)_p$ (2.6.16)

Specific isochoric heat capacity: $C_v = \left(\frac{\partial u}{\partial T}\right)_v$ (2.6.17)

Region 4 of the equation defines saturation region. The basic equation is given as a polynomial

$$\beta^2\vartheta^2 + n_1\beta^2\vartheta + n_2\beta^2 + n_3\beta\vartheta^2 + n_4\beta\vartheta + n_5\beta + n_6\vartheta^2 + n_7\vartheta + n_8 = 0 \quad (2.6.18)$$

Where

$$\beta = \left(\frac{P_S}{P^*}\right)^{0.25} \quad (2.6.19)$$

$$\vartheta = \frac{T_S}{T^*} + \frac{n_9}{\left(\frac{T_S}{T^*}\right) - n_{10}} \quad (2.6.20)$$

From this equation both saturation pressure and saturation temperature equation can be derived.

$$\frac{P_S}{P^*} = \left[\frac{2C}{-B + (B^2 - 4AC)^{0.5}}\right]^4 \quad (2.6.21)$$

Where P^*=1 MPa

$$A = \vartheta^2 + n_1\vartheta + n_2$$

$$B = n_3\vartheta^2 + n_4\vartheta + n_5$$

$$C = n_6\vartheta^2 + n_7\vartheta + n_8$$

Table 2.6.5 coefficients of eqn. 2.6.21

i	n_i	i	n_i
1	1.1670521453E+03	6	1.4915108614E+01
2	-7.2421316703E+05	7	-4.8232657362E+03
3	-1.7073846940E+01	8	4.0511340542E+05
4	1.2020824702E+04	9	-2.3855557568E-01
5	-3.2325550322E+06	10	6.5017534845E+02

It is also possible to drive saturation temperature equation from the basic polynomial as:

$$\frac{T_s}{T^*} = \frac{n_{10} + D - [(n_{10} + D)^2 - 4(n_9 + n_{10}D)]^{0.5}}{2} \quad (2.6.22)$$

Where T^*=1 K

$$D = \frac{2G}{-F - (F^2 - 4EG)^{0.5}}$$

$$E = \beta^2 + n_3\beta + n_6$$

$$F = n_1\beta^2 + n_4\beta + n_7$$

$$G = n_2\beta^2 + n_5\beta + n_8$$

And the final region for steam is region 5, again given as gibbs free equation type EOS

$$\frac{g_5(P,T)}{RT} = \gamma(\pi,\tau) = \gamma^0(\pi,\tau) + \gamma^r(\pi,\tau) \quad (2.6.23)$$

Where $\pi = \frac{P}{P^*}$ $\tau = \frac{T^*}{T}$ R=0.461526 kJ/(kgK), $\gamma^0(\pi,\tau)$is the ideal gas part of EOS, and $\gamma^r(\pi,\tau)$is the real gas difference of the EOS. İdeal gas part equation:

$$\gamma^0(\pi,\tau) = \ln(\pi) + \sum_{i=1}^{9} n_i^0 \tau^{J_i} \quad (2.6.24)$$

Where p^*=1MPa and T^*=1000 K

Table 2.6.6 coefficients of eqn. 2.6.24

i	J_i^0	n_i^0	i	J_i^0	n_i^0
1	0	-13.1799836742	4	-2	0.3690153498
2	1	6.8540841634	5	-1	-3.1161318214
3	-3	-0.0248051489	6	2	-0.3296162654

The real gas part of the equation

$$\gamma^r(\pi,\tau) = \sum_{i=1}^{43} n_i \pi^{I_i} \tau^{J_i} \quad (2.6.25)$$

Table 2.6.7 coefficients of eqn. 2.6.25

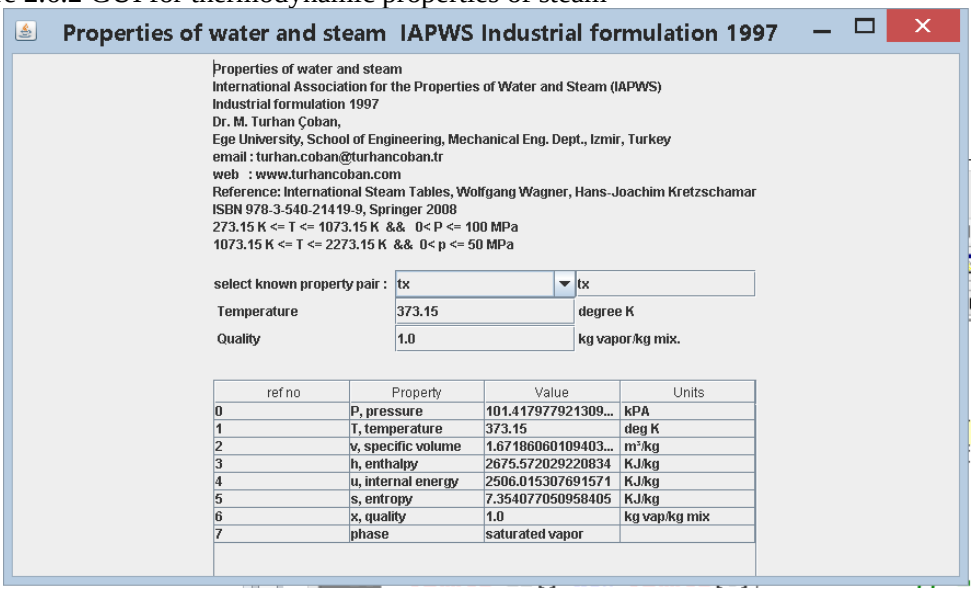
i	I_i	J_i	n_i	i	I_i	J_i	n_i
1	1	1	1.5736404855E-03	4	2	3	2.2440037409E-06
2	1	2	9.0153761674E-04	5	2	9	-4.1163275453E-06
3	1	3	-5.0270077678E-03	6	3	7	3.7919454823E-08

At reference [29] more detailed set of equations are given for this calculations, for example reverse equation sets are given so that instead of additional curve fitting to get the variables this additional sets can be utilized A program in java (steamIAPWS_IF97) is prepared to use this set of EOS. A user interface is also prepared (steamTableIF97.java). The user interface input frame is given in the following figure

Program List of IAPWS IF97 EOS:

steamIAPWS_IF97	Basic Equation of state
steamTableIF97	Graphic user interface program
steamModelIF97	A subprogram for steamTableIF97

Figure 2.6.2 GUI for thermodynamic properties of steam



Sample code to show utilisation of class:

```
 public class steamIAPWS_IF97_test1
{ public static void main(String arg[])
  {steamIAPWS_IF97 ap=new steamIAPWS_IF97();
   double T=300.0;  //degree K;
   double P=100.0;  //kPa
   double a[]=ap.property("tp",T,P);
   double h=a[3];
   double s=a[5];
   double Cp=a[7];
   String ss="T = "+T+" degree K"+" P = "+P+" kPa"+" h = "+h+" kJ/kg"+" s= "+s+"kJ/kgK "+"Cp ="+Cp+"kJ/kgK ";
   System.out.println(ss);
  }
}
```

```
---------- Capture Output ----------
> "E:\co\java\bin\java.exe" steamIAPWS_IF97_test1
T = 300.0 degree K P = 100.0 kPa h = 112.66382328242584 kJ/kg s= 0.3930970472619295kJ/kgK Cp
=4.181101061896756kJ/kgK

> Terminated with exit code 0.
```

Input known pairs of method property:

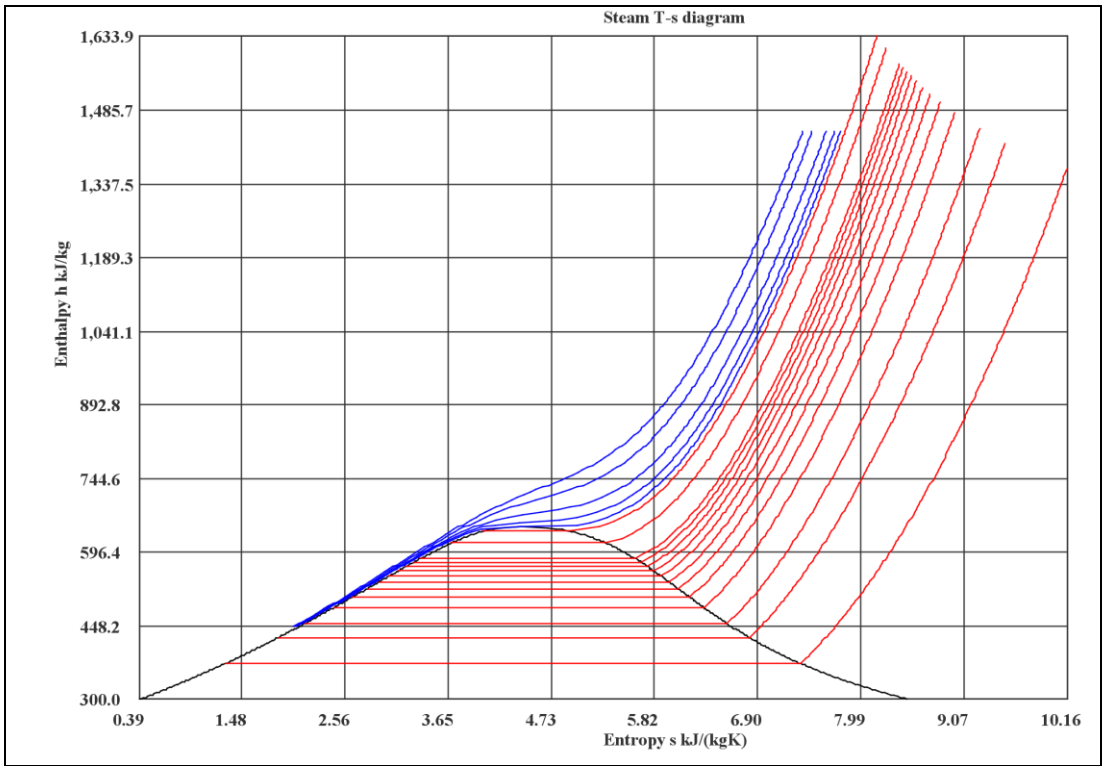
tp	tro	vp	hp
vt	pv	rop	ps
rot	pro	ph	sp

Output array of property method:

0	P, pressure	kPa
1	T, temperature	Degree K
2	v, specific volume	m^3/kg
3	h, enthalpy	kJ/kg
4	u, internal energy	kJ/kg
5	s, entropy	kJ/(kgK)
6	x, quality	kg vapor/kg total
7	Cp, specific heat at constant	kJ/(kgK)
8	Cv, specific heat at constant	kJ/(kgK)
9	α isobaric thermal expansion	1/K
10	κ isothermal compressibility	kPa
11	μ Dynamic viscosity	Pa.s
12	k Thermal conductivity	W/(mK)
13	Surface tension	N/m
14	Prandtl number	
15	density	kg/m^3
16	speed of sound	m/s

Sample code to plot T-s diagram by using steamIAPWS_IF97

```
 public class steamtest2B
{
public static void main(String arg[])
{steamIAPWS_IF97 st=new steamIAPWS_IF97();
double T2[];
double s2[];
double t=0.1;
int nn=st.ats.length;
double aa[][]=st.TS_data(300.0);
graph2D p=new graph2D();
p.addData(aa[0],aa[1]);
double pp[]={100,500,1000,2000,3000,4000,5000,6000,7000,8000,9000,10000,15000,20000};
double pp1[]={22064,25000,30000,40000,50000};
int n=200;
T2=new double[n];
s2=new double[n];
for(int j=0;j<pp.length;j++)
{   double a[]=st.property("px",pp[j],0.0);
    T2[0]=a[1];
    s2[0]=a[5];
    a=st.property("px",pp[j],1.0);
    T2[1]=a[1];
    s2[1]=a[5];
              for(int i=2;i<n;i++)
  {t=T2[0]+i*5;
   T2[i]=t;
   a=st.property("tp",t,pp[j]);
   s2[i]=a[5];
  }
  p.addData(s2,T2,0,"red");
}
for(int j=0;j<pp1.length;j++)
{   double t1=st.Tc-200;
              for(int i=0;i<n;i++)
  {t=t1+i*5;
   T2[i]=t;
   double a[]=st.property("tp",t,pp1[j]);
   s2[i]=a[5];
  }
  p.addData(s2,T2,0,"blue");
}
p.setPlabel("Steam T-s diagram");
p.setXlabel("Entropy s kJ/(kgK)");
p.setYlabel("Enthalpy h kJ/kg");
p.setGrid();
p.plot();
}}
```



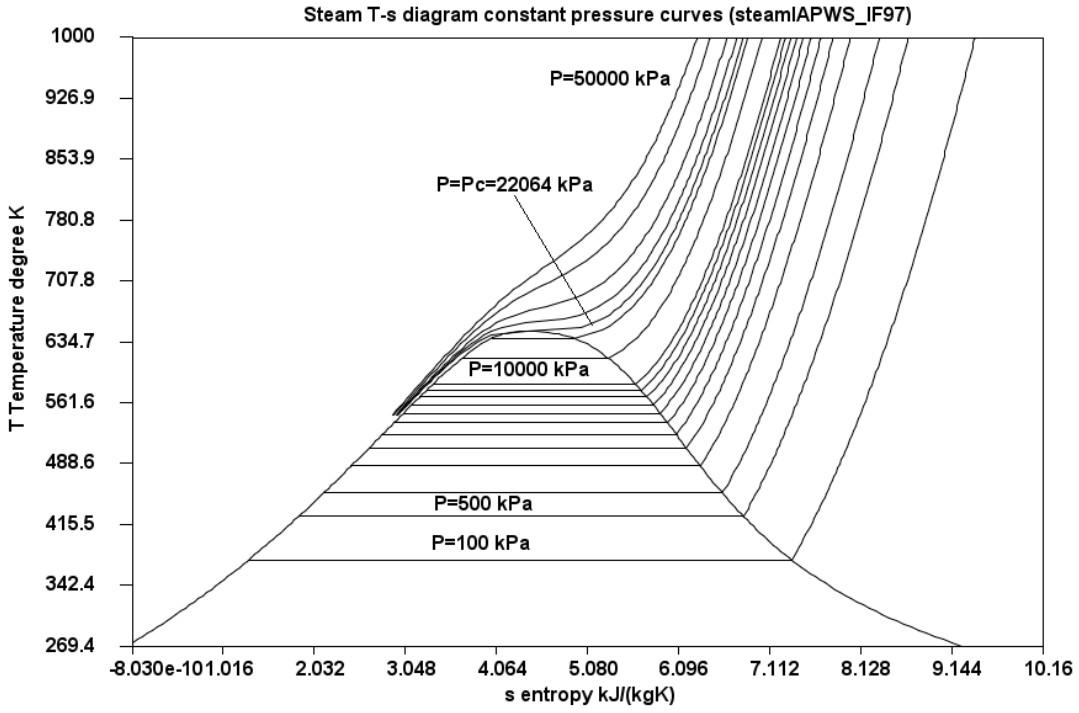


Figure 2.6.4 T-s diagram of steam created by using steamIAPWS_IF97

```
public class steamtest_sh
{
public static void main(String arg[])
{
steamIAPWS_IF97 st=new steamIAPWS_IF97();

double t=0.1;
int nn=st.ats.length;
graph2D p=new graph2D();
double Tmin=273.16;
double a1[][]=st.sh_data(Tmin);
p.addData(a1[0],a1[1]);
```

```
int n=10;
double T2[]=new double[n];
double x2[]=new double[n];
double s2[]=new double[n];
double h2[]=new double[n];
double P2[]=new double[n];
double xx[]={0.1,0.2,0.3,0.4,0.5,0.6,0.7,0.8,0.9};
T2[0]=273.16;
double dT=(st.Tc-T2[0])/n;
// quality lines
for(int j=0;j<xx.length;j++)
{ for(int i=0;i<n;i++)
   {t=t=T2[0]+i*dT;
    T2[i]=t;
    x2[i]=xx[j];
    double a[]=st.property("tx",T2[i], x2[i]);
    s2[i]=a[5];
    h2[i]=a[3];
   }
  p.addData(s2,h2,0,"blue");
}
//pressure lines
double
pp[]={10,50,100,500,1000,2000,3000,4000,5000,6000,7000,8000,9000,10000,15000,20000,22064,25000,30000,40000,500
00};

for(int j=0;j<pp.length;j++)
{   double a[]=st.property("px",pp[j],0.0);
    s2[0]=a[5];
    h2[0]=a[3];
    T2[0]=a[1];
    a=st.property("px",pp[j],1.0);
    s2[1]=a[5];
    h2[1]=a[3];
              for(int i=2;i<n;i++)
   { t=T2[0]+i*50;
     T2[i]=t;
   a=st.property("tp",t,pp[j]);
   s2[i]=a[5];
   h2[i]=a[3];
   }
  p.addData(s2,h2,0,"red");
}
p.setPlabel("Molliere h-s diagram");
   p.setXlabel("Entropy s kJ/(kgK)");
   p.setYlabel("Enthalpy h kJ/kg");
   p.setGrid();
p.plot();
}
}
```

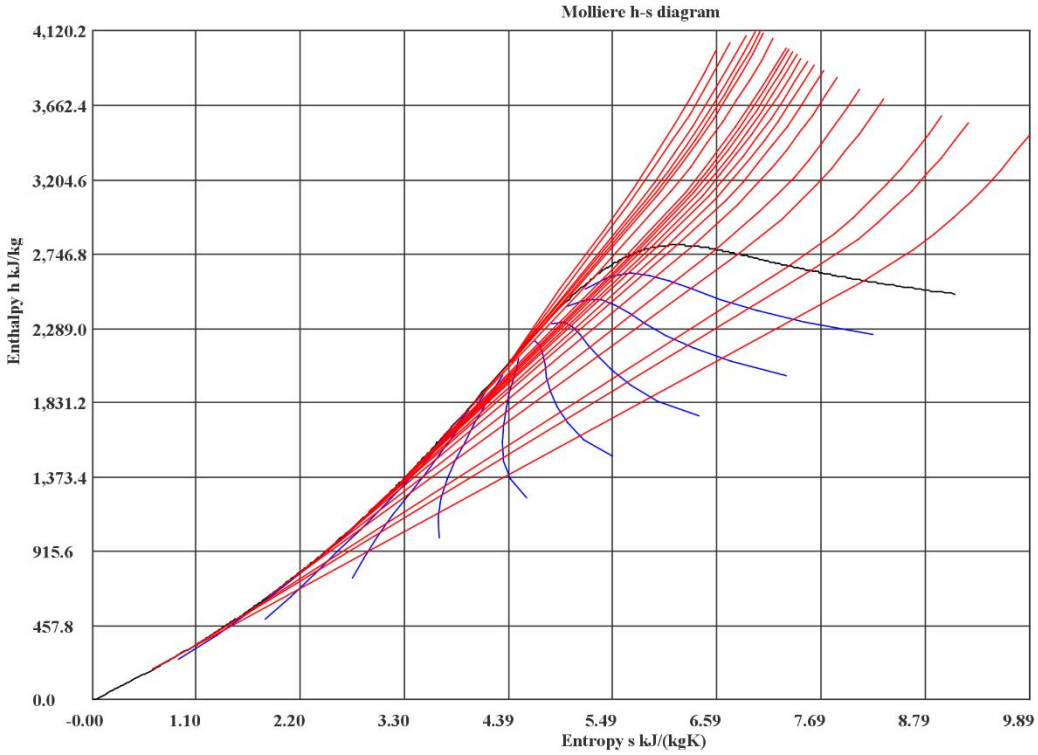


Figure 2.6.5 Mollier h-s diagram of steam created by using steamIAPWS_IF97

Another set of equation of state is given by International Association for the Properties of Water and Steam(IAPWS). It is called **IAPWS 95 EoS**[58]. The previous equation is for the industrial use for power plants industry, so it covers very high pressure and temperatures and furthermore divide the EOS to simple calculatable regions. IAPWS 95 is designed for scientific calculations. It has only one basics EoS to cover regions from ice to steam.
The water-vapor part is given by IAPWS-95 Helmholtz free energy for water and steam[2]. This equation has the following form:

$$\frac{f^V(T,\rho^V)}{RT} = \phi^V(\delta,\tau) = \phi^{V0}(\delta,\tau) + \phi^{Vr}(\delta,\tau) \quad (2.6.26)$$

Where $\delta = \rho^V/\rho_c^V$ and $\tau = T_c^V/T$ for steam and water T_c^V=647.096 K, ρ_c^V=322 kg/m^3 and R=0.46151805 kJ/(kgK). The equation for the ideal gas part of the Helmholtz free energy $\phi^{V0}(\delta,\tau)$ can be obtained by using the following equation:

$$\phi^{V0}(\delta,\tau) = \ln(\delta) + n_1^0 + n_2^0\tau + n_3^0 \ln(\tau) \sum_{i=4}^{8} n_i^0 ln\left[1 - e^{-(\gamma_i^o)^2}\right] \quad (2.6.27)$$

Table 2.6.8 Coefficients of Equation 2.6.27

i	n_i^0	γ_i^o	i	n_i^0	γ_i^o
1	-8.32044648201	-	5	0.97315	3.53734222
2	6.6832105268	-	6	1.27950	7.7473708
3	3.00632	-	7	0.966956	9.24437796
4	0.012436	1.28728967	8	0.24873	27.5075105

The equation for the residual part of the Helmholtz free energy $\phi^{Vr}(\delta,\tau)$ can be obtained by using the following equation:

$$\phi^{Vr}(\delta,\tau) = \sum_{i=1}^{7} n_i \delta^{d_i} \tau^{t_i} + \sum_{i=8}^{51} n_i \delta^{d_i} \tau^{t_i} e^{-\delta^{c_i}} + \sum_{i=52}^{54} n_i \delta^{d_i} \tau^{t_i} e^{-\alpha_i(\delta-\epsilon_i)^2-\beta_i(\tau-\gamma_i)^2} + \sum_{i=55}^{56} n_i \Delta^{b_i} \Psi \quad (2.6.28)$$

With $\Delta = \theta^2 + B_i[(\delta-1)^2]^{a_i}$ (2.6.29)

$\theta = (1-\tau) + A_i[(\delta-1)^2]^{1/(2\beta_i)}$ (2.6.30)

$\Psi = e^{-C_i(\delta-1)^2-D_i(\tau-1)^2}$ (2.6.31)

Table 2.6.9 Coefficients of Equation 2.6.28-31

i	a_i	b_i	B_i	c_i	d_i	t_i	n_i	α_i	β_i	γ_i	ϵ_i	C_i	D_i	A_i
1	0	0	0	0	1	-0.5	1.25335479355230E-02	0	0	0	0	0	0	0
2	0	0	0	0	1	0.875	7.89576347228280E+00	0	0	0	0	0	0	0
3	0	0	0	0	1	1	-8.78032033035610E+00	0	0	0	0	0	0	0
4	0	0	0	0	2	0.5	3.18025093454180E-01	0	0	0	0	0	0	0
5	0	0	0	0	2	0.75	-2.61455338593580E-01	0	0	0	0	0	0	0
6	0	0	0	0	3	0.375	-7.81997516879810E-03	0	0	0	0	0	0	0
7	0	0	0	0	4	1	8.80894931021340E-03	0	0	0	0	0	0	0
8	0	0	0	1	1	4	-6.68565723079650E-01	0	0	0	0	0	0	0
9	0	0	0	1	1	6	2.04338109509650E-01	0	0	0	0	0	0	0
10	0	0	0	1	1	12	-6.62126050396870E-05	0	0	0	0	0	0	0
11	0	0	0	1	2	1	-1.92327211560020E-01	0	0	0	0	0	0	0
12	0	0	0	1	2	5	-2.57090430034380E-01	0	0	0	0	0	0	0
13	0	0	0	1	3	4	1.60748684862510E-01	0	0	0	0	0	0	0
14	0	0	0	1	4	2	-4.00928289258070E-02	0	0	0	0	0	0	0
15	0	0	0	1	4	13	3.93434226032540E-07	0	0	0	0	0	0	0
16	0	0	0	1	5	9	-7.59413770881440E-06	0	0	0	0	0	0	0
17	0	0	0	1	7	3	5.62509793518880E-04	0	0	0	0	0	0	0
18	0	0	0	1	9	4	-1.56086522571350E-05	0	0	0	0	0	0	0
19	0	0	0	1	10	11	1.15379964229510E-09	0	0	0	0	0	0	0
20	0	0	0	1	11	4	3.65821651442040E-07	0	0	0	0	0	0	0
21	0	0	0	1	13	13	-1.32511800746680E-12	0	0	0	0	0	0	0
22	0	0	0	1	15	1	-6.26395869124540E-10	0	0	0	0	0	0	0
23	0	0	0	2	1	7	-1.07936009089320E-01	0	0	0	0	0	0	0
24	0	0	0	2	2	1	1.76114910087520E-02	0	0	0	0	0	0	0
25	0	0	0	2	2	9	2.21322951675460E-01	0	0	0	0	0	0	0
26	0	0	0	2	2	10	-4.02476697635280E-01	0	0	0	0	0	0	0
27	0	0	0	2	3	10	5.80833999857590E-01	0	0	0	0	0	0	0
28	0	0	0	2	4	3	4.99691469908060E-03	0	0	0	0	0	0	0
29	0	0	0	2	4	7	-3.13587007125490E-02	0	0	0	0	0	0	0
30	0	0	0	2	4	10	-7.43159297103410E-01	0	0	0	0	0	0	0
31	0	0	0	2	5	10	4.78073299154800E-01	0	0	0	0	0	0	0
32	0	0	0	2	6	6	2.05279408959480E-02	0	0	0	0	0	0	0
33	0	0	0	2	6	10	-1.36364351103430E-01	0	0	0	0	0	0	0
34	0	0	0	2	7	10	1.41806344006170E-02	0	0	0	0	0	0	0
35	0	0	0	2	9	1	8.33265048807130E-03	0	0	0	0	0	0	0
36	0	0	0	2	9	2	-2.90523360095850E-02	0	0	0	0	0	0	0
37	0	0	0	2	9	3	3.86150855742060E-02	0	0	0	0	0	0	0
38	0	0	0	2	9	4	-2.03934865137040E-02	0	0	0	0	0	0	0
39	0	0	0	2	9	8	-1.65540500637340E-03	0	0	0	0	0	0	0
40	0	0	0	2	10	6	1.99555719795410E-03	0	0	0	0	0	0	0
41	0	0	0	2	10	9	1.58703083241570E-04	0	0	0	0	0	0	0
42	0	0	0	2	12	8	-1.63885683425300E-05	0	0	0	0	0	0	0
43	0	0	0	3	3	16	4.36136157238110E-02	0	0	0	0	0	0	0
44	0	0	0	3	4	22	3.49940054637650E-02	0	0	0	0	0	0	0
45	0	0	0	3	4	23	-7.67881978446210E-02	0	0	0	0	0	0	0
46	0	0	0	3	5	23	2.24462773320060E-02	0	0	0	0	0	0	0
47	0	0	0	4	14	10	-6.26897104146850E-05	0	0	0	0	0	0	0
48	0	0	0	6	3	50	-5.57111185656450E-10	0	0	0	0	0	0	0
49	0	0	0	6	6	44	-1.99057183544080E-01	0	0	0	0	0	0	0
50	0	0	0	6	6	46	3.17774973307380E-01	0	0	0	0	0	0	0
51	0	0	0	6	6	50	-1.18411824259810E-01	0	0	0	0	0	0	0
52	0	0	0	0	3	0	-3.13062603234350E+01	20	150	1.2	1	0	0	0
53	0	0	0	0	3	1	3.15461402377810E+01	20	150	1.2	1	0	0	0
54	0	0	0	0	3	4	-2.52131543416950E+03	20	250	1.3	1	0	0	0
55	3.5	0.9	0.2	0	0	0	-1.48746408567240E-01	0	0.3	0	0	28	700	0.32
56	3.5	1	0.2	0	0	0	3.18061108784440E-01	0	0.3	0	0	32	800	0.32

All thermodynamic values for water-steam can be derived from equation 2.6.26

Pressure (kPa): $\frac{P(\delta,\tau)}{\rho RT} = 1 + \delta \frac{\partial \phi^{Vr}(\delta,\tau)}{\partial \delta}$ (2.6.32)

Entropy (kJ/(kgK)) : $\frac{s(\delta,\tau)}{R} = \tau \frac{\partial[\phi^{V0}(\delta,\tau)+\phi^{Vr}(\delta,\tau)]}{\partial \tau} - \left[\phi^{V0}(\delta,\tau) + \phi^{Vr}(\delta,\tau)\right]$ (2.6.33)

Enthalpy (kJ/kg) : $\frac{h(\delta,\tau)}{RT} = 1 + \tau \frac{\partial[\phi^{V0}(\delta,\tau)+\phi^{Vr}(\delta,\tau)]}{\partial \tau} + \delta \frac{\partial \phi^{Vr}(\delta,\tau)}{\partial \delta}$ (2.6.34)

Isochoric specific heat C_p(kJ/(kgK)) :

$$\frac{C_p(\delta,\tau)}{R} = -\tau^2\left[\phi^{V0}(\delta,\tau) + \phi^{Vr}(\delta,\tau)\right] + \frac{\left(1+\delta\frac{\partial\phi^{Vr}(\delta,\tau)}{\partial\delta} - \delta\frac{\partial^2\phi^{Vr}(\delta,\tau)}{\partial\delta\partial\tau}\right)^2}{1+2\delta\frac{\partial\phi^{Vr}(\delta,\tau)}{\partial\delta} + \delta^2\frac{\partial^2\phi^{Vr}(\delta,\tau)}{\partial\delta\partial\delta}} \quad (2.6.35)$$

For saturation region following equations are given:

$$\frac{P_\sigma(\delta',\tau)}{\rho' RT} = 1 + \delta'\frac{\partial\phi^{Vr}(\delta',\tau)}{\partial\delta} \quad (2.6.36)$$

$$\frac{P_\sigma(\delta^{"},\tau)}{\rho^{"} RT} = 1 + \delta^{"}\frac{\partial\phi^{Vr}(\delta^{"},\tau)}{\partial\delta} \quad (2.6.37)$$

$$\frac{P_\sigma(\delta',\tau)}{RT}\left(\frac{1}{\rho^{"}} - \frac{1}{\rho'}\right) - ln\left(\frac{\rho'}{\rho^{"}}\right) \quad (2.6.38)$$

Where ρ' is saturated liquid density and $\rho^{"}$ is saturated vapor density.

Auxiliary equations for P_σ, ρ' , $\rho^{"}$, h' , $h^{"}$, , s' , $s^{"}$ is also given in IAPWS-95 as:

$$\left(\frac{P_\sigma}{P_c}\right) = exp\left(\frac{T_c}{T}(a_1\vartheta + a_2\vartheta^{1.5} + a_3\vartheta^3 + a_4\vartheta^{3.5} + a_5\vartheta^4 + a_6\vartheta^{7.5})\right) \quad (2.6.39)$$

$$\left(\frac{dP_\sigma}{dT}\right) = -\frac{P_\sigma}{T}\left(ln\left(\frac{P_\sigma}{P_c}\right) + a_1\vartheta + 1.5a_2\vartheta^{0.5} + 3a_3\vartheta^2 + 3.5a_4\vartheta^{2.5} + 4a_5\vartheta^3 + 7.5a_6\vartheta^{6.5}\right) \quad (2.6.40)$$

$$\left(\frac{\rho'}{\rho_c}\right) = \left(1 + b_1\vartheta^{1/3} + b_2\vartheta^{2/3} + b_3\vartheta^{5/3} + b_4\vartheta^{16/35} + b_5\vartheta^{43/3} + b_6\vartheta^{110/3}\right) \quad (2.6.41)$$

$$ln\left(\frac{\rho^{"}}{\rho_c}\right) = exp\left(c_1\vartheta^{2/6} + c_2\vartheta^{4/6} + c_3\vartheta^{8/6} + c_4\vartheta^{18/6} + c_5\vartheta^{37/6} + c_6\vartheta^{71/6}\right) \quad (2.6.42)$$

$$\left(\frac{\alpha}{\alpha_0}\right) = (d_0 + d_1\theta^{-19} + d_2\theta + d_3\theta^{4.5} + d_4\theta^5 + d_5\theta^{54.5}) \quad (2.6.43)$$

$$\left(\frac{\varphi}{\varphi_0}\right) = \left(e_0 + \frac{19}{20}d_1\theta^{-20} + d_2 ln\theta + \frac{9}{7}d_3\theta^{3.5} + \frac{5}{4}d_4\theta^4 + \frac{109}{107}d_5\theta^{53.5}\right) \quad (2.6.44)$$

$$h' = \left(\frac{\alpha}{\alpha_0}\right) + \frac{T}{\rho'}\left(\frac{dP_\sigma}{dT}\right) \quad (2.6.45)$$

$$h^{"} = \left(\frac{\alpha}{\alpha_0}\right) + \frac{T}{\rho^{"}}\left(\frac{dP_\sigma}{dT}\right) \quad (2.6.46)$$

$$s' = \varphi + \frac{1}{\rho'}\left(\frac{dP_\sigma}{dT}\right) \quad (2.6.47)$$

$$s^{"} = \varphi + \frac{1}{\rho^{"}}\left(\frac{dP_\sigma}{dT}\right) \quad (2.6.48)$$

Where $\vartheta = \left(1 - \frac{T}{T_c}\right)$ and $\theta = \frac{T}{T_c}$ Coefficient of the Auxiliary equations are as follows:

Table 2.6.10 Coefficients of Auxiliary Equations 2.39-2.6.42

i	A	b	c
1	-7.85951783	1.99274064	-2.0315024
2	1.84408259	1.09965342	-2.68302940
3	-11.7866497	-0.510839303	-5.38626492
4	22.6807411	-1.75493479	-17.2991605
5	-15.9618719	-45.5170352	-44.7586581
6	1.80122502	-6.74694450e5	-63.9201063

Table 2.6.11 Coefficients of Auxiliary Equations 2.6.43-2.6.48

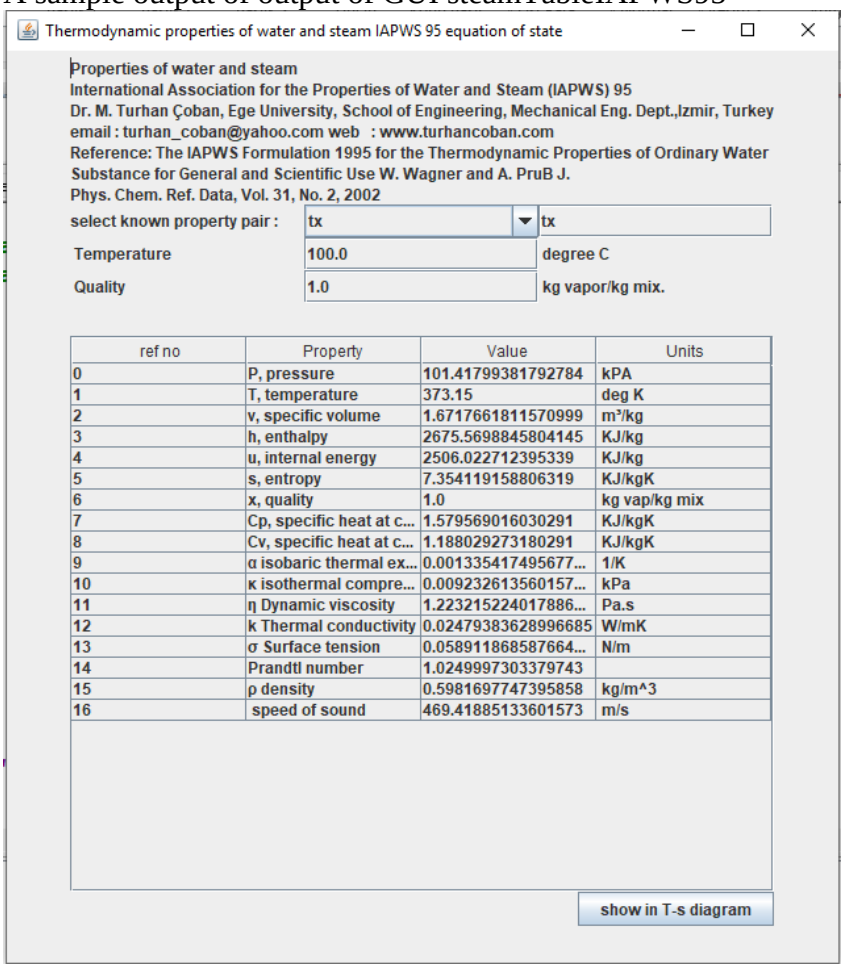
i	d	e
0	-1135.905627715	2319.5246
1	-5.65134998e-8	
2	2690.66631	
3	127.287297	
4	-135.003439	

Program List of IAPWS95 EOS:

steam_IAPWS95	Basic Equation of state

steamTableIAPWS95	Graphic user interface program
steamModel_IAPWS95	A subprogram for steamTableIAPWS95

A sample output of output of GUI steamTableIAPWS95



Sample code to show utilisation of class:

```
public class steam_IAPWS95_test1
{ public static void main(String arg[])
  {steam_IAPWS95 ap=new steam_IAPWS95();
   double T=300.0;  //degree K;
   double P=100.0;  //kPa
   double a[]=ap.property("tp",T,P);
   double h=a[3];
   double s=a[5];
   double Cp=a[7];
   String ss="T = "+T+" degree K"+" P = "+P+" kPa"+" h = "+h+" kJ/kg"+" s= "+s+"kJ/kgK "+"Cp ="+Cp+"kJ/kgK ";
   System.out.println(ss);
  }
}
```

```
---------- Capture Output ----------
> "E:\co\java\bin\java.exe" steam_IAPWS95_test1
T = 300.0 degree K P = 100.0 kPa h = 112.65367945804118 kJ/kg s= 0.393062433009772kJ/kgK Cp =4.180639522003496kJ/kgK

> Terminated with exit code 0.
```

Input known pairs of method property:

tp	tro	vp	hp
vt	pv	rop	ps
rot	pro	ph	sp

Output array of property method:

0	P, pressure	kPa
1	T, temperature	Degree K
2	v, specific volume	m^3/kg
3	h, enthalpy	kJ/kg
4	u, internal energy	kJ/kg
5	s, entropy	kJ/(kgK)
6	x, quality	kg vapor/kg total
7	Cp, specific heat at constant pressure	kJ/(kgK)
8	Cv, specific heat at constant volume	kJ/(kgK)
9	α isobaric thermal expansion coefficient	1/K
10	κ isothermal compressibility	kPa
11	μ Dynamic viscosity	Pa.s
12	k Thermal conductivity	W/(mK)
13	Surface tension	N/m
14	Prandtl number	
15	density	kg/m^3
16	speed of sound	m/s

Sample code to create T-s diagram

```
public class steamtest2A
{public static void main(String arg[])
{
steam_IAPWS95 st=new steam_IAPWS95();
double T2[];
double s2[];
double t=0.1;
int nn=st.ats.length;
Plot p=new Plot(st.ass,st.ats);
double pp[]={100,500,1000,2000,3000,4000,5000,6000,7000,8000,9000,10000,15000,20000};
double pp1[]={22064,30000,40000,50000};
int n=200;
T2=new double[n];
s2=new double[n];
for(int j=0;j<pp.length;j++)
{   double a[]=st.property("px",pp[j],0.0);
    T2[0]=a[1];
    s2[0]=a[5];
    a=st.property("px",pp[j],1.0);
    T2[1]=a[1];
    s2[1]=a[5];
    //System.out.println("T2="+T2[0]+"s20="+s2[0]+"s21="+s2[1]);
            for(int i=2;i<n;i++)
  {t=T2[0]+i*5;
   T2[i]=t;
   a=st.property("tp",t,pp[j]);
   s2[i]=a[5];
  }
  p.addData(s2,T2);
}
for(int j=0;j<pp1.length;j++)
{   double t1=st.Tc-200;
    for(int i=0;i<n;i++)
  {t=t1+i*5;
   T2[i]=t;
   double a[]=st.property("tp",t,pp1[j]);
   s2[i]=a[5];
  }
  p.addData(s2,T2);
}
p.plot();
}}
```

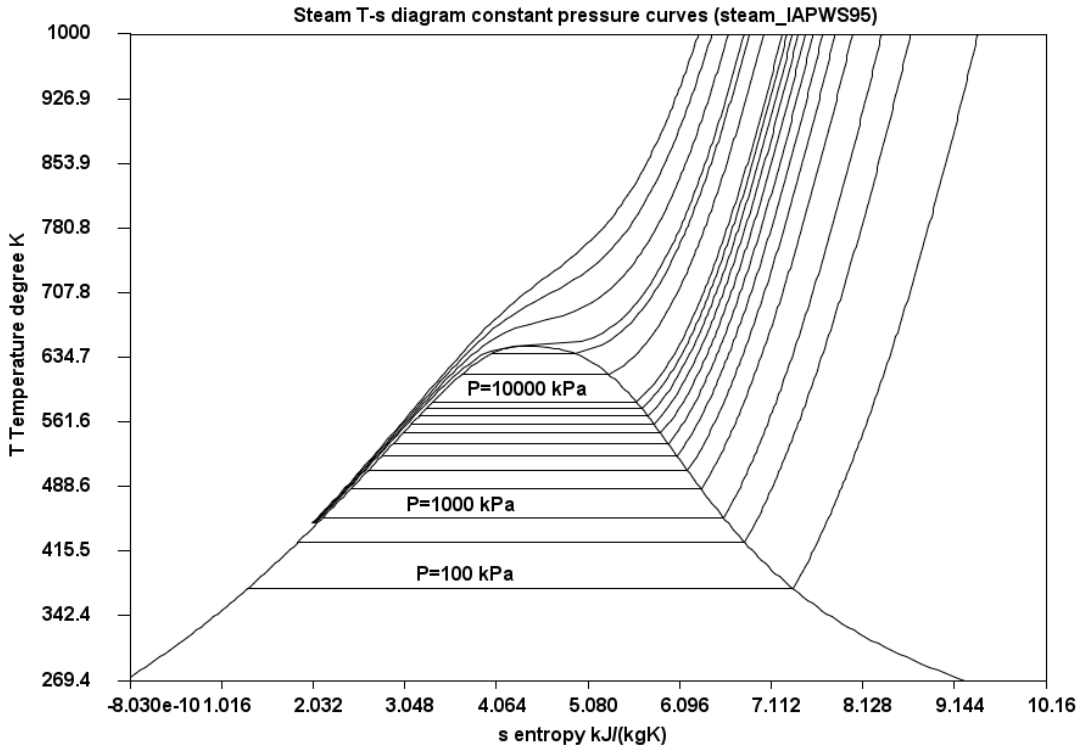


Figure 2.6.5 T-s diagram of steam created by using steamIAPWS95

Ice Equation of State (crystal structure Ih)

The equation of state for ice Ih is in the Gibbs energy form. Independent variables are temperature and pressure.

$$g(T,P) = g_0(P) - s_0 T_t \tau + T_t Re\left(\sum_{k=1}^{2} r_k \left[(t_k - \tau)ln(t_k - \tau) + \left[(t_k + \tau)ln(t_k + \tau) - 2t_k \ln(t_k) - \frac{\tau^2}{t_k}\right]\right]\right) \quad (2.6.49)$$

$$g_0(P) = \sum_{k=0}^{4} g_{0k}(\pi - \pi_0)^k$$

$$r_2(P) = \sum_{k=0}^{2} r_{2k}(\pi - \pi_0)^k$$

With the reduced temperature $\tau = T/T_t$, the reduced pressure $\pi = P/P_t$ and the reduced normal pressure $\pi_0 = P_0/P_t$, where T_t, P_t and P_0 are given in the table below. In the equation inside of the parenthesis is complex number and the real part of the complex number (Re) is taken out of the parenthesis.

Table 2.6.12. Coefficients of equation of states

Quantity	Symbol	Value	Unit
Experimental triple point pressure	P_t	611.657(10)	Pa
Numerical triple point pressure	P_t^{num}	611.654771007894	Pa
Normal presure	P_0	101.325	Pa
Triple point temperature	T_t	273.16	Degree K

Table 2.6.13 Coefficients of equation of states

Coefficient	Real part	Imaginary part	Unit
g_{00}	-6.320202E+05		J/kg
g_{01}	6.550222E-01		J/kg

g_{02}	-1.893699E-08		J/kg
g_{03}	3.397461E-15		J/kg
g_{04}	-5.564649E-22		J/kg
s_0(absolute)	1.891300E+02		J/kgK
g_{00}(IAPWS-95)	-3.327338E+03		J/kgK
t_1	3.680171E-02	5.108781E-02	
r_1	4.470507E+01	6.568768E+01	J/kgK
t_2	3.373157E-01	3.354494E-01	
r_{20}	-7.259746E+01	-7.810084E+01	J/kgK
r_{21}	-5.571077E-05	4.645786E-05	J/kgK
r_{22}	2.348014E-11	-2.856511E-11	J/kgK

In the table2.6.13, 18 coefficients are listed. All thermodynamic properties can be calculated as derivatived of the gibbs energy function.

Density

$$\rho(T,P)=\frac{1}{v(T,P)}=-\left(\frac{\partial g(T,P)}{\partial P}\right)^{-1}_{T} \quad (2.6.50)$$

Specific entropy

$$s(T,P)=\frac{1}{v(T,P)}=-\left(\frac{\partial g(T,P)}{\partial T}\right)_{P} \quad (2.6.51)$$

Specific isobaric heat capacity

$$C_p(T,P)=T\left(\frac{\partial s(T,P)}{\partial T}\right)_{P} \quad (2.6.52)$$

Specific enthalpy

$h(T,P)=g(T,P)-Ts(T,P)$ (2.6.53)

Specific helmholtz energy

$$u(T,P)=g(T,P)+Ts(T,P)-Pv \quad (2.6.54)$$

Cubic expansion coefficient

$$\alpha(T,P)=\frac{1}{v}\left(\frac{\partial v}{\partial T}\right)_{P} \quad (2.6.55)$$

Isothermal compressibility

$$\kappa_T(T,P)=-\frac{1}{v}\left(\frac{\partial v}{\partial P}\right)_{T} \quad (2.6.56)$$

A computer code ice_IAPWSR10_06 is developed to use equation of state for ice Ih

```
public static void main(String arg[])
{  ice_IAPWSR10_06 ice=new ice_IAPWSR10_06();
   double T=0;
   double t=0;
   double P=0;
   double b[][]=new double[10][10];
   System.out.println("Saturated solid-liquid interface");
   int i=0;
    for(t=0;t>-10;t-=1)
   {  T=t+273.15;
                P=ice.Pmelt(T);
                double
a[]={T,P,1.0/ice.romelts_T(T),1.0/ice.romeltl_T(T),ice.umelts_T(T),ice.umeltl_T(T),ice.hmelts_T(T),ice.hmeltl_T(T),ice.s
melts_T(T),ice.smeltl_T(T)};
                b[i]=a;
                i++;
}
                String s[]={"T degree K","P kPa","vs m^3/kg","vl m^3/kg","us kJ/kg","ul kJ/kg","hs kJ/kg","hl kJ/kg","ss
kJ/kgK","sl kJ/kgK"};
                String s1="Saturated solid ice(Ih) - liquid water";
                Text.print(b,s,s1);
}
```

Saturated solid ice(Ih) - liquid water

T degree K	P kPa	vs m^3/kg	vl m^3/kg	us kJ/kg	ul kJ/kg	hs kJ/kg	hl kJ/kg	ss kJ/kgK	sl kJ/kgK
273.15	136.21371658203321	0.0010908389603100 18	0.0010002082039539683	-333.4723460124997	-0.042199534364594636	-333.32375878352343	-0.041588194113371196	-1.2207947443948206	-1.5448926937043305E-4
272.15	13386.449495847924	0.0010889730932596626	0.0010002856162881755	-336.17551450549666	-4.263929331787773	-321.5980311902389	-4.263360959952767	-1.2307553383730383	-0.015638573582834894
271.15	26054.643625685647	0.0010872020610589543	0.0010003819400036 54	-338.8060200715866	-8.489411664025402	-310.47935782158464	-8.488883549525122	-1.2405672517929498	-0.031193472338249 33
270.15	38185.95860669486	0.0010855177305498062	0.0010004978940824053	-341.3708777047252	-12.718945792362012	-299.9193425791169	-12.718455373993038	-1.250243721810621	-0.046820814578223 47
269.15	49822.72694678303 4	0.0010839126738000956	0.0010006342498027308	-343.8762409577099	-16.952857941738 27	-289.8727557768102	-16.952402800771047	-1.259796745582271	-0.062522347468085 55
268.15	61004.62769002148	0.0010823800991352501	0.0010007918359077 11	-346.3275121652838	-21.191504583609554	-280.2973171984493	-21.191082437177055	-1.2692372032849795	-0.078299950275186 91
267.15	71768.85273195231	0.0010809137899551798	0.0010009715444145658	-348.72943749024654	-25.43527622652954	-271.15349488301683	-25.434888492175 75	-1.278574967871325	-0.094155650506204 34
266.15	82150.26348291922	0.0010795080503337388	0.0010011743371562 53	-351.0861890187659	-29.684601802629 51	-262.4043182519168	-29.684239309466218	-1.2878190030688015	-0.110091642582163 89
265.15	92181.53841220759	0.0010781576566398418	0.0010014012531640 78	-353.4014357836869	-33.939953754940255	-254.01520435294373	-33.939618160802 4	-1.2969774509576475	-0.126110309503281 36
264.15	101893.3119783569	0.0010768578737399362	0.0010016534170099133	-355.6784053048747	-38.201853949782084	-245.95379613314	-38.201543452752574	-1.3060577103044204	-0.142214248040726 52

Ice Ih-liquid boundary Melting pressure equation(Temperature range from 273.16 to 251.165 K) [7]

$$\frac{P_{m,Ih}}{P_n} = 1 - 0.626x10^6(1-\theta^{-3}) + 0.197135x10^6(1-\theta^{21.2}) \quad (2.6.57)$$

Where $\theta = \frac{T}{T_n}$ $T_n = 273.16$ (triple point) and $P_n = 0.611657\ kPa$

Ice III-liquid boundary Melting pressure equation(Temperature range from 251.165 to 256.164 K)

$$\frac{P_{m,III}}{P_n} = 1 - 0.295252(1-\theta^{60}) \quad (2.6.58)$$

Where $\theta = \frac{T}{T_n}$ $T_n = 251.165$ and $P_n = 209900\ kPa$

Ice V-liquid boundary Melting pressure equation(Temperature range from 256.164 to 273.31 K)

$$\frac{P_{m,V}}{P_n} = 1 - 1.18721(1-\theta^{8}) \quad (2.6.59)$$

Where $\theta = \frac{T}{T_n}$ $T_n = 256.64\ K$ and $P_n = 350100\ kPa$

Ice VI-liquid boundary Melting pressure equation(Temperature range from 273.31 to 355 K)

$$\frac{P_{m,VI}}{P_n} = 1 - 1.07476(1-\theta^{4.6}) \quad (2.6.59)$$

Where $\theta = \frac{T}{T_n}$ $T_n = 273.31\ K$ and $P_n = 632400\ kPa$

Ice VII-liquid boundary Melting pressure equation(Temperature range from 355to 715 K)

$$ln\left(\frac{P_{m,VI}}{P_n}\right) = 1.73683(1-\theta^{-1}) - 0.0544606(1-\theta^{5}) + 0.806106x10^{-7}(1-\theta^{22}) \quad (2.6.59)$$

Where $\theta = \frac{T}{T_n}$ $T_n = 355\ K$ and $P_n = 2216000\ kPa$

The sublimation pressure equation (Temperature range from 0 K to 273.16 K)

$$ln\left(\frac{P_{subl}}{P_n}\right) = -13.928169(1-\theta^{-1.5}) + 34.707823(1-\theta^{-1.25}) \quad (2.6.60)$$

Where $\theta = \frac{T}{T_n}$ $T_n = 273.16\ K$ and $P_n = 0.611657\ kPa$

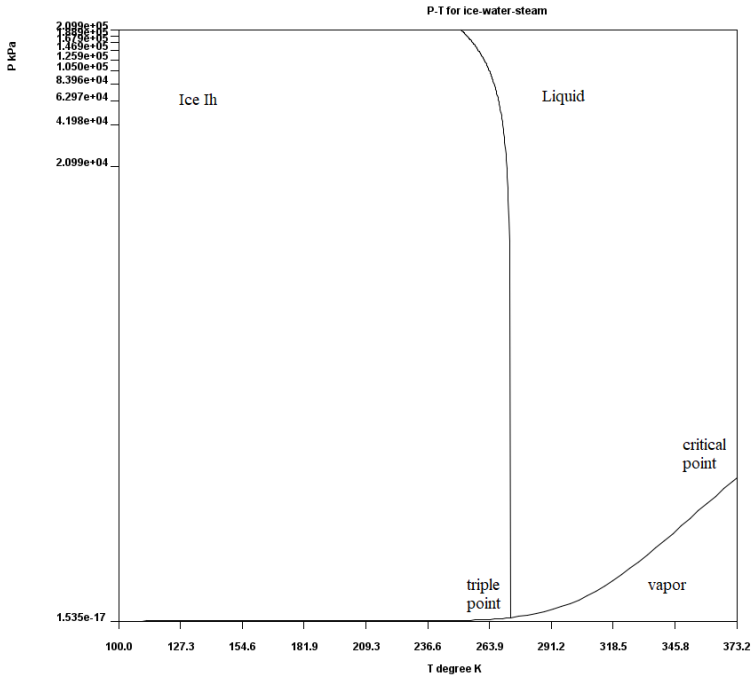


Figure 2.6.6a P-T diagram of steam using steamIAPWS95

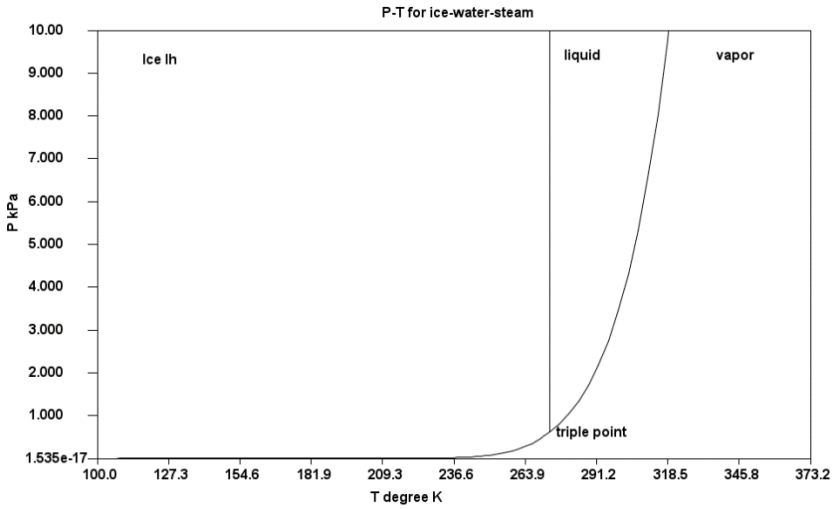


Figure 2.6.6b P-T diagram of steam using steamIAPWS95

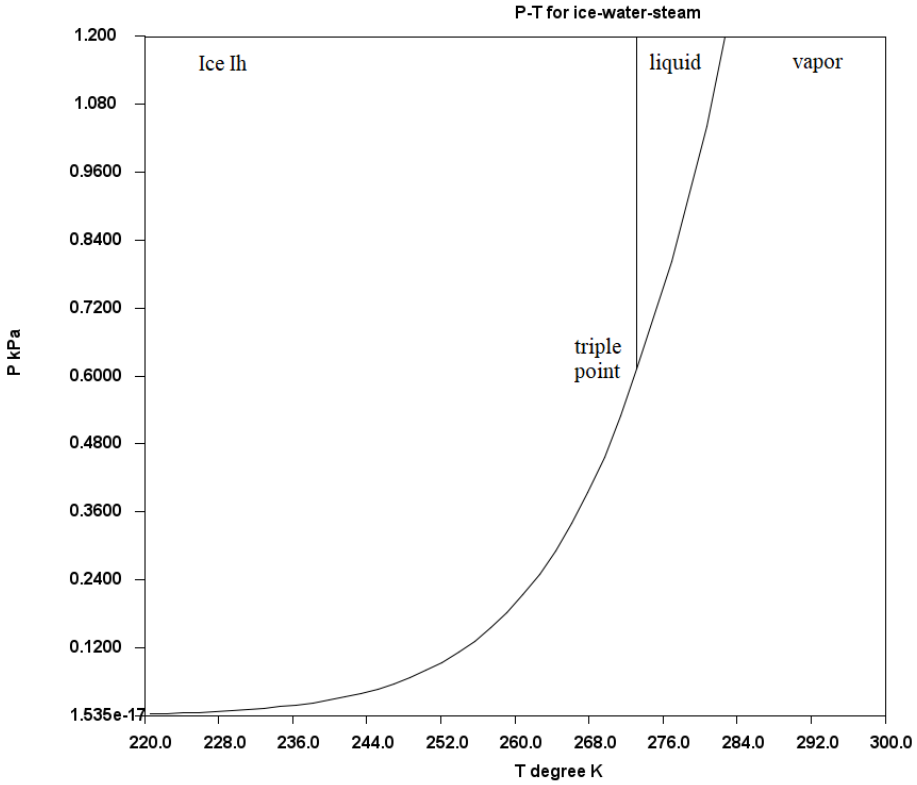


Figure 2.6.6c P-T diagram of steam using steamIAPWS95

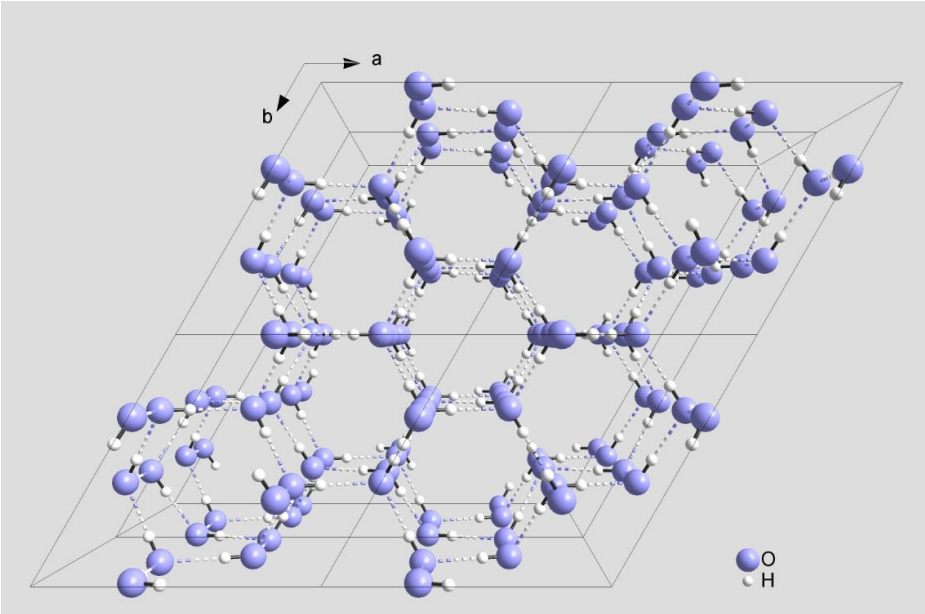


Figure 2.6.7 cyristal structure of ice Ih
Source: https://en.wikipedia.org/wiki/Ice_Ih

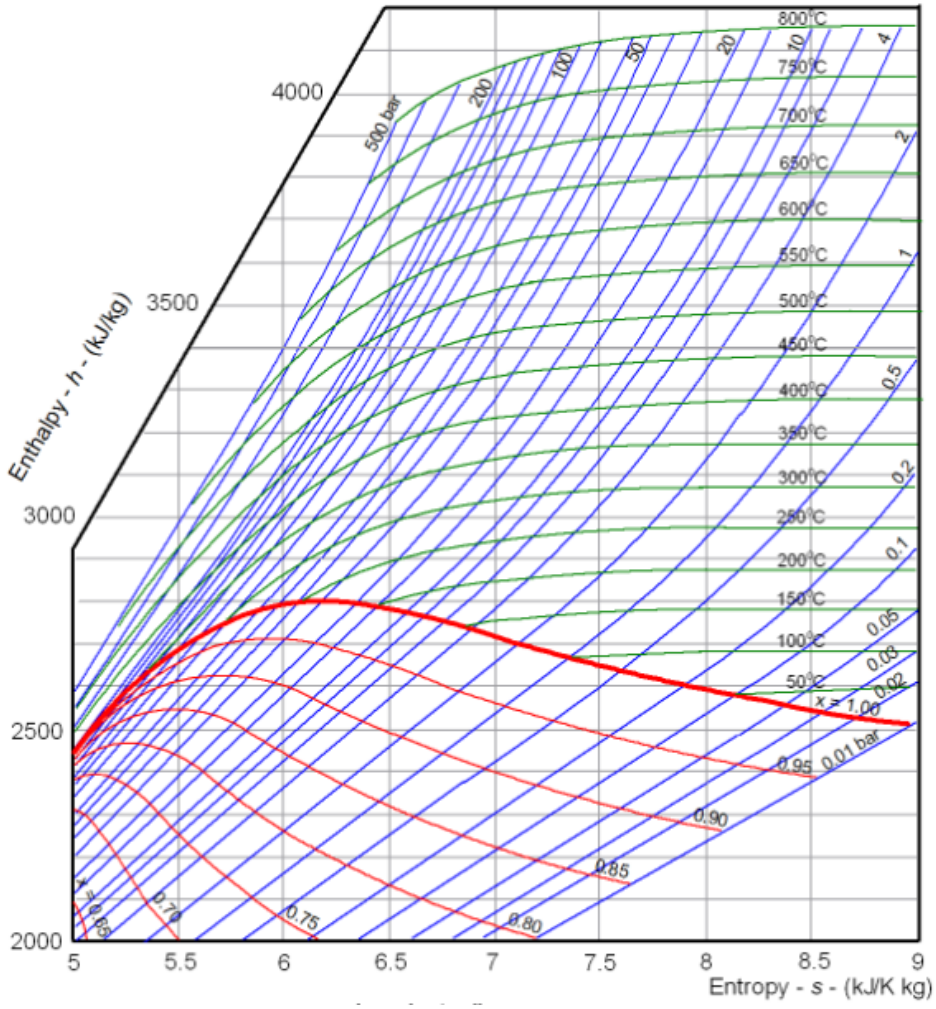


Figure 2.6.7 h-s Enthalpy-entropy diagram (Molier diagram) of water

2.7 MARTIN-HOU EQUATION OF STATE

The Martin-Hou Equation of State is originally published in [31] is

$P(T,V) = \frac{RT}{V-b} + \sum_{i=2}^{5} \frac{(A_i + B_i T + C_i e^{-\gamma})}{(V-b)^i}$ (2.7.1) where $\gamma = \frac{KT}{T_c}$ (2.7.2)

In later use of this equation, specially for refrigerants, additional terms are added to obtain more accurate fit and equation becomes

$$P(T,V) = \frac{RT}{V-b} + \sum_{i=2}^{5} \frac{(A_i + B_i T + C_i e^{-\gamma})}{(V-b)^i} + \frac{(A_6 + B_6 T + C_6 e^{-\gamma})}{e^{uV}(1 + C' e^{uV})} \quad (2.7.3)$$

As examples to the coefficients of these equation R22 coefficients are given in the table below:

Table 2.6.1 R22 Martin Hou denklemi A,B,C kat sayıları

i	A	B	C
1	0.0	0.0	0.0
2	-0.87466122549	8.7054394311E-4	-8.8533742184
3	-0.018939998642	1.4890632572E-4	1.6091656668
4	0.013524290185	-3.7996244181E-5	0.0
5	-0.0011768746066	3.0463870398E-6	-0.0058307411777
6	940018.96962	-2075.798446	0.0

For R22 refrigerant the other coefficients are: K=4.2, u=101.55456431, and C'=0.0. It should be note that this equation is only valid for gas state. In order to model saturation state and liquid state, additional equations are required such as saturation pressure and liquid saturation density.

$$\log_{10}(P_s) = A_1 + \frac{A_2}{T} + \frac{A_3}{T^2} + A_4\log_{10}(T) + A_5T + A_6T^2 + A_7T^3 + \frac{A_8(A_9 - T)}{T}\log_{10}[(A_9 - T)A_{10}] \quad (2.7.4)$$

Saturation data for R22:

i	A_i
1	0.25189356867e2
2	-0.21362184178e4
3	0.0
4	-0.786103122e1
5	0.39436902792e-2
6	0.0
7	0.0
8	0.445746703
9	0.38116666667e3
10	1.8

In order to calculate properties such as enthalpy and entropy, ideal gas specific heat is also required, It can be calculated from data as curve fitting as mentioned in ideal gas section or if a curve fittig equation is already given, for the specified fluid, it can be used. For example for R22, equation in the form

$$C_v(T) = G_1 + G_2T + G_3T^2 + G_4T^3 + \frac{G_5}{T^2} + G_6T^4 \quad (2.7.5)$$

is given as an accurate approximation, with coefficients

i	G_i
1	0.10183212e2
2	0.14697171
3	-0.76354723e-4
4	0.0
5	0.28754121e5
6	0.0

Density of saturated fluid is an important properties to calculate liquid and saturation properties. Martin-Hou equation of State is defined for gases only, therefore an accurate estimation for the liquid and saturated region is not available. For the liquid region of R22 saturated liquid density is given as:

$$\frac{1}{V_f} = \rho_f = E_1 + \sum_{i=2}^{5} E_i\,\tau^{(i-1)/3} \quad (2.7.6)$$

Where $\tau = 1 - \frac{T}{T_c}$ (2.7.7)

With coefficients

i	E_i
1	0.6068821e1
2	0.1012108e2
3	0.6807772e1
4	-0.4129719e1
5	0.3792696e1

Saturation vapor pressure is not required as an additional equation to be given, it can be obtained directly from eqaution of state, but in order to avoid additional curve fitting calculations, it can be given as an additional equation. For R22

$$ln\left(\frac{1}{V_g}\right) = ln(\rho_g) = \sum_{i=-10}^{13} F_{i+11}\,\tau^{1/3} + F_{25}\ln(1-\tau) \quad (2.7.8)$$

i	F_i
7	-0.1680024749e-4
11	0.1344723347
12	-2.845461973
14	0.1292966614e3
20	0.3106835616e4
21	-0.1189422522e5
22	0.1914759823e5
23	-0.1472081631e5
24	0.4527640563e4
25	0.1306817379e3

Unlisted coefficients in the table is zero. In order to calculate other thermodynamic properties, equations previously derived will be utilized as:

$$ds = \frac{C_v}{T}dT + \left(\frac{\partial P}{\partial T}\right)_v dv \quad (1.47)$$

$$du = C_v dT + \left(T\left(\frac{\partial P}{\partial T}\right)_v - P\right)dv \quad (1.48)$$

If integration carried out for the given equations enthalpy and entropy functions will be as follows:

$$h = \sum_{i=2}^{5}\left[A_i\frac{(V-b)^{(1-i)}}{(1-i)}\right] + \frac{JA_6}{ue^{uV}} + \sum_{i=2}^{5} J\frac{(1+\gamma)C_i e^{-\gamma}(V-b)^{(1-i)}}{1-i} + JPV + G_1T + G_2\frac{T^2}{2} + G_3\frac{T^3}{3} + G_3\frac{T^4}{4} + G_4\frac{T^4}{4} - G_5\frac{1}{T} + G_6\frac{T^5}{5} + X \quad (2.7.9)$$

$$s = JR\,ln(V-b) + \sum_{i=2}^{5} J\left[\frac{B_i}{i-1}\right](V-b)^{(i-1)} - \frac{JB_6}{ue^{uV}} - \sum_{i=2}^{5}\frac{JKC_i e^{-\gamma}(V-b)^{(1-i)}}{T_c(1-i)} + G_1\,ln(T) + G_2T + G_3\frac{T^2}{2} + G_4\frac{T^3}{3} - G_5\frac{1}{2T^2} + G_6\frac{T^4}{4} + Y \quad (2.7.10)$$

Saturation state can be solved from Classius-Clapeyron equation

$$\left(\frac{h_g - h_f}{T(v_g - v_f)}\right)_T = \left(\frac{\partial P}{\partial T}\right)_v$$

This EOS codes are listed in refrigerant.java program. Program code is given in A5.

Program List of refrgerant:

refrigerant	Basic Equation of state
ref_CS3	Saturation properties by using cubic spline
refTable	User graphic interface

2.8 BENEDİCT-WEBB-RUBİN (BWR) EQUATION OF STATES:

One of the most used real gas equation of state is Benedict-Webb-Rubin (BWR) equation of state which is published in 1940. While looking Lee-Kesler equation, which is a modified form of Benedict-Webb_rubin equation, we mentioned this EOS. EOS originally developed to predict real gas thermodynamic properties of hydrocarbons. A Modified form of BWR equation is also frequently used for refrigerant applications. A java program refrigerantEN is developed to predict several refrigerants an some of them are using BWR EOS. The listing of the program is given in appendix 5. In this program Refrigerants R134a, R123, R124, R125,R50, R702, R720,R732, R740 used a MBWR EOS. In this section, description of MBWR EOS will be introduced.

Modified BWR Equation of state

$$\frac{P}{100} = \sum_{n=1}^{9} \frac{a_n}{V^n} + exp\left(-\frac{V_c^2}{V^2}\right) \sum_{n=10}^{15} \frac{a_n}{V^{(2n-17)}} \quad (2.8.1)$$

a_n coefficients in the equation is function of temperature and given in the table below

Table 2.8-1 a_n coefficientss for MBWR EOS

i	a_i
1	RT
2	$b_1T + b_2T^{0.5} + b_3 + \frac{b_4}{T} + \frac{b_5}{T^2}$
3	$b_6T + b_7 + \frac{b_8}{T} + \frac{b_9}{T^2}$
4	$b_{10}T + b_{11} + \frac{b_{12}}{T}$
5	b_{13}
6	$\frac{b_{14}}{T} + \frac{b_{15}}{T^2}$
7	$\frac{b_{16}}{T}$
8	$\frac{b_{17}}{T} + \frac{b_{18}}{T^2}$
9	$\frac{b_{19}}{T^2}$
10	$\frac{b_{20}}{T^2} + \frac{b_{21}}{T^3}$
11	$\frac{b_{23}}{T^2} + \frac{b_{23}}{T^4}$
12	$\frac{b_{24}}{T^2} + \frac{b_{25}}{T^3}$
13	$\frac{b_{26}}{T^2} + \frac{b_{27}}{T^4}$
14	$\frac{b_{28}}{T^2} + \frac{b_{29}}{T^3}$
15	$\frac{b_{30}}{T^2} + \frac{b_{31}}{T^3} + \frac{b_{32}}{T^4}$

In this equation T is temperature, degree K, V is specific volume in m^3/kmole, V_c is critical specific volume in m^3/kmole, P is pressure in kPa, R is gas constant. As an example constants for R134a refrigerant is given below:

Table 2.7-2 R134a b_i MWBR EOS coefficients

i	b_i		i	b_i
1	–6.545 523 5227 E–02		17	–1.015 436 8796 E–02
2	5.889 375 1817 E+00		18	1.173 423 3787 E+00
3	–1.376 178 8409 E+02		19	–2.730 176 6113 E–02
4	2.269 316 8845 E+04		20	–6.633 850 2898 E+05
5	–2.926 261 3296 E+06		21	–6.475 479 9101 E+07

6	–1.192 377 6190 E–04		22	–3.729 521 9382 E+04
7	–2.721 419 4543 E+00		23	1.261 473 5899 E+09
8	1.629 525 3680 E+03		24	–6.474 220 0070 E+02
9	7.294 220 3182 E+05		25	1.236 245 0399 E+05
10	–1.172 451 9115 E–04		26	–1.569 919 6293 E+00
11	8.686 451 0013 E–01		27	–5.184 893 2204 E+05
12	–3.066 016 8246 E+02		28	–8.139 632 1392 E–02
13	–2.566 404 7742 E–02		29	3.032 516 8842 E+01
14	–2.438 183 5971 E+00		30	1.339 904 2297 E–04
15	–3.160 316 3961 E+02		31	–1.585 619 2849 E–01
16	3.432 165 1521 E–01		32	9.067 958 3743 E+00

Table 2.7-3 R134a other coefficients MWBR EOS coefficients

Chemical formula	Sembol	CH_2FCF_3
Molecular weight	M	102.03
Boiling temperature		-26.06
Critical temperature	T_c	101.08 C = 374.23K
Critical pressure	P_c	4060.3 kPa
Critical density	ρ_c	515.3 kg/m^3
Critical volume	v_c	0.00194 m^3/kg

Table 2.7-4 R134a other referance values

Name of constant		
Ideal Gas constant	R	8.314 J/(mole)K
Referance enthalpy	h_f	200 kJ/kg O C de
Referance entropy	s_f	1 kJ/kg 0 C de
Atmospheric pressure	P_{atm}	101.325 kPa

Saturation pressure equation is defined as partially continious in the range $T_{li} \leq T \leq T_{hi}$

$$\log_{10}(P_s) = A_1 + \frac{A_2}{T} + \frac{A_3}{T^2} + A_4 \log_{10}(T) + A_5 T + A_6 T^2 + A_7 T^3 + \left[\frac{A_8(A_9 - T)}{T}\right] \log_{10}[(A_9 - T)A_9] \quad (2.8.2)$$

This equation divided into 6 temperature zones. For R134a for $-100\,C \leq T \leq 101C$ range, A_i coefficients are given in the table below

Table 2.7-5 R134a Saturation pressure equation coefficients at range $-100\,C \leq T \leq 101C$

i	A_i
1	4.069889 E+01
2	–2.362540 E+03
3	0.0
4	–1.306883 E+01
5	7.616005 E–03
6	0.0
7	0.0
8	2.342564 E–01
9	3.761111 E+02
10	1.0

Ideal gas specific heat at constant volume is another important propery to create EOS

$$C_v^0(T) = G_1 + G_2 T + G_3 T^2 + G_4 T^3 + \frac{G_5}{T^2} + \frac{G_6}{T^4} \quad (2.7\text{-}3)$$

Saturated liquid density (a property used in Classius-Clapeyron equation for saturation enthapy and entropy) can be calculated directly from MBWR EOS, and additional equation can still be given for mathematical simplicity

$$\frac{1}{V'} = \rho' = E_1 + \sum_{i=2}^{5} E_i \tau^{(i-1)/3} \quad (2.8.4)$$

In this equation is defined as (T and T_c in degree K) $\tau = 1 - \frac{T}{T_c}$ (2.8.5)

For R134a saturated liquid density equation coefficients :

Table 2.8.6 -R134a MBWR EOS additional saturated liquid density equation coefficients

i	E_i
1	5.281464E+02
2	7.551834E+02
3	1.028676E+03
4	–9.491172E+02
5	5.935660E+02

Program List of refrgerant:

refrigerant	Basic Equation of state
ref_CS3	Saturation properties by using cubic spline
refTable	User graphic interface

2.9 IAPWS EQUATION OF STATE FOR DRY AIR

International Assosiation for Properties of Water and Steam (IAPWS) published and equation for dry air [60]. Equation of State (the specific Helmholtz energy) for dry air has the following form:

$$f^A(T,\rho^A) = \frac{R^L T}{M_A}\alpha^A(\delta,\tau) = \frac{R^L T}{M_A}[\alpha^{A0}(\delta,\tau) + \alpha^{Ar}(\delta,\tau)] \qquad (2.9.1)$$

Where R_L=8.31451 J/(molK) molar gas constant, M_A=0.02896546 kg/mol Molar mass of dry air. $\alpha^{A0}(\delta,\tau)$ is ideal gas part of the reduced specific Helmholtz energy and $\alpha^{Ar}(\delta,\tau)$ is the residual part of the reduced specific Helmholtz energy.

$\alpha^{A0}(\delta,\tau) = \ln(\delta) + \sum_{i=1}^{5} n_i^0 \tau^{i-4} + n_6^0 \tau^{1.5} + n_7^0 \ln(\tau) + n_8^0 \ln[1 - \exp(-n_{11}^0 \tau)] + n_9^0 \ln[1 - \exp(-n_{12}^0 \tau)] + n_{10}^0 \ln[2/3 - \exp(-n_{13}^0 \tau)]$ (2.9.2)

$\alpha^{Ar}(\delta,\tau) = \sum_{k=1}^{10} n_k \delta^{i_k} \tau^{j_k} + \sum_{k=11}^{19} n_k \delta^{i_k} \tau^{j_k} \exp(-n_k \delta^{t_k})$ (2.9.3)

Where $\tau = \frac{T_A^*}{T}$ $T_A^* = 132.6312\ K$ $\delta = \frac{\rho^A}{\rho_A^*}$ $\rho_A^* = 10447.7\ mol/m^3$

Table (2.9.1 Coefficients of Equation 2.9.2

i	n_i^0
1	6.05719400E-08
2	-2.10274769E-05
3	-1.58860716E-04
4	9.74502517E+00
5	1.00986147E+01
6	-1.95363420E-04
7	2.49088803E+00
8	7.91309509E-01
9	2.12236768E-01
10	-1.97938904E-01
11	2.53636500E+01
12	1.69074100E+01
13	8.73127900E+01

Table (2.9.2 Coefficients of Equation 2.9.3

i	i_k	j_k	l_k	n_k
1	1	0	0	1.181607472290E-01
2	1	0.33	0	7.131163920790E-01

3	1	1.01	0	-1.618241920670E+00
4	2	0	0	7.141401789710E-02
5	3	0	0	-8.654213966460E-02
6	3	0.15	0	1.342111767040E-01
7	4	0	0	1.126267042180E-02
8	4	0.2	0	-4.205332288420E-02
9	4	0.35	0	3.490084319820E-02
10	6	1.35	0	1.649571831860E-04
11	1	1.6	1	-1.013650379120E-01
12	3	0.8	1	-1.738136909700E-01
13	5	0.95	1	-4.721031837310E-02
14	6	1.25	1	-1.225235542530E-02
15	1	3.6	3	-1.466296097130E-01
16	3	6	2	-3.160558798210E-02
17	11	3.25	2	2.335948061420E-04
18	1	3.5	3	1.482878919780E-02
19	3	15	3	-9.387828846670E-03

All thermodynamic values for dry air can be derived from equation (2.9.4)

Pressure (kPa): $P(T,ro) = \rho^2 \frac{\partial f^A(T,\rho^A)}{\partial \rho}$ (2.9.5)

Entropy (kJ/(kgK)) : $s(T,ro) = -\frac{\partial f^A(T,\rho^A)}{\partial T}$ (2.9.6)

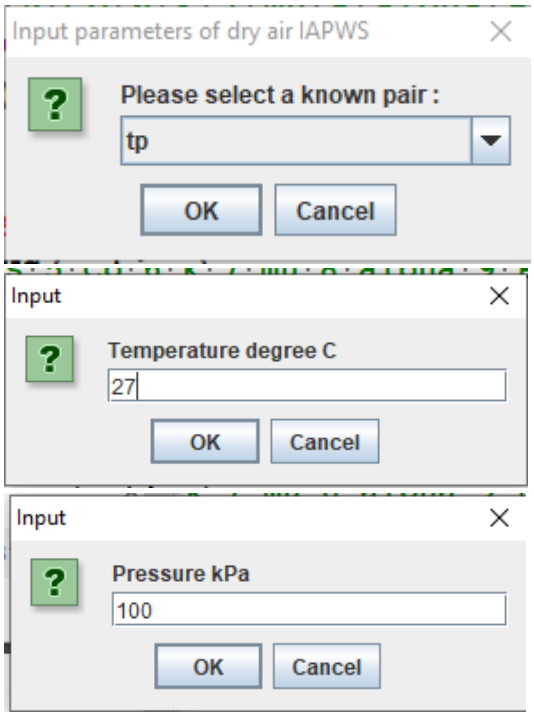
Enthalpy (kJ/kg) : $h(T,ro) = f^A(T,\rho^A) - T\frac{\partial f^A(T,\rho^A)}{\partial T} + \rho\frac{\partial f^A(T,\rho^A)}{\partial \rho}$ (2.9.7)

Isochoric specific heat C_p(kJ/(kgK)) :

$$C_p(T,ro) = -T\frac{\partial^2 f^A(T,\rho^A)}{\partial T \partial T} + \frac{T\rho\left[\frac{\partial^2 f^A(T,\rho^A)}{\partial T \partial \rho}\right]^2}{\left(2\frac{\partial f^A(T,\rho^A)}{\partial \rho} + \rho\frac{\partial^2 f^A(T,\rho^A)}{\partial \rho \partial \rho}\right)} \quad (2.9.8)$$

Program List of air_IAPWS:

Air_IAPWS	Basic Equation of state & input-output main program



Properties of dry air IAPWS formulation based

array no	property name	dry air property	unit
0	P, pressure	100.0	kPA
1	T, temperature	300.15	deg K
2	v, specific volume	0.861313392231...	m^3/kg
3	h, enthalpy	27.16456955287...	KJ/kg
4	u, internal energy	27.07843821365...	KJ/kg
5	s, entropy	0.098612058192...	KJ/kgK
6	x, quality	1.0	kg vap/kg mix
7	Cp, specific heat...	1.006358824854...	KJ/kgK
8	Cv, specific heat...	0.717975421774...	KJ/kgK
9	α isobaric therm...	0.003340424771...	1/K
10	κ isothermal co...	9.986863533996...	kPa
11	η Dynamic visco...	1.855104508520...	Pa.s
12	k Thermal condu...	0.026222311846...	W/mK
13	σ Surface tension	0.0	N/m
14	Prandtl number	0.713455013951...	
15	ρ density	1.161017591296...	kg/m^3
16	speed of sound	347.4048703936...	m/s

Our basic reason to define equation of states is using them in thermodynamic processes fro that reason user of the book should have the ability to use them in their own codes. In order to do that a small program example is given here on how to use dry air EOS in your own.

```
 public class air_IAPWS_test1
{ public static void main(String arg[])
  {air_IAPWS ap=new air_IAPWS();
   double T=300.0;  //degree K;
   double P=100.0;  //kPa
   double a[]=ap.property("tp",T,P);
   double h=a[3];
   double s=a[5];
   double Cp=a[7];
   String ss="T = "+T+" degree K"+" P = "+P+" kPa"+" h = "+h+" kJ/kg"+" s= "+s+"kJ/kgK "+"Cp ="+Cp+"kJ/kgK ";
   System.out.println(ss);
  }
}
```

```
---------- Capture Output ----------
> "E:\co\java\bin\java.exe" air_IAPWS_test1
T = 300.0 degree K P = 100.0 kPa h = 27.013616191103285 kJ/kg s= 0.09810900607279928kJ/kgK Cp
=1.0063533483681613kJ/kgK

> Terminated with exit code 0.
```

Input known pairs of method property:

tp	tro	vp	hp	ts
vt	pv	rop	ps	st
rot	pro	ph	sp	

Output array of property method:

0	P, pressure	kPa
1	T, temperature	Degree K
2	v, specific volume	m^3/kg
3	h, enthalpy	kJ/kg
4	u, internal energy	kJ/kg
5	s, entropy	kJ/(kgK)

6	x, quality	kg vapor/kg total
7	Cp, specific heat at constant	kJ/(kgK)
8	Cv, specific heat at constant	kJ/(kgK)
9	α isobaric thermal expansion	1/K
10	κ isothermal compressibility	kPa
11	μ Dynamic viscosity	Pa.s
12	k Thermal conductivity	W/(mK)
13	Surface tension	N/m
14	Prandtl number	
15	density	kg/m^3
16	speed of sound	m/s

2.10 IAPWS EQUATION OF STATE FOR HUMID AIR (DRY AIR – WATER VAPOR MIXTURE)

Equation of state formulations of humid air will be considered three basic parts. The first part is for water properties, the second part is for the dry air properties and the third part is for the mixed interactions of dry air and water(air-water cross virial part). The equation of state is represented here in terms of the specific Helmholtz free energy of humid air, f ^{AV}, expressed as a function of dry-air mass fraction A (kg dry air/kg total), temperature T (degree K) and humid-air mass density ρ(kg/m^3).[58]

$$f^{AV}(A,T,\rho) = (1-A)f^{V}(T,\rho^{V}) + Af^{A}(T,\rho^{A}) + f^{mix}(A,T,\rho) \quad (2.10.1)$$

In this equation $\rho^{A} = A\rho$ (2.10.2) and $\rho^{V} = (1-A)\rho$ (2.10.3)
The water-vapor part is given by IAPWS-95 Helmholtz free energy for water and steam[58]. Which was given in chapter 2.4. Dry air part of the equation is given in chapter 2.8[60]. In real gas mixtures interaction between different molecules requaires additional terms in the equation. Equation of state for water-dry air interaction:

$$f^{mix}(A,T,\rho) = \frac{2A(1-A)\rho RT}{M_A M_W}\left\{B^{AW}(T) + \frac{3}{4}\rho\left[\frac{A}{M_A}C^{AAW}(T) + \frac{(1-A)}{M_W}C^{AWW}(T)\right]\right\} \quad (2.10.4)$$

$$B^{AW}(T) = b^{*}\sum_{i=1}^{3} c_i \bar{T}^{d_i} \quad (2.10.5)$$

$$C^{AAW}(T) = c^{*}\sum_{i=0}^{4} a_i \bar{T}^{-i} \quad (2.10.6)$$

$$C^{AWW}(T) = -c^{*}exp\left[\sum_{i=0}^{4} b_i \bar{T}^{-i}\right] \quad (2.10.7)$$

Where $\bar{T} = \frac{T}{100}$ Coefficients of the equation is given in Table 2.10.1

Table (2.10.1 Coefficients of Equation 2.10.4-2.10.7

i	a_i	b_i	c_i	d_i
0	4.827370E-04	-1.072888E+01	0.000000E+00	0.000000E+00
1	1.056780E-03	3.478020E+01	6.656870E+01	-2.370000E-01
2	-6.563940E-03	-3.833830E+01	-2.388340E+02	-1.048000E+00
3	2.944420E-02	3.340600E+01	-1.767550E+02	-3.183000E+00
4	-3.193170E-02	0.000000E+00	0.000000E+00	0.000000E+00

It should be note that this equation is also function of dry-air mass fraction A(kg dry air/kg total). Now we can look at termodynamic properties of humid air as combined equation.

Pressure (kPa): $P(A,T,ro) = \rho^2 \frac{\partial f^{AV}(A,T,\rho)}{\partial \rho}$ (2.10.8)

Entropy (kJ/(kgK)) : $s(T,ro) = -\frac{\partial f^{AV}(A,T,\rho)}{\partial T}$ (2.10.9)

Enthalpy (kJ/kg) : $h(T,ro) = f^{AV}(A,T,\rho) - T\frac{\partial f^{AV}(A,T,\rho)}{\partial T} + \rho\frac{\partial f^{AV}(A,T,\rho)}{\partial \rho}$ (2.10.10)

Isochoric specific heat C_p(kJ/(kgK)) :

$$C_p(T,ro) = -T\frac{\partial^2 f^{AV}(A,T,\rho)}{\partial T \partial T} + \frac{T\rho\left[\frac{\partial^2 f^{AV}(A,T,\rho)}{\partial T \partial \rho}\right]^2}{\left(2\frac{\partial f^{AV}(A,T,\rho)}{\partial \rho} + \rho\frac{f^{AV}(A,T,\rho)}{\partial \rho \partial \rho}\right)} \quad (2.10.11)$$

Humid air As a gas mixture, usually utilised in air conditioning industry, definitions used in standart air conditioning industry should also be expressed. Some of these properties are:

Humidity ratio $w = \left(\frac{1}{A}\right) - 1$ (2.10.12)

Mole fraction of dry air $x_A = \frac{A(M_W/M_A)}{1-A[1-(M_W/M_A)]}$ (2.10.13)

Mole fraction of water: $1 - x_A$ (2.10.14)

Mass fraction of dry air: $A = \frac{x_A}{1-(1-x_A)[1-(M_W/M_A)]}$ (2.10.15)

Partial pressure of water: $P^W = (1 - x_A)P$ (2.10.16)

Partial pressure of saturated water: $P^{Wsat} = (1 - x_A^{sat})P = P_\sigma(\mathrm{T})$ (2.10.17)

Relative humidity: $RH = \frac{P^W}{P^{Wsat}} = \frac{(1-x_A)}{(1-x_A^{sat})}$ (2.10.18)

Degree of saturation $DOS = \frac{m^W}{m^{Wsat}} = \frac{1-A}{1-A^{sat}}$ (2.10.19)

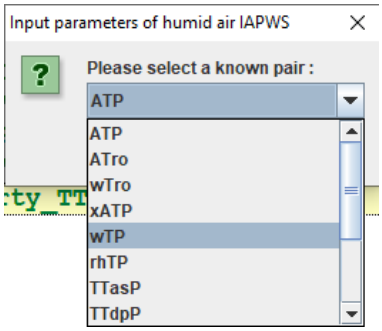
Dew point temperature (saturation temperature at partial pressure of water) $T_\sigma(P^W) = T_\sigma\left((1 - x_A)P\right)$ (2.10.20)

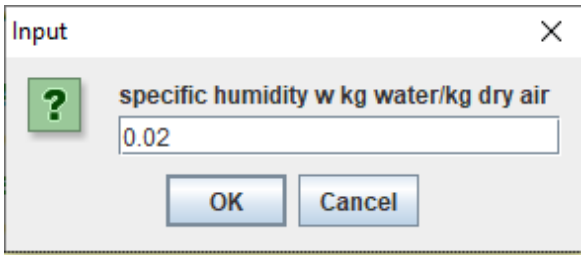
Program List of humid_air_IAPWS:

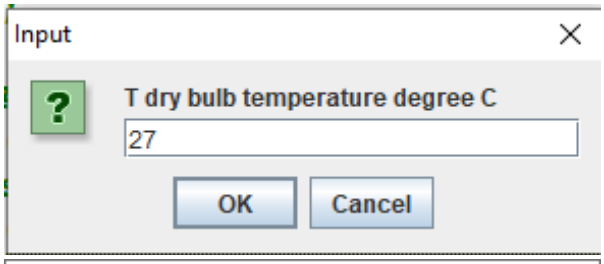
air_IAPWS	Basic Equation of state & input-output main program for dry air
steam_IAPWS95	Basic Equation of state for water & steam
humid_air_IAPWS	Basic Equation of state & input-output main program for humid air

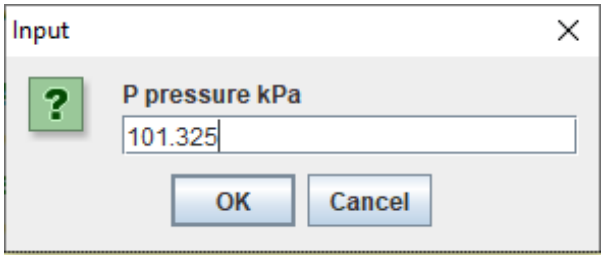
```
public static void main(String arg[])

{humid_air_IAPWS.open("English");}
```









Properties of humid air IAPWS formulation based

array no	property name	value humid air mass fr base	unit	value dry air mass fr base	unit
0	P, pressure	101.325	kPA		
1	T, temperature	300.15	deg C		
2	v, specific volume	1.6440568974407666	m³/kg humid air	1.676938035389582	m³/kg dry air
3	h, enthalpy	360.47196042398656	KJ/kg humid air	367.6813996324663	kJ/(kg dry air)
4	s, entropy	0.9410074463024197	KJ/kg humid air K	0.9598275952284682	kJ/(kg dry air K)
5	w, specific humidity, hu...	0.02	kg vap/kg dry air		
6	A, mass fraction of dry air	0.9803921568627451	kg dry air/kg humid air		
7	xA, mole fraction of dry air	0.9688452609721077	kmol dry air/kmol humid...		
8	relative humidity	3.668072052863495E-4			
9	Adyabatic saturation tem...	NaN	degree C		
10	dew point temperature	24.9307162072634	degree C		
11	η Dynamic viscosity	5.015915176909041E-5	Pa.s		
12	k Thermal conductivity	0.06598847625199475	W/mK		
13	degree of saturation	0.019607843137254943			
14	Pa dry air partial pressure	98.16824606799881	kPA		
15	Pv water moisture partial ...	3.1567539320011915	kPa		
16	ρ density	0.608251455017559	kg humid air/m³	0.5963249558995676	kg dry air/m³
17	Cp isochoric specific heat	1.063883731814466	kJ/(kg humid air K)	1.0851614064507553	kJ/(kg dry air K)
18	Cv isovolumetric specific...	0.7731139654903227	kJ/(kg humid air K)	0.7885762448001292	kJ/(kg dry air K)
19	ha enthalpy of dry air KJ/...	306.14318787583017	kJ/(kg humid air)	312.26605163334676	kJ/(kg dry air K)
20	hv enthalpy of vapor KJ/...	3077.2032084265547	kJ/(kg water)		

Our basic purpose of giving EOS codes are creating processes by using them. In order to do this a sample code of using the class and the key input and outputs are given below:

```
public class humid_air_IAPWS_test2
{public static void main(String arg[])
 { humid_air_IAPWS ha=new humid_air_IAPWS();
   double w=0.01;    //kg su buharı/kg kuru hava
   double T=300.0;   // derece Kelvin
   double P=101.325; // kPa
   double a[]=ha.property("wTP",w,T,P);
   double h=a[3];
   double s=a[5];
   double rh=a[8];
   double Tas=a[9];
   double Td=a[10];
   String ss="T = "+T+" degree K"+" P = "+P+" kPa"+"w = "+w+"kg water vapor/kg dry air"+" h = "+h+" kJ/kg humid
air"+" s= "+s+"kJ/kg humid airK "+"rh="+rh+" Tas="+Tas+" degree K"+" Tdew="+Td+" degree K";
   System.out.println(ss);
   }
}
```

```
---------- Capture Output ----------
> "E:\co\java\bin\java.exe" humid_air_IAPWS_test2
T = 300.0 degree K P = 101.325 kPaw = 0.01kg water vapor/kg dry air h = 51.98926992503951 kJ/kg humid air s=
0.01kJ/kg humid airK rh=0.453344930359494 Tas=18.657901835175323 degree K Tdew=14.04311399616131 degree K
> Terminated with exit code 0.
```

Input property triplets of property class

String variable	First variable	Unit of first variable	second variable	Unit of secondvariable	third variable	Unit of third variable
ATP	A mass ratio	Kg dry air/kg moist air	T temperature	Degree K	P pressure	kPa
ATro	A mass ratio	Kg dry air/kg moist air	T temperature	Degree K	ρ density	Kg moist air/m^3
wTro	Specific humidity	Kg water vapor/kg dry air	T temperature	Degree K	ρ density	Kg moist air/m^3
xATP	xA molar ratio	Kmol dry air/kmol moist air	T temperature	Degree K	P pressure	kPa
wTP	Specific humidity	Kg water vapor/kg dry air	T temperature	Degree K	P pressure	kPa
TwP	T temperature	Degree K	P pressure		P pressure	kPa
rhTP	Relative humidity	0-1	T temperature	Degree K	P pressure	kPa
TrhP	T temperature	Degree K	Relative humidity	0-1	P pressure	kPa
TTasP	T temperature	Degree K	T adiabatic saturation temperature	Degree K	P pressure	kPa
TTdpP	T temperature	Degree K	T dew point temperature	Degree K	P pressure	kPa
AhP	A mass ratio	Kg dry air/kg moist air	enthalpy	kJ/kg moist air	P pressure	kPa
whP	Specific humidity	Kg water vapor/kg dry air	enthalpy	kJ/kg moist air	P pressure	kPa
hwP	enthalpy	kJ/kg moist air	Specific humidity	Kg water vapor/kg dry air	P pressure	kPa
ArhP	A mass ratio	Kg dry air/kg moist air	enthalpy	kJ/kg moist air	P pressure	kPa
wrhP	Specific humidity	Kg water vapor/kg dry air	enthalpy	kJ/kg moist air	P pressure	kPa
rhwP			Specific humidity	Kg water vapor/kg dry air	P pressure	kPa

Output properties of property class

No	name	unit
0	P, pressure	kPA
1	T, temperature	deg C
2	v, specific volume	m^3/kg humid air
3	h, enthalpy	KJ/kg humid air
4	s, entropy	KJ/kg humid air K
5	w, specific humidity, humidity ratio	kg vap/kg dry air
6	A, mass fraction of dry air	kg dry air/kg humid air
7	xA, mole fraction of dry air	kmol dry air/kmol humid
8	relative humidity	
9	Adyabatic saturation temperature	degree C
10	dew point temperature	degree C
11	η Dynamic viscosity	Pa.s
12	k Thermal conductivity	W/mK
13	degree of saturation	
14	Pa dry air partial pressure	kPA
15	Pv water moisture partial pressure	kPa

16	ρ density	kg humid air/m³
17	Cp isochoric specific heat	kJ/(kg humid air K)
18	Cv isovolumetric specific heat	kJ/(kg humid air K)
19	ha enthalpy of dry air KJ/kg dry air	kJ/(kg humid air)
20	hv enthalpy of vapor KJ/kg water	kJ/(kg water)

2.11 PENG-ROBINSON-STRYJEK-VERA (PRSV) EQUATION OF STATES:

Peng-Robinson-Stryjek-Vera (PRSV) equation of state is usually employed fluid mixtures rather than the pure fluids. This equation can be used as EOS for refrigerants R404A, R407C, R401A, R401B, R402A, R402B

$$P(v,T) = \frac{RT}{v-b} - \frac{a}{(v^2+2bv+b^2)} \qquad (2.11\text{-}1)$$

Coefficients defined as

$$a = \sum_{i=1}^{3}\sum_{j=1}^{3} x_i\, x_j a_{ij} \quad (2.11-2)$$

ve

$$b = \sum_{i=1}^{3} x_i b_i \quad (2.11-3)$$

where

x_i=i mole percentages of fluid i in the mixture.

$$a_i = \left(0.457235\frac{R^2T_{ci}^2}{P_{ci}}\right)\alpha_i \quad (2.11-4)$$

$$b_i = \left(0.077796\frac{RT_{ci}}{P_{ci}}\right) \quad (2.11-5)$$

k_{ij}=interaction coefficient for species i and j

$$\alpha_i = \left[1+\kappa_i T_{ri}^{0.5}\right]^2 \quad (2.11-6)$$

$$\kappa_i = \kappa_{0i} + \kappa_{1i}\left[\left(1+T_{ri}^{0.5}\right)(0.7-T_{ri})\right]$$

Note: $k_i=k_{oi}$ for Tr>0.7

$$\kappa_{0i} = 0.378893 + 1.4897153\omega_i - 0.17131848\omega_i^2 + 0.019655\omega_i^3 \quad (2.11\text{-}8)$$

Where the value of κ_{1i} is the single pure compound adjustable parameter. The value of κ_{1i} is depends on the values for critical constants T_c, P_c and acentric factor ω. T_{ri} is the reduced temperature and defined as

$$T_{ri} = \frac{T_i}{T_{ci}} \quad (2.11\text{-}9)$$

As an example component values for R407C refrigerants is as follows

Table 2.10-1 R407C component values

Component	i	T_{ci}	P_{ci}	ω_i	κ_{1i}	x_i
R32	1	351.60	5830.0	0.2763	-0.0250	0.38110
R125	2	339.19	3595.0	0.3023	0.0310	0.17956
R134a	3	374.20	4056.0	0.3266	-0.0060	0.43934

For R407C κ_{ij} values:

κ_{11} = 0.00000	κ_{12} = -0.00028	κ_{13} =-0.00815
κ_{21} = -0.00028	κ_{22} = 0.00000	κ_{23} =-0.00240

K_{31} = -0.00815	K_{32} = -0.00240	K_{33} = 0.00000

Ideal gas specific heat should also be given for the EOS. For R407 refrigerant these equation is:

$$C_p = \sum_{i=1}^{3} x_i C_{pi} \quad (2.11.10)$$

$$C_{pi} = 4.184(A_i + B_i T + C_i T^2 + D_i T^3 + E_i T^4 + F_i T^5 \quad (2.11-11)$$

Table 2.11-2 –for R407C Cpi coefficients Ai,Bi,Ci,Di,Ei,Fi :

A_1= 1.226880E+01	B_1=-0.699113E-01
A_2= 1.170140E+01	B_2= 0.216411E-01
A_3= 0.463685e+01	B_3= 0.617914E-01
C_1= 0.394642E-03	D_1=-0.837462E-06
C_2= 0.868526E-04	D_2=-0.112776E-06
C_3=-0.309907E-04	D_3= 0.000000E+00
E_1= 0.859548E -09	F_1=0.000000E+00
E_2= 0.000000E+00	F_2=0.000000E+00
E_3= 0.000000E+00	F_3=0.000000E+00

Saturation pressure equation:

$\log_n P_{sat} = A + B/T + C \log_n T + D T^2$ (2.11-12)

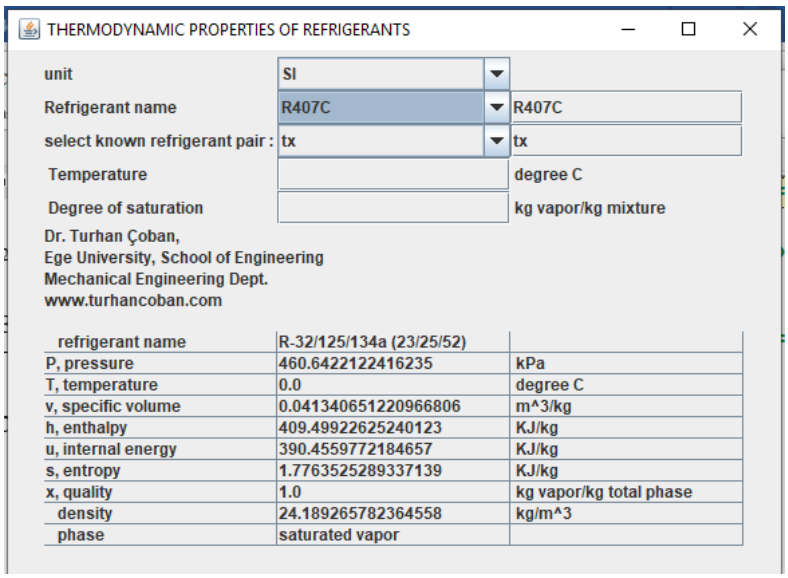
In this equation P_{sat} is saturation pressure

For R407C A,B,C,D coefficient of the equation.

A=4.27103E+01 B=-3.34460E+03 C=-4.39387E+00 D=6.86997E-06

Program List of refrgerant:

refrigerant	Basic Equation of state
ref_CS3	Saturation properties by using cubic spline
refTable	User graphic interface



2.12 THERMODYNAMIC PROPERTIES OF AMMONIA WATER MIXTURE

Ammonia-Water mixture is used in absorbtion refrigeration systems and Kalina power cycles, therefore its properties are important for technical applications. So proporties of this mixture will be investigated in this chapter. Pure water and ammonia equation of states were introduced previously. Mixture properties of this pair can not be obtain direcly by using simple mixing rules due to additional energy transfers while mixing up this two fluids. In order to investigate thermodynamic properties of the mixture, we will start up from liquid saturation temperature of mixture. The equation is:

$$T(P,x) = T_0 \sum_i a_i \, (1-x)^{m_i} \left[ln\left(\frac{P_0}{P}\right) \right]^{n_i} \quad (2.12-1)$$

In this equation x is the molar percentage of ammonia in liquid phase. P is pressure in MPa, T_0= 100 K, p_0= 2 MPa are reference values. a_i, m_i and n_i, are curve fitting values of the equation and given in the table below.

Table 2.12-1 liquid- saturation temperature equation coefficients of ammonia-water mix

i	m_i	n_i	a_i
1	0	0	3.223020000000E+00
2	0	1	-3.842060000000E-01
3	0	2	4.609650000000E-02
4	0	3	-3.789450000000E-03
5	0	4	1.356100000000E-04
6	1	0	4.877550000000E-01
7	1	1	-1.201080000000E-01
8	1	2	1.061540000000E-02
9	2	3	-5.335890000000E-04
10	4	0	7.850410000000E+00
11	5	0	-1.159410000000E+01
12	5	1	-5.231500000000E-02
13	6	0	4.895960000000E+00
14	13	1	4.210590000000E-02

vapor saturation temperature of mixture is

$$(P,y) = T_0 \sum_i a_i \, (1-y)^{\frac{m_i}{4}} \left[ln\left(\frac{P_0}{P}\right) \right]^{n_i} \quad (2.12-2)$$

In this equation y is the molar percentage of ammonia in gas phase. P is pressure in MPa, T_0= 100 K, p_0= 2 MPa are reference values. a_i, m_i and n_i, are curve fitting values of the equation and given in the table below.

Table 2.12-2 vapor- saturation temperature equation coefficients of ammonia-water mix

i	m_i	n_i	a_i
1	0	0	3.240040E+00
2	0	1	-3.959200E-01
3	0	2	4.356240E-02
4	0	3	-2.189430E-03
5	1	0	-1.435260E+00
6	1	1	1.052560E+00
7	1	2	-7.192810E-02
8	2	0	1.223620E+01
9	2	1	-2.243680E+00
10	3	0	-2.017800E+01
11	3	1	1.108340E+00

12	4	0	1.453990E+01
13	4	2	6.443120E-01
14	5	0	-2.212460E+00
15	5	2	-7.562660E-01
16	6	0	-1.355290E+00
17	7	2	1.835410E-01

Liquid phase enthalpy of the mixture :

$$h_L(T,x) = h_0 \sum_i a_i \left(\frac{T}{T_0} - 1\right)^{m_i} x^{n_i} \quad (2.12-3)$$

In this equation x is the molar percentage of ammonia in liquid phase. T_0= 273.16 K, h_0= 100 kJ/kg, are reference values. a_i, m_i and n_i, are curve fitting values of the equation and given in the table below.

Table 2.12-3 mixture liquid enthalpy

i	m_i	n_i	a_i
1	0	1	-7.61080000E+00
2	0	4	2.56905000E+01
3	0	8	-2.47092000E+02
4	0	9	3.25952000E+02
5	0	12	-1.58854000E+02
6	0	14	6.19084000E+01
7	1	0	1.14314000E+01
8	1	1	1.18157000E+00
9	2	1	2.84179000E+00
10	3	3	7.41609000E+00
11	5	3	8.91844000E+02
12	5	4	-1.61309000E+03
13	5	5	6.22106000E+02
14	6	2	-2.07588000E+02
15	6	4	-6.87393000E+00
16	8	0	3.50716000E+00

Mixture gas enthalpy :

$$h_G(T,y) = h_0 \sum_i a_i \left(\frac{T}{T_0} - 1\right)^{m_i} (1-y)^{n_i/4} \quad (2.12-4)$$

In this equation y is the molar percentage of ammonia in gas phase. T_0=324.0 K, h_0=1000 kJ/kg, are referance temperature and pressure. a_i, m_i and n_i, are curve fitting values of the equation and given in the table below.

Table 2.12-4 mixture vapor enthalpy

i	m_i	n_i	a_i
1	0	0	1.288270000E+00
2	1	0	1.252470000E-01
3	2	0	-2.087480000E+00
4	3	0	2.176960000E+00
5	0	2	2.356870000E+00
6	1	2	-8.869870000E+00
7	2	2	1.026350000E+01
8	3	2	-2.374400000E+00
9	0	3	-6.701550000E+00
10	1	3	1.645080000E+01
11	2	3	-9.368490000E+00
12	0	4	8.422540000E+00
13	1	4	-8.588070000E+00
14	0	5	-2.770490000E+00
15	4	6	-9.612480000E-01
16	2	7	9.880090000E-01
17	1	10	3.084820000E-01

Mixture specific volume can directly calculated from pure component specific volumes. For comparisons, table form of the mixture specific volüme values are given in the table below.

Table 2.12-5 Saturated liquid phase ammonia vapor mixture specific volume m^3/kg

T.	Ammonia concentration % mass											T
C	0	10	20	30	40	50	60	70	80	90	100	C
-10	0.00100	0.00103	0.00106	0.00109	0.00114	0.00118	0.00122	0.00128	0.00135	0.00142	0.00151	-10
0	0.00100	0.00103	0.00107	0.00110	0.00114	0.00119	0.00124	0.00130	0.00137	0.00146	0.00156	0
10	0.00100	0.00104	0.00107	0.00111	0.00115	0.00120	0.00125	0.00132	0.00139	0.00149	0.00160	10
20	0.00100	0.00104	0.00108	0.00112	0.00116	0.00121	0.00127	0.00133	0.00142	0.00152	0.00164	20
30	0.00100	0.00105	0.00108	0.00113	0.00117	0.00123	0.00128	0.00135	0.00145	0.00156	0.00168	30
40	0.00101	0.00105	0.00109	0.00114	0.00119	0.00124	0.00130	0.00138	0.00148	0.00159	0.00173	40
50	0.00101	0.00106	0.00110	0.00115	0.00120	0.00125	0.00132	0.00140	0.00151	0.00163	0.00177	50
60	0.00102	0.00106	0.00111	0.00116	0.00121	0.00127	0.00134	0.00143	0.00154	0.00167	0.00183	60
70	0.00102	0.00107	0.00112	0.00117	0.00122	0.00129	0.00136	0.00146	0.00158	0.00172	0.00190	70
80	0.00103	0.00108	0.00113	0.00118	0.00124	0.00130	0.00139	0.00149	0.00162	0.00178	0.00198	80
90	0.00104	0.00109	0.00114	0.00119	0.00125	0.00132	0.00141	0.00153	0.00167	0.00184	0.00208	90
100	0.00104	0.00110	0.00115	0.00121	0.00127	0.00135	0.00145	0.00157	0.00172	0.00191	0.00219	100

A program developed to solve water-ammonia mixture.
Program List of refrgerant:

ammonia	Basic Equation of state
ammoniaTable	Graphical user interface of ammonia
amoniaModel	Subprogram of GUI
steam	Basic equation of state
water_ammonia_mix	Basic Equation of state (utilises ammonia and steam classses)

Program 2.12-1 Water ammonia saturation phase diagram at 1 atm pressure

```
 import java.io.*;
import javax.swing.*;
public class water_ammonia_mix_test{

        public static void main(String arg[])
{
        water_ammonia_mix1 wamix=new water_ammonia_mix1();
        double T1[]=new double[501];
        double T2[]=new double[501];
        double X[]=new double[501];
        double x1[]=new double[2];
        double x2[]=new double[2];
        double t1[]=new double[2];
        double t2[]=new double[2];
        double P=101.325; //kPa
        for(int i=0;i<T1.length;i++)
        {X[i]=i/500.0;
        T1[i]=wamix.T_l(P,X[i])-273.15;
        T2[i]=wamix.T_g(P,X[i])-273.15;
        }
        graph2D p=new graph2D();
        p.addData(X,T1);
```

```
            p.addData(X,T2);
            x1[0]=0.3;
            x1[1]=0.3;
            x2[0]=0.7;
            x2[1]=0.7;
            t1[0]=wamix.T_l(P,x1[0])-273.15;
            t1[1]=wamix.T_g(P,x1[1])-273.15;
            t2[0]=wamix.T_l(P,x2[0])-273.15;
            t2[1]=wamix.T_g(P,x2[1])-273.15;
            p.addData(x1,t1,0,"blue");
            p.addData(x2,t2,0,"red");
            p.setPlabel("Water-Ammonia mixture phase diagram P ="+P+" kPa");
    p.setXlabel("x molar fraction of NH3 in the liquid phase");
    p.setYlabel("Temperature degree C");
            p.plot();

            }
}
```

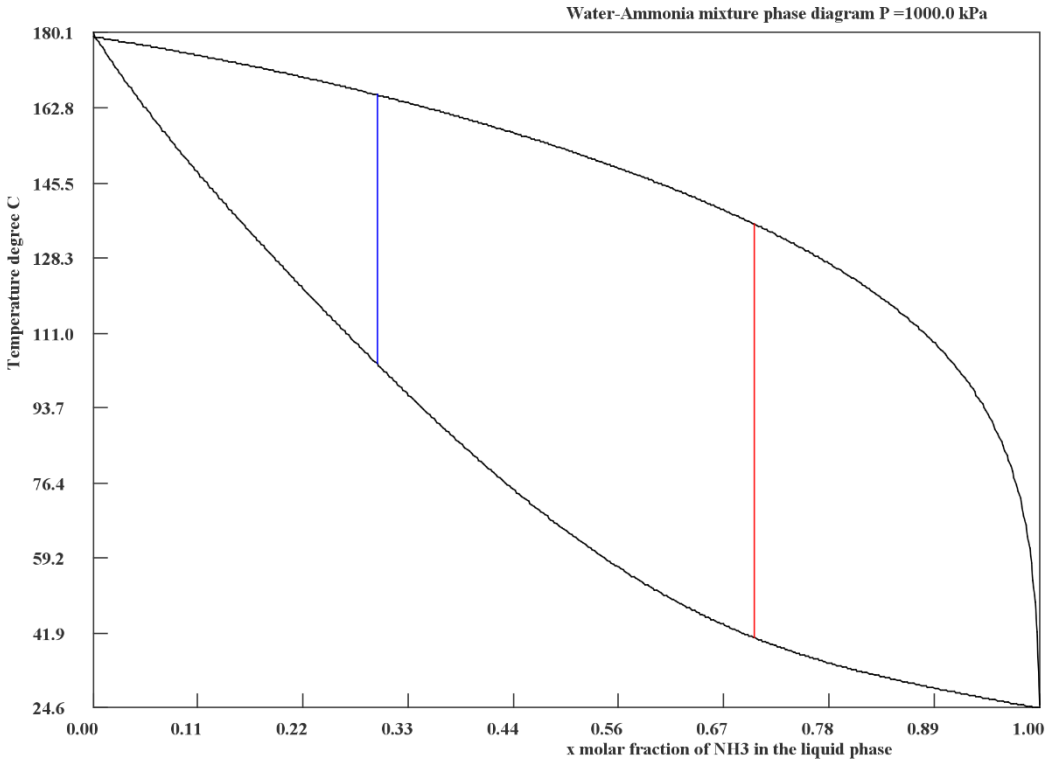


Figure 2.14-1 Water ammonia saturation phase diagram at 101.325 kPa pressure

2.13 THERMODYNAMIC PROPERTIES OF LITHIUM BROMIDE WATER MIXTURE

2.13 THERMODYNAMIC PROPERTIES OF LITHIUM BROMIDE WATER MIXTURE

Lithium bromide-water mixture can also be used in absorbtion cycle and liquid air drying process, therefore it is important to obtain thermodynamic properties of this mixture as well Similar to Ammonia-Water mixture, mixture properties of lithiumbromide-water can not be obtain direcly by using simple mixing rules due to additional energy transfers while mixing up this two fluids. The pure

Lithiumbromide is a salt and can not be obtain in liquid form, therefore the properties of pure liquid is not available to us. We will be looked at how to obtain thermodynamic properties of this mixture.

$P(T,x) = P_\sigma, \overleftrightarrow{\leftarrow}\overleftrightarrow{\leftarrow}\overleftrightarrow{\leftarrow}\overleftrightarrow{\leftarrow}\ \theta = T - \sum_{i=1}^{8} a_i x^{m_i}(0.4-x)^{n_i}\left(\frac{T}{T_c}\right)$ (2.10.1)

Where T_c=647.096 K

Table 2.10-1 Coefficients of P(T,x)

i	m_i	n_i	t_i	a_i
1	3	0	0	-2.41303000E+02
2	4	5	0	1.91750000E+07
3	4	6	0	-1.75521000E+08
4	8	3	0	3.25430000E+07
5	1	0	1	3.92571000E+02
6	1	2	1	-2.12626000E+03
7	4	6	1	1.85127000E+08
8	6	0	1	1.91216000E+03

Mixture density :

$\rho(T,x) = (1-x)\rho'(T) + \rho_c \sum_{i=1}^{2} a_i x^{m_i}\left(\frac{T}{T_c}\right)^{t_i}$ (2.10.2)

Where ρ_c=17.873 kmol/m^3 , T_c=647.096 K

Table 2.10-2 Coefficients of ρ(T,x)

i	m_i	t_i	a_i
1	1	0	1.746
2	1	6	4.709

$C_p(T,x) = (1-x)C_p'(T) + C_{p,t} \sum_{i=1}^{8} a_i x^{m_i}(0.4-x)^{n_i}\left(\frac{T_c}{T-T_0}\right)^{t_i}$ (2.10.3)

Where T_c=647.096 K, $C_{p,t}$=76.0226 kJ/(kmol K), T_0=221 K

Table 2.10-3 Coefficients of C_p(T,x)

i	mi	ni	ti	ai
1	2	0	0	-1.420940E+01
2	3	0	0	4.049430E+01
3	3	1	0	1.111350E+02
4	3	2	0	2.299800E+02
5	3	3	0	1.345260E+03
6	2	0	2	-1.410100E-02
7	1	3	3	1.249770E-02
8	1	2	4	-6.832090E-04

$h(T,x) = (1-x)h'(T) + h_c \sum_{i=1}^{30} a_i x^{m_i}(0.4-x)^{n_i}\left(\frac{T_c}{T-T_0}\right)^{t_i}$ (2.10.4)

Where h_c=37548.5 kJ//kmol, T_c=647.096 K, T_0=221 K

Table 2.10-4 Coefficients of h(T,x)

i	mi	ni	ti	ai
1	1	0	0	2.274310E+00
2	1	1	0	-7.995110E+00
3	2	6	0	3.852390E+02
4	3	6	0	-1.639400E+04
5	6	2	0	-4.225620E+02
6	1	0	1	1.133140E-01
7	3	0	1	-8.334740E+00
8	5	4	1	-1.738330E+04
9	4	0	2	6.497630E+00
10	5	4	2	3.245520E+03
11	5	5	2	-1.346430E+04
12	6	5	2	3.993220E+04

13	6	6	2	-2.588770E+05
14	1	0	3	-1.930460E-03
15	2	3	3	2.806160E+00
16	2	5	3	-4.044790E+01
17	2	7	3	1.453420E+02
18	5	0	3	-2.748730E+00
19	6	3	3	-4.497430E+02
20	7	1	3	-1.217940E+01
21	1	0	4	-5.837390E-03
22	1	4	4	2.339100E-01
23	2	2	4	3.418880E-01
24	2	6	4	8.852590E+00
25	2	7	4	-1.787310E+01
26	3	0	4	7.351790E-02
27	1	0	5	-1.794300E-04
28	1	1	5	1.842610E-03
29	1	2	5	-6.242820E-03
30	1	3	5	6.847650E-03

$$s(T,x) = (1-x)s'(T) + s_c \sum_{i=1}^{29} a_i x^{m_i} (0.4-x)^{n_i} \left(\frac{T_c}{T-T_0}\right)^{t_i} \qquad (2.10.5)$$

Where s_c=79.3933 kJ/(kmolK) , T_c=647.096 K, T_0=221 K

Table 2.10-5 Coefficients of s(T,x)

i	mi	ni	ti	ai
1	1	0	0	1.530910E+00
2	1	1	0	-4.525640E+00
3	2	6	0	6.983020E+02
4	3	6	0	-2.166640E+04
5	6	2	0	-1.475330E+03
6	1	0	1	8.470120E-02
7	3	0	1	-6.595230E+00
8	5	4	1	-2.953310E+04
9	1	0	2	9.563140E-03
10	2	0	2	-1.886790E-01
11	2	4	2	9.317520E+00
12	4	0	2	5.781040E+00
13	5	4	2	1.389310E+04
14	5	5	2	-1.717620E+04
15	6	2	2	4.151080E+02
16	6	5	2	-5.556470E+04
17	1	0	3	-4.234090E-03
18	3	4	3	3.052420E+01
19	5	0	3	-1.676200E+00
20	7	1	3	1.482830E+01
21	1	0	4	3.030550E-03
22	1	2	4	-4.018100E-02
23	1	4	4	1.492520E-01
24	2	7	4	2.592400E+00
25	3	1	4	-1.774210E-01
26	1	0	5	-6.996500E-05
27	1	1	5	6.050070E-04
28	1	2	5	-1.652280E-03
29	1	3	5	1.229660E-03

In these equations T is in degree K, T_0=221 K is a nonlinear parameter of equations. the pressure p(T,x) is expressed in terms of the pure water saturation pressure p_s(T). Pure water properties will be taken from equation of state described above. The properties of saturated pure liquid water as a function of temperature are denoted by apostrophe (r' (T), Cp'(T),h'(T) and s'(T)) in the equations. whereas quantities describing the solution have no subscript (p(T,x), h(T,x) etc.). In these set of

equations x is the mole fraction of LiBr in solution. Specific quantities and mass fraction w used in common computational practice are related to the molar quantities by the following relations:

$$w = \frac{xM_{LiBr}}{xM_{LiBr}+(1-x)M_w} \quad (2.10.6)$$

$$x = \frac{w/M_{LiBr}}{w/M_{LiBr}+(1-w)/M_w} \quad (2.10.7)$$

$$h(w) = \frac{h(x)}{xM_{LiBr}+(1-x)M_w} \quad (2.10.8)$$

and similarly for density, entropy and isobaric heat capacity. Graphical representation of the enthalpy-mass percentages of LiBr- Water mixture is given in figure 2.10.1

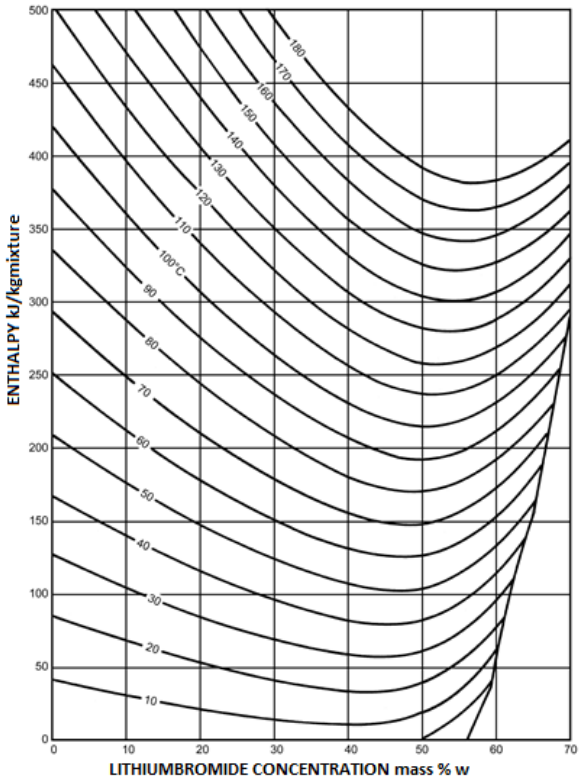


Figure 2.10-1 LithiumBromide-Water mixture enthalpy diagram

Program List of: water_lithiumbromide_mix

water_lithiumbromide_mix	Basic Equation of state
steam	Basic equation of state

Sample test program:

```
public class water_lithiumbromide_mixtest
{
public static void main(String arg[])
{
water_lithiumbromide_mix wlb=new water_lithiumbromide_mix();
System.out.println("hl(8.550668891085447,0.5)="+wlb.h_lt(8.550668891085447,0.5));
System.out.println("hlw(30,0.5)="+wlb.h_ltw(30.0,0.5));
System.out.println("T_l(30.0,0.5)="+wlb.t_l(30.0,0.5));
System.out.println("T_w(8.550668891085447,0.5)="+wlb.t_w(8.550668891085447,0.5));
System.out.println("P(30)="+wlb.P(30));
System.out.println("tp(4.245832460932154)="+wlb.t_P(4.245832460932154));
}}
```

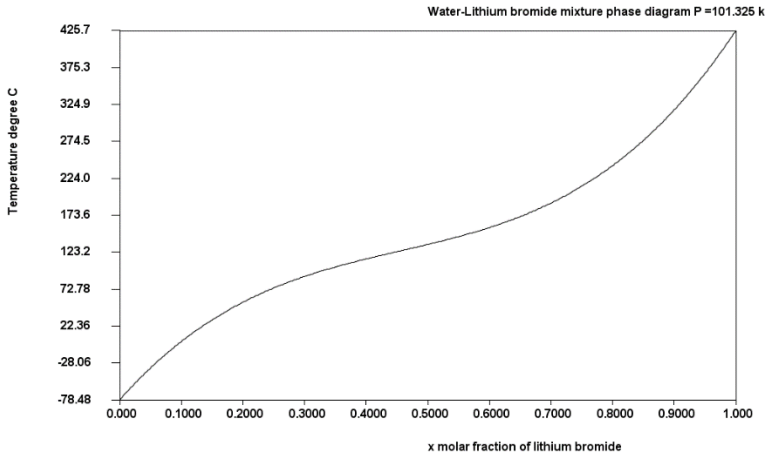
```
---------- Capture Output ----------
> "E:\co\java\bin\java.exe" water_lithiumbromide_mixtest
hl(8.550668891085447,0.5)=60.10609196250109
```

```
hlw(30,0.5)=60.10609196250109
T_l(30.0,0.5)=8.550668891085447
T_w(8.550668891085447,0.5)=30.0
P(30)=4.245832460932154
tp(4.245832460932154)=29.999999999999968

> Terminated with exit code 0.
```

```
import java.io.*;
import javax.swing.*;
public class water_lithiumbromide_mixtest2{

public static void main(String arg[])
{    water_lithiumbromide_mix wamix=new water_lithiumbromide_mix();
            double T1[]=new double[501];
            double T2[]=new double[501];
            double X[]=new double[501];
            double P=101.325;  //kPa
            for(int i=0;i<T1.length;i++)
            {X[i]=i/500.0;
            T1[i]=wamix.t_w(P,X[i]);
            T2[i]=wamix.t_w(P,X[i]);
            }
            Plot p=new Plot(X,T1);
            //p.addData(X,T2);
            p.setPlabel("Water-Lithium bromide mixture phase diagram P ="+P+" kPa");
    p.setXlabel("x molar fraction of lithium bromide");
    p.setYlabel("Temperature degree C");
            p.plot();
            }
}
```



2.14 THERMODYNAMIC PROPERTIES OF SECONDARY COOLANTS & HEAT TRANSFER FLUIDS

As a practical heat transfer fluid a secondary coolant or heat transfer fluid is utilized. In some of the refrigeration applications, it is required that transfer medium should performed in low temperature ranges (sub zero temperatures). The most common heat transfer fluid is water and air. For low temperature applications most common heat transfer fluids are brines (mixture of sodiumchloride NaCl and water or potasiumchloride and water), ethylene gylicol-water and propylene gylicol-water. Since they are binary mixtures, their properties depends on the mixture rate. In addition to that several synthetic oils are used as heat transfer fluids both on heat applications and cold applications. Curve fitting functions are used to express properties of binary mixtures. For example freezing points of a binary mixture is expressed as:

$$T_f(x) = a_0 + a_1x + a_2x^2 + a_3x^3 + a_4x^4 + a_5x^5 + a_6x^6 + a_7x^7 + a_8x^8 + a_9x^9 + + \frac{a_{10}}{x+0.01} + \frac{a_{11}}{x^2+0.01} + \frac{a_{12}}{x^3+0.01} + \frac{a_{13}}{x^4+0.01} + a_{15}\, log(x + 0.01)$$ (2.14.1)

In such a binary mixture, properties of density will be given with a curve fitting function again as:

$$\rho(T,x) = a_0 + a_1x + a_2T + a_3x^2 + a_4T^2 + a_5xT + a_6T^3 + a_7xT^2 + a_8T^4 + a_9xT^3 + +a_{10}T^5 + a_{11}xT^4 + a_{12}T^6 + a_{13}xT^5 + a_{14}T^7$$ (2.14.2)

$$C_p(T,x) = a_0 + a_1x + a_2T + a_3x^2 + a_4T + a_5xT + a_6T^3 + a_7xT^2 + a_8T^4 + a_9xT^3 + +a_{10}T^5 + a_{11}xT^4 + a_{12}T^6 + a_{13}xT^5 + a_{14}T^7$$ (2.14.3)

For the enthalpy and entropy general thermodynamic relation is used

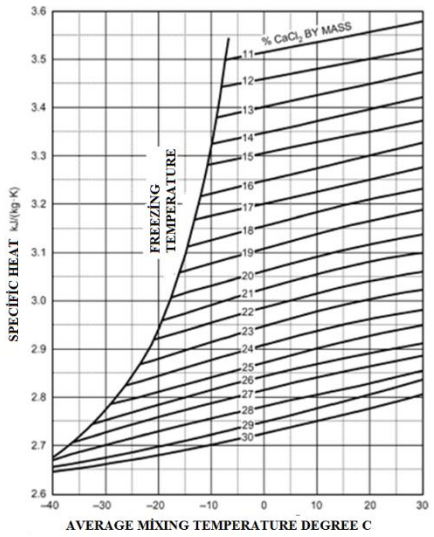
$h(T,x) = \int_{T_{ref}}^{T} C_p(T,x)dT$ where T_{ref}=0 degree C (2.14.4)

$s(T,x) = \int_{T_{ref}}^{T} \frac{C_p(T,x)}{T} dT$ (2.14.5)

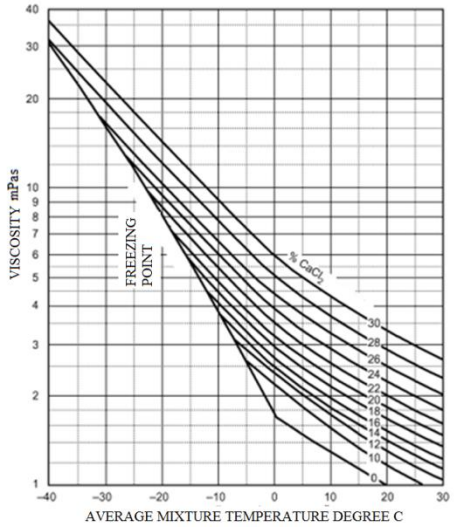
Some of the basic properties for such binary mixtures are given below as table and graphic forms

Table 2.14.1 Salt (CaCl) -water mixture

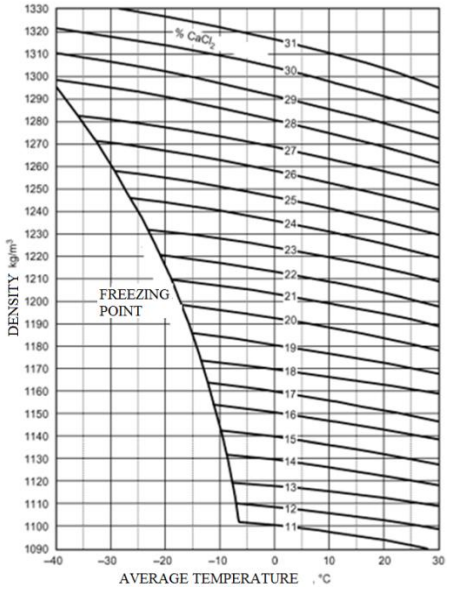
CaCl	Cp 15 C	Freezing temperatur	Density at 16 C		Densities at different temperatures			
%	J/kgK	degree C	NaCl	water	-20 C	-10	0 C	10 C
0	4184	0	0	999			,	
5	3866	-2.4	52.2	1044			1042	1041
6	3824	-2.9	63	1049			1051	1050
7	3757	-3.4	74.2	1059			1060	1059
8	3699	-4.1	85.5	1068			1070	1068
9	3636	-4.7	96.9	1078			1079	1077
10	3577	-5.4	108.6	1087			1088	1086
11	3523	-6.2	120.5	1095			1097	1095
12	3464	-7.1	132.5	1104			1107	1104
13	3414	-8	144.8	1113			1116	1114
14	3364	-9.2	157.1	1123			1126	1123
15	3318	-10.3	169.8	1132		1140	1136	1133
16	3259	-11.6	182.6	1141		1150	1145	1142
17	3209	-13	195.7	1152		1160	1155	1152
18	3163	-14.5	209	1161		1170	1165	1162
19	3121	-16.2	222.7	1171		1179	1175	1172
20	3084	-18	236	1180		1189	1185	1182
21	3050	-19.9	249.6	1189				
22	2996	-22.1	264.3	1201	1214	1210	1206	1202
23	2958	-24.4	278.7	1211				
24	2916	-26.8	293.5	1223	1235	1231	1227	1223
25	2882	-29.4	308.2	1232				
26	2853	-32.1	323.1	1242				
27	2816	-35.1	338.5	1253				
28	2782	-38.8	354	1264				
29	2753	-45.2	369.9	1275				
29.87	2741	-55	378.8	1289				
30	2732	-46	358.4	1294				
32	2678	-28.6	418.1	1316				
34	2636	-15.4	452	1339				



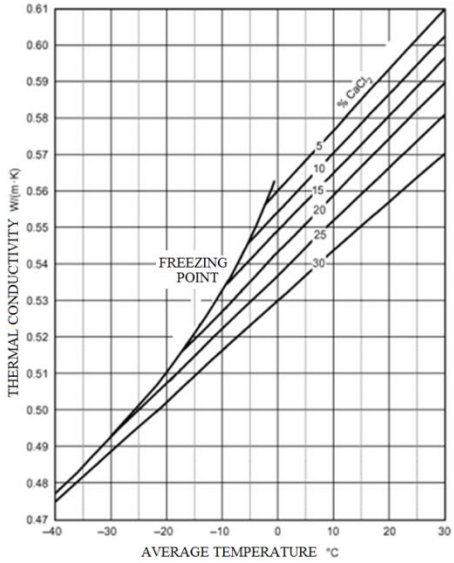
SPECIFIC HEAT OF CALCIUM CHLORIDE



VISCOSITY OF CALCIUM CHLORIDE



DENSITY OF CALCIUM CHLORIDE

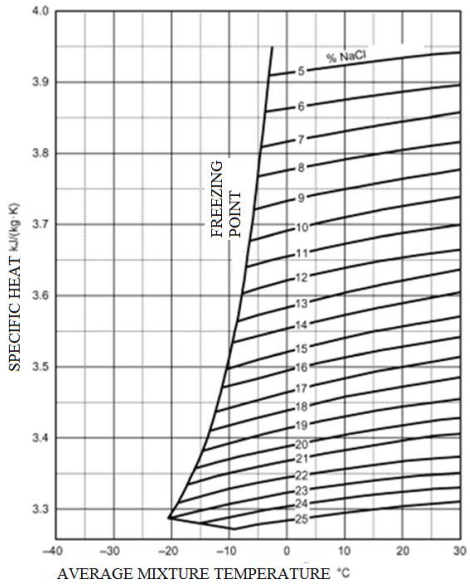


THERMAL CONDUCTIVITY OF CALCIUM CHLORIDE

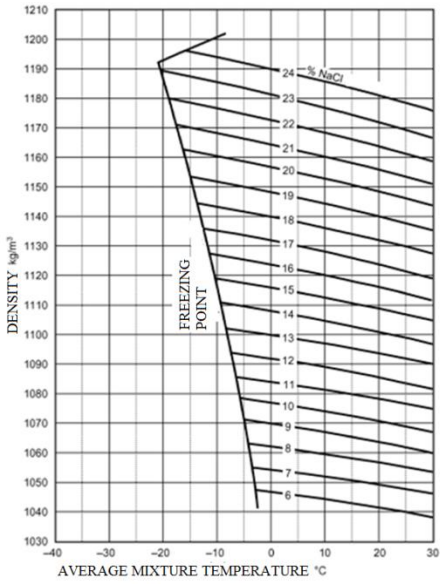
Table 2.14.2 Salt (NaCl) –water mixture

NaCl	Spec. heat 15 C	Freezing point	Density at 16 C		Density in various temperatures			
% by	J/kgK	degree C	NaCl	water	-10 C	0 C	10 C	20 C
0	4184	0	0	1000				
5	3925	-2.9	51.7	1035		1038.1	1036.5	1034
6	3879	-3.6	62.5	1043		1045.8	1043.9	1041.2
7	3836	-4.3	73.4	1049		1053.7	1051.4	1048.5
8	3795	-5	84.6	1057		1061.2	1058.9	1055.8
9	3753	-5.8	95.9	1065		1069	1066.4	1063.2
10	3715	-6.6	107.2	1072		1076.8	1074	1070.6
11	3678	-7.3	118.8	1080		1084.8	1081.6	1078.1
12	3640	-8.2	130.3	1086		1092.4	1089.6	1085.6
13	3607	-9.1	142.2	1094		1100.3	1097	1093.2
14	3573	-10.1	154.3	1102		1108.2	1104.7	1100.8
15	3544	-10.9	166.5	1110	1119.4	1116.2	1112.5	1108.5
16	3515	-11.9	178.9	1118	1127.6	1124.2	1120.4	1116.2

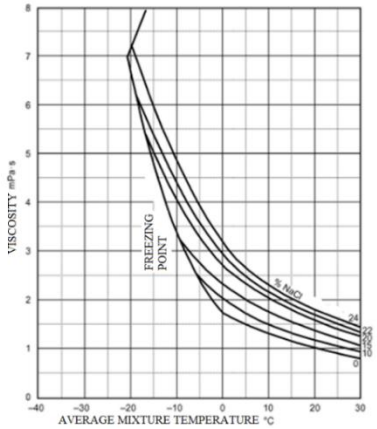
17	3485	-13	191.4	1126	1135.8	1132.2	1128.3	1124
18	3456	-14.1	204.1	1134	1144.1	1140.3	1136.2	1131.8
19	3427	-15.3	217	1142	1153.4	1148.5	1144.3	1139.7
20	3402	-16.5	230	1150	1160.7	1156.7	1154.1	1147.7
21	3376	-17.8	243.2	1158	1169.1	1165	1160.5	1155.8
22	3356	-19.1	256.6	1166	1177.6	1173.3	1168.7	1163.9
23	3330	-20.6	270	1174	1186.1	1181.7	1177	1172
24	3310	-15.7	283.7	1182	1194.7	1190.1	1185.3	1180.3
25	3289	-8.8	297.5	1190				
25.2		0						



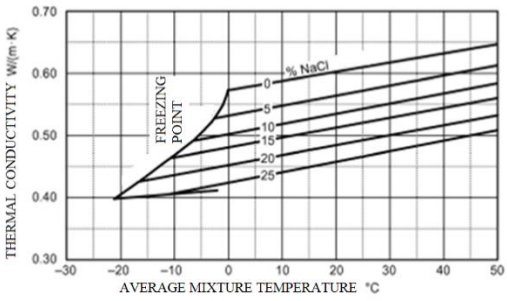
SODIUM CHLORIDE(SALT)-WATER MIXTURE SPECIFIC HEAT



SODIUM CHLORIDE(SALT)-WATER MIXTURE DENSITY



SODIUM CHLORIDE(SALT)-WATER MIXTURE VISCOSITY

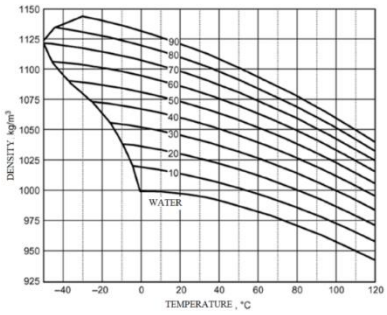


SODIUM CHLORIDE(SALT)-WATER MIXTURE THERMAL CONDUCTIVITY

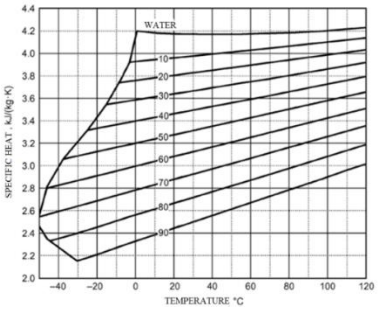
Table 2.14.3 Ethylene Gylicol -water mixture properties

ETHYLENE GYLICOL FREEZING AND BOILING POINTS				ETHYLENE GYLICOL FREEZING AND BOILING POINTS			
% by mass	% by mass	% by mass	% by mass	% by mass	volumetric	T freezing	Tboiling
0	0	0	100	0	0	0	100
5	4.4	-1.4	100.6	5	4.8	-1.6	100
10	8.9	-3.2	101.1	10	9.6	-3.3	100
15	13.6	-5.4	101.7	15	14.5	-5.1	100

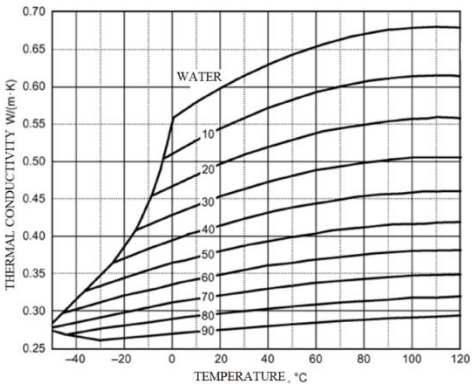
20	18.1	-7.8	102.2	20	19.4	-7.1	100.6
21	19.2	-8.4	102.2	21	20.4	-7.6	100.6
22	20.1	-8.9	102.2	22	21.4	-8	100.6
23	21	-9.5	102.8	23	22.4	-8.6	100.6
24	22	-10.2	102.8	24	23.4	-9.1	100.6
25	22.9	-10.7	103.3	25	24.4	-9.6	101.1
26	23.9	-11.4	103.3	26	25.3	-10.2	101.1
27	24.8	-12	103.3	27	26.4	-10.8	101.1
28	25.8	-12.7	103.9	28	27.4	-11.4	101.7
29	26.7	-13.3	103.9	29	28.4	-12	101.7
30	27.7	-14.1	104.4	30	29.4	-12.7	102.2
31	28.7	-14.8	104.4	31	30.4	-13.4	102.2
32	29.6	-15.4	104.4	32	31.4	-14.1	102.2
33	30.6	-16.2	104.4	33	32.4	-14.8	102.2
34	31.6	-17	104.4	34	33.5	-15.6	102.2
35	32.6	-17.9	105	35	34.4	-16.4	102.8
36	33.5	-18.6	105	36	35.5	-17.3	102.8
37	34.5	-19.4	105	37	36.5	-18.2	102.8
38	35.5	-20.3	105	38	37.5	-19.1	103.3
39	36.5	-21.3	105	39	38.5	-20.1	103.3
40	37.5	-22.3	105.6	40	39.6	-21.1	103.9
41	38.5	-23.2	105.6	41	40.6	-22.1	103.9
42	39.5	-24.3	105.6	42	41.6	-23.2	103.9
43	40.5	-25.3	106.1	43	42.6	-24.3	103.9
44	41.5	-26.4	106.1	44	43.7	-25.5	103.9
45	42.5	-27.5	106.7	45	44.7	-26.7	104.4
46	43.5	-28.8	106.7	46	45.7	-27.9	104.4
47	44.5	-29.8	106.7	47	46.8	-29.3	104.4
48	45.5	-31.1	106.7	48	47.8	-30.6	105
49	46.6	-32.6	106.7	49	48.9	-32.1	105
50	47.6	-33.8	107.2	50	49.9	-33.5	105.6
51	48.6	-35.1	107.2	51	50.9	-35	105.6
52	49.6	-36.4	107.2	52	51.9	-36.6	105.6
53	50.6	-37.9	107.8	53	53	-38.2	106.1
54	51.6	-39.3	107.8	54	54	-39.8	106.1
55	52.7	-41.1	108.3	55	55	-41.6	106.1
56	53.7	-42.6	108.3	56	56	-43.3	106.1
57	54.7	-44.2	108.9	57	57	-45.2	106.7
58	55.7	-45.6	108.9	58	58	-47.1	106.7
59	56.8	-47.1	109.4	59	59	-49	106.7
60	57.8	-48.3	110	60	60	-51.1	107.2
65	62.8	*	112.8	65	65	*	108.3
70	68.3	*	116.7	70	70	*	110
75	73.6	*	120	75	75	*	113.9
80	78.9	-46.8	123.9				
85	84.3	-36.9	133.9				
90	89.7	-29.8	140.6				
95	95	-19.4	158.3				



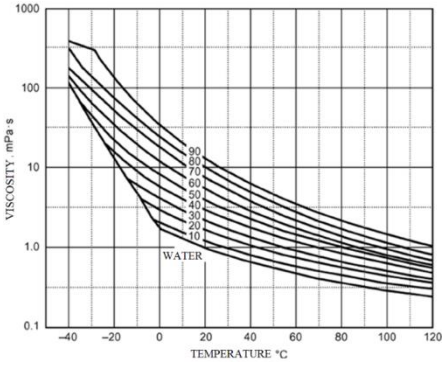
WATER ETHYLENE GYLICOL MIXTURE(% VOLUME) DENSITY



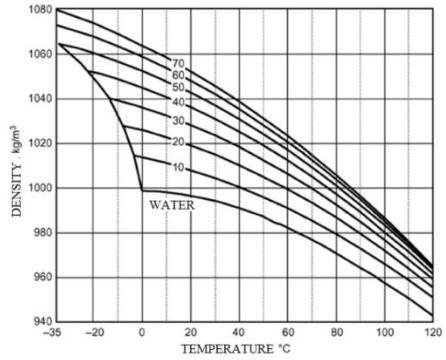
WATER ETHYLENE GYLICOL MIXTURE(% VOLUME)) SPECIFIC HEAT



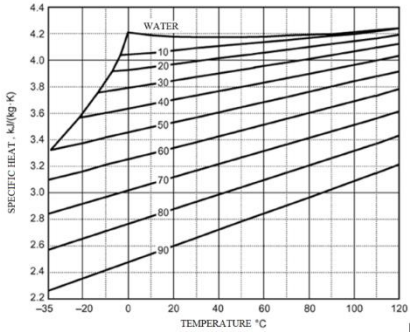
WATER ETHYLENE GYLICOL MIXTURE(% VOLUME) THERMAL CONDUCTIVITY



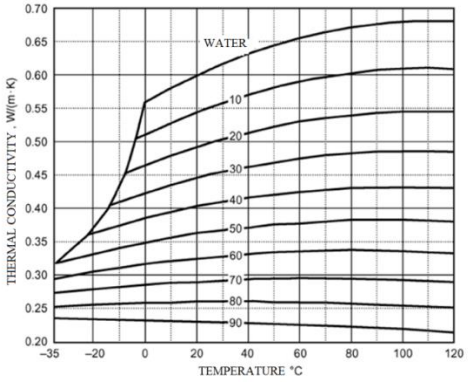
WATER ETHYLENE GYLICOL MIXTURE(% VOLUME) VISCOSITY



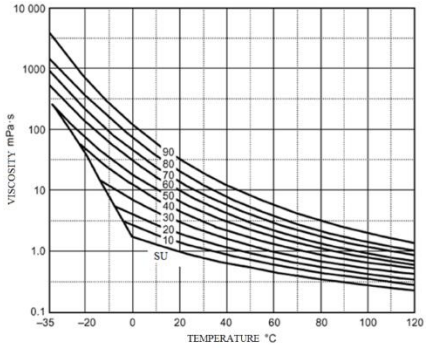
WATER PROPYLENE GYLICOL MIXTURE(% VOLUME) DENSITY



WATER PROPYLENE GYLICOL MIXTURE(% VOLUME) SPECIFIC HEAT



WATER PROPYLENE GYLICOL MIXTURE(% VOLUME) THERMAL CONDUCTIVITY



WATER PROPYLENE GYLICOL MIXTURE(% VOLUME) VISCOSITY

Table 2.14.4 polydimetilsiloaksan heat transfer fluid properties

Flame temperature: 46.7 C
Boiling point : 175 C
Freezing point : -111.1 C
Temperature range : -73.3 to 260 degree C

Temperature	Vapor pressure		density	Specific heat	Thermal conductivity
°C	kPa	mPa·s	kg/m3	kJ/(kg·K)	W/(m·K)
-73	0	12.4	924.6	1.41	0.1294
-70	0	11.2	922.1	1.418	0.1288
-60	0	8.26	913.5	1.443	0.1269
-50	0	6.24	905	1.469	0.1251

-40	0	4.83	896.4	1.495	0.1231
-30	0	3.81	887.9	1.52	0.1212
-20	0	3.07	879.3	1.546	0.1192
-10	0.01	2.51	870.7	1.572	0.1171
0	0.03	2.09	862	1.597	0.115
10	0.08	1.76	853.3	1.623	0.1129
20	0.16	1.49	844.5	1.649	0.1108
30	0.32	1.29	835.5	1.674	0.1086
40	0.61	1.12	826.5	1.7	0.1064
50	1.09	0.98	817.3	1.726	0.1042
60	1.85	0.86	807.9	1.751	0.1019
70	3.02	0.77	798.4	1.777	0.0996
80	4.76	0.69	788.7	1.803	0.0973
90	7.25	0.62	778.8	1.828	0.0949
100	10.73	0.56	768.7	1.854	0.0925
110	15.45	0.51	758.3	1.88	0.0901
120	21.75	0.47	747.7	1.905	0.0877
130	29.95	0.43	736.8	1.931	0.0852
140	40.45	0.4	725.6	1.957	0.0827
150	53.67	0.37	714.1	1.982	0.0802
160	70.06	0.34	702.3	2.008	0.0777
170	90.1	0.32	690.2	2.033	0.0751
180	114.29	0.3	677.7	2.059	0.0725
190	143.17	0.28	664.8	2.085	0.0699
200	177.27	0.26	651.6	2.11	0.0673
210	217.14	0.25	638	2.136	0.0646
220	263.36	0.24	623.9	2.162	0.062
230	316.47	0.22	609.5	2.187	0.0593
240	377.03	0.21	594.5	2.213	0.0566
250	445.61	0.2	579.1	2.239	0.0538
260	522.74	0.19	563.3	2.264	0.0511

Properties of d-limonen heat transfer fluid

Flame temperature : 46.1 C
Boiling point : 154.4 C
Freezing point : -96.7 C

Temperature degree C	Specific heat kJ/(kg K)	Viscosity mPas	Density kg/m^3	Thermal conductivity W/(m K)
-73	1.27	3.8	914.3	0.137
-50	1.39	3	897.1	0.133
-25	1.51	2.3	878.3	0.128
0	1.65	1.8	859.2	0.124
25	1.78	1.4	839.8	0.119
50	1.91	1.1	820.1	0.114
75	2.04	0.8	800	0.11
100	2.17	0.7	779.5	0.105
125	2.3	0.5	758.4	0.1
150	2.41	0.4	736.6	0.096

Table 2.14.5 Properties of Duratherm XLT-50 heat transfer oil

Temperature range : -46 to 177 C

Temperature degree C	Density (kg/m3)	Viscosity (Centipoise)	Specific heat (W/m.K)	Thermal (kJ/kgK)	Pressure (kPa)
-46	871	87.42	0.14	1.926	0
-40	868	59.49	0.138	1.926	0
-34	865	32.2	0.138	1.938	0
-29	863	13.91	0.138	1.95	0
-23	860	4.46	0.137	1.962	0
-18	857	3.87	0.137	1.974	0

-12	854	3.41	0.137	1.986	0
-7	852	3.03	0.136	1.998	0
-1	849	2.72	0.136	2.01	0
4	846	2.45	0.135	2.022	0
10	844	2.24	0.135	2.034	0
16	841	2.04	0.135	2.046	0
21	838	1.88	0.134	2.058	0
27	835	1.74	0.134	2.069	0
32	833	1.62	0.134	2.081	0
38	830	1.51	0.133	2.093	0.14
43	827	1.41	0.133	2.11	0.67
49	824	1.33	0.132	2.127	1.21
54	820	1.25	0.132	2.144	1.74
60	817	1.18	0.131	2.16	2.28
66	814	1.12	0.131	2.177	2.41
71	811	1.05	0.13	2.194	3.61
77	808	1	0.13	2.211	4.81
82	805	0.96	0.129	2.227	6.01
88	801	0.91	0.129	2.244	7.21
93	798	0.87	0.128	2.261	8.41
99	795	0.83	0.128	2.278	11.35
104	784	0.79	0.127	2.294	14.29
110	773	0.75	0.127	2.311	17.22
116	761	0.72	0.126	2.328	20.16
121	750	0.68	0.125	2.345	23.1
127	739	0.65	0.125	2.361	26.41
132	728	0.62	0.124	2.378	29.72
138	716	0.59	0.124	2.395	33.03
143	705	0.56	0.123	2.412	36.34
149	694	0.53	0.123	2.428	39.64
154	683	0.51	0.123	2.445	44.48
160	672	0.49	0.119	2.462	49.33
166	661	0.48	0.119	2.474	54.17
171	651	0.45	0.119	2.487	59.01
176	641	0.43	0.116	2.5	63.85

In order to convert the data to computer programs general least square curve fitting programs are used

Program 2.11.1 Surface least square curve fitting program

```
import java.io.*;
import java.util.*;
import javax.swing.*;
import java.awt.event.*;

// surface curve fitting
//f(x,y)=a0+a1*x+a2*T+a3*x^2+a4*T^2+a5*xT+a6*T^3+a7*xT^2+a8*T^2
//      +a9*xT^3+a10*T^5+a11*xT^4+a12*T^6+a13xT^5+a14*T^7
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[1]*x[1]*x[1];}
   else if (i==7) {xx=x[0]*x[1]*x[1];}
   else if (i==8) {xx=x[1]*x[1]*x[1]*x[1];}
   else if (i==9) {xx=x[0]*x[1]*x[1]*x[1];}
   else if (i==10){xx=x[1]*x[1]*x[1]*x[1]*x[1];}
   else if (i==11){xx=x[0]*x[1]*x[1]*x[1]*x[1];}
   else if (i==12){xx=x[1]*x[1]*x[1]*x[1]*x[1]*x[1];}
   else if (i==13){xx=x[0]*x[1]*x[1]*x[1]*x[1]*x[1];}
   else if (i==14){xx=x[1]*x[1]*x[1]*x[1]*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 : ");
  int n=Integer.parseInt(JOptionPane.showInputDialog("polinom derecesi (n)"));
  double c[][]=Text.readDouble(s1);
  Text.print(c,"a");
  double b[]=GeneralLeastSquare(c,n);
  Text.print(b,"b");
  double xi[][]=cikti(c,n,1);
  String s="";
  System.out.println("n_xi = "+xi.length+"n_xi[0] = "+xi[0].length);
  for(int i=0;i<xi.length;i++)
  { for(int j=0;j<xi[0].length;j++)
   {s+=xi[i][j]+" ";}
   s+="\n";
  }
  System.out.println(s);
  //System.out.println("output \n"+Matrix.toString(xi));
  //System.out.println("error \n"+Matrix.toString(hata(c,b)));

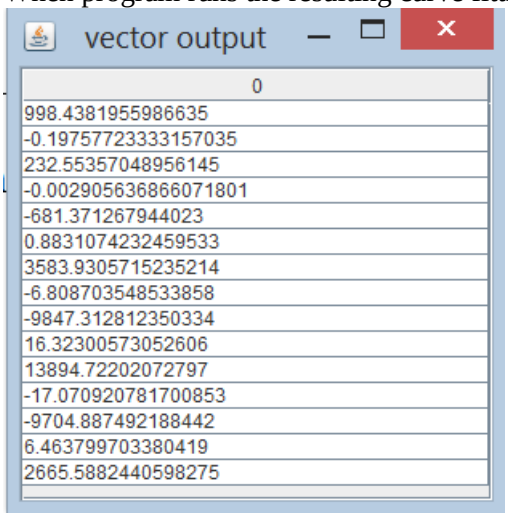
  }
}
```

A sample data for the curve fitting program:

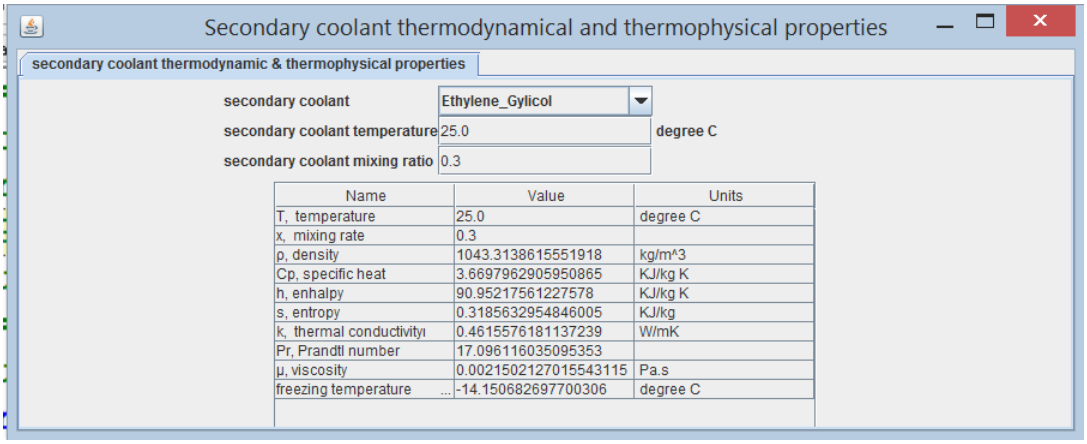
Temp	%volume ethylene gylicol density								
Degree	10%	20%	30%	40%	50%	60%	70%	80%	90%
-35						1089.9	1104.6	1118.6	1132.1
-30						1089	1103.5	1117.4	1130.7
-25					1088	1102.4	1116	1129.2	1141.9
-20				1072	1086.9	1101.1	1114.6	1127.6	1140.1
-15				1070.9	1085.6	1099.6	1113	1125.8	1138.1
-10			1054.3	1069.6	1084.2	1098.1	1111.3	1123.9	1136.1
-5		1036.9	1053.1	1068.3	1082.7	1096.4	1109.5	1121.9	1133.9
0	1018.7	1035.7	1051.8	1066.8	1081.1	1094.6	1107.5	1119.8	1131.6
5	1017.6	1034.4	1050.3	1065.2	1079.3	1092.7	1105.4	1117.6	1129.2
10	1016.3	1032.9	1048.8	1063.5	1077.5	1090.7	1103.2	1115.2	1126.7
15	1014.9	1031.4	1047.1	1061.7	1075.5	1088.5	1100.9	1112.7	1124
20	1013.3	1029.7	1045.3	1059.7	1073.4	1086.3	1098.5	1110.1	1121.2

25	1011.7	1027.9	1043.3	1057.6	1071.1	1083.9	1095.9	1107.4	1118.3
30	1009.9	1026	1041.3	1055.4	1068.8	1081.4	1093.2	1104.6	1115.3
35	1008	1024	1039.1	1053.1	1066.3	1078.7	1090.4	1101.6	1112.2
40	1006	1021.8	1036.8	1050.6	1063.7	1076	1087.5	1098.5	1108.9
45	1003.9	1019.6	1034.4	1048.1	1060.9	1073.1	1084.5	1095.3	1105.5
50	1001.6	1017.2	1031.8	1045.4	1058.1	1070.1	1081.3	1091.9	1102
55	999.23	1014.6	1029.2	1042.5	1055.1	1066.9	1078	1088.5	1098.4
60	996.72	1012	1026.4	1039.6	1052	1063.7	1074.6	1084.9	1094.6
65	994.1	1009.2	1023.5	1036.6	1048.8	1060.3	1071.1	1081.2	1090.7
70	991.35	1006.4	1020.4	1033.4	1045.5	1056.8	1067.4	1077.4	1086.7
75	988.49	1003.3	1017.3	1030.1	1042	1053.2	1063.6	1073.4	1082.6
80	985.5	1000.2	1014	1026.7	1038.5	1049.5	1059.7	1069.4	1078.4
85	982.39	996.96	1010.6	1023.1	1034.8	1045.6	1055.7	1065.2	1074
90	979.15	993.59	1007.1	1019.4	1031	1041.7	1051.6	1060.9	1069.5
95	975.8	990.1	1003.5	1015.7	1027	1037.6	1047.3	1056.4	1064.9
100	972.32	986.48	999.69	1011.7	1023	1033.3	1042.9	1051.9	1060.2
105	968.73	982.75	995.81	1007.7	1018.8	1029	1038.4	1047.2	1055.4
110	965.01	978.89	991.81	1003.6	1014.5	1024.5	1033.8	1042.4	1050.4
115	961.17	974.91	987.68	999.29	1010	1019.9	1029.1	1037.5	1045.3
120	957.21	970.81	983.43	994.9	1005.5	1015.2	1024.2	1032.5	1040.1
125	953.12	966.59	979.07	990.38	1000.8	1010.4	1019.2	1027.3	1034.7

When program runs the resulting curve fitting coefficients are:

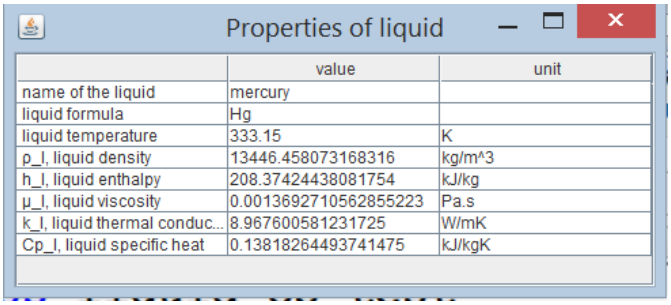


By using this curve fittings a program for the fluid properties, secondarycoolantEN.java and user interface program secondarycoolantTableEN are prepared. Program codes are given in Appendix 6.4

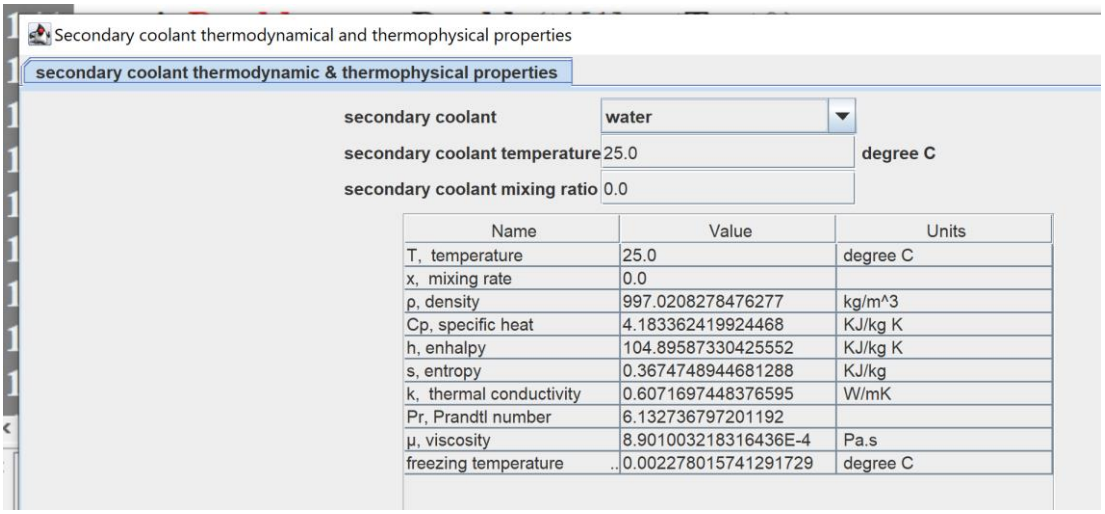


Another liquid properties pogram is developed by using cubic splines. This set is only one dimensional curve fitting (temperature dependent only). Program lists liquid_CS_Data, liquid_CS_EN, liquid_CS_Model_EN and a GUI program liquid_CS_Table_EN is given in Appendix 6.5. A sample program and a GUI output is shown below:

```
public class liquid_CS_test
{
public static void main(String arg[])
{ liquid_CS_EN l=new liquid_CS_EN("mercury");
  String s1[]={"","value","unit"};
  double T=60.0+273.15;//degree K
  Text.print(l.toString(T),s1,"Properties of liquid");
}}
```



	value	unit
name of the liquid	mercury	
liquid formula	Hg	
liquid temperature	333.15	K
ρ_l, liquid density	13446.458073168316	kg/m^3
h_l, liquid enthalpy	208.37424438081754	kJ/kg
μ_l, liquid viscosity	0.0013692710562855223	Pa.s
k_l, liquid thermal conduc...	8.967600581231725	W/mK
Cp_l, liquid specific heat	0.13818264493741475	kJ/kgK



Name	Value	Units
T, temperature	25.0	degree C
x, mixing rate	0.0	
ρ, density	997.0208278476277	kg/m^3
Cp, specific heat	4.183362419924468	KJ/kg K
h, enhalpy	104.89587330425552	KJ/kg K
s, entropy	0.3674748944681288	KJ/kg
k, thermal conductivity	0.6071697448376595	W/mK
Pr, Prandtl number	6.132736797201192	
μ, viscosity	8.901003218316436E-4	Pa.s
freezing temperature	0.002278015741291729	degree C

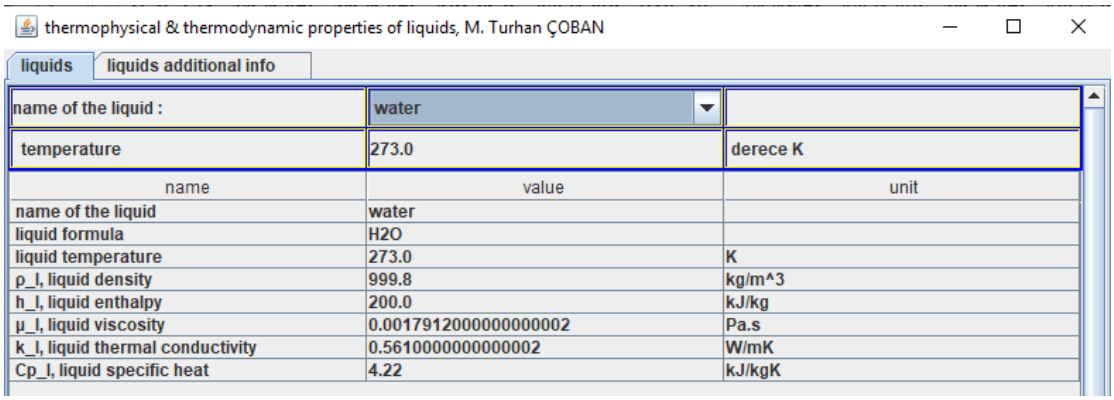
Program List of: secondatyCoolantEN

secondarycoolantEN	Basic Equation of state(based on curve fitting of data)
secondarycoolantTableEN	Graphical user interface for secondarycoolantEN
secondarycoolantModelEN	GUI support program(creates table format of output data)

Fluids defined in secondaryCoolantEN

Water	
NaCl brine	Salt+water
CaCl brine	CaCl salt+water
Ethylene Gylicol-water	Antifreeze
Propylene Gylicol-water	Antifreeze
polydimethylsilohexane	

There is another class given thermodynamic & thermophysical properties of some liquids. It is called liquid_CS_EN. It is basically gather information from tables for liquids and apply cubic spline curve fitting to convert values into functions to be used in computer environment.



Program List of: liquid_CS_EN

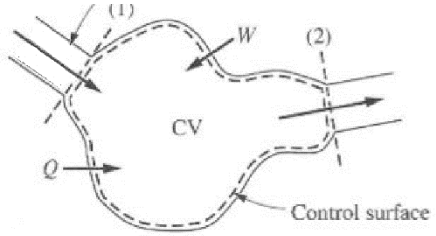
liquid_CS_EN	Basic Equation of state(based on curve fitting of data)
liquid_CS_ Table_EN	Graphical user interface for liquid_CS_EN
liquid_CS_Model_EN	GUI support program(creates table format of output data)

liquids defined in liquid_CS_EN

water H2O	glycerine
engineoil	xceltherm_MK1
methanol	Polydimethylsiloxane
ethanol	therminolD12
tetrachloromethane	therminolVLT
trichloroetylene	

3 POWER CYCLES

3.0 FLOW SYSTEM PROCESSES FLOW SYSTEM PROCESSES



Conservation of mass for Flow System:

$$\left.\frac{dm}{dt}\right|_{in} - \left.\frac{dm}{dt}\right|_{out} = \left.\frac{dm}{dt}\right|_{control\ volume}$$

$$\dot{m}\big|_{in} - \dot{m}\big|_{out} = \dot{m}\big|_{system}$$

Or considering more than one inlet and outlet

$$\sum \dot{m}\big|_{in} - \sum \dot{m}\big|_{out} = \dot{m}\big|_{system}$$

For steady state systems

$$\dot{m}\big|_{in} - \dot{m}\big|_{out} = 0$$

$$\sum \dot{m}\big|_{in} - \sum \dot{m}\big|_{out} = 0$$

Conservation of Energy for Flow System:

$$\left.\frac{dE}{dt}\right|_{in} - \left.\frac{dE}{dt}\right|_{out} = \left.\frac{dE}{dt}\right|_{control\ volume}$$

$$\frac{dE}{dt} = \frac{d(KE + PE + U + PV + W + Q)}{dt}$$

Kinetic Energy=KE=$\dot{m}\frac{V^2}{2}$ where V is velocity in m/s

Potential energy=PE=$\dot{m}gz$ where z is height from a specified datum state

Heat transfer : $\frac{dQ}{dt} = \dot{Q}$ dQ=Tds

Internal Energy: U=$\dot{m}u$ atomic level kinetic energy

$PV = \dot{m}Pv$ pressure energy

Enthalpy H=$\dot{m}h$=$\dot{m}(u + Pv)$

Work Transfer: $\frac{dW}{dt} = \dot{W}$ $dw = -Pdv + \sum \mu dn_i$ mechanical + chemical work where v is specific volume

$$\left[\frac{(\dot{m}_{out}V_{out}^2 - \dot{m}_{out}V_{in}^2)}{2} + g(\dot{m}_{out}z_{out} - \dot{m}_{in}z_{in}) + (\dot{m}_{out}h_{out} - \dot{m}_{in}h_{in}) - \dot{Q} + \dot{W}\right] = \left.\frac{dE}{dt}\right|_{control\ volume}$$

Steady state (also considers steady state mass balance)

$$\dot{m}\left[\frac{(V_{out}^2 - V_{in}^2)}{2} + g(z_{out} - z_{in}) + (h_{out} - h_{in})\right] = \dot{Q} - \dot{W}$$

Note that it is important to draw system boundaries. If a suden change in mass flow rates or heat transfer or mass transfer happens border materials will be part of energy balance, so it should be considered as a part of system. For steady state systems, it can be assumed that

state of border materials is not changing so we can limit system with working fluid only. It should be also note that there could be other form of energies created in control volüme or passed from the border.

Isentropic (constant entropy process)

Case 1: Isentropic Expansion, Turbine, expander

$dU = TdS - PdV$

$$dQ = TdS$$

Isentropic process dQ=0 dS=0 (reversible process)

Adiabatic process dQ=0 dS>0 (irreversible process)

Ignoring kinetic and potential energies

$$\dot{m}[(h_2 - h_1)] = \dot{Q} - \dot{W}$$

If $\dot{Q} = 0 \;\; W = \dot{m}[(h_1 - h_2)]$

Working fluid: steam P1=3000 kPa T1=500 C

P2=80 kPa S2=S1 (isentropic)

Working fluid calculation source: steamTableIF97

$$\eta_{isentropic} = \frac{h_1 - h_2}{h_1 - h_{2i}} = 0.9$$

$$h_2 = h_1 - \eta_{isentropic}(h_1 - h_{2i})$$



```
/*
steamIAPWS_IF97 property vector
  a[0]="P, pressure        ";       kPa
  a[1]="T, temperature     ";   degree K
  a[2]="v, specific volume ";   m^3/kg
  a[3]="h, enthalpy        ";       kJ/kg
  a[4]="u, internal energy ";   kJ/kg
  a[5]="s, entropy         ";       kJ/kgK
  a[6]="x, quality         ";       kg/kg total
  a[7]="Cp, specific heat at constant pressure ";  kJ/kgK
  a[8]="Cv, specific heat at constant volume ";     kJ/kgK
  a[9]="isobaric thermal expansion coefficient ";  1/K
  a[10]="isothermal compressibility" ; 1/kPa
  a[11]="Dynamic viscosity" ;  Pas
  a[12]="k Thermal conductivity" ; W/mK
  a[13]="Surface tension" ;  N/m
  a[14]="Prandtl number" ;
```

```
   a[15]="density" ;  kg/m^3
   a[16]="speed of sound" ; m/s
*/
public class isentropic1
{ steamIAPWS_IF97 steam;
  double TT1[];  //isentropic process temperature array
  double ss1[];    //isentropic processentropy array
  double TT2[];  //adiabatic process temperature array
  double ss2[];   //adiabatic process entropy array
  double s1,P1,T1,x1,h1;
  double s2i,P2i,T2i,x2i,h2i;
  double s2,P2,T2,x2,h2;
  double Wisentropic;
  double Wadiabatic;
  double eta_isentropic;
  double T0=273.15;  //degree K
public isentropic1()
{steam=new steamIAPWS_IF97();}
//T1i,T2i degree K, P1i,P2i kPa
public void calculate1(double T1in,double P1in,double T2in, double P2in)
{
    T1=T1in;
    P1=P1in;
    T2=T2in;
    P2=P2in;
    double a1[]=steam.property("tp",T1,P1);
          s1=a1[5];
    h1=a1[3];
    x1=a1[6];
    //double P2=80.0; //kPa
    s2i=s1;
    P2i=P2;
    double a2[]=steam.property("ps",P2i,s2i);
    h2i=a2[3];
    x2i=a2[6];
    T2i=a2[1];
    double a3[]=steam.property("tp",T2,P2);
    h2=a3[3];
    s2=a3[5];
    x2=a3[6];
    Wisentropic = h1-h2i;
    Wadiabatic = h1-h2;
    eta_isentropic=Wadiabatic/Wisentropic;
}
public void plot1()
{
    Plot p=new Plot(steam.ass,steam.ats);
     p.setPlabel("Adiabatic process, Working fluid: Steam");
```

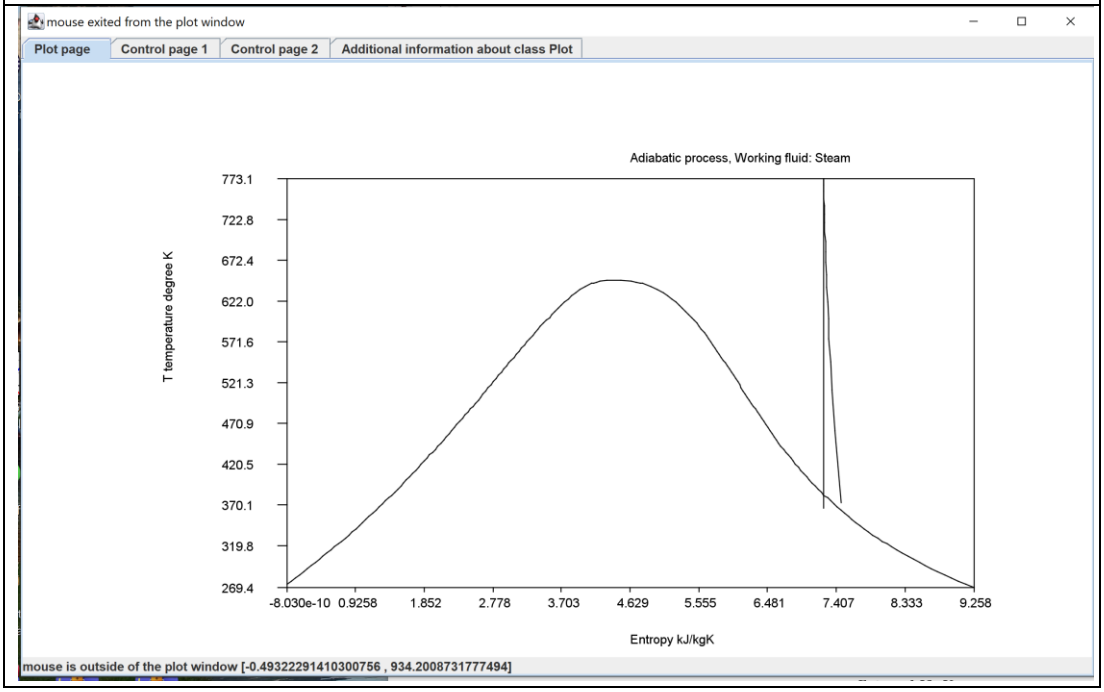
```
    p.setXlabel("Entropy kJ/kgK");
    p.setYlabel("T temperature degree K");
    double ss1[]={s1,s2i};
    double TT1[]={T1,T2i};
    p.addData(ss1,TT1);
    double ss2[]=new double[20];
    double TT2[]=new double[20];
    ss2[0]=s1;
    TT2[0]=T1;
    double dP=(P1-P2)/19;
    double PP=P1;
    for(int i=1;i<20;i++)
    {PP=PP-dP;
      double ai[]=steam.property("ps",PP,s1);
      double hi=ai[3];
      double ha=h1-eta_isentropic*(h1-hi);
      double aa[]=steam.property("ph",PP,ha);
      ss2[i]=aa[5];
      TT2[i]=aa[1];
    }
     p.addData(ss2,TT2);
    p.plot();
}
public void plot2()
{
    graph2D p=new graph2D();
    p.addData(steam.ass,steam.ats);
    p.setPlabel("Adiabatic process, Working fluid: Steam");
    p.setXlabel("Entropy kJ/kgK");
    p.setYlabel("T temperature degree K");
    double ss1[]={s1,s2i};
    double TT1[]={T1,T2i};
    p.addData(ss1,TT1);
    double ss2[]=new double[20];
    double TT2[]=new double[20];
    ss2[0]=s1;
    TT2[0]=T1;
    double dP=(P1-P2)/19;
    double PP=P1;
    for(int i=1;i<20;i++)
    {PP=PP-dP;
      double ai[]=steam.property("ps",PP,s1);
      double hi=ai[3];
      double ha=h1-eta_isentropic*(h1-hi);
      double aa[]=steam.property("ph",PP,ha);
      ss2[i]=aa[5];
      TT2[i]=aa[1];
    }
```

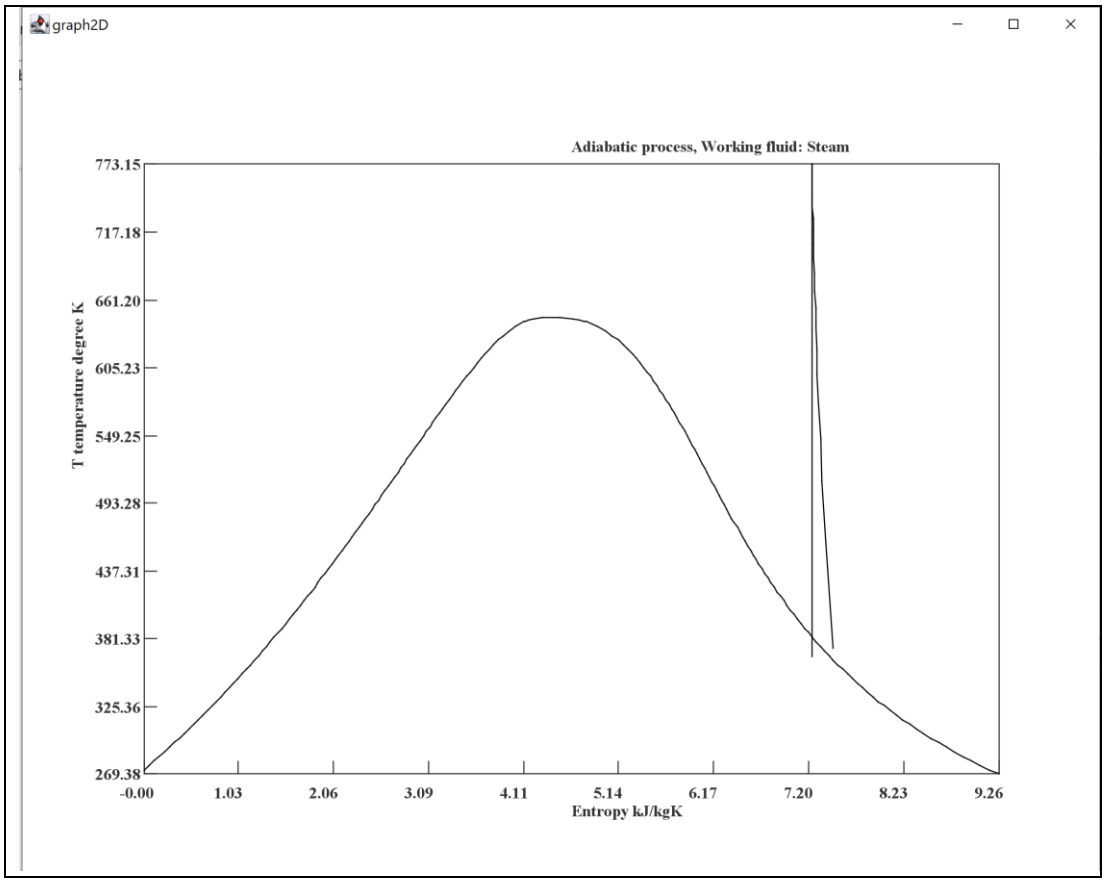
```
      p.addData(ss2,TT2);
     p.plot();
}

public void calculate2(double T1in,double P1in, double P2in,double eta_isentropicin)
{   eta_isentropic=eta_isentropicin;
     T1=T1in;
     P1=P1in;
     P2=P2in;
     double a1[]=steam.property("tp",T1,P1);
            s1=a1[5];
     h1=a1[3];
     x1=a1[6];
     s2i=s1;
     P2i=P2;
     double a2[]=steam.property("ps",P2i,s2i);
     h2i=a2[3];
     x2i=a2[6];
     T2i=a2[1];
      Wisentropic = h1-h2i;
      Wadiabatic=Wisentropic*eta_isentropic;
      h2=h1-Wadiabatic;
     double a3[]=steam.property("ph",P2,h2);
     T2=a3[1];
     s2=a3[5];
     x2=a3[6];
}
public String toString()
{ String s="Adiabatic Process working fluid: steam\n";
  s+="P1= "+P1+" kPa "+"T1= "+T1+" degree K \n";
  s+="h1= "+h1+" kJ/kg "+"s1= "+s1+" kJ/kgK x1="+x1+" kg vapor/kg \n";
  s+="P2i= "+P2i+" kPa "+"T2i= "+T2i+" degree K \n";
  s+="h2i= "+h2i+" kJ/kg "+"s2i= "+s2i+" kJ/kgK x2i="+x2i+" kg vapor/kg \n";
  s+="P2= "+P2+" kPa "+"T2= "+T2+" degree K \n";
  s+="h2= "+h2+" kJ/kg "+"s2= "+s2+" kJ/kgK x2="+x2+" kg vapor/kg \n";
  s+="Wisentropic = "+Wisentropic+" kJ/kg\n";
  s+="Wadiabatic = "+Wadiabatic+" kJ/kg\n";
  s+="eta_isentropic = "+eta_isentropic;
  return s;
}
public static void main(String arg[])
{
     double T1=500.0+273.15; //degree Celcius
     double P1=3000.0; //kPa
     double P2=80.0; //kPa
     double eta_isentropic=0.9;
      isentropic1 is1=new isentropic1 ();
      is1.calculate2(T1,P1,P2,eta_isentropic);
```

```
    System.out.println(is1);
    is1.plot1();
    is1.plot2();
}
}
```

```
---------- Capture Output ----------
> "C:\Users\Turhan Çoban\.p2\pool\plugins\org.eclipse.justj.openjdk.hotspot.jre.full.win32.x86_64_19.0.2.v20230129-1123\jre\bin\javaw.exe" isentropic1
Adiabatic Process working fluid: steam
P1= 3000.0 kPa T1= 773.15 degree K
h1= 3457.0405348564645 kJ/kg s1= 7.235592163176318 kJ/kgK x1=2.0 kg vapor/kg
P2i= 80.0 kPa T2i= 366.6353541287056 degree K
h2i= 2592.4732190640925 kJ/kg s2i= 7.235592163176318 kJ/kgK
x2i=0.9680214499294163 kg vapor/kg
P2= 80.0 kPa T2= 373.3714573759581 degree K
h2= 2678.92995064333 kJ/kg s2= 7.471044594025981 kJ/kgK x2=2.0 kg vapor/kg
Wisentropic = 864.567315792372 kJ/kg
Wadiabatic = 778.1105842131349 kJ/kg
eta_isentropic = 0.9
```





Working fluid: butane P1=3000 kPa T1=150 C

P2=80 kPa T2=40 C (adiabatic)

Working fluid calculation source: **refrigerant_coolProp + coolProp library**

$$\eta_{isentropic} = \frac{h_1 - h_2}{h_1 - h_{2i}}$$

$$h_2 = h_1 - \eta_{isentropic}(h_1 - h_{2i})$$



```
 *
refrigerant_coolProp property vector
  a[0]="P, pressure        ";      kPa
  a[1]="T, temperature     ";  degree K
  a[2]="v, specific volume ";  m^3/kg
  a[3]="h, enthalpy        ";      kJ/kg
  a[4]="u, internal energy ";  kJ/kg
  a[5]="s, entropy         ";      kJ/kgK
  a[6]="x, quality         ";       kg/kg total
```

```
   a[7]="Cp, specific heat at constant pressure ";  kJ/kgK
   a[8]="Cv, specific heat at constant volume ";     kJ/kgK
   a[9]="isobaric thermal expansion coefficient ";  1/K
   a[10]="isothermal compressibility" ; 1/kPa
   a[11]="Dynamic viscosity" ;  Pas
   a[12]="k Thermal conductivity" ; W/mK
   a[13]="Surface tension" ;  N/m
   a[14]="Prandtl number" ;
   a[15]="density" ;  kg/m^3
   a[16]="speed of sound" ; m/s
*/
public class isentropic2
{ refrigerant_coolProp refr;
  double TT1[];  //isentropic process temperature array
  double ss1[];    //isentropic processentropy array
  double TT2[];  //adiabatic process temperature array
  double ss2[];   //adiabatic process entropy array
  double s1,P1,T1,x1,h1;
  double s2i,P2i,T2i,x2i,h2i;
  double s2,P2,T2,x2,h2;
  double Wisentropic;
  double Wadiabatic;
  double eta_isentropic;
  double T0=273.15;  //degree K
  String refname;
public isentropic2(String refnamei)
{   refname=refnamei;
          refr=new refrigerant_coolProp(refname);}
//T1i,T2i degree K, P1i,P2i kPa
public void calculate1(double T1in,double P1in,double T2in, double P2in)
{
    T1=T1in;
    P1=P1in;
    T2=T2in;
    P2=P2in;
    double a1[]=refr.property("tp",T1,P1);
           s1=a1[5];
    h1=a1[3];
    x1=a1[6];
    //double P2=80.0; //kPa
    s2i=s1;
    P2i=P2;
    double a2[]=refr.property("ps",P2i,s2i);
    h2i=a2[3];
    x2i=a2[6];
    T2i=a2[1];
    double a3[]=refr.property("tp",T2,P2);
    h2=a3[3];
```

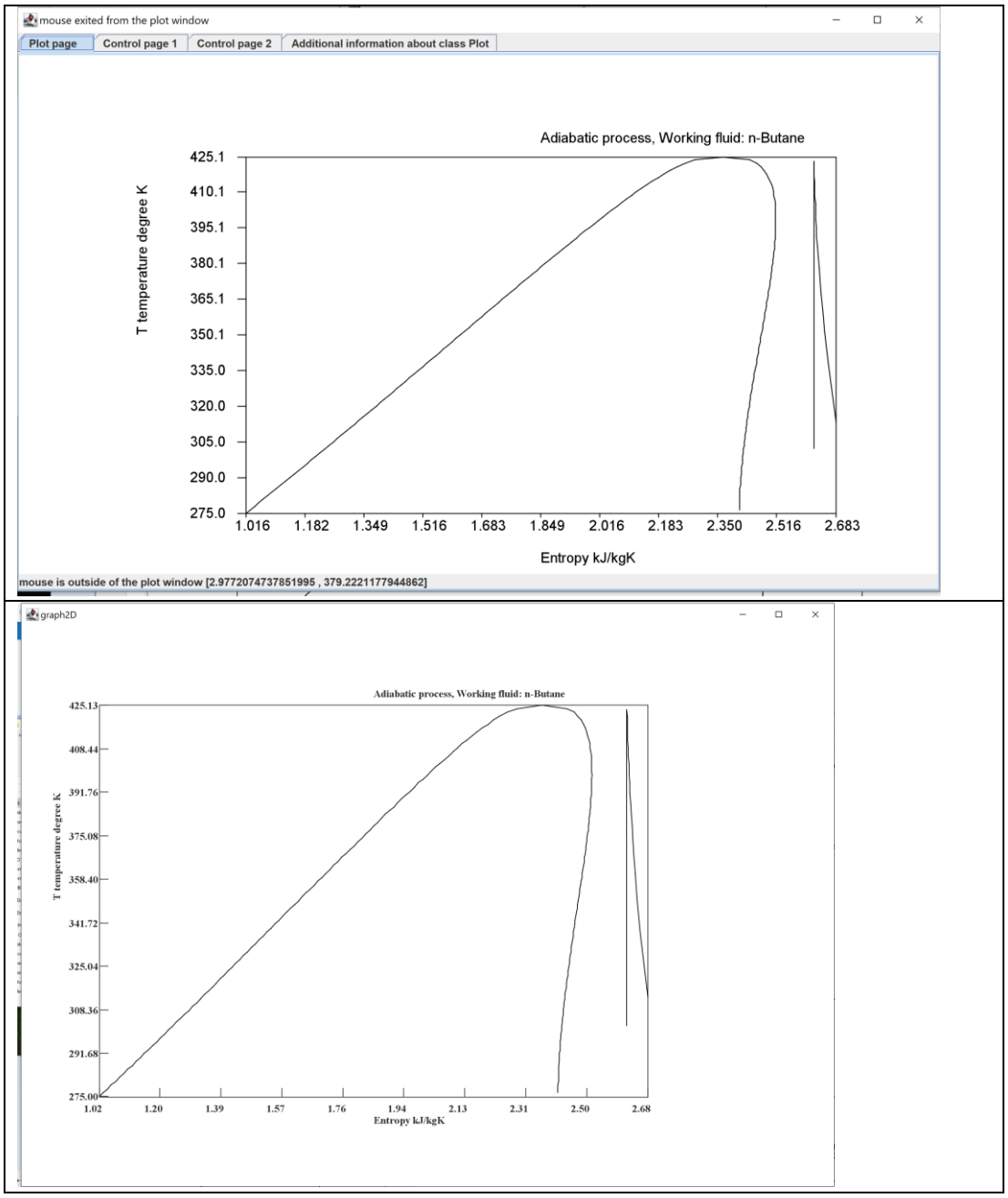
```
    s2=a3[5];
    x2=a3[6];
    Wisentropic = h1-h2i;
    Wadiabatic = h1-h2;
    eta_isentropic=Wadiabatic/Wisentropic;
}
public void plot1()
{
    Plot p=new Plot(refr.TS_data(275.0));
     p.setPlabel("Adiabatic process, Working fluid: "+refname);
    p.setXlabel("Entropy kJ/kgK");
    p.setYlabel("T temperature degree K");
    double ss1[]={s1,s2i};
    double TT1[]={T1,T2i};
    p.addData(ss1,TT1);
    double ss2[]=new double[20];
    double TT2[]=new double[20];
    ss2[0]=s1;
    TT2[0]=T1;
    double dP=(P1-P2)/19;
    double PP=P1;
    for(int i=1;i<20;i++)
    {PP=PP-dP;
      double ai[]=refr.property("ps",PP,s1);
      double hi=ai[3];
      double ha=h1-eta_isentropic*(h1-hi);
      double aa[]=refr.property("ph",PP,ha);
      ss2[i]=aa[5];
      TT2[i]=aa[1];
    }
     p.addData(ss2,TT2);
    p.plot();
}
public void plot2()
{
    graph2D p=new graph2D();
    p.addData(refr.TS_data(275.0));
    p.setPlabel("Adiabatic process, Working fluid: "+refname);
    p.setXlabel("Entropy kJ/kgK");
    p.setYlabel("T temperature degree K");
    double ss1[]={s1,s2i};
    double TT1[]={T1,T2i};
    p.addData(ss1,TT1);
    double ss2[]=new double[20];
    double TT2[]=new double[20];
    ss2[0]=s1;
    TT2[0]=T1;
    double dP=(P1-P2)/19;
```

```
    double PP=P1;
    for(int i=1;i<20;i++)
    {PP=PP-dP;
      double ai[]=refr.property("ps",PP,s1);
      double hi=ai[3];
      double ha=h1-eta_isentropic*(h1-hi);
      double aa[]=refr.property("ph",PP,ha);
      ss2[i]=aa[5];
      TT2[i]=aa[1];
    }
     p.addData(ss2,TT2);
    p.plot();
}

public void calculate2(double T1in,double P1in, double P2in,double eta_isentropicin)
{   eta_isentropic=eta_isentropicin;
    T1=T1in;
    P1=P1in;
    P2=P2in;
    double a1[]=refr.property("tp",T1,P1);
           s1=a1[5];
    h1=a1[3];
    x1=a1[6];
    s2i=s1;
    P2i=P2;
    double a2[]=refr.property("ps",P2i,s2i);
    h2i=a2[3];
    x2i=a2[6];
    T2i=a2[1];
     Wisentropic = h1-h2i;
     Wadiabatic=Wisentropic*eta_isentropic;
     h2=h1-Wadiabatic;
    double a3[]=refr.property("ph",P2,h2);
    T2=a3[1];
    s2=a3[5];
    x2=a3[6];
}
public String toString()
{ String s="Adiabatic Process working fluid:"+refname+"\n";
 s+="P1= "+P1+" kPa "+"T1= "+T1+" degree K \n";
 s+="h1= "+h1+" kJ/kg "+"s1= "+s1+" kJ/kgK x1="+x1+" kg vapor/kg \n";
 s+="P2i= "+P2i+" kPa "+"T2i= "+T2i+" degree K \n";
 s+="h2i= "+h2i+" kJ/kg "+"s2i= "+s2i+" kJ/kgK x2i="+x2i+" kg vapor/kg \n";
 s+="P2= "+P2+" kPa "+"T2= "+T2+" degree K \n";
 s+="h2= "+h2+" kJ/kg "+"s2= "+s2+" kJ/kgK x2="+x2+" kg vapor/kg \n";
 s+="Wisentropic = "+Wisentropic+" kJ/kg\n";
 s+="Wadiabatic = "+Wadiabatic+" kJ/kg\n";
 s+="eta_isentropic = "+eta_isentropic;
```

```
  return s;
}
public static void main(String arg[])
{   double T1=150.0+273.15; //degree Celcius
    double P1=3000.0; //kPa
    double T2=40+273.15;
    double P2=80.0; //kPa
    double eta_isentropic=0.9;
     isentropic2 is2=new isentropic2 ("n-Butane");
     is2.calculate1(T1,P1,T2,P2);
     System.out.println(is2);
     is2.plot1();
     is2.plot2();
}
}
```

```
 ---------- Capture Output ----------
> "C:\Users\Turhan
Çoban\.p2\pool\plugins\org.eclipse.justj.openjdk.hotspot.jre.full.win32.x86_64_19.0.2.v2
0230129-1123\jre\bin\javaw.exe" isentropic2
Adiabatic Process working fluid:n-Butane
P1= 3000.0 kPa T1= 423.15 degree K
h1= 795.8271184920371 kJ/kg s1= 2.620800459414021 kJ/kgK
x1=1.3083206116505914 kg vapor/kg
P2i= 80.0 kPa T2i= 302.2707366130761 degree K
h2i= 635.5933897893714 kJ/kg s2i= 2.620800459414021 kJ/kgK
x2i=1.1524920556054956 kg vapor/kg
P2= 80.0 kPa T2= 313.15 degree K
h2= 654.7589493585888 kJ/kg s2= 2.6830867546234596 kJ/kgK
x2=1.201504638496226 kg vapor/kg
Wisentropic = 160.23372870266576 kJ/kg
Wadiabatic = 141.06816913344835 kJ/kg
eta_isentropic = 0.8803899795355723
```



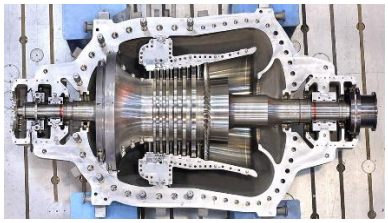
Working fluid: air P1=3000 kPa T1=300 C

P2=100 kPa S2i=S1 (isentropic)

Working fluid calculation source: **air_PG**

$$\eta_{isentropic} = \frac{h_1 - h_2}{h_1 - h_{2i}} = 0.9$$

$$h_2 = h_1 - \eta_{isentropic}(h_1 - h_{2i})$$



```
 *
air_PG property vector
  a[0]="P, pressure        ";       kPa
  a[1]="T, temperature     ";   degree K
  a[2]="v, specific volume ";   m^3/kg
  a[3]="h, enthalpy        ";       kJ/kg
  a[4]="u, internal energy ";   kJ/kg
  a[5]="s, entropy         ";       kJ/kgK
  a[6]="x, quality         ";         kg/kg total
  a[7]="Cp, specific heat at constant pressure ";  kJ/kgK
  a[8]="Cv, specific heat at constant volume ";     kJ/kgK
  a[9]="isobaric thermal expansion coefficient ";  1/K
  a[10]="isothermal compressibility" ; 1/kPa
  a[11]="Dynamic viscosity" ;  Pas
  a[12]="k Thermal conductivity" ; W/mK
  a[13]="Surface tension" ;  N/m
  a[14]="Prandtl number" ;
  a[15]="density" ;  kg/m^3
  a[16]="speed of sound" ; m/s
*/
public class isentropic3
{ air_PG refr;
  double TT1[];  //isentropic process temperature array
  double ss1[];    //isentropic processentropy array
  double TT2[];  //adiabatic process temperature array
  double ss2[];   //adiabatic process entropy array
  double s1,P1,T1,x1,h1;
  double s2i,P2i,T2i,x2i,h2i;
  double s2,P2,T2,x2,h2;
  double Wisentropic;
  double Wadiabatic;
  double eta_isentropic;
  double T0=273.15;  //degree K
  String refname;
public isentropic3()
{   refname="air";
          refr=new air_PG();}
//T1i,T2i degree K, P1i,P2i kPa
public void calculate1(double T1in,double P1in,double T2in, double P2in)
{
    T1=T1in;
    P1=P1in;
    T2=T2in;
    P2=P2in;
    double a1[]=refr.property("tp",T1,P1);
           s1=a1[5];
    h1=a1[3];
    x1=a1[6];
```

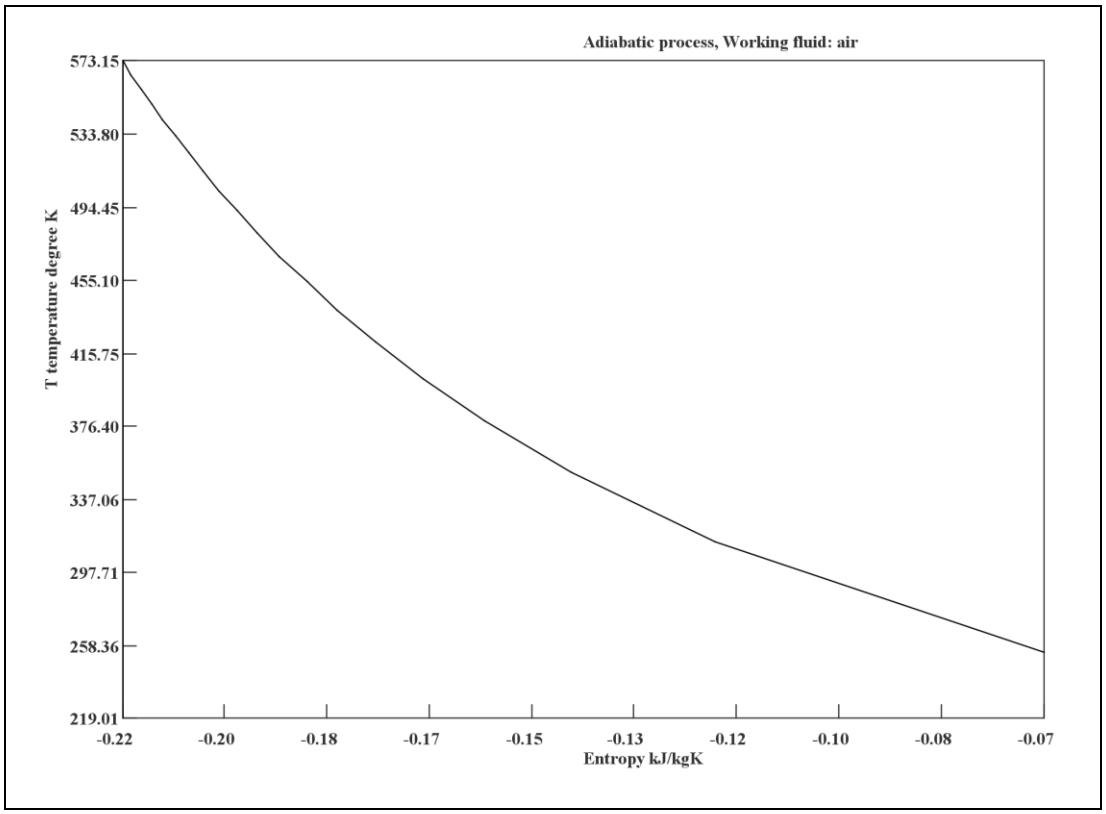
```
    //double P2=80.0; //kPa
    s2i=s1;
    P2i=P2;
    double a2[]=refr.property("ps",P2i,s2i);
    h2i=a2[3];
    x2i=a2[6];
    T2i=a2[1];
    double a3[]=refr.property("tp",T2,P2);
    h2=a3[3];
    s2=a3[5];
    System.out.println("s2="+s2);
    x2=a3[6];
    Wisentropic = h1-h2i;
    Wadiabatic = h1-h2;
    eta_isentropic=Wadiabatic/Wisentropic;
}
public void plot1()
{       double ss1[]={s1,s2i};
    double TT1[]={T1,T2i};
    Plot p=new Plot(ss1,TT1);
     p.setPlabel("Adiabatic process, Working fluid: "+refname);
    p.setXlabel("Entropy kJ/kgK");
    p.setYlabel("T temperature degree K");
    double ss2[]=new double[20];
    double TT2[]=new double[20];
    ss2[0]=s1;
    TT2[0]=T1;
    double dP=(P1-P2)/19;
    double PP=P1;
    for(int i=1;i<20;i++)
    {PP=PP-dP;
      double ai[]=refr.property("ps",PP,s1);
      double hi=ai[3];
      double ha=h1-eta_isentropic*(h1-hi);
      double aa[]=refr.property("ph",PP,ha);
      ss2[i]=aa[5];
      TT2[i]=aa[1];
    }
     p.addData(ss2,TT2);
    p.plot();
}
public void plot2()
{
    graph2D p=new graph2D();
    //p.addData(refr.TS_data(275.0));
    p.setPlabel("Adiabatic process, Working fluid: "+refname);
    p.setXlabel("Entropy kJ/kgK");
    p.setYlabel("T temperature degree K");
```

```
    double ss1[]={s1,s2i};
    double TT1[]={T1,T2i};
    p.addData(ss1,TT1);
    double ss2[]=new double[20];
    double TT2[]=new double[20];
    ss2[0]=s1;
    TT2[0]=T1;
    double dP=(P1-P2)/19;
    double PP=P1;
    for(int i=1;i<20;i++)
    {PP=PP-dP;
      double ai[]=refr.property("ps",PP,s1);
      double hi=ai[3];
      double ha=h1-eta_isentropic*(h1-hi);
      double aa[]=refr.property("ph",PP,ha);
      ss2[i]=aa[5];
      TT2[i]=aa[1];
    }
     p.addData(ss2,TT2);
    p.plot();
}

public void calculate2(double T1in,double P1in, double P2in,double eta_isentropicin)
{   eta_isentropic=eta_isentropicin;
    T1=T1in;
    P1=P1in;
    P2=P2in;
    double a1[]=refr.property("tp",T1,P1);
          s1=a1[5];
    h1=a1[3];
    x1=a1[6];
    s2i=s1;
    P2i=P2;
    double a2[]=refr.property("ps",P2i,s2i);
    h2i=a2[3];
    x2i=a2[6];
    T2i=a2[1];
     Wisentropic = h1-h2i;
     Wadiabatic=Wisentropic*eta_isentropic;
     h2=h1-Wadiabatic;
    double a3[]=refr.property("ph",P2,h2);
    T2=a3[1];
    s2=a3[5];
    System.out.println("s2="+s2);
    double x2=a3[6];
}
public String toString()
{ String s="Adiabatic Process working fluid:"+refname+"\n";
```

```
  s+="P1= "+P1+" kPa "+"T1= "+T1+" degree K \n";
  s+="h1= "+h1+" kJ/kg "+"s1= "+s1+" kJ/kgK x1="+x1+" kg vapor/kg \n";
  s+="P2i= "+P2i+" kPa "+"T2i= "+T2i+" degree K \n";
  s+="h2i= "+h2i+" kJ/kg "+"s2i= "+s2i+" kJ/kgK x2i="+x2i+" kg vapor/kg \n";
  s+="P2= "+P2+" kPa "+"T2= "+T2+" degree K \n";
  s+="h2= "+h2+" kJ/kg "+"s2= "+s2+" kJ/kgK x2="+x2+" kg vapor/kg \n";
  s+="Wisentropic = "+Wisentropic+" kJ/kg\n";
  s+="Wadiabatic = "+Wadiabatic+" kJ/kg\n";
  s+="eta_isentropic = "+eta_isentropic;
  return s;
}
public static void main(String arg[])
{   double T1=300.0+273.15; //degree Celcius
    double P1=3000.0; //kPa
    double eta=0.9;
    double P2=100.0; //kPa
    double eta_isentropic=0.9;
     isentropic3 is2=new isentropic3 ();
     is2.calculate2(T1,P1,P2,eta);
     System.out.println(is2);
     is2.plot1();
     is2.plot2();
}
}
```

```
---------- Capture Output ----------
> "C:\Users\Turhan
Çoban\.p2\pool\plugins\org.eclipse.justj.openjdk.hotspot.jre.full.win32.x86_64_19.0.2.v2
0230129-1123\jre\bin\javaw.exe" isentropic3
s2=-0.06595876484322367
Adiabatic Process working fluid:air
P1= 3000.0 kPa T1= 573.15 degree K
h1= 305.98240572047223 kJ/kg s1= -0.21837623017797814 kJ/kgK x1=2.0 kg vapor/kg
P2i= 100.0 kPa T2i= 219.0083396947801 degree K
h2i= -54.446055832303756 kJ/kg s2i= -0.21837623017797814 kJ/kgK x2i=2.0 kg
vapor/kg
P2= 100.0 kPa T2= 254.84857911069383 degree K
h2= -18.403209677026155 kJ/kg s2= -0.06595876484322367 kJ/kgK x2=0.0 kg vapor/kg
Wisentropic = 360.42846155277596 kJ/kg
Wadiabatic = 324.3856153974984 kJ/kg
eta_isentropic = 0.9
```



In the calculation above which utilises Gas_PG class, constant entropy is calculates by using root finding process.

$$s(T_1, P_1) - s_{2i} = s_{ref} + \int_{T_{ref}}^{T_1} \frac{C_p(T)}{T} dT - Rln\left(\frac{P_1}{P_{ref}}\right) - s_{2i} = 0$$

In air_PG

It is also possible to use reduced pressure and reduced volume concepts.

$$s(T, P) = s_{ref} + \int_{T_{ref}}^{T} \frac{C_p(T)}{T} dT - Rln\left(\frac{P}{P_{ref}}\right)$$

$$s^0(T) = s_{ref} + \int_{T_{ref}}^{T} \frac{C_p(T)}{T} dT$$

$$s(T, P) = s^0(T) - Rln\left(\frac{P}{P_{ref}}\right)$$

For the isentropic process

$$s(T, P) - s\left(T^{ref}, P^{ref}\right) = \left[s^0(T) - Rln\left(\frac{P}{P_{ref}}\right)\right] - \left[s^0\left(T^{ref}\right) - Rln\left(\frac{P^{ref}}{P_{ref}}\right)\right] = 0$$

Reduced pressure in isentropic processes

$$P_r = \left(\frac{P}{P_{ref}}\right) = \exp\left(\frac{s^0(T) - s^0\left(T^{ref}\right)}{R}\right)$$

$$\frac{P_2}{P_1} = \frac{P_{r2}}{P_{r1}}$$

Another approximation to the process can be carried out by assuming Cp and Cv is constant

$$s_2(T_2) - s_1(T_1) = + \int_{T_1}^{T_2} \frac{C_p(T)}{T} dT - Rln\left(\frac{P_2}{P_1}\right)=0$$

$$s_2(T_2) - s_1(T_1) = C_p ln\left(\frac{T_2}{T_1}\right) - Rln\left(\frac{P_2}{P_1}\right) = 0$$

$$R = C_p - C_v$$

$$C_p ln\left(\frac{T_2}{T_1}\right) - (C_p - C_v) ln\left(\frac{P_2}{P_1}\right) = 0$$

Isentropic constant for constant C_p and C_v case $\gamma = \frac{C_p}{C_v}$

$$ln\left(\frac{T_2}{T_1}\right)^{C_p} = ln\left(\frac{P_2}{P_1}\right)^{(C_p - C_v)}$$

$$\left(\frac{T_2}{T_1}\right) = \left(\frac{P_2}{P_1}\right)^{\frac{(C_p - C_v)}{C_p}} = \left(\frac{P_2}{P_1}\right)^{\left(1 - \frac{C_v}{C_p}\right)} = \left(\frac{P_2}{P_1}\right)^{\left(1 - \frac{1}{\gamma}\right)} = \left(\frac{P_2}{P_1}\right)^{\left(\frac{\gamma - 1}{\gamma}\right)}$$

Gas Turbine:

Working fluid: mixing exhaust gases

gas	**Ni**
CO_2	**0.15**
H_2O	**0.1**
N_2	**0.70**
O_2	**0.05**

P1=3000 kPa T1=300 C

P2=100 kPa S2i=S1 (isentropic)

$$\eta_{isentropic} = \frac{h_1 - h_2}{h_1 - h_{2i}} = 0.9$$

Working fluid calculation source: **Gas_PG (Pure Ideal Gases) , Gmix_PG (Ideal Gas mixtures)**

```
/*  Gas Turbine
Gmix_PG property vector
  a[0]="P, pressure        ";      kPa
  a[1]="T, temperature     ";   degree K
  a[2]="v, specific volume ";   m^3/kg
  a[3]="h, enthalpy        ";      kJ/kg
  a[4]="u, internal energy ";   kJ/kg
  a[5]="s, entropy         ";      kJ/kgK
  a[6]="x, quality         ";      kg/kg total
  a[7]="Cp, specific heat at constant pressure ";  kJ/kgK
  a[8]="Cv, specific heat at constant volume ";     kJ/kgK
  a[9]="isobaric thermal expansion coefficient ";  1/K
  a[10]="isothermal compressibility" ; 1/kPa
  a[11]="Dynamic viscosity" ;  Pas
  a[12]="k Thermal conductivity" ; W/mK
  a[13]="Surface tension" ;  N/m
  a[14]="Prandtl number" ;
  a[15]="density" ;  kg/m^3
  a[16]="speed of sound" ; m/s
*/
```

```
public class isentropic4
{ Gmix_PG refr;
  double TT1[];  //isentropic process temperature array
  double ss1[];    //isentropic processentropy array
  double TT2[];  //adiabatic process temperature array
  double ss2[];   //adiabatic process entropy array
  double s1,P1,T1,x1,h1;
  double s2i,P2i,T2i,x2i,h2i;
  double s2,P2,T2,x2,h2;
  double Wisentropic;
  double Wadiabatic;
  double eta_isentropic;
  double T0=273.15;  //degree K
  String refname;
public isentropic4()
{   refname="Exhaust Gases";
   String name[]={"CO2","H2O","N2","O2"};
   double n[]={0.15,0.1,0.70,0.05};
         refr=new Gmix_PG("gmix IG",name,n);
         refr.mole=false;
}
//T1i,T2i degree K, P1i,P2i kPa
public void calculate1(double T1in,double P1in,double T2in, double P2in)
{
    T1=T1in;
    P1=P1in;
    T2=T2in;
    P2=P2in;
    double a1[]=refr.property("tp",T1,P1);
          s1=a1[5];
    h1=a1[3];
    x1=a1[6];
    //double P2=80.0; //kPa
    s2i=s1;
    P2i=P2;
    double a2[]=refr.property("ps",P2i,s2i);
    h2i=a2[3];
    x2i=a2[6];
    T2i=a2[1];
    double a3[]=refr.property("tp",T2,P2);
    h2=a3[3];
    s2=a3[5];
    x2=a3[6];
    Wisentropic = h1-h2i;
    Wadiabatic = h1-h2;
    eta_isentropic=Wadiabatic/Wisentropic;
}
public void plot1()
```

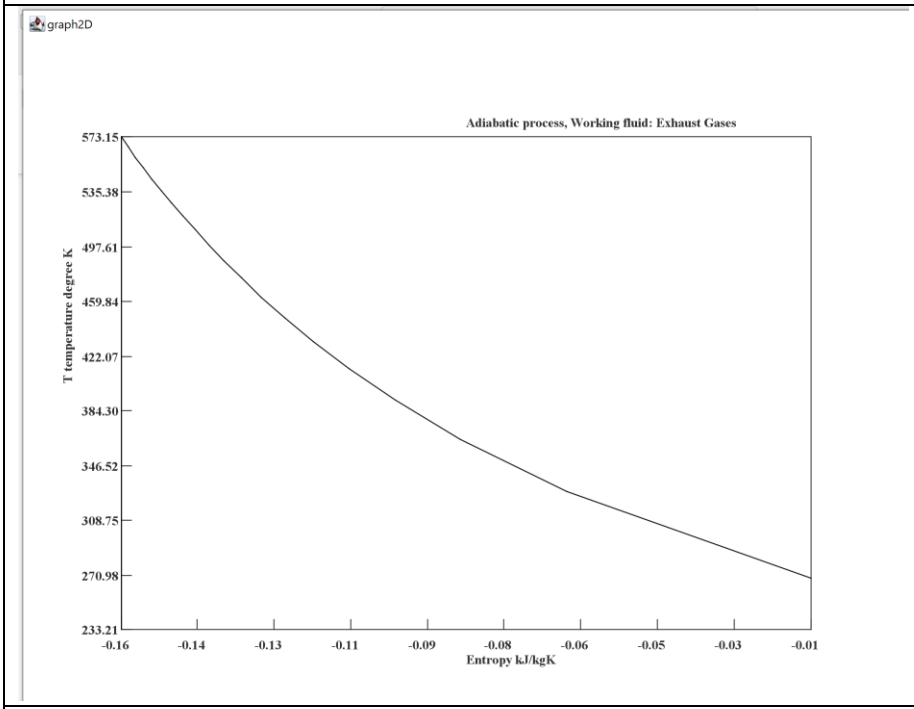
```
{       double ss1[]={s1,s2i};
    double TT1[]={T1,T2i};
    Plot p=new Plot(ss1,TT1);
     p.setPlabel("Adiabatic process, Working fluid: "+refname);
    p.setXlabel("Entropy kJ/kgK");
    p.setYlabel("T temperature degree K");
    double ss2[]=new double[20];
    double TT2[]=new double[20];
    ss2[0]=s1;
    TT2[0]=T1;
    double dP=(P1-P2)/19;
    double PP=P1;
    for(int i=1;i<20;i++)
    {PP=PP-dP;
      double ai[]=refr.property("ps",PP,s1);
      double hi=ai[3];
      double ha=h1-eta_isentropic*(h1-hi);
      double aa[]=refr.property("ph",PP,ha);
      ss2[i]=aa[5];
      TT2[i]=aa[1];
    }
     p.addData(ss2,TT2);
    p.plot();
}
public void plot2()
{
    graph2D p=new graph2D();
    //p.addData(refr.TS_data(275.0));
    p.setPlabel("Adiabatic process, Working fluid: "+refname);
    p.setXlabel("Entropy kJ/kgK");
    p.setYlabel("T temperature degree K");
    double ss1[]={s1,s2i};
    double TT1[]={T1,T2i};
    p.addData(ss1,TT1);
    double ss2[]=new double[20];
    double TT2[]=new double[20];
    ss2[0]=s1;
    TT2[0]=T1;
    double dP=(P1-P2)/19;
    double PP=P1;
    for(int i=1;i<20;i++)
    {PP=PP-dP;
      double ai[]=refr.property("ps",PP,s1);
      double hi=ai[3];
      double ha=h1-eta_isentropic*(h1-hi);
      double aa[]=refr.property("ph",PP,ha);
      ss2[i]=aa[5];
      TT2[i]=aa[1];
```

```
    }
     p.addData(ss2,TT2);
    p.plot();
}

public void calculate2(double T1in,double P1in, double P2in,double eta_isentropicin)
{   eta_isentropic=eta_isentropicin;
    T1=T1in;
    P1=P1in;
    P2=P2in;
    double a1[]=refr.property("tp",T1,P1);
           s1=a1[5];
    h1=a1[3];
    x1=a1[6];
    s2i=s1;
    P2i=P2;
    double a2[]=refr.property("ps",P2i,s2i);
    h2i=a2[3];
    x2i=a2[6];
    T2i=a2[1];
     Wisentropic = h1-h2i;
     Wadiabatic=Wisentropic*eta_isentropic;
     h2=h1-Wadiabatic;
    double a3[]=refr.property("ph",P2,h2);
    double T2=a3[1];
    double s2=a3[5];
    double x2=a3[6];
}
public String toString()
{ String s="Adiabatic Process working fluid:"+refname+"\n";
  s+="P1= "+P1+" kPa "+"T1= "+T1+" degree K \n";
  s+="h1= "+h1+" kJ/kg "+"s1= "+s1+" kJ/kgK x1="+x1+" kg vapor/kg \n";
  s+="P2i= "+P2i+" kPa "+"T2i= "+T2i+" degree K \n";
  s+="h2i= "+h2i+" kJ/kg "+"s2i= "+s2i+" kJ/kgK x2i="+x2i+" kg vapor/kg \n";
  s+="P2= "+P2+" kPa "+"T2= "+T2+" degree K \n";
  s+="h2= "+h2+" kJ/kg "+"s2= "+s2+" kJ/kgK x2="+x2+" kg vapor/kg \n";
  s+="Wisentropic = "+Wisentropic+" kJ/kg\n";
  s+="Wadiabatic = "+Wadiabatic+" kJ/kg\n";
  s+="eta_isentropic = "+eta_isentropic;
  return s;
}
public static void main(String arg[])
{   double T1=300.0+273.15; //degree Celcius
    double P1=3000.0; //kPa
    double eta=0.9;
    double P2=100.0; //kPa
    double eta_isentropic=0.9;
     isentropic4 is2=new isentropic4 ();
```

```
    is2.calculate2(T1,P1,P2,eta);
    System.out.println(is2);
    is2.plot1();
    is2.plot2();
}
}
```

```
---------- Capture Output ----------
> "C:\Users\Turhan Çoban\.p2\pool\plugins\org.eclipse.justj.openjdk.hotspot.jre.full.win32.x86_64_19.0.2.v20230129-1123\jre\bin\javaw.exe" isentropic4
Adiabatic Process working fluid:Exhaust Gases
P1= 3000.0 kPa T1= 573.15 degree K
h1= 322.4228377726355 kJ/kg s1= -0.15867768298943197 kJ/kgK x1=2.0 kg vapor/kg
P2i= 100.0 kPa T2i= 233.21107641110146 degree K
h2i= -41.03002108687559 kJ/kg s2i= -0.15867768298943197 kJ/kgK x2i=2.0 kg vapor/kg
P2= 100.0 kPa T2= 0.0 degree K
h2= -4.684735200924479 kJ/kg s2= 0.0 kJ/kgK x2=0.0 kg vapor/kg
Wisentropic = 363.4528588595111 kJ/kg
Wadiabatic = 327.10757297355997 kJ/kg
eta_isentropic = 0.9
```



Constant pressure heating-cooling

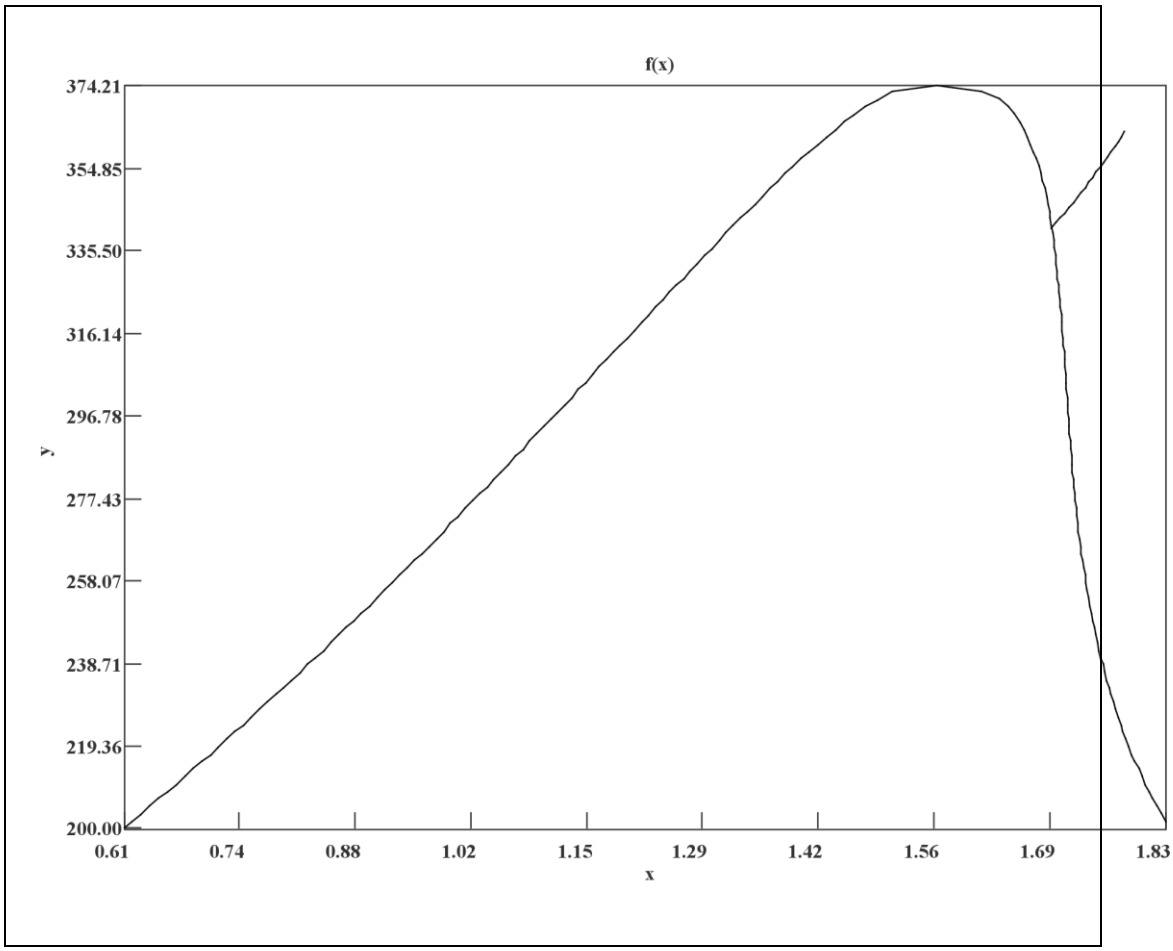
$$\dot{m}\left[\frac{\left(V_{out}^2 - V_{in}^2\right)}{2} + g(z_{out} - z_{in}) + (h_{out} - h_{in})\right] = \dot{Q} - \dot{W}$$

$$Q = \dot{m}[(h_{out} - h_{in})]$$

```
public class pressure1
{
public static void main(String arg[])
{
refrigerant_coolProp r=new refrigerant_coolProp("R134a");
double T0=273.15;
double P1=2000.0; //kPa
double T1=90.0+T0; //degree C
double P2=P1;
double x2=1.0;
double a1[]=r.property("tp",T1,P1);
double h1=a1[3];
double s1=a1[5];
double a2[]=r.property("px",P2,x2);
double h2=a2[3];
double s2=a2[5];
double T2=a2[1];
double dQ=(h1-h2);
System.out.println("dQ="+dQ+"T2="+T2);

double dT=(T1-T2)/19;
System.out.println("dT="+dT);
    double ss2[]=new double[20];
    double TT2[]=new double[20];
 ss2[0]= s1;
  TT2[0]=T1;
  for(int i=1;i<19;i++)
  {TT2[i]=TT2[i-1]-dT;
   double aa[]=r.property("tp",TT2[i],P1);
   ss2[i]=aa[5];
  }
  TT2[19]=T2;
  ss2[19]=s2;
graph2D p=new graph2D();
p.addData(r.TS_data(200.0));
p.addData(ss2,TT2);
p.plot();
}
}
```

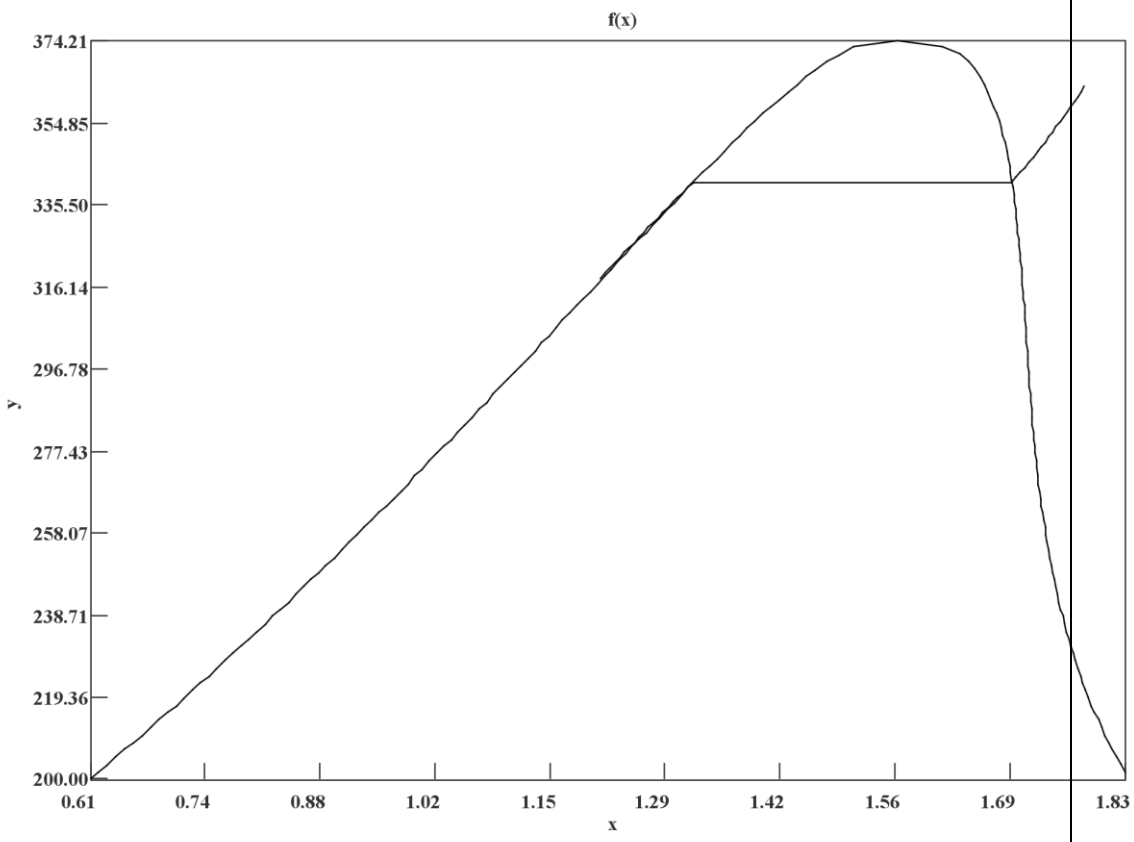
```
---------- Capture Output ----------
> "C:\Users\Turhan
Çoban\.p2\pool\plugins\org.eclipse.justj.openjdk.hotspot.jre.full.win32.x86_64_19.0.2.v2
0230129-1123\jre\bin\javaw.exe" pressure1
dQ=30.104427039694826T2=340.63075060197804
dT=1.1852236525274706
```




```
public class pressure2
{
public static void main(String arg[])
{
refrigerant_coolProp r=new refrigerant_coolProp("R134a");
double T0=273.15;
double P1=2000.0; //kPa
double T1=90.0+T0; //degree C
double P2=P1;
double x2=1.0;
double a1[]=r.property("tp",T1,P1);
double h1=a1[3];
double s1=a1[5];
double a2[]=r.property("px",P2,x2);
double h2=a2[3];
double s2=a2[5];
double T2=a2[1];
double dQ=(h1-h2);
System.out.println("dQ="+dQ+"T2="+T2);

double dT=(T1-T2)/19;
System.out.println("dT="+dT);
```

```
    double ss2[]=new double[20];
    double TT2[]=new double[20];
 ss2[0]= s1;
  TT2[0]=T1;
  for(int i=1;i<19;i++)
  {TT2[i]=TT2[i-1]-dT;
   double aa[]=r.property("tp",TT2[i],P1);
   ss2[i]=aa[5];
  }
  TT2[19]=T2;
  ss2[19]=s2;
   double aa1[]=r.property("tx",T2,0.0);
   double s3=aa1[5];
   double ss3[]={s2,s3};
   double TT3[]={T2,T2};
   double T3=300.0;
   dT=(T1-T2)/19;
   double ss4[]=new double[20];
   double TT4[]=new double[20];
   TT4[0]=T2;
   ss4[0]=s3;
   for(int i=1;i<20;i++)
  {TT4[i]=TT4[i-1]-dT;
   double aa[]=r.property("tp",TT4[i],P1);
   ss4[i]=aa[5];
  }
graph2D p=new graph2D();
p.addData(r.TS_data(200.0));
p.addData(ss2,TT2);
p.addData(ss3,TT3);
p.addData(ss4,TT4);
p.plot();
}
}
```



R134a in condition P=2000 kPa and T=80 ℃ cool to x=1.0 (saturated vapor) . R134a flows inside a pipe D=0.02 m and m=0.1 kg/s and L=? m long. Pipe surface temperature is constant at T=20 ℃. Calculate the total length of pipe, heat transfer, pressure drop and Plot the process in T-x diagram

Note: for plotting and carrying out the processes, you can use small finite difference path points

```
/* refrigerant_coolProp property vector
  a[0]="P, pressure        ";       kPa
  a[1]="T, temperature     ";   degree K
  a[2]="v, specific volume ";   m^3/kg
  a[3]="h, enthalpy        ";       kJ/kg
  a[4]="u, internal energy ";   kJ/kg
  a[5]="s, entropy         ";       kJ/kgK
  a[6]="x, quality         ";       kg/kg total
  a[7]="Cp, specific heat at constant pressure ";  kJ/kgK
  a[8]="Cv, specific heat at constant volume ";     kJ/kgK
  a[9]="isobaric thermal expansion coefficient ";  1/K
  a[10]="isothermal compressibility" ; 1/kPa
  a[11]="Dynamic viscosity" ;  Pas
  a[12]="k Thermal conductivity" ; W/mK
  a[13]="Surface tension" ;  N/m
  a[14]="Prandtl number" ;
  a[15]="density" ;  kg/m^3
  a[16]="speed of sound" ; m/s
```

```
*/
public class pressure3
{
public static double f(double Re,double eod)
 {
 //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;
 }
  // inside of pipe heat transfer
   public static double Nu(double Re,double Pr,double f)
   {double xx=(f/8.0)*(Re-
1000.0)*Pr/(1.07+12.7*Math.sqrt(f/8.0)*(Math.pow(Pr,(2.0/3.0))-1));
    return xx;
   }
public static void main(String arg[])
{String refname="R134a";
double ss1[]=new double[200];
double TT1[]=new double[200];
double PP1[]=new double[200];
refrigerant_coolProp r=new refrigerant_coolProp(refname);
double T0=273.15;
double P1=2000.0; //kPa
double T1=80.0+T0; //degree K R134a temperature
double D=0.02; //m
double A=Math.PI*D*D/4.0;
double m=0.1; //kg/s
double dL=0.01; // m
double Ts=20.0+T0; //degree K surface temp.
double a1[]=r.property("tp",T1,P1);
double h1=a1[3];
double s1=a1[5];
double x1=a1[6];
TT1[0]=T1;
ss1[0]=s1;
PP1[0]=P1;
int n=0;
double ro=a1[15];
double mu=a1[11];
double Pr=a1[14];
double k=a1[12];
double V=m/(ro*A); //m/s
double eps=1.2e-4; //m
double eod=eps/D;
double L=0.0;
```

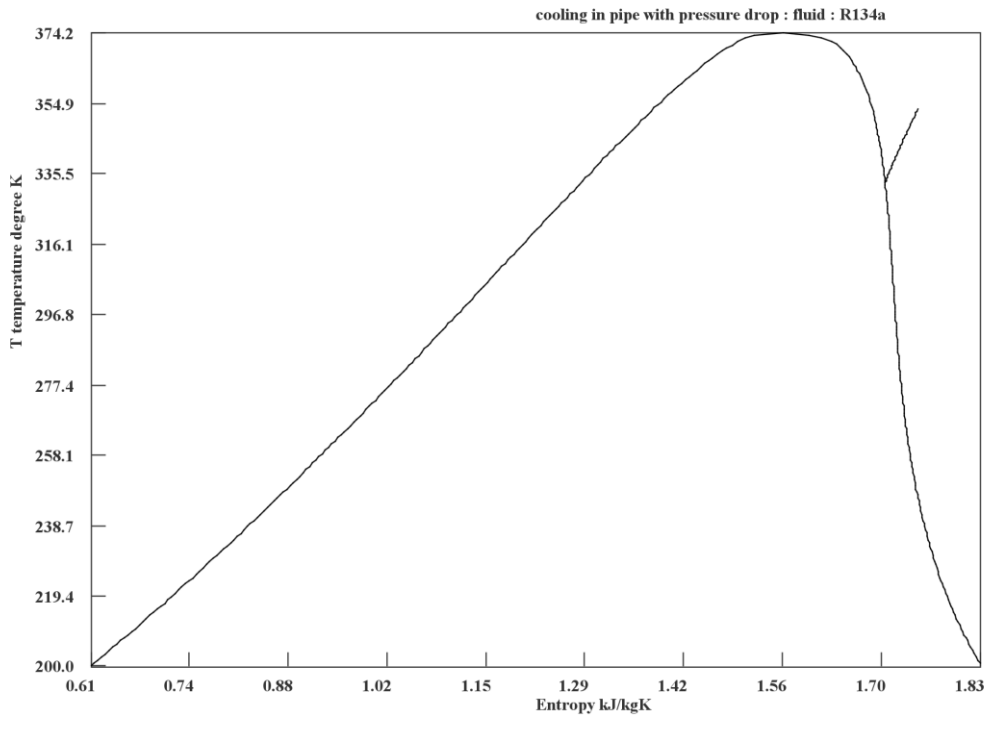
```
double Re=ro*V*D/mu;
double f=f(Re,eod);
double Nu=Nu(Re,Pr,f);
double h=Nu*k/D;
double dA=Math.PI*D*dL;
double dQ=h*dA*(T1-Ts);  //W
double dh=dQ*1e-3/m;
double dP=f*dL/D*ro*V*V/2.0;
double Q=dQ;

while(x1>1.0)
{  n++;
          L+=dL;
   h1=h1-dh;
   P1=P1-dP;
   a1=r.property("ph",P1,h1);
   x1=a1[6];
   T1=a1[1];
   s1=a1[5];
   TT1[n]=T1;
    ss1[n]=s1;
    PP1[n]=P1;
   ro=a1[15];
   mu=a1[11];
   Pr=a1[14];
   k=a1[12];
   V=m/(ro*A);
   Re=ro*V*D/mu;
   f=f(Re,eod);
   Nu=Nu(Re,Pr,f);
   h=Nu*k/D;
   dQ=h*dA*(T1-Ts);  //W
   dh=dQ*1e-3/m;
   Q+=dQ;
}
System.out.println("Q="+Q+"L="+L+"m");
double ss2[]=new double[n];
double TT2[]=new double[n];
double PP2[]=new double[n];
double x2[]=new double[n];
for(int i=0;i<n;i++)
{   x2[i]=i*dL;
          TT2[i]=TT1[i];
   ss2[i]=ss1[i];
   PP2[i]=PP1[i];
   System.out.println("i="+i+"x="+x2[i]+"m T="+TT2[i]+"degree K P="+PP2[i]+" kPa
s="+ss2[i]+" kJ/kgK");
}
graph2D p=new graph2D();
p.addData(r.TS_data(200.0));
p.addData(ss2,TT2);
p.setPlabel("cooling in pipe with pressure drop : fluid : "+refname);
p.setXlabel("Entropy kJ/kgK");
p.setYlabel("T temperature degree K");
p.plot();
```

```
}
}
```

```
---------- Capture Output ----------
> "C:\Users\Turhan Çoban\.p2\pool\plugins\org.eclipse.justj.openjdk.hotspot.jre.full.win32.x86_64_19.0.2.v20230129-1123\jre\bin\javaw.exe" pressure3
Q=1997.1567416965945L=0.3900000000000002m
i=0x=0.0m T=353.15degree K P=2000.0 kPa s=1.7480505623113984 kJ/kgK
i=1x=0.01m T=352.5531056359859degree K P=1991.3739820134113 kPa s=1.746608490219468 kJ/kgK
i=2x=0.02m T=351.96017815375876degree K P=1982.7479640268225 kPa s=1.745179999184538 kJ/kgK
i=3x=0.03m T=351.371178061256degree K P=1974.1219460402338 kPa s=1.7437650687825654 kJ/kgK
i=4x=0.04m T=350.7860660427883degree K P=1965.495928053645 kPa s=1.7423636796281876 kJ/kgK
i=5x=0.05m T=350.2048031302007degree K P=1956.8699100670563 kPa s=1.740975813604837 kJ/kgK
i=6x=0.06m T=349.6273507017962degree K P=1948.2438920804675 kPa s=1.7396014538699818 kJ/kgK
i=7x=0.07m T=349.0536704813463degree K P=1939.6178740938788 kPa s=1.7382405848599947 kJ/kgK
i=8x=0.08m T=348.4837245371868degree K P=1930.99185610729 kPa s=1.7368931922945803 kJ/kgK
i=9x=0.09m T=347.9174752814057degree K P=1922.3658381207013 kPa s=1.7355592631807488 kJ/kgK
i=10x=0.1m T=347.35488546912296degree K P=1913.7398201341125 kPa s=1.7342387858162913 kJ/kgK
i=11x=0.11m T=346.7959181978657degree K P=1905.1138021475238 kPa s=1.7329317497927188 kJ/kgK
i=12x=0.12m T=346.2405369070401degree K P=1896.487784160935 kPa s=1.7316381459976256 kJ/kgK
i=13x=0.13m T=345.68870537750445degree K P=1887.8617661743463 kPa s=1.7303579666164435 kJ/kgK
i=14x=0.14m T=345.1403877312416degree K P=1879.2357481877575 kPa s=1.729091205133543 kJ/kgK
i=15x=0.15m T=344.5955484311353degree K P=1870.6097302011688 kPa s=1.7278378563326358 kJ/kgK
i=16x=0.16m T=344.0541522808513degree K P=1861.98371221458 kPa s=1.726597916296455 kJ/kgK
i=17x=0.17m T=343.5161644248239degree K P=1853.3576942279913 kPa s=1.7253713824056507 kJ/kgK
i=18x=0.18m T=342.9815503483482degree K P=1844.7316762414025 kPa s=1.7241582533368756 kJ/kgK
i=19x=0.19m T=342.45027587778134degree K P=1836.1056582548138 kPa s=1.7229585290600111 kJ/kgK
i=20x=0.2m T=341.9223071808502degree K P=1827.479640268225 kPa s=1.7217722108344962 kJ/kgK
i=21x=0.21m T=341.3976107670678degree K P=1818.8536222816363 kPa s=1.720599301204718 kJ/kgK
i=22x=0.22m T=340.8761534882576degree K P=1810.2276042950475 kPa s=1.7194398039944225 kJ/kgK
i=23x=0.23m T=340.3579025391845degree K P=1801.6015863084588 kPa s=1.718293724300105 kJ/kgK
i=24x=0.24m T=339.8428254582936degree K P=1792.97556832187 kPa s=1.717161068483345 kJ/kgK
i=25x=0.25m T=339.33089012834796degree K P=1784.3495503352813 kPa s=1.7160418441612193 kJ/kgK
i=26x=0.26m T=338.8220647783109degree K P=1775.7235323486925 kPa s=1.7149360602001398 kJ/kgK
```

i=27x=0.27m T=338.31631798277334degree K P=1767.0975143621038 kPa
s=1.7138437266983042 kJ/kgK
i=28x=0.28m T=337.81361866439977degree K P=1758.471496375515 kPa
s=1.7127648549786043 kJ/kgK
i=29x=0.29m T=337.3139360947655degree K P=1749.8454783889263 kPa
s=1.7116994575734819 kJ/kgK
i=30x=0.3m T=336.8172398957058degree K P=1741.2194604023375 kPa
s=1.7106475482101882 kJ/kgK
i=31x=0.31m T=336.3235000407557degree K P=1732.5934424157488 kPa
s=1.7096091417947499 kJ/kgK
i=32x=0.32m T=335.8326868566759degree K P=1723.96742442916 kPa s=1.7085842543946088 kJ/kgK
i=33x=0.33m T=335.3447710250637degree K P=1715.3414064425713 kPa
s=1.7075729032199305 kJ/kgK
i=34x=0.34m T=334.8597235840432degree K P=1706.7153884559825 kPa
s=1.7065751066035546 kJ/kgK
i=35x=0.350m T=334.3775159300301degree K P=1698.0893704693938 kPa
s=1.7055908839795744 kJ/kgK
i=36x=0.36m T=333.89811981956774degree K P=1689.463352482805 kPa
s=1.7046202558605419 kJ/kgK
i=37x=0.37m T=333.4215073712279degree K P=1680.8373344962163 kPa
s=1.7036632438132793 kJ/kgK
i=38x=0.38m T=332.9476510675708degree K P=1672.2113165096275 kPa
s=1.7027198704333102 kJ/kgK



R134a in condition P=2000 kPa and T=80 ℃ cool to T=40 ℃ (liquid stat) . R134a flows inside a pipe D=0.02 m and m=1.5 kg/s and L=? m long. Pipe surface temperature is constant at T=20 ℃. Calculate the total length of pipe, heat transfer, pressure drop and Plot the process in T-x diagram

```
/* refrigerant_coolProp property vector
  a[0]="P, pressure       ";      kPa
  a[1]="T, temperature    ";   degree K
```

```
  a[2]="v, specific volume ";   m^3/kg
  a[3]="h, enthalpy       ";      kJ/kg
  a[4]="u, internal energy ";   kJ/kg
  a[5]="s, entropy        ";       kJ/kgK
  a[6]="x, quality        ";        kg/kg total
  a[7]="Cp, specific heat at constant pressure ";  kJ/kgK
  a[8]="Cv, specific heat at constant volume ";     kJ/kgK
  a[9]="isobaric thermal expansion coefficient ";  1/K
  a[10]="isothermal compressibility" ; 1/kPa
  a[11]="Dynamic viscosity" ;  Pas
  a[12]="k Thermal conductivity" ; W/mK
  a[13]="Surface tension" ;  N/m
  a[14]="Prandtl number" ;
  a[15]="density" ;  kg/m^3
  a[16]="speed of sound" ; m/s
*/
public class pressure4
{
public static double f(double Re,double eod)
 {
 //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;
 }
  // inside of pipe heat transfer single phase
   public static double Nu(double Re,double Pr,double f)
   { double xx=(f/8.0)*(Re-
1000.0)*Pr/(1.07+12.7*Math.sqrt(f/8.0)*(Math.pow(Pr,(2.0/3.0))-1));
     return xx;
   }
   // inside heat transfer with a phase change
public static double two_phase_homogeneous_Pressure_drop(String name,double
m,double Tsat,double x,double d,double L)
{refrigerant_coolProp st=new refrigerant_coolProp(name);
   double al[]=st.property("tx",Tsat,0.0);
   double av[]=st.property("tx",Tsat,1.0);
   double mul=al[11];  //kg/(m.s)=Pa.s
   double muv=av[11];  //kg/(m.s)=Pa.s
   double rol=al[15];  //kg/m^3
   double rov=av[15];  //kg/m^3
   double Psat=al[0]*1e3; //Pa
   double eH=1.0/(1.0+(1.0-x)/x*rov/rol);
```

```
   double roH=rol*(1.0-eH)+rov*eH;
   double mutp=x*muv+(1.0-x)*mul;
   double g=9.806; //m/s^2
   double A=Math.PI*d*d/4.0;
   double G=m/A;
   double Re=G*d/mutp;
   double ftp=0.079/Math.pow(Re,.25);
   double dP_friction=2.0*ftp*L*G*G/d/roH;
   return dP_friction;
}
public static double bisection(if_x f,double a,double b)
  {double b1=2.0*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>=100) r=bisection(f,a,b1);
   return r;
  }
public static void main(String arg[])
{String refname="R134a";
double ss1[]=new double[6000];
double TT1[]=new double[6000];
double PP1[]=new double[6000];
refrigerant_coolProp r=new refrigerant_coolProp(refname);
double T0=273.15;
double P1=2000.0; //kPa
double P0=P1;
double T1=80.0+T0; //degree K R134a temperature
double D=0.02; //m
double A=Math.PI*D*D/4.0;
double T2=40.0+273.15;
double m=1.5; //kg/s
double dL=0.005; // m
double Ts=20.0+T0; //degree K wall temp.
double Tw=0.0;
HT_K1_ref ff;
double a1[]=r.property("tp",T1,P1);
double h1=a1[3];
double s1=a1[5];
double x1=a1[6];
TT1[0]=T1;
ss1[0]=s1;
PP1[0]=P1;
int n=0;
double ro=a1[15];
```

```
double mu=a1[11];
double Pr=a1[14];
double k=a1[12];
double V=m/(ro*A); //m/s
double eps=1.2e-4; //m
double eod=eps/D;
double L=0.0;
double Re=ro*V*D/mu;
double f=f(Re,eod);
double Nu=Nu(Re,Pr,f);
double h=Nu*k/D;
double dA=Math.PI*D*dL;
double dQ=h*dA*(T1-Ts);  //W
double dh=dQ*1e-3/m;
double dP=f*dL/D*ro*V*V/2.0;
double Q=dQ;

while(x1>1.0)
{  n++;
          L+=dL;
  h1=h1-dh;
  P1=P1-dP*1e-3;
  a1=r.property("ph",P1,h1);
  x1=a1[6];
  T1=a1[1];
  s1=a1[5];
  TT1[n]=T1;
   ss1[n]=s1;
   PP1[n]=P1;
  ro=a1[15];
  mu=a1[11];
  Pr=a1[14];
  k=a1[12];
  V=m/(ro*A);
  Re=ro*V*D/mu;
  f=f(Re,eod);
  Nu=Nu(Re,Pr,f);
  h=Nu*k/D;
  dQ=h*dA*(T1-Ts);  //W
  dh=dQ*1e-3/m;
  Q+=dQ;
  dP=f*dL/D*ro*V*V/2.0; //Pa
  //System.out.println("n="+n+"x1="+x1+"T1="+T1);
}
while(x1>=0 && x1<=1.0)
{ n++;
  L+=dL;
  h1=h1-dh;
  P1=P1-dP*1e-3;
  a1=r.property("ph",P1,h1);
  x1=a1[6];
  T1=a1[1];
```

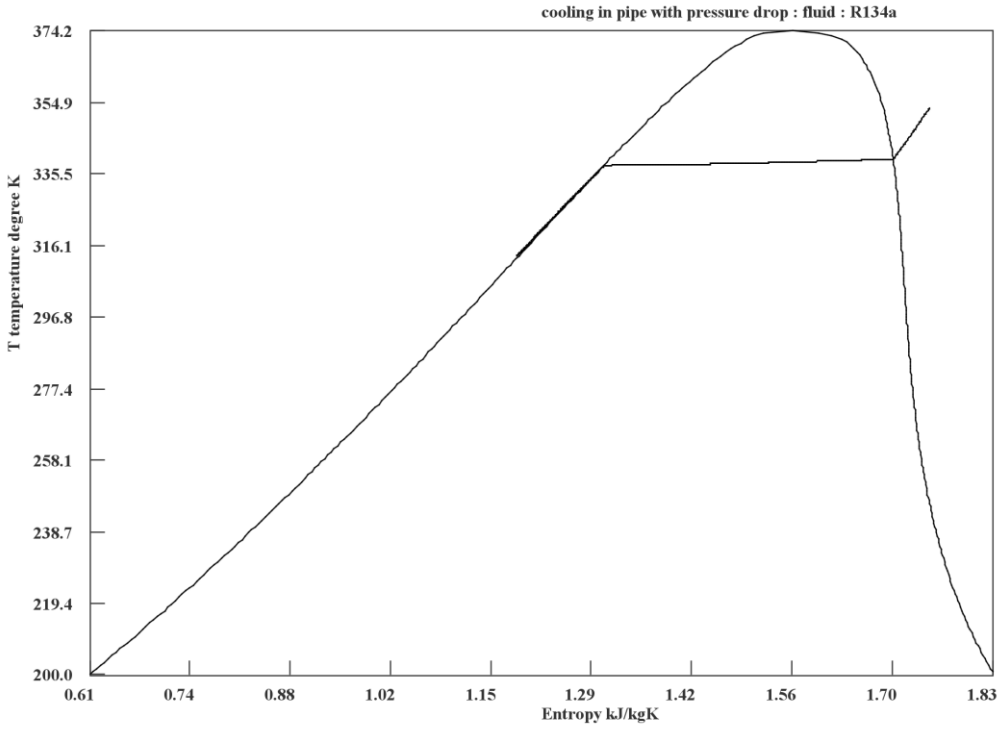
```
  s1=a1[5];
  TT1[n]=T1;
  ss1[n]=s1;
  PP1[n]=P1;
  ff=new  HT_K1_ref(refname,Ts,T1,m,x1,D,eps,"horizontal");
  h=bisection(ff,100.0,50000.0);
  dQ=h*dA*(T1-Ts);  //W
  dh=dQ*1e-3/m;
  Q+=dQ;
  dP=two_phase_homogeneous_Pressure_drop(refname,m,T1,x1,D,dL);
  //System.out.println("n="+n+"x1="+x1+"T1="+T1);
}
while(x1<=0  && T1>=T2)
{  n++;
          L+=dL;
  h1=h1-dh;
  P1=P1-dP*1e-3;
  a1=r.property("ph",P1,h1);
  x1=a1[6];
  T1=a1[1];
  s1=a1[5];
  TT1[n]=T1;
   ss1[n]=s1;
   PP1[n]=P1;
  ro=a1[15];
  mu=a1[11];
  Pr=a1[14];
  k=a1[12];
  V=m/(ro*A);
  Re=ro*V*D/mu;
  f=f(Re,eod);
  Nu=Nu(Re,Pr,f);
  h=Nu*k/D;
  dQ=h*dA*(T1-Ts);  //W
  dh=dQ*1e-3/m;
  Q+=dQ;
  dP=f*dL/D*ro*V*V/2.0;
  //System.out.println("n="+n+"x1="+x1+"T1="+T1);
}

System.out.println("Q = "+Q+" W L="+L+" m dP = "+(P0-P1)+" kPa");
double ss2[]=new double[n];
double TT2[]=new double[n];
double PP2[]=new double[n];
double x2[]=new double[n];
for(int i=0;i<n;i++)
{   x2[i]=i*dL;
          TT2[i]=TT1[i];
  ss2[i]=ss1[i];
  PP2[i]=PP1[i];
  System.out.println("i="+i+"x="+x2[i]+"m T="+TT2[i]+"degree K P="+PP2[i]+" kPa
s="+ss2[i]+" kJ/kgK");
```

```
}
graph2D p=new graph2D();
p.addData(r.TS_data(200.0));
p.addData(ss2,TT2);
p.setPlabel("cooling in pipe with pressure drop : fluid : "+refname);
p.setXlabel("Entropy kJ/kgK");
p.setYlabel("T temperature degree K");
p.plot();
}
}
```

Q = 284403.2739229959 W L=5.114999999999913 m dP = 166.22593861277778 kPa

i=0x=0.0m T=353.15degree K P=2000.0 kPa s=1.7480505623113984 kJ/kgK
i=1x=0.005m T=352.90545677773423degree K P=1999.035088870425 kPa s=1.7472328806151847 kJ/kgK
i=2x=0.01m T=352.66197684265086degree K P=1998.0714586854317 kPa s=1.7464172523532533 kJ/kgK
i=3x=0.015m T=352.4195561414038degree K P=1997.109108128158 kPa s=1.7456036648496531 kJ/kgK
……
i=42x=0.21m T=343.74869978197825degree K P=1960.5639769170677 kPa s=1.7153246152279156 kJ/kgK
i=43x=0.215m T=343.5454279003922degree K P=1959.651990190695 kPa s=1.7145817012952682 kJ/kgK
i=44x=0.22m T=343.3430602262504degree K P=1958.7412491809107 kPa s=1.7138402670153159 kJ/kgK
i=45x=0.225m T=343.1415934139944degree K P=1957.8317536223328 kPa s=1.7131002975059741 kJ/kgK
i=46x=0.23m T=342.9410241397933degree K P=1956.9235032805045 kPa s=1.712361777807773 kJ/kgK
i=47x=0.23500000000000001m T=342.74134909919337degree K P=1956.0164979522444 kPa s=1.711624692871781 kJ/kgK
i=48x=0.24m T=342.5425650088972degree K P=1955.1107374660278 kPa s=1.7108890275645783 kJ/kgK
i=49x=0.245m T=342.3446686062815degree K P=1954.2062216823592 kPa s=1.7101547666637908 kJ/kgK
i=50x=0.25m T=342.1476566494207degree K P=1953.302950494159 kPa s=1.7094218948556281 kJ/kgK
…….



Constant pressure heating-cooling Heat exchangers

$$\dot{m}\left[\frac{\left(V_{out}^2 - V_{in}^2\right)}{2} + g(z_{out} - z_{in}) + (h_{out} - h_{in})\right] = \dot{Q} - \dot{W}$$

$$Q = \dot{m}_h\left[\left(h_{h_out} - h_{h_in}\right)\right]$$
$$Q = \dot{m}_c\left[\left(h_{c_out} - h_{c_in}\right)\right]$$

```
public class double_pipe_heat_exchanger_2022
{  public double din,dout;
   public double mh,mc;
   public double L;
   public double dL;
   public int n;
   steamIAPWS_IF97 water;
   double Th[];
   double Tc[];
   double x[];
   double Ah;
   double Ac;
   double dh;
   double eps;
   double dA;
   public double Q;
   public double_pipe_heat_exchanger_2022(double dini,double douti,double mhi,double
mci,double Li)
   {din=dini;dout=douti;mh=mhi;mc=mci;L=Li;n=100;dL=L/(n-1);
     water=new steamIAPWS_IF97();
     Th=new double[n];
      Tc=new double[n];
      x=new double[n];
      Ah=Math.PI*din*din/4.0;
      Ac=Math.PI*(dout*dout-din*din)/4.0;
      dh=dout-din;
      eps=1.2e-4; //m
      dA=dL*Math.PI*din;
     }
    // eps     : pipe equivalent roughness m
 //         ----------------------------------------
 //         value of equivalent roughness for some pipes:
 //         Pipe             equivalent rougness m
 //         -------------    ---------------------
 //         Riveted steel    0.9e-3 - 9e-3
 //         Concrete         0.3e-3 - 3e-3
 //         Wood stave       0.18e-3 - 0.9 e-3
 //         Cast iron        0.26e-3
 //         Galvanised iron  0.15e-3
 //         Commercial steel 0.045e-3
 //         wrought iron     0.045e-3
 //         drawn tubing     0.0015e-3
```

```
//          Plastic,glass    0.0 (smooth)
//          ----------------------------------------
public double f(double Re,double eod)
{
//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;
}
 public static double f_Haaland(double Re,double eod)
 {double xx=-1.8*Math.log10(Math.pow((eod/3.7) ,1.11)+6.9/Re);
   double f=1.0/(xx*xx);
   return f;
  }
  public static double Nu(double Re,double Pr,double f)
  {double xx=(f/8.0)*(Re-
1000.0)*Pr/(1.07+12.7*Math.sqrt(f/8.0)*(Math.pow(Pr,(2.0/3.0))-1));
   return xx;
  }
  //Parallel flow
  public void calculate_parallel(double Thi,double Tci,double Phi,double Pci)
  { Th[0]=Thi;
     Tc[0]=Tci;
      x[0]=0.0;
     Q=0.0;
     for(int i=1;i<n;i++)
     { x[i]=dL*i;
               double a[]=water.propertyC("tp",Th[0],Phi);
     double roh=1.0/a[2];
     double vish=a[11];
     double kh=a[12];
     double Cph=a[7];
      double Prh=a[14];//Cph*1e3*vish/kh;
     double b[]=water.propertyC("tp",Tc[0],Pci);
     double roc=1.0/b[2];
     double visc=b[11];
     double kc=b[12];
      double Cpc=b[7];
      double Prc=b[14];//Cpc*1e3*visc/kc;
```

```
    double Vc=mc/(roc*Ac);  //cold site velocity m/s
    double Vh=mh/(roh*Ah); //hot side velocity m/s
    double Reh=roh*Vh*din/vish;
     double Rec=roc*Vc*dh/visc;
     double eodh=eps/din;
     double eodc=eps/dh;
     double fh=f(Reh,eodh);
     double fc=f(Rec,eodc);
     double Nuh=Nu(Reh,Prh,fh);
     double Nuc=Nu(Rec,Prc,fc);
     double hh=Nuh*kh/din;
      double hc=Nuc*kc/dh;
      double U=1.0/(1.0/hh+1.0/hc);
      double dQ=U*dA*(Th[i-1]-Tc[i-1]);
      Q+=dQ;
      Th[i]=Th[i-1]-dQ/(mh*Cph*1e3);
      Tc[i]=Tc[i-1]+dQ/(mc*Cpc*1e3);
    }
  }
    //Counter flow Thi (hot side inlet) and Tco (cold site outlet) is given
  public double calculate1(double Thi,double Tco,double Phi,double Pci)
  { Th[0]=Thi;
    Tc[0]=Tco;
     x[0]=0.0;
    Q=0.0;
    for(int i=1;i<n;i++)
    { x[i]=dL*i;
              double a[]=water.propertyC("tp",Th[0],Phi);
    double roh=1.0/a[2];
    double vish=a[11];
    double kh=a[12];
    double Cph=a[7];
     double Prh=a[14];//Cph*1e3*vish/kh;
    double b[]=water.propertyC("tp",Tc[0],Pci);
    double roc=1.0/b[2];
    double visc=b[11];
    double kc=b[12];
     double Cpc=b[7];
     double Prc=b[14];//Cpc*1e3*visc/kc;
    double Vc=mc/(roc*Ac);  //cold site velocity m/s
    double Vh=mh/(roh*Ah); //hot side velocity m/s
    double Reh=roh*Vh*din/vish;
     double Rec=roc*Vc*dh/visc;
     double eodh=eps/din;
     double eodc=eps/dh;
     double fh=f(Reh,eodh);
     double fc=f_Haaland(Rec,eodc);
     double Nuh=Nu(Reh,Prh,fh);
```

```
      double Nuc=Nu(Rec,Prc,fc);
      double hh=Nuh*kh/din;
       double hc=Nuc*kc/dh;
       double U=1.0/(1.0/hh+1.0/hc);
       double dQ=U*dA*(Th[i-1]-Tc[i-1]);
       Q+=dQ;
       Th[i]=Th[i-1]-dQ/(mh*Cph*1e3);
       Tc[i]=Tc[i-1]-dQ/(mc*Cpc*1e3);
     }
     return Tc[n-1];
  }
  //Counter flow Thi (hot side inlet) and Tci (cold site inlet) is given
  //bisection root finding is applied in limits of Tci and Thi to find Tco
   public void calculate_counter(double Thi,double Tci,double Phi,double Pci)
   { if_x f=(x)->calculate1(Thi,x,Phi,Pci) - Tci;
    double Tco=f.bisection(Tci,Thi);
    calculate1(Thi,Tco,Phi,Pci);
   }
   public static void main(String arg[])
   {  double dini=12.7e-3; //m
      double douti=30e-3; //m
      double mhi=0.02; //kg/s
      double mci=0.05; //kg/s
      double Li=10.0; //m
            double_pipe_heat_exchanger_2022 ib=new
double_pipe_heat_exchanger_2022(dini,douti,mhi,mci,Li);
            double Thi=70.0; //degree C
            double Tci=20.0; //kPa
            double Phi=200.0; //kPa
            double Pci=300.0; //kPa
            //Parallel flow heat exchanger
            ib.calculate_parallel(Thi,Tci,Phi,Pci);
            double cc[][]={ib.x,ib.Tc,ib.Th};
            String s[]={"x m","cold water temp.","hot water temp."};
            Text.printT(cc,s,"Parallel flow heat exchanger");
            Plot p=new Plot(ib.x,ib.Th);
            p.addData(ib.x,ib.Tc);
            p.setPlabel("Parallel Flow double pipe heat exchanger");
      p.setXlabel("pipe length x m");
      p.setYlabel("Temperature degree C");
            p.plot();
            // counter flow heat exchanger
            ib.calculate_counter(Thi,Tci,Phi,Pci);
            double cc1[][]={ib.x,ib.Tc,ib.Th};
            String s1[]={"x m","cold water temp.","hot water temp."};
            Text.printT(cc1,s1,"Counter flow heat exchanger");
            Plot p1=new Plot(ib.x,ib.Th);
            p1.addData(ib.x,ib.Tc);
```

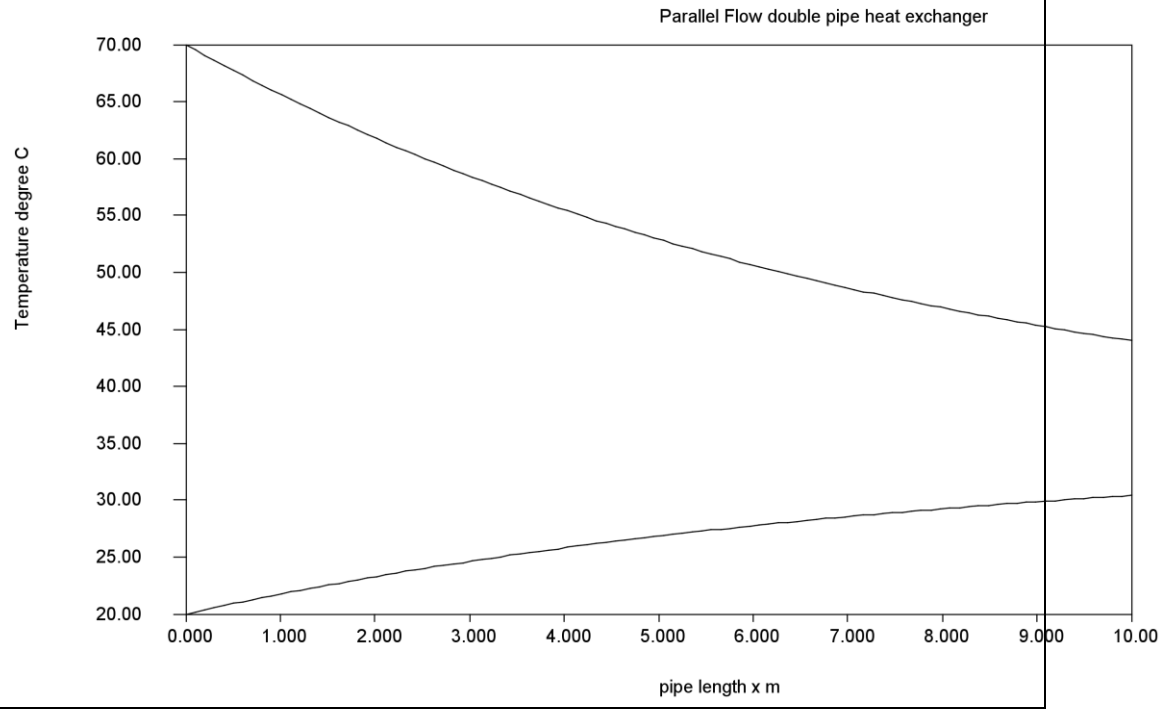
```
            p1.setPlabel("CounterFlow double pipe heat exchanger");
        p1.setXlabel("pipe length x m");
        p1.setYlabel("Temperature degree C");
            p1.plot();
    }
}
```

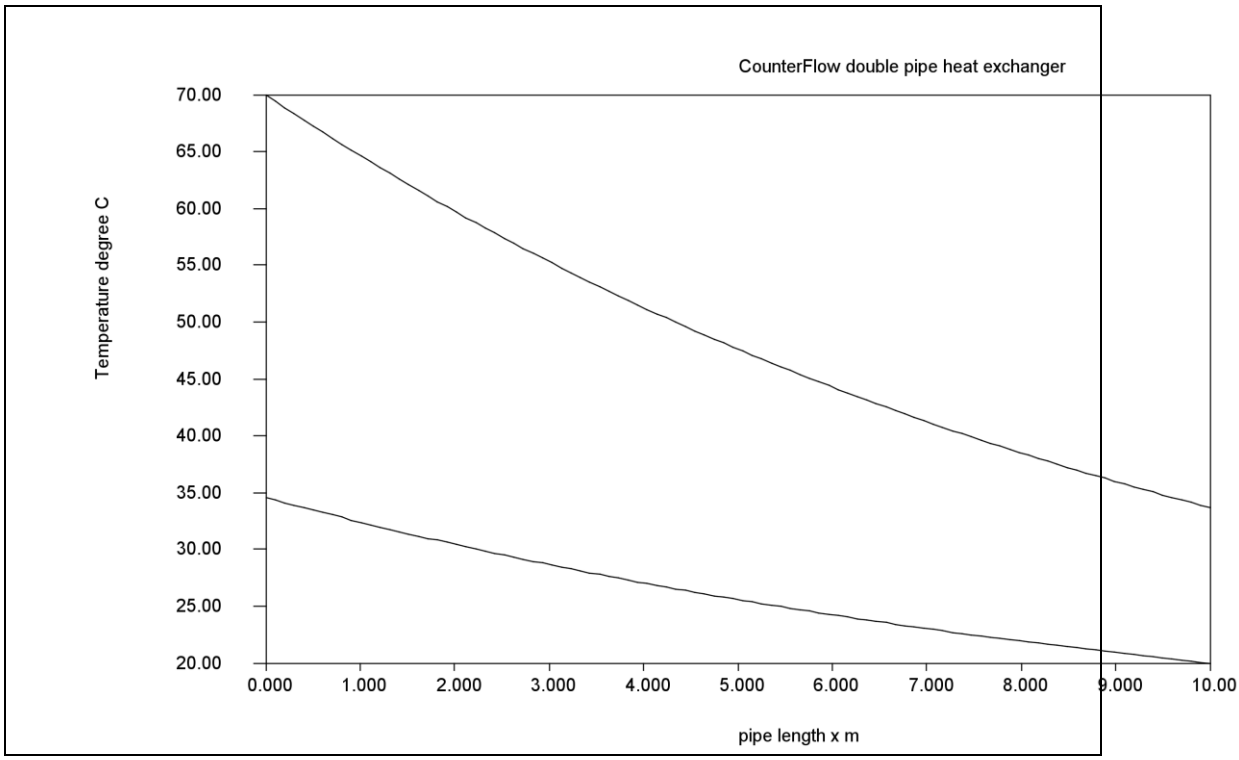
Parallel flow heat exchanger

x m	cold water temp.	hot water temp.
0.0	20.0	70.0
0.10101010101010101	20.186848160078842	69.53329263127046
0.20202020202020202	20.37125400719634	69.07268564616429
0.30303030303030304	20.553249465095277	68.6180993058926
0.40404040404040403	20.732866040239653	68.16945491394101
0.5050505050505051	20.910134827269005	67.726674802446
0.6060606060606061	21.085086514381395	67.28968231874927
0.7070707070707071	21.25775138864605	66.8584018121279
0.8080808080808081	21.428159341246573	66.43275862069791
0.9090909090909091	21.596339872655594	66.0126790584891
1.0101010101010102	21.762322097741812	65.59809040268871
1.1111111111111112	21.92613475081025	65.18892088105191
1.2121212121212122	22.08780619057665	64.78509965947683
1.3131313131313131	22.24736440507686	64.38655682974195
1.4141414141414141	22.404837016511998	63.993223397403874
1.5151515151515151	22.56025128603035	63.60503126985319
1.6161616161616161	22.7136341184467 16	63.221913244526526
1.7171717171717171	22.865012066900082	62.843802997272626
1.8181818181818181	23.01441133745042	62.470635070870564
1.9191919191919191	23.161857793615393	62.10234486369799
2.0202020202020203	23.307376960847773	61.73886861854748
2.121212121212121	23.450994030954305	61.380143411589145
2.2222222222222223	23.59273386645685	61.026107141477425
2.323232323232323	23.732621004896494	60.676698518600304
2.4242424242424243	23.870679663081408	60.331857054469076

Counter flow heat exchanger

x m	cold water temp.	hot water temp.
0.0	34.55762152481887	70.0
0.10101010101010101	34.330785764914296	69.43418063343024
0.20202020202020202	34.10611954349221	68.87377296601423
0.30303030303030304	33.88360211030924	68.31872523832224
0.40404040404040403	33.66321291358475	67.76898618597033
0.5050505050505051	33.44493159810271	67.22450503488544
0.6060606060606061	33.22873800333168	66.68523149661596
0.7070707070707071	33.01461216156274	66.15111576368705
0.8080808080808081	32.8025342960653	65.62210850500038
0.9090909090909091	32.592484819260534	65.09816086127798
1.0101010101010102	32.3844443309122	64.57922444054945
1.1111111111111112	32.1783936163349	64.0652513136826
1.2121212121212122	31.974313644619347	63.55619400995656
1.3131313131313131	31.772185566874697	63.05200551267751
1.4141414141414141	31.571990714487658	62.552639254836095
1.5151515151515151	31.373710597398237	62.058049114806536
1.6161616161616161	31.177326902392	61.56818941208682
1.7171717171717171	30.98282149140866	61.083014903079594
1.8181818181818181	30.79017639986684	60.60248077691351
1.9191919191919191	30.59937383500485	60.126542651304426
2.0202020202020203	30.410396174237356	59.655156568456285
2.121212121212121	30.22322596352773	59.18827899100113
2.2222222222222223	30.03784591577602	58.72586679797799
2.323232323232323	29.854238909222282	58.26787728085023
2.4242424242424243	29.67238798586522	57.814268139560944





heating-cooling with pressure drop: Heat exchangers

$$\dot{m}\left[\frac{\left(V_{out}^2 - V_{in}^2\right)}{2} + g(z_{out} - z_{in}) + (h_{out} - h_{in})\right] = \dot{Q} - \dot{W}$$

$$Q = \dot{m}_h\left[\left(h_{h_out} - h_{h_in}\right)\right]$$
$$Q = \dot{m}_c\left[\left(h_{c_out} - h_{c_in}\right)\right]$$

```
//This class adapting nonlinear system of equation
// function to optimisation problem function
class f2 implements if_xj
{//adaptation function
ifi_xj ff1;
public f2(ifi_xj ffi) {ff1=ffi;}

public double func(double x[])
{
double g=0.0;
double fa[]=ff1.func(x);
for(int i=0;i<fa.length;i++)  g+=fa[i]*fa[i];
return g;
}
}
// Double pipe heat exchanger with pressure drop in each
// finite difference step
public class double_pipe_HE_with_PressureD
{  public double din,dout; //ic ve dis cap
   public double mh,mc; //sicak ve soguk sivi ic sicak dis soguk
```

```
  public double L;
  public double dL;
  public int n;
  // both side of the heat exchanger utilises water as warking fluid
  steamIAPWS_IF97 water;  //water and steam properties
  double Th[];
  double Tc[];
  double Ph[];
  double Pc[];
  double x[];
  double Ah;
  double Ac;
  double dh;
  double eps;
  double dA;
  public double Q;
  public double dPh;
  public double dPc;
  public double_pipe_HE_with_PressureD(double dini,double douti,double mhi,double
mci,double Li)
  {din=dini;dout=douti;mh=mhi;mc=mci;L=Li;n=100;dL=L/(n-1);
    water=new steamIAPWS_IF97();
    Th=new double[n];
     Tc=new double[n];
     Ph=new double[n];
     Pc=new double[n];
     x=new double[n];
     Ah=Math.PI*din*din/4.0;
     Ac=Math.PI*(dout*dout-din*din)/4.0;
     dh=dout-din;
     eps=1.2e-4; //m
     dA=dL*Math.PI*din;
    }
    // eps     : pipe equivalent roughness m
//         ------------------------------------------
//         value of equivalent roughness for some pipes:
//         Pipe             equivalent rougness m
//         -------------    ---------------------
//         Riveted steel    0.9e-3 - 9e-3
//         Concrete         0.3e-3 - 3e-3
//         Wood stave       0.18e-3 - 0.9 e-3
//         Cast iron        0.26e-3
//         Galvanised iron  0.15e-3
//         Commercial steel 0.045e-3
//         wrought iron     0.045e-3
//         drawn tubing     0.0015e-3
//         Plastic,glass    0.0 (smooth)
//         ------------------------------------------
```

```
public double f(double Re,double eod)
{
//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;
}
 public static double f_Haaland(double Re,double eod)
 {double xx=-1.8*Math.log10(Math.pow((eod/3.7) ,1.11)+6.9/Re);
  double f=1.0/(xx*xx);
  return f;
  }
  public static double Nu(double Re,double Pr,double f)
  {double xx=(f/8.0)*(Re-
1000.0)*Pr/(1.07+12.7*Math.sqrt(f/8.0)*(Math.pow(Pr,(2.0/3.0))-1));
   return xx;
  }
  //Parallel flow
  public void calculate_parallel(double Thi,double Tci,double Phi,double Pci)
  { Th[0]=Thi;
    Tc[0]=Tci;
    Ph[0]=Phi;
    Pc[0]=Pci;
     x[0]=0.0;
    Q=0.0;
    dPc=0.0;
    dPh=0.0;
    for(int i=1;i<n;i++)
    { x[i]=dL*i;
              double a[]=water.propertyC("tp",Th[i-1],Ph[i-1]);
    double roh=1.0/a[2];
    double vish=a[11];
    double kh=a[12];
    double Cph=a[7];
     double Prh=Cph*1e3*vish/kh;
    double b[]=water.propertyC("tp",Tc[i-1],Pc[i-1]);
    double roc=1.0/b[2];
    double visc=b[11];
    double kc=b[12];
```

```
    double Cpc=b[7];
    double Prc=b[14];//Cpc*1e3*visc/kc;
   double Vc=mc/(roc*Ac);  //cold site velocity m/s
   double Vh=mh/(roh*Ah); //hot side velocity m/s
   double Reh=roh*Vh*din/vish;
    double Rec=roc*Vc*dh/visc;
    double eodh=eps/din;
    double eodc=eps/dh;
    double fh=f(Reh,eodh);
    double fc=f(Rec,eodc);
    double Nuh=Nu(Reh,Prh,fh);
    double Nuc=Nu(Rec,Prc,fc);
    double hh=Nuh*kh/din;
     double hc=Nuc*kc/dh;
     double U=1.0/(1.0/hh+1.0/hc);
     double dQ=U*dA*(Th[i-1]-Tc[i-1]);
     double dPc1=fc*L/dh*roc*Vc*Vc/2.0;
     double dPh1=fh*L/din*roh*Vh*Vh/2.0;
     dPc+=dPc1*1e-3;
     dPh+=dPh1*1e-3;
     Pc[i]=Pc[i-1]-dPc1*1e-3;
     Ph[i]=Ph[i-1]-dPh1*1e-3;
     Q+=dQ;
     Th[i]=Th[i-1]-dQ/(mh*Cph*1e3);
     Tc[i]=Tc[i-1]+dQ/(mc*Cpc*1e3);
   }
}
   //Counter flow Thi (hot side inlet) and Tco (cold site outlet) is given
public double[] calculate1(double Thi,double Tco,double Phi,double Pco)
{ Th[0]=Thi;
   Tc[0]=Tco;
   Ph[0]=Phi;
   Pc[0]=Pco;
    x[0]=0.0;
   Q=0.0;
   for(int i=1;i<n;i++)
   { x[i]=dL*i;
           double a[]=water.propertyC("tp",Th[i-1],Ph[i-1]);
   double roh=1.0/a[2];
   double vish=a[11];
   double kh=a[12];
   double Cph=a[7];
    double Prh=a[14];//Cph*1e3*vish/kh;
   double b[]=water.propertyC("tp",Tc[i-1],Pc[i-1]);
   double roc=1.0/b[2];
   double visc=b[11];
   double kc=b[12];
    double Cpc=b[7];
```

```
      double Prc=b[14];//Cpc*1e3*visc/kc;
     double Vc=mc/(roc*Ac);  //cold site velocity m/s
     double Vh=mh/(roh*Ah); //hot side velocity m/s
     double Reh=roh*Vh*din/vish;
      double Rec=roc*Vc*dh/visc;
      double eodh=eps/din;
      double eodc=eps/dh;
      double fh=f(Reh,eodh);
      double fc=f_Haaland(Rec,eodc);
      double Nuh=Nu(Reh,Prh,fh);
      double Nuc=Nu(Rec,Prc,fc);
      double hh=Nuh*kh/din;
       double hc=Nuc*kc/dh;
       double U=1.0/(1.0/hh+1.0/hc);
       double dQ=U*dA*(Th[i-1]-Tc[i-1]);
       Q+=dQ;
       double dPc1=fc*L/dh*roc*Vc*Vc/2.0;
       double dPh1=fh*L/din*roh*Vh*Vh/2.0;
       dPc+=dPc1*1e-3;
       dPh+=dPh1*1e-3;
       Pc[i]=Pc[i-1]-dPc1*1e-3;
       Ph[i]=Ph[i-1]-dPh1*1e-3;
       Th[i]=Th[i-1]-dQ/(mh*Cph*1e3);
       Tc[i]=Tc[i-1]-dQ/(mc*Cpc*1e3);
     }
     double d[]={Tc[n-1],Pc[n-1]};
     return d;
  }
 //Counter flow Thi (hot side inlet) and Tci (cold site inlet) is given
 //bisection root finding is applied in limits of Tci and Thi to find Tco
  public void calculate_counter(double Thi,double Tci,double Phi,double Pci)
  { ifi_xj f1=(double x[])->{double a[]=calculate1(Thi,x[0],Phi,x[1]);
                                double ff[]=new double[2];
                                ff[0]=a[0]-Tci;
                                ff[1]=a[1]-Pci;
                               return ff;};
   f2 f=new f2(f1);
    double [] x0={(Thi+Tci)/2.0,Pci};
    double [] dx0=new double[x0.length];
    for(int i=0;i<x0.length;i++) dx0[i]=0.2*x0[i];
    double a[]=nelderi.nelder(f,x0,dx0);
    double Tco=a[0];
    double Pco=a[1];
    calculate1(Thi,Tco,Phi,Pci);
  }
  public static void main(String arg[])
  {  double dini=12.7e-3; //m
     double douti=30e-3; //m
```

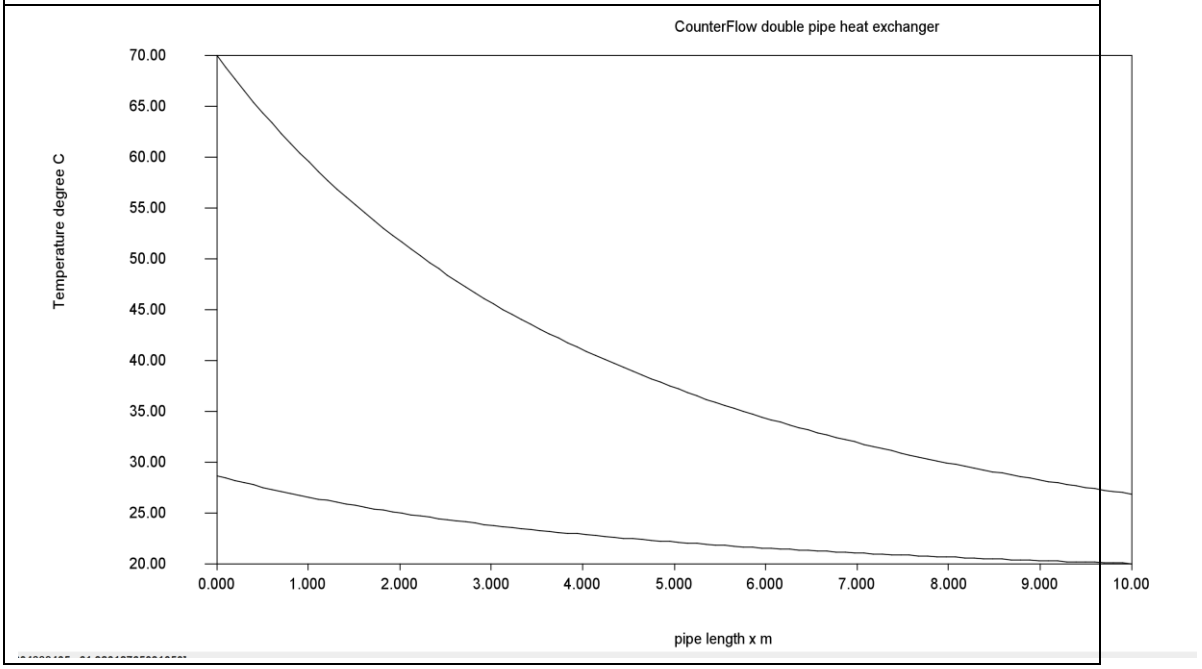
```
      double mhi=0.02; //kg/s
      double mci=0.1; //kg/s
      double Li=10.0; //m
            double_pipe_HE_with_PressureD ib=new
double_pipe_HE_with_PressureD(dini,douti,mhi,mci,Li);
            double Thi=70.0; //degree C
            double Tci=20.0; //kPa
            double Phi=200.0; //kPa
            double Pci=300.0; //kPa
            //Parallel flow heat exchanger
            ib.calculate_parallel(Thi,Tci,Phi,Pci);
            System.out.println("Q_parallel ="+ib.Q);
            double cc[][]={ib.x,ib.Tc,ib.Th,ib.Pc,ib.Ph};
            String s[]={"x m","cold water temp.","hot water temp.","cold water
pressure","hot water pressure"};
            Text.printT(cc,s,"Parallel flow heat exchanger");
            Plot p=new Plot(ib.x,ib.Th);
            p.addData(ib.x,ib.Tc);
            p.setPlabel("Parallel Flow double pipe heat exchanger");
      p.setXlabel("pipe length x m");
      p.setYlabel("Temperature degree C");
            p.plot();
            Plot p1=new Plot(ib.x,ib.Ph);
            p1.addData(ib.x,ib.Pc);
            p1.setPlabel("Parallel Flow double pipe heat exchanger");
      p1.setXlabel("pipe length x m");
      p1.setYlabel("Pressure kPa");
            p1.plot();
            // counter flow heat exchanger
            ib.calculate_counter(Thi,Tci,Phi,Pci);
            System.out.println("Q_counter ="+ib.Q);
            double cc1[][]={ib.x,ib.Tc,ib.Th,ib.Pc,ib.Ph};
            String s1[]={"x m","cold water temp.","hot water temp."};
            Text.printT(cc1,s1,"Counter flow heat exchanger");
            Plot p2=new Plot(ib.x,ib.Th);
            p2.addData(ib.x,ib.Tc);
            p2.setPlabel("CounterFlow double pipe heat exchanger");
      p2.setXlabel("pipe length x m");
      p2.setYlabel("Temperature degree C");
            p2.plot();
            Plot p3=new Plot(ib.x,ib.Ph);
            p3.addData(ib.x,ib.Pc);
            p3.setPlabel("Counter Flow double pipe heat exchanger");
      p3.setXlabel("pipe length x m");
      p3.setYlabel("Pressure kPa");
            p3.plot();
   }
}
```

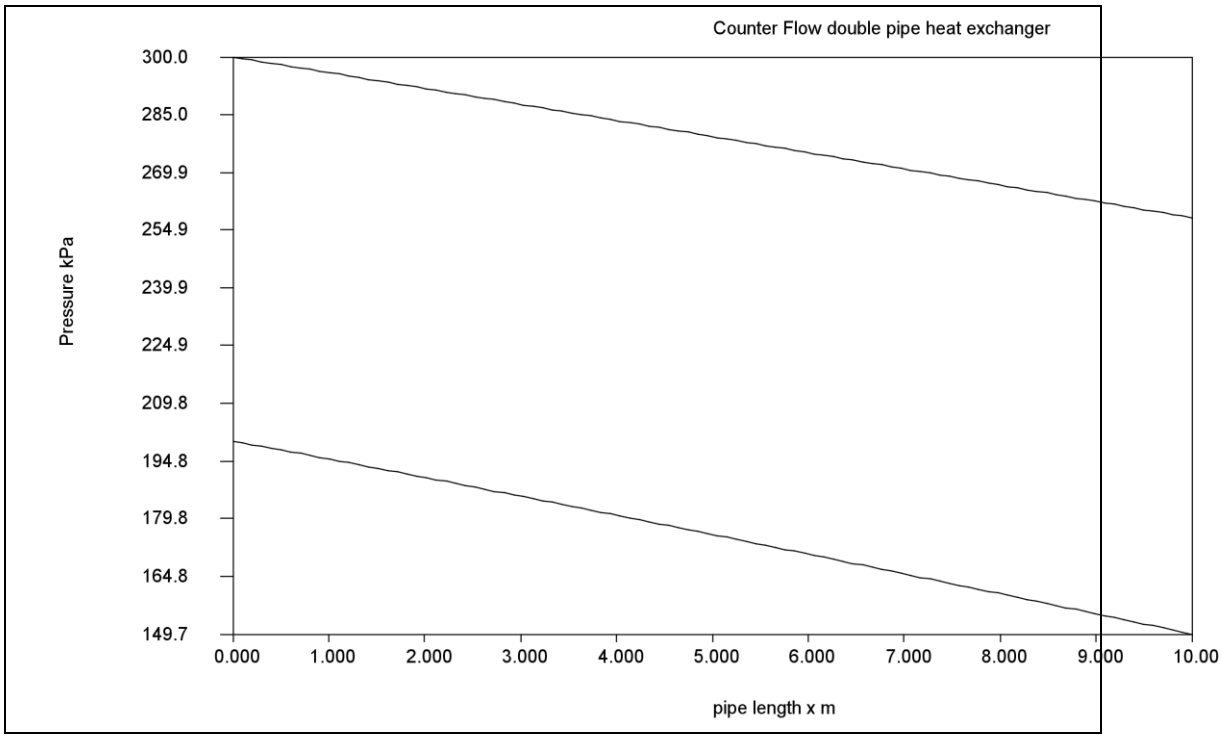
Parallel flow heat exchanger

x m	cold water temp.	hot water temp.	cold water pres..	hot water press..
0.0	20.0	70.0	300.0	200.0
0.101010101...	20.25666656...	68.71780191...	299.5741103...	199.5301259...
0.202020202...	20.50536785...	67.47520951...	299.1488085...	199.0592251...
0.303030303...	20.74629886...	66.27127430...	298.7240717...	198.5872943...
0.404040404...	20.97965690...	65.10503336...	298.2998781...	198.1143317...
0.505050505...	21.20564077...	63.97551421...	297.8762065...	197.6403367...
0.606060606...	21.42444982...	62.88173920...	297.4530369...	197.1653104...
0.707070707...	21.63628329...	61.82272951...	297.0303498...	196.6892549...
0.808080808...	21.84133952...	60.79750870...	296.6081267...	196.2121736...
0.909090909...	22.03981545...	59.80510585...	296.1863499...	195.7340712...
1.010101010...	22.23190603...	58.84455841...	295.7650022...	195.2549531...
1.111111111...	22.41780380...	57.91491460...	295.3440673...	194.7748259...
1.212121212...	22.59769845...	57.01523556...	294.9235295...	194.2936971...
1.313131313...	22.77177650...	56.14459721...	294.5033739...	193.8115749...
1.414141414...	22.94022100...	55.30209177...	294.0835860...	193.3284684...
1.515151515...	23.10321130...	54.48682912...	293.6641520...	192.8443873...
1.616161616...	23.26092283...	53.69793788...	293.2450586...	192.3593420...
1.717171717...	23.41352697...	52.93456628...	292.8262933...	191.8733436...
1.818181818...	23.56119088...	52.19588287...	292.4078438...	191.3864035...
1.919191919...	23.70407746...	51.48107706...	291.9896986...	190.8985336...
2.020202020...	23.84234523...	50.78935949...	291.5718463...	190.4097463...
2.121212121...	23.97614834...	50.11996231...	291.1542765...	189.9200544...
2.222222222...	24.10563654...	49.47213926...	290.7369786...	189.4294710...
2.323232323...	24.23095516...	48.84516577...	290.3199431...	188.9380093...
2.424242424...	24.35224515...	48.23833882...	289.9031602...	188.4456830...

Counter flow heat exchanger

x m	cold water temp.	hot water temp.	3	4
0.0	28.63596540...	70.0	300.0	200.0
0.101010101...	28.39334911...	68.78923028...	299.5898516...	199.5301259...
0.202020202...	28.15836972...	67.61633769...	299.1792382...	199.0592833...
0.303030303...	27.93071400...	66.47978801...	298.7681710...	198.5874712...
0.404040404...	27.71008464...	65.37812342...	298.3566610...	198.1146897...
0.505050505...	27.49619928...	64.30995804...	297.9447189...	197.6409394...
0.606060606...	27.28878968...	63.27397371...	297.5323549...	197.1662217...
0.707070707...	27.08760080...	62.26891614...	297.1195792...	196.6905387...
0.808080808...	26.89239008...	61.29359123...	296.7064015...	196.2138932...
0.909090909...	26.70292671...	60.34686171...	296.2928314...	195.7362884...
1.010101010...	26.51899093...	59.42764392...	295.8788780...	195.2577280...
1.111111111...	26.34037341...	58.53490483...	295.4645504...	194.7782164...
1.212121212...	26.16687466...	57.66765926...	295.0498573...	194.2977583...
1.313131313...	25.99830448...	56.82496724...	294.6348071...	193.8163588...
1.414141414...	25.83448145...	56.00593156...	294.2194082...	193.3340233...
1.515151515...	25.67523243...	55.20969550...	293.8036685...	192.8507579...
1.616161616...	25.52039211...	54.43544061...	293.3875958...	192.3665685...
1.717171717...	25.36980263...	53.68238473...	292.9711979...	191.8814617...
1.818181818...	25.22331310...	52.94978006...	292.5544819...	191.3954442...
1.919191919...	25.08077930...	52.23691137...	292.1374552...	190.9085229...
2.020202020...	24.94206332...	51.54309435...	291.7201248...	190.4207051...
2.121212121...	24.80703318...	50.86767396...	291.3024973...	189.9319980...
2.222222222...	24.67556259...	50.21002301...	290.8845795...	189.4424093...
2.323232323...	24.54753062...	49.56954071...	290.4663777...	188.9519466...
2.424242424...	24.42282142...	48.94565136...	290.0478983...	188.4606179...





Polytropic process $Pv^n = constant \quad -\infty \leq n \leq \infty$

$$\frac{P_1}{P_2} = \left(\frac{V_2}{V_1}\right)^n$$

$$\frac{T_1}{T_2} = \left(\frac{V_2}{V_1}\right)^{n-1}$$

$$\frac{P_1}{P_2} = \left(\frac{T_1}{T_2}\right)^{n/(n-1)}$$

Boundary work done by a polytropic process:

$$W = \frac{n}{1-n}(P_2V_2 - P_1V_1)$$

For a perfect gas:

$$W = \frac{n}{1-n}R(T_2 - T_1)$$

Example: Air is polytropically expanded through a reversible turbine from 1500 kPa to 100 kPa pressure. The inlet temperature is T=1200 K. The poytropic expansion coefficient n=1.5. Determine the power produced and the heat transfer.

$$P_1 = 1500\ kPa$$
$$T_1 = 1200\ K$$
$$P_2 = 100\ kPa$$
n=1.5

```
/*
air_PG property vector
  a[0]="P, pressure       ";      kPa
  a[1]="T, temperature    ";   degree K
  a[2]="v, specific volume ";   m^3/kg
  a[3]="h, enthalpy       ";      kJ/kg
  a[4]="u, internal energy ";   kJ/kg
```

```
   a[5]="s, entropy         ";       kJ/kgK
   a[6]="x, quality         ";       kg/kg total
   a[7]="Cp, specific heat at constant pressure ";  kJ/kgK
   a[8]="Cv, specific heat at constant volume ";     kJ/kgK
   a[9]="isobaric thermal expansion coefficient ";  1/K
   a[10]="isothermal compressibility" ; 1/kPa
   a[11]="Dynamic viscosity" ;  Pas
   a[12]="k Thermal conductivity" ; W/mK
   a[13]="Surface tension" ;  N/m
   a[14]="Prandtl number" ;
   a[15]="density" ;  kg/m^3
   a[16]="speed of sound" ; m/s
*/
public class polytropic1
{ air_PG refr;
  double TT1[];  //isentropic process temperature array
  double ss1[];    //isentropic processentropy array
  double TT2[];  //adiabatic process temperature array
  double ss2[];   //adiabatic process entropy array
  double s1,P1,T1,x1,h1;
  double s2i,P2i,T2i,x2i,h2i;
  double s2,P2,T2,x2,h2;
  double W;
  double Q;
  double eta_isentropic;
  double T0=273.15;  //degree K
  double n;  //Polytropic constant
  String refname;
public polytropic1()
{   refname="air";
           refr=new air_PG();}
//T1i,T2i degree K, P1i,P2i kPa
public void calculate1(double T1in,double P1in, double P2in,double ni)
{    n=ni;
     T1=T1in;
     P1=P1in;
     P2=P2in;
     double a1[]=refr.property("tp",T1,P1);
           s1=a1[5];
     h1=a1[3];
     x1=a1[6];
     T2=T1*Math.pow((P2/P1),(n-1)/n);
     double a2[]=refr.property("tp",T2,P2);
     s2=a2[5];
     h2=a2[3];
     x2=a2[6];
     W = n/(1-n)*refr.RL*1e-3/refr.M*(T2-T1);
     double dh=(h2-h1);
```

```
    System.out.println("dh="+dh);
    Q=(h2-h1)+W;
}


public String toString()
{ String s="Polytropic Process working fluid:"+refname+"\n";
 s+="P1= "+P1+" kPa "+"T1= "+T1+" degree K \n";
 s+="h1= "+h1+" kJ/kg "+"s1= "+s1+" kJ/kgK x1="+x1+" kg vapor/kg \n";
 s+="P2= "+P2+" kPa "+"T2= "+T2+" degree K \n";
 s+="h2= "+h2+" kJ/kg "+"s2= "+s2+" kJ/kgK x2="+x2+" kg vapor/kg \n";
 s+="W = "+W+" kJ/kg\n";
 s+="Q = "+Q+" kJ/kg\n";
 return s;
}
public static void main(String arg[])
{   double T1=1200.0; //degree K
    double P1=1500.0; //kPa
    double P2=100.0; //kPa
    double n=1.5;
     polytropic1 p1=new polytropic1();
     p1.calculate1(T1,P1,P2,n);
     System.out.println(p1);
}
}
```

```
---------- Capture Output ----------
> "C:\Users\Turhan
Çoban\.p2\pool\plugins\org.eclipse.justj.openjdk.hotspot.jre.full.win32.x86_64_19.0.2.v2
0230129-1123\jre\bin\javaw.exe" polytropic1
dh=-788.5921275547804
Polytropic Process working fluid:air
P1= 1500.0 kPa T1= 1200.0 degree K
h1= 1004.8893619454096 kJ/kg s1= 0.798453221234683 kJ/kgK x1=2.0 kg vapor/kg
P2= 100.0 kPa T2= 486.57615964587205 degree K
h2= 216.29723439062926 kJ/kg s2= 0.5883351014191621 kJ/kgK x2=2.0 kg vapor/kg
W = 614.3630712092403 kJ/kg
Q = -174.22905634554013 kJ/kg

> Terminated with exit code 0.
```

Pumps

$$\dot{m}\left[\frac{\left(V_{out}^2 - V_{in}^2\right)}{2} + g(z_{out} - z_{in}) + (h_{out} - h_{in})\right] = \dot{Q} - \dot{W}$$

A water pump delivers 1.0 m^3/min at P=150 kPa absulte pressure through $25x10^{-3}$ m diameter pipe. If the water inlet is at 100 kPA and temperature T=26.85 C and inlet velocity

is negligible. In addition heat transfer through the pump is negligible. The code difference between input and exit is 25 m. Determine the required power input.

$$\dot{m}\left[\frac{\left(V_{out}^2 - V_{in}^2\right)}{2} + g(z_{out} - z_{in}) + (u_{out} - u_{in}) + ((P_{out}v_{out} - P_{in}v_{in}))\right] = \dot{Q} - \dot{W}$$

Since there is no heat transfer, $T_{out} = T_{in} = 300\ K$

$$P_{in} = 100\ kPa$$
$$P_{out} = 150\ kPa$$
$$\dot{V}_{out} = \frac{1}{60}\ m^3/s$$
$$\dot{m} = \dot{V}_{out}\rho_{out} = \dot{V}_{out}/v_{out}$$

```
/*
steamIAPWS_IF97 property vector
   a[0]="P, pressure        ";       kPa
   a[1]="T, temperature     ";   degree K
   a[2]="v, specific volume ";   m^3/kg
   a[3]="h, enthalpy        ";       kJ/kg
   a[4]="u, internal energy ";   kJ/kg
   a[5]="s, entropy         ";       kJ/kgK
   a[6]="x, quality         ";         kg/kg total
   a[7]="Cp, specific heat at constant pressure ";  kJ/kgK
   a[8]="Cv, specific heat at constant volume ";     kJ/kgK
   a[9]="isobaric thermal expansion coefficient ";  1/K
   a[10]="isothermal compressibility" ; 1/kPa
   a[11]="Dynamic viscosity" ;  Pas
   a[12]="k Thermal conductivity" ; W/mK
   a[13]="Surface tension" ;  N/m
   a[14]="Prandtl number" ;
   a[15]="density" ;  kg/m^3
   a[16]="speed of sound" ; m/s
*/
public class pump3
{ steamIAPWS_IF97 steam;
  double s1,P1,T1,h1,ro1;
  double s2,P2,T2,h2,ro2;
  double V1,V2;
  double m;
  double V;
  double W;
  double Q;
  double T0=273.15;  //degree K
  double d;
  double dz;
  double A; //m^2
public pump3(double di)
```

```
{steam=new steamIAPWS_IF97();d=di;}
//T1i,T2i degree K, P1i,P2i kPa
public void calculate1(double T1in,double P1in,double T2in, double P2in,double
Vi,double dzi)
{    V=Vi; //m^3/s   volumetric flow rate
    dz=dzi;
    T1=T1in;  //inlet temp.
    P1=P1in;  //inlet pressure
    T2=T2in;  //exit temp.
    P2=P2in; //exit pressure
    double a1[]=steam.property("tp",T1,P1);
          s1=a1[5]; //inlet entropy
    h1=a1[3];  //inlet enthalpy
    ro1=a1[15]; //inlet density
    double a2[]=steam.property("tp",T2,P2);
    h2=a2[3]; //exit enthalpy
    s2=a2[5]; //exit entropy
    ro2=a2[15]; //exit density
    m=V*ro2;  //mass flow rate kg/s
    A=Math.PI*d*d/4.0;
    V1=0.0; //m/s
    V2=V/A;  //m/s exit velocity
    Q=0.0;
    double g=9.806; // m/s^2
    W=m*(V2*V2/2.0+g*dz+(h2-h1)*1e3)*1e-3;
}
```

```
---------- Capture Output ----------
> "C:\Users\Turhan
Çoban\.p2\pool\plugins\org.eclipse.justj.openjdk.hotspot.jre.full.win32.x86_64_19.0.2.v2
0230129-1123\jre\bin\javaw.exe" pump3
Pump working fluid:water
m = 16.60966468819861kg/s V=0.016666666666666666m^3/s
P1= 100.0 kPa T1= 300.0 degree K V1 = 0.0
h1= 112.66382328242584 kJ/kg s1= 0.3930970472619295 kJ/kgK
ro1=996.5574824996614 kg/m^3
P2= 150.0 kPa T2= 300.0 degree K V2 = 33.953054526271
h2= 112.70986527236666 kJ/kg s2= 0.3930832800396584 kJ/kgK
ro2=996.5798812919166 kg/m^3
W = 14.410494353791174 kW
Q = 0.0 kW

> Terminated with exit code 0.
```

A water pump delivers 1.0 m^3/min at P=150 kPa absulte pressure through $25x10^{-3}$ m diameter pipe. If the water inlet is at 100 kPA and temperature T=26.85 C and inlet velocity is negligible. Water exit temperature is T=27.0 C. The code difference between input and exit is 25 m. Determine the required power input.

```
 *
```

```
steamIAPWS_IF97 property vector
  a[0]="P, pressure        ";       kPa
  a[1]="T, temperature     ";   degree K
  a[2]="v, specific volume ";   m^3/kg
  a[3]="h, enthalpy        ";       kJ/kg
  a[4]="u, internal energy ";   kJ/kg
  a[5]="s, entropy         ";       kJ/kgK
  a[6]="x, quality         ";         kg/kg total
  a[7]="Cp, specific heat at constant pressure ";  kJ/kgK
  a[8]="Cv, specific heat at constant volume ";     kJ/kgK
  a[9]="isobaric thermal expansion coefficient ";  1/K
  a[10]="isothermal compressibility" ; 1/kPa
  a[11]="Dynamic viscosity" ;  Pas
  a[12]="k Thermal conductivity" ; W/mK
  a[13]="Surface tension" ;  N/m
  a[14]="Prandtl number" ;
  a[15]="density" ;  kg/m^3
  a[16]="speed of sound" ; m/s
*/
public class pump3
{ steamIAPWS_IF97 steam;
  double s1,P1,T1,h1,ro1;
  double s2,P2,T2,h2,ro2;
   double s3,P3,T3,h3,ro3;
  double V1,V2,V3;
  double m;
  double V;
  double W;
  double Q;
  double T0=273.15;  //degree K
  double d;
  double dz;
  double A; //m^2
public pump3(double di)
{steam=new steamIAPWS_IF97();d=di;}
//T1i,T2i degree K, P1i,P2i kPa
public void calculate1(double T1in,double P1in,double T2in, double P2in,double
Vi,double dzi)
{    V=Vi; //m^3/s   volumetric flow rate
     dz=dzi;
     T1=T1in;  //inlet temp.
     P1=P1in;  //inlet pressure
     T2=T2in;  //exit temp.
     P2=P2in; //exit pressure
     P3=P2;
     double a1[]=steam.property("tp",T1,P1);
            s1=a1[5]; //inlet entropy
     h1=a1[3];  //inlet enthalpy
```

```
    ro1=a1[15]; //inlet density
    double a2[]=steam.property("tp",T1,P2);
    h2=a2[3]; //exit enthalpy
    s2=a2[5]; //exit entropy
    ro2=a2[15]; //exit density
    double a3[]=steam.property("tp",T2,P2);
    h3=a3[3]; //exit enthalpy
    s3=a3[5]; //exit entropy
    ro3=a3[15]; //exit density
    m=V*ro3;  //mass flow rate kg/s
    A=Math.PI*d*d/4.0;
    V1=0.0; //m/s
    V2=V/A;  //m/s exit velocity
    V3=V2;
    double g=9.806; // m/s^2
    W=m*(V2*V2/2.0+g*dz+(h2-h1)*1e3)*1e-3;
    double WplusQ=m*(V2*V2/2.0+g*dz+(h3-h1)*1e3)*1e-3;
    System.out.println("WplusQ="+WplusQ);
    Q=WplusQ-W;
}


public String toString()
{ String s="Pump working fluid:water\n";
 s+="m = "+m+"kg/s V="+V+"m^3/s\n";
 s+="P1= "+P1+" kPa "+"T1= "+T1+" degree K V1 = "+V1+"\n";
 s+="h1= "+h1+" kJ/kg "+"s1= "+s1+" kJ/kgK ro1="+ro1+" kg/m^3 \n";
 s+="P2= "+P2+" kPa "+"T2= "+T2+" degree K V2 = "+V2+"\n";
 s+="h2= "+h2+" kJ/kg "+"s2= "+s2+" kJ/kgK  ro2="+ro2+" kg/m^3 \n";
s+="P3= "+P3+" kPa "+"T2= "+T3+" degree K V2 = "+V3+"\n";
 s+="h3= "+h3+" kJ/kg "+"s2= "+s3+" kJ/kgK  ro2="+ro3+" kg/m^3 \n";

   s+="W = "+W+" kW\n";
 s+="Q = "+Q+" kW";
 return s;
}
public static void main(String arg[])
{
    double T1=26.85+273.15; //degree K
    double P1=100.0; //kPa
    double P2=150.0; //kPa
    double T2=27.0+273.15;  //degree K
    double d=25.0e-3; //m
    double dz=25.0; //m
    double V=1.0/60.0; //m^3/s
   pump3 p3=new pump3(d);
   p3.calculate1(T1,P1,T2, P2,V,dz);
   System.out.println(p3);
```

```
}
}
```

```
---------- Capture Output ----------
> "C:\Users\Turhan
Çoban\.p2\pool\plugins\org.eclipse.justj.openjdk.hotspot.jre.full.win32.x86_64_19.0.2.v2
0230129-1123\jre\bin\javaw.exe" pump3
WplusQ=24.826048214251326
Pump working fluid:water
m = 16.608979251416198kg/s V=0.016666666666666666m^3/s
P1= 100.0 kPa T1= 300.0 degree K V1 = 0.0
h1= 112.66382328242584 kJ/kg s1= 0.3930970472619295 kJ/kgK
ro1=996.5574824996614 kg/m^3
P2= 150.0 kPa T2= 300.15 degree K V2 = 33.953054526271
h2= 112.70986527236666 kJ/kg s2= 0.3930832800396584 kJ/kgK
ro2=996.5798812919166 kg/m^3
P3= 150.0 kPa T2= 0.0 degree K V2 = 33.953054526271
h3= 113.33700486870464 kJ/kg s2= 0.3951732229198204 kJ/kgK
ro2=996.5387550849719 kg/m^3
W = 14.409899670932232 kW
Q = 10.416148543319094 kW

> Terminated with exit code 0.
```

Expansion valve

$$\dot{m}\left[\frac{\left(V_{out}^2 - V_{in}^2\right)}{2} + g(z_{out} - z_{in}) + (h_{out} - h_{in})\right] = \dot{Q} - \dot{W}$$

$$(h_{out} - h_{in}) = 0 \quad (if\ there\ is\ no\ heat\ transfer$$

$$(h_{out} - h_{in}) = Q \quad if\ there\ is\ heat\ transfer$$

R134 a expands through a T-x type expansion valve from P=1500 kPa x=0 to P=100 kPa. Calculate exit temperature.

```
/*
refrigerant_coolProp property vector
  a[0]="P, pressure        ";      kPa
  a[1]="T, temperature     ";   degree K
  a[2]="v, specific volume ";   m^3/kg
  a[3]="h, enthalpy        ";      kJ/kg
  a[4]="u, internal energy ";   kJ/kg
  a[5]="s, entropy         ";      kJ/kgK
  a[6]="x, quality         ";        kg/kg total
  a[7]="Cp, specific heat at constant pressure ";  kJ/kgK
  a[8]="Cv, specific heat at constant volume ";     kJ/kgK
  a[9]="isobaric thermal expansion coefficient ";  1/K
  a[10]="isothermal compressibility" ; 1/kPa
  a[11]="Dynamic viscosity" ;  Pas
  a[12]="k Thermal conductivity" ; W/mK
```

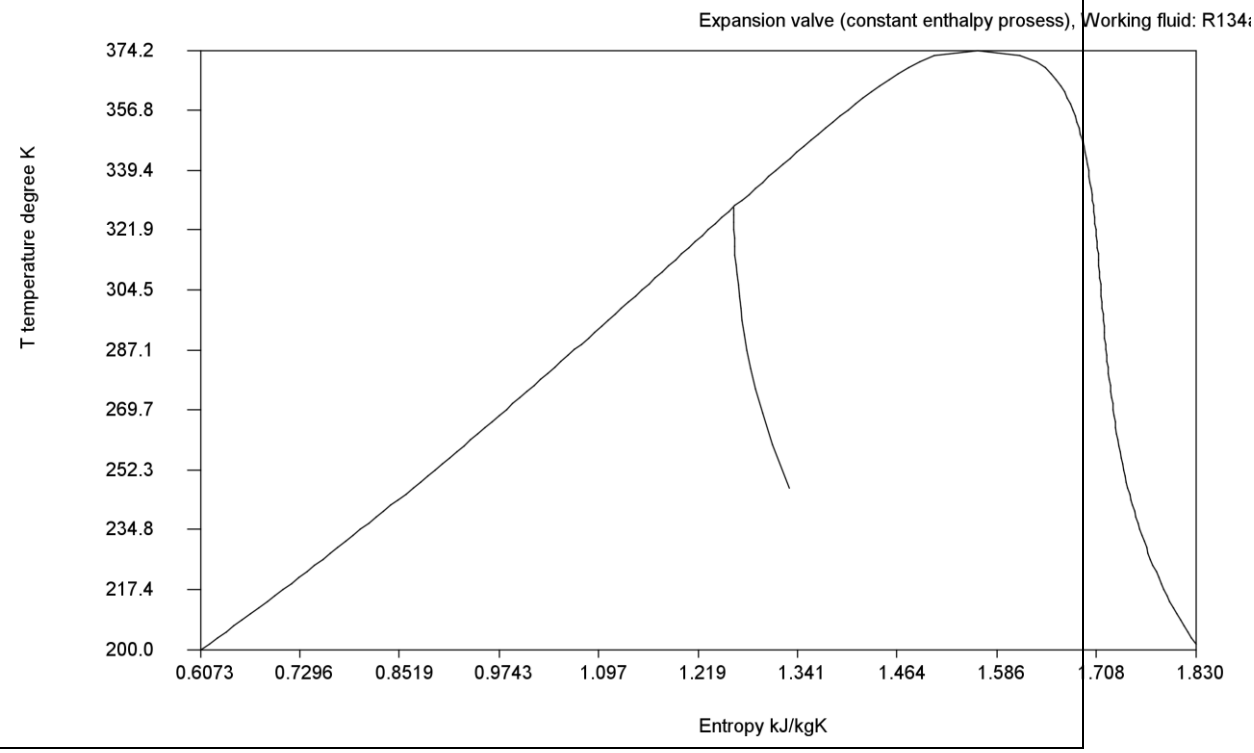
```
   a[13]="Surface tension" ;  N/m
   a[14]="Prandtl number" ;
   a[15]="density" ;  kg/m^3
   a[16]="speed of sound" ; m/s
*/
public class expansion_valve1
{ refrigerant_coolProp refr;
  double TT1[];  //h=constant process temperature array
  double ss1[];    //h=constant process entropy array
  double s1,P1,T1,x1,h1;
  double s2,P2,T2,x2,h2;
  double T0=273.15;  //degree K
  String refname;
public expansion_valve1(String refnamei)
{   refname=refnamei;
          refr=new refrigerant_coolProp(refname);}
//T1i,T2i degree K, P1i,P2i kPa
public void calculate1(double P1in,double x1i,double P2in)
{
    x1=x1i;
    P1=P1in;
    P2=P2in;
    double a1[]=refr.property("px",P1,x1);
    T1=a1[1];
          s1=a1[5];
    h1=a1[3];
    h2=h1;
    double a2[]=refr.property("ph",P2,h2);
    T2=a2[1];
    x2=a2[6];
    s2=a2[5];
}
public void plot1()
{
    Plot p=new Plot(refr.TS_data(200.0));
     p.setPlabel("Expansion valve (constant enthalpy prosess), Working fluid: "+refname);
    p.setXlabel("Entropy kJ/kgK");
    p.setYlabel("T temperature degree K");
    double ss2[]=new double[20];
    double TT2[]=new double[20];
    ss2[0]=s1;
    TT2[0]=T1;
    double dP=(P1-P2)/19;
    double PP=P1;
    for(int i=1;i<20;i++)
    {PP=PP-dP;
      double ai[]=refr.property("ph",PP,h1);
      ss2[i]=ai[5];
```

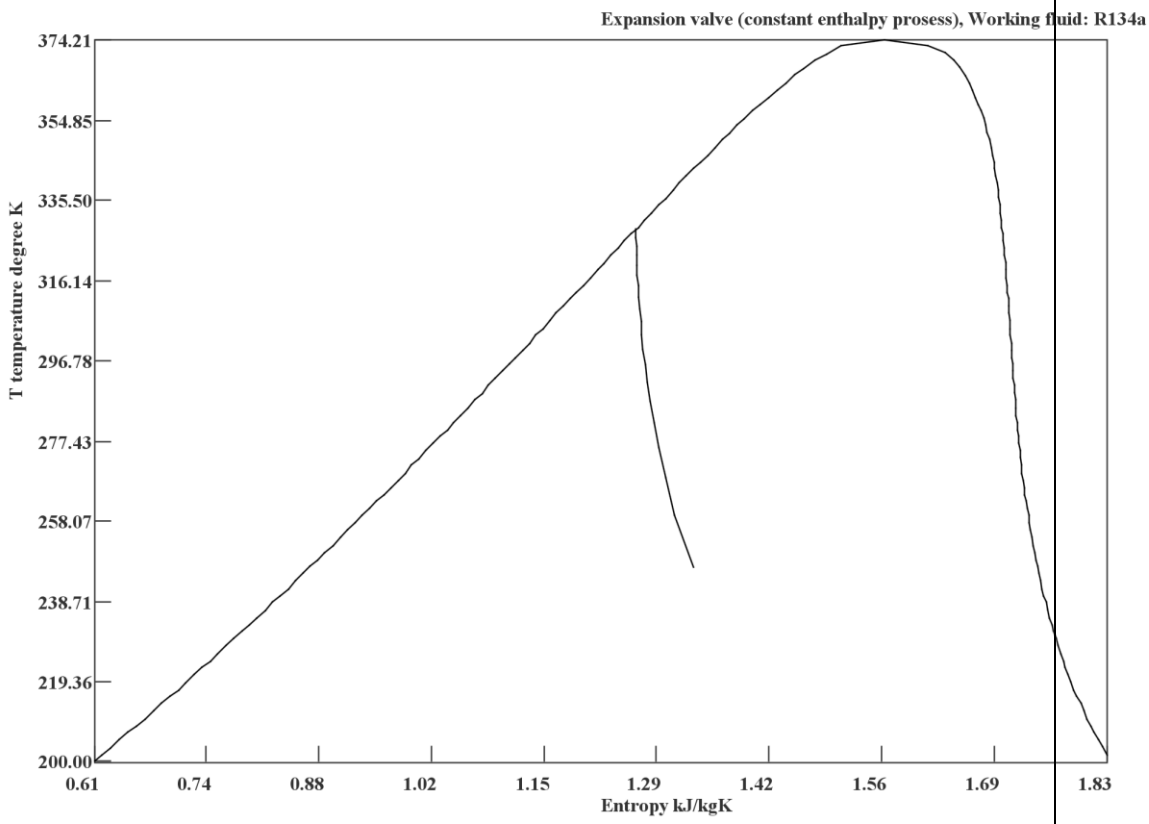
```
        TT2[i]=ai[1];
      }
       p.addData(ss2,TT2);
      p.plot();
}
public void plot2()
{
     graph2D p=new graph2D();
     p.addData(refr.TS_data(200.0));
     p.setPlabel("Expansion valve (constant enthalpy prosess), Working fluid: "+refname);
     p.setXlabel("Entropy kJ/kgK");
     p.setYlabel("T temperature degree K");
     double ss2[]=new double[20];
     double TT2[]=new double[20];
      ss2[0]=s1;
     TT2[0]=T1;
     double dP=(P1-P2)/19;
     double PP=P1;
     for(int i=1;i<20;i++)
     {PP=PP-dP;
        double ai[]=refr.property("ph",PP,h1);
        ss2[i]=ai[5];
        TT2[i]=ai[1];
     }
      p.addData(ss2,TT2);
     p.plot();
}


public String toString()
{ String s="Expansion valve (constant enthalpy prosess), Working fluid: "+refname+"\n";
  s+="P1= "+P1+" kPa "+"T1= "+T1+" degree K \n";
  s+="h1= "+h1+" kJ/kg "+"s1= "+s1+" kJ/kgK x1="+x1+" kg vapor/kg \n";
  s+="P2= "+P2+" kPa "+"T2= "+T2+" degree K \n";
  s+="h2= "+h2+" kJ/kg "+"s2= "+s2+" kJ/kgK x2="+x2+" kg vapor/kg \n";
  return s;
}
public static void main(String arg[])
{
     double P1=1500.0; //kPa
     double x1=0.0;
     double P2=100.0; //kPa
      expansion_valve1 ev1=new expansion_valve1("R134a");
      ev1.calculate1(P1,x1,P2);;
      System.out.println(ev1);
       ev1.plot1();
       ev1.plot2();
}
```

}

---------- Capture Output ----------
> "C:\Users\Turhan Çoban\.p2\pool\plugins\org.eclipse.justj.openjdk.hotspot.jre.full.win32.x86_64_19.0.2.v20230129-1123\jre\bin\javaw.exe" expansion_valve1
Expansion valve (constant enthalpy prosess), Working fluid: R134a
P1= 1500.0 kPa T1= 328.38293754720064 degree K
h1= 279.8389668131603 kJ/kg s1= 1.262160368167155 kJ/kgK x1=0.0 kg vapor/kg
P2= 100.0 kPa T2= 246.78881174998173 degree K
h2= 279.8389668131603 kJ/kg s2= 1.331103563055461 kJ/kgK
x2=0.5267935536830876 kg vapor/kg





R134a in condition P=2000 kPa and dryness fraction x=0 expands in an T-x expansion valve to P= P=200 kPa. Assume that expansion valve is a cylinder D=0.05 m and L=0.15 m, and environmental condition is T=40 °C., P=100 kPa and natural convective heat loses is existed from expansion valve to environment , Plot the process in T-x diagram

```
/*
refrigerant_coolProp property vector
  a[0]="P, pressure        ";       kPa
  a[1]="T, temperature     ";   degree K
  a[2]="v, specific volume ";   m^3/kg
  a[3]="h, enthalpy        ";       kJ/kg
  a[4]="u, internal energy ";   kJ/kg
  a[5]="s, entropy         ";       kJ/kgK
  a[6]="x, quality         ";         kg/kg total
  a[7]="Cp, specific heat at constant pressure ";  kJ/kgK
  a[8]="Cv, specific heat at constant volume ";     kJ/kgK
  a[9]="isobaric thermal expansion coefficient ";  1/K
  a[10]="isothermal compressibility" ; 1/kPa
  a[11]="Dynamic viscosity" ;  Pas
  a[12]="k Thermal conductivity" ; W/mK
  a[13]="Surface tension" ;  N/m
  a[14]="Prandtl number" ;
  a[15]="density" ;  kg/m^3
  a[16]="speed of sound" ; m/s
*/
public class expansion_valve2
```

```
{ refrigerant_coolProp refr;
  air_PG air;
  double TT1[];
  double hh1[]; //enthalpy change with heat trnsfer
  double ss1[]; //entropy change with heat transfer
  double ss1h[];  //constant enthalpy
  double s1,P1,T1,x1,h1;
  double s2,P2,T2,x2,h2;
  double T0=273.15;  //degree K
  String refname;
  double Q,dQ;
  double Tinf,Pinf;
  double D; //diameter of expansion valve
  double A,dA;
  double L;  //length of expansion valve
  //Note expansion valve is assumed to be a cylinder
  int n;  //number of finite difference
public expansion_valve2(String refnamei,double Tinfi, double Pinfi,double Di,double Li)
{   refname=refnamei;
    air=new air_PG();
    Tinf=Tinfi;
    Pinf=Pinfi;
    D=Di;
    L=Li;
          refr=new refrigerant_coolProp(refname);
          air=new air_PG();
          A=Math.PI*D*L;
          n=20;
          hh1=new double[n];
          ss1=new double[n];
          ss1h=new double[n];
   TT1=new double[n];
          }
//T1i,T2i degree K, P1i,P2i kPa
  public  double h(double Ts,double Tinf,double P,double D,int select)
  {//Nusselt number and h convective heat transfer coefficient
   double T=(Tinf+Ts)/2.0;
   double a[]=air.property("tp",T,P);
   double Ra=Ra(Ts,Tinf,P,D);
   double Pr=a[14];
   double k=a[12];
   double Gr=Ra/Pr;
   double Nu=0.0;
   double Racritic=1e9;
   double C=0,n=0;
   double Pr_05=Math.sqrt(Pr);

System.out.println("T="+T+"Pr="+Pr+"k="+k+"Gr="+Gr+"Ra="+Ra+"Pr_05="+Pr_05);
   if(select==1) {
             double x1=0.387*Math.pow(Ra,(1.0/6.0));

Nu=(0.60+0.387*Math.pow(Ra,(1.0/6.0))/Math.pow((1+Math.pow((0.559/Pr),(9.0/16.0)))
,(8.0/27.0)));
      Nu=Nu*Nu;
   }
```

```
   else if(select==2)
   {  if(Ra>=1e-10 && Ra<1e-2)    {C=0.675;n=0.058;}
      else if(Ra>=1e-2 && Ra<1e2) {C=1.02;n=0.148;}
      else if(Ra>=1e2 && Ra<1e4) {C=0.85;n=0.188;}
      else if(Ra>=1e4 && Ra<1e7)  {C=0.480;n=0.250;}
      else if (Ra>=1e7 && Ra<=1e12){C=0.125;n=0.333;}
      System.out.println("Ra="+Ra);
            Nu=C*Math.pow(Ra,n);
   }
   double h=Nu*k/D;
   return h;
   }

  public  double Ra(double Ts,double Tinf,double P,double D)
  {   double T=(Ts+Tinf)/2.0;
      System.out.println("T="+T+"P="+P+"D="+D);
     double a[]=air.property("tp",T,P);
            double mu=a[11];
      double ro=a[15];
      double Cp=a[7]*1e3;;
      double k=a[12];
      double beta=a[9];
      double nu=mu/ro;
      double alpha=k/(ro*Cp);
      double g=9.806; //m/s^2
      double Ra=g*beta*(Tinf-Ts)*D*D*D/alpha/nu;
            return Ra;
  }

public void calculate1(double P1in,double x1i,double P2in)
{
     x1=x1i;
     P1=P1in;
      double PP=P1;
     P2=P2in;
     double a1[]=refr.property("px",P1,x1);
     TT1[0]=a1[1];
           ss1[0]=a1[5];
           ss1h[0]=a1[5];
     hh1[0]=a1[3];
     double Ts=(T1+Tinf)/2.0;
     double hh=h(Ts,Tinf,Pinf,D,1);
     dA=A/n;
     Q=hh *A*(Tinf-Ts)*1e-3;
     dQ=Q/n;
     System.out.println("Q="+Q+" dQ="+dQ);
      double dP=(P1-P2)/19;
     for(int i=1;i<20;i++)
     { hh1[i]= hh1[i-1]+dQ;
       PP=PP-dP;
        double ai[]=refr.property("ph",PP,hh1[0]);
        ss1h[i]=ai[5];
         double aj[]=refr.property("ph",PP,hh1[i]);
         TT1[i]=aj[1];
         ss1[i]=aj[5];
```

```
System.out.println("i="+i+"h1="+hh1[0]+"hh1[i]="+hh1[i]+"TT1[i]="+TT1[i]+"ss1[i]="
+ss1[i]);
    }
}
public void plot1()
{
    Plot p=new Plot(refr.TS_data(200.0));
     p.setPlabel("Expansion valve (constant enthalpy prosess), Working fluid: "+refname);
    p.setXlabel("Entropy kJ/kgK");
    p.setYlabel("T temperature degree K");
    p.addData(ss1h,TT1);
     p.addData(ss1,TT1);
    p.plot();
}
public void plot2()
{
    graph2D p=new graph2D();
    p.addData(refr.TS_data(200.0));
     p.setPlabel("Expansion valve (natural convection on valve), Working fluid:
"+refname);
    p.setXlabel("Entropy kJ/kgK");
    p.setYlabel("T temperature degree K");
    p.addData(ss1h,TT1);
     p.addData(ss1,TT1);
    p.plot();
}


public static void main(String arg[])
{
    double P1=2000.0; //kPa
    double x1=0.0;
    double P2=200.0; //kPa
    double Tinfi=40.0+273.15;
    double Pinfi=100.0;
    double Di=0.05;
    double Li=0.15;
     expansion_valve2 ev1=new expansion_valve2("R134a",Tinfi, Pinfi,Di,Li);
     ev1.calculate1(P1,x1,P2);
      ev1.plot1();
      ev1.plot2();
}
}
```

```
---------- Capture Output ----------
> "C:\Users\Turhan
Çoban\.p2\pool\plugins\org.eclipse.justj.openjdk.hotspot.jre.full.win32.x86_64_19.0.2.v2
0230129-1123\jre\bin\javaw.exe" expansion_valve2
T=234.86249999999998P=100.0D=0.05
T=234.86249999999998Pr=0.7251745050223167k=0.021189555508420804Gr=7726187.
149918998Ra=5602833.942152293Pr_05=0.8515717850083554
Q=0.03736571565922021 dQ=0.0018682857829610104
i=1h1=299.9529698356858hh1[i]=299.95483812146875TT1[i]=338.5041860852862ss1[i
]=1.3211665846946032
```

i=2h1=299.9529698356858hh1[i]=299.9567064072517TT1[i]=336.29578940661366ss1[i]=1.3215505126830671
i=3h1=299.9529698356858hh1[i]=299.9585746930347TT1[i]=333.9981197547427ss1[i]=1.3220152306255983
i=4h1=299.9529698356858hh1[i]=299.96044297881764TT1[i]=331.60256113468625ss1[i]=1.3225725064948375
i=5h1=299.9529698356858hh1[i]=299.9623112646006TT1[i]=329.0990575744597ss1[i]=1.3232364716062455
i=6h1=299.9529698356858hh1[i]=299.96417955038356TT1[i]=326.47576744789444ss1[i]=1.3240242694474624
i=7h1=299.9529698356858hh1[i]=299.9660478361665TT1[i]=323.7186050245048ss1[i]=1.3249569376446866
i=8h1=299.9529698356858hh1[i]=299.9679161219495TT1[i]=320.81062149765853ss1[i]=1.3260606298535011
i=9h1=299.9529698356858hh1[i]=299.96978440773245TT1[i]=317.7311521125837ss1[i]=1.327368345563831
i=10h1=299.9529698356858hh1[i]=299.9716526935154TT1[i]=314.45461323244075ss1[i]=1.3289224403879984
i=11h1=299.9529698356858hh1[i]=299.97352097929837TT1[i]=310.9487590539954ss1[i]=1.330778375119787
i=12h1=299.9529698356858hh1[i]=299.97538926508133TT1[i]=307.1720735760538ss1[i]=1.3330105067653149
i=13h1=299.9529698356858hh1[i]=299.9772575508643TT1[i]=303.069718125528ss1[i]=1.3357213999861302
i=14h1=299.9529698356858hh1[i]=299.97912583664726TT1[i]=298.56693821110065ss1[i]=1.3390575454304283
i=15h1=299.9529698356858hh1[i]=299.9809941224302TT1[i]=293.5577076652213ss1[i]=1.3432375428534316
i=16h1=299.9529698356858hh1[i]=299.9828624082132TT1[i]=287.8836944179526ss1[i]=1.348606675564577
i=17h1=299.9529698356858hh1[i]=299.98473069399614TT1[i]=281.29135781839284ss1[i]=1.3557539410285167
i=18h1=299.9529698356858hh1[i]=299.9865989797791TT1[i]=273.3318089752556ss1[i]=1.3658015798982568
i=19h1=299.9529698356858hh1[i]=299.98846726556206TT1[i]=263.07372753976694ss1[i]=1.3812963392934419

Nozzle

$$\dot{m}\left[\frac{\left(V_{out}^2 - V_{in}^2\right)}{2} + g(z_{out} - z_{in}) + (h_{out} - h_{in})\right] = \dot{Q} - \dot{W}$$

$$\dot{Q} = 0$$
$$\dot{W} = 0$$
$$z_{out} = z_{in}$$

$$m = \rho_{in} V_{in} A_{in} = \rho_{out} V_{out} A_{out}$$

Air flow through the converging nozzle with an inlet velocity of 240 m/s. The density of air is found to be 0.48 kg/m^3 at inlet and 1.12 kg/m^3 at exit. If the area of the nozzle in the inlet is 0.1 m^2 and at the exit 0.05 m^2, determine

a) Mass flow rate of the air through the nozzle
b) Velocity of air at the exit

$$V_1 = 240\ m/s$$

$$\rho_1 = 0.48\ kg/m^3$$
$$\rho_2 = 1.12\ kg/m^3$$
$$A_1 = 0.1\ m^2$$
$$A_2 = 0.05\ m^2$$

```
/*
air_PG property vector
  a[0]="P, pressure       ";      kPa
  a[1]="T, temperature    ";   degree K
  a[2]="v, specific volume ";   m^3/kg
  a[3]="h, enthalpy       ";      kJ/kg
  a[4]="u, internal energy ";   kJ/kg
  a[5]="s, entropy        ";      kJ/kgK
  a[6]="x, quality        ";       kg/kg total
  a[7]="Cp, specific heat at constant pressure ";  kJ/kgK
  a[8]="Cv, specific heat at constant volume ";     kJ/kgK
  a[9]="isobaric thermal expansion coefficient ";  1/K
  a[10]="isothermal compressibility" ; 1/kPa
  a[11]="Dynamic viscosity" ;  Pas
  a[12]="k Thermal conductivity" ; W/mK
  a[13]="Surface tension" ;  N/m
  a[14]="Prandtl number" ;
  a[15]="density" ;  kg/m^3
  a[16]="speed of sound" ; m/s
*/
public class nozzle1
{ air_PG refr;
 double TT1[];  //isentropic process temperature array
 double ss1[];    //isentropic processentropy array
 double TT2[];  //adiabatic process temperature array
 double ss2[];   //adiabatic process entropy array
 double s1,P1,T1,x1,h1,ro1;
 double s2,P2,T2,x2,h2,ro2;
 double A1,A2;
 double m; //kg/s
 double V1,V2;
 double T0=273.15;  //degree K
 String refname;
public nozzle1()
{   refname="air";
          refr=new air_PG();}
//T1i,T2i degree K, P1i,P2i kPa
public void calculate1(double V1in,double ro1in,double ro2in,double A1in,double A2in)
{   ro1=ro1in;
    ro2=ro2in;
    V1=V1in;
    A1=A1in;
```

```
    A2=A2in;
    m=ro1*A1*V1; //kg/s
    V2=m/(ro2*A2);
}

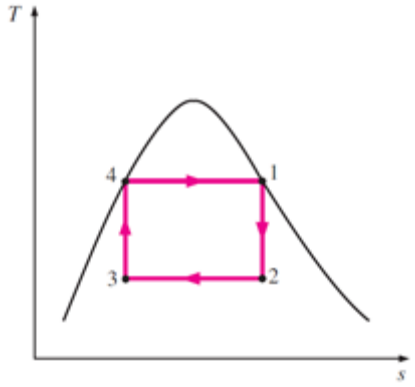
public String toString()
{ String s="nozzle working fluid :"+refname+"\n";
s+="m="+m+" kg/s \n";
  s+="A1= "+A1+" m^2a "+"ro1= "+ro1+" kg/m^3 V1= "+V1+" m/s\n";
 s+="A2= "+A2+" m^2a "+"ro2= "+ro2+" kg/m^3 V2= "+V2+" m/s\n";
  return s;
}
public static void main(String arg[])
{   double V1=240.0; //m/s
    double ro1=0.48; //kg/m^3
    double ro2=1.12; //kg/m^3
    double A1=0.1; //m^2
    double A2=0.05; //m^2
    nozzle1 n1=new nozzle1();
    n1.calculate1(V1,ro1,ro2,A1,A2);
    System.out.println(n1);
}
}
```

```
---------- Capture Output ----------
> "C:\Users\Turhan
Çoban\.p2\pool\plugins\org.eclipse.justj.openjdk.hotspot.jre.full.win32.x86_64_19.0.2.v2
0230129-1123\jre\bin\javaw.exe" nozzle1
nozzle working fluid :air
m=11.52 kg/s
A1= 0.1 m^2a ro1= 0.48 kg/m^3 V1= 240.0 m/s
A2= 0.05 m^2a ro2= 1.12 kg/m^3 V2= 205.71428571428567 m/s

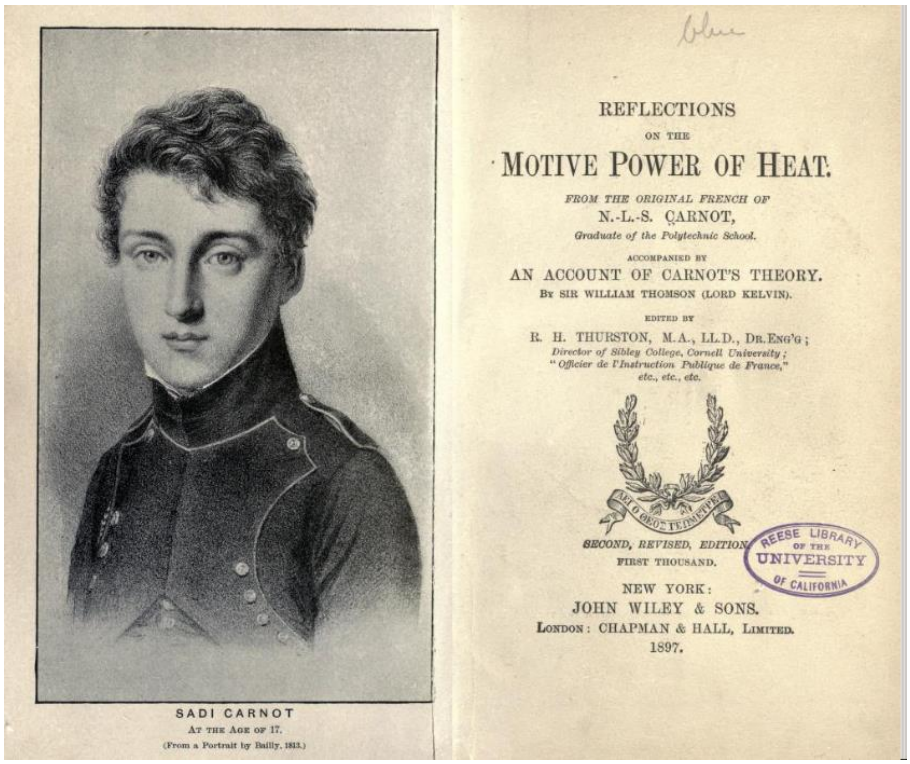

> Terminated with exit code 0.
```

Now that equation of states information and information about processes are established, actual power cycles can be investigated. Cycles are combinations of processes that ended up in the beginning point. If total cycle direction is clockwise, this kind of cycles are called power cycles.

3.1 CARNOT CYCLE



Carnot cycle is an ideal cycle and it is important as a reference cycle to give maximum possible cycle efficincy. It consists of two isothermal and two isentropic processes. If a carnot cycle constructed by using a boiler (evaporator), an isentropic turbine, a condenser and an isentropic compressor,



Nicolas Leonard Sadi Carnot

$$W_{isentropic\ compressor} = m(h_4 - h_3) \qquad s_4 = s_3 \qquad (3.1.1)$$

$$Q_{boiler} = m(h_1 - h_4) = mT_1(s_1 - s_4) \qquad (3.1.2)$$

$$Q_{condenser} = m(h_3 - h_2) = mT_2(s_2 - s_3) = mT_2(s_1 - s_4) \qquad (3.1.3)$$

$$W_{isentropic\ turbine} = m(h_1 - h_2) \qquad s_1 = s_2 \qquad (3.1.4)$$

$$\eta_{cycle} = \frac{W_{turbine} - W_{compressor}}{Q_{boiler}} = \frac{Q_{boiler} - Q_{condenser}}{Q_{boiler}} = \frac{mT_1(s_1 - s_4) - mT_2(s_1 - s_4)}{mT_1(s_1 - s_4)}$$

$$\eta_{cycle} = 1 - \frac{T_2}{T_1} \qquad (3.1.5)$$

A computer model is developed to analyse this cycle in details

Program 3.1.1 Carnot.java Carnot's cycle with steam

```
public class carnot
{
// a steam ideal carnot cycle
// steam b=new steam();
// double a[]=b.property("tp",30.0,1.01325);
// girdi parametreleri
// termodinamik set "tv" "tp" "th" "tu" "ts"tx"
//                  "pv" "pt" "ph" "pu" "ps" "px"
// çıkış parametreleri(a)
// a[0]=P a[1]=T a[2]=V a[3]=h
// a[4]u a[5]=s a[6]=x
double T[];
double P[];
double h[];
double s[];
double x[];
double etait,etaip;
double etacevrim,etacevrim1,etaisi,etatoplam;
double m;//kg/s
double Wt,Wp,W;//iş
double Qkazan,Qyogusturucu;
steam b;
double c1[][];
double c3[][];
public carnot(double mi,double P1, double P2)
{
m=mi;
b=new steam();
T=new double[9];
P=new double[9];
h=new double[9];
s=new double[9];
x=new double[9];

P[1]=P1;//turbine inlet
P[2]=P2;//turbine outlet
c1=new double[2][20];
c3=new double[2][20];
}
public void cycle()
{
// Turbine input
double a1[]=b.property("px",P[1],1.0);
h[1]=a1[3];
s[1]=a1[5];
T[1]=a1[1];
// isentropik turbine output
s[2]=s[1];
P[2]=P[2];
double a2[]=b.property("ps",P[2],s[2]);
T[2]=a2[1];
h[2]=a2[3];
//compressor output
P[4]=P[1];
double a4[]=b.property("px",P[4],0.0);
h[4]=a4[3];
s[4]=a4[5];
```

```
T[4]=a4[1];
// condenser output
P[3]=P[2];
s[3]=s[4];
double a3[]=b.property("ps",P[3],s[3]);
h[3]=a3[3];
s[3]=a3[5];
T[3]=a3[1];
P[3]=P[4];
x[3]=a3[6];


// isentropik pompa
Wt=m*(h[1]-h[2]);
Wp=m*(h[4]-h[3]);
W=Wt-Wp;
Qkazan=m*(h[1]-h[4]);
Qyogusturucu=m*(h[2]-h[3]);
etacevrim=W/Qkazan;
etacevrim1=1.0-(T[2]+273.15)/(T[1]+273.15);
etaisi=Qyogusturucu/Qkazan;
etatoplam=etacevrim+etaisi;
}

public String toString()
{
cycle();
String ss="steam carnot cycle\n";
ss+=" cycle efficiency by using eta=1-Tlow/Thigh = "+etacevrim1+"\n";
ss+=" cycle efficiency    = " + etacevrim+"\n";
ss+=" heat energy utilisation rate = "+etaisi+"\n";
ss+=" total heat+power efficiency = "+etatoplam+"\n";
ss+=" Turbine power output    = " + Wt+" kW\n";
ss+=" Net power output    = " + W +" kW\n";
ss+=" Boiler heat input = " + Qkazan +" kW\n";
ss+=" Condenser heat  = "+Qyogusturucu +" kW\n";
ss+=" h1 turbine input = "+h[1] +" kJ/kg\n";
ss+=" T1 turbine input = "+T[1] +" derece C\n";
ss+=" P1 turbine input = "+P[1] +" bar \n";
ss+=" s1 turbine input = "+s[1] +" kJ/kgK\n";
ss+=" x1 turbine input = "+x[1] +" kgvapor/kgtotal"+"\n";
ss+=" T2 turbine output = "+T[3] +" derece C\n";
ss+=" P2 turbine output = "+P[3] +" bar \n";
ss+=" s2 turbine output = "+s[2] +" kJ/kgK\n";
ss+=" x2 turbine output = "+x[2] +" kgvapor/kgtotal"+"\n";
ss+=" h3 condenser output = "+h[3] +" kJ/kg\n";
ss+=" T3 condenser output = "+T[3] +" derece C\n";
ss+=" P3 condenser output = "+P[3] +" bar \n";
ss+=" s3 condenser output = "+s[3] +" kJ/kgK\n";
ss+=" x3 condenser output = "+x[3] +" kgvapor/kgtotal"+"\n";
ss+=" h4 pump output = "+h[4] +" kJ/kg\n";
ss+=" T4 pump output = "+T[4] +" derece C\n";
ss+=" s4 pump output = "+s[4] +" kJ/kgK\n";
ss+=" x4 pump output = "+x[4] +" kgvapor/kgtotal"+"\n";
return ss;
}
public double[][] TS1()
{
```

```
double a[][]=Text.readDoubleT("steam.txt");
return a;
}

public double[][] TS()
{
double a[][]=new double[2][201];
double tc=b.Tc-273.15;
double dt=(tc-10.0)/100.0;
int i=0;
double t;
for(t=10.0;t<=tc;t+=dt)
{double a1[]=b.property("tx",t,0.0);
 a[1][i]=t;
 a[0][i]=a1[5];
 i++;
}
double a3[]=b.property("tx",tc,1.0);
 a[1][i]=t;
 a[0][i]=a3[5];
 i++;
for(t=tc;t>=10;t-=dt)
{double a2[]=b.property("tx",t,1.0);
 a[1][i]=t;
 a[0][i]=a2[5];
 i++;
}
return a;
}

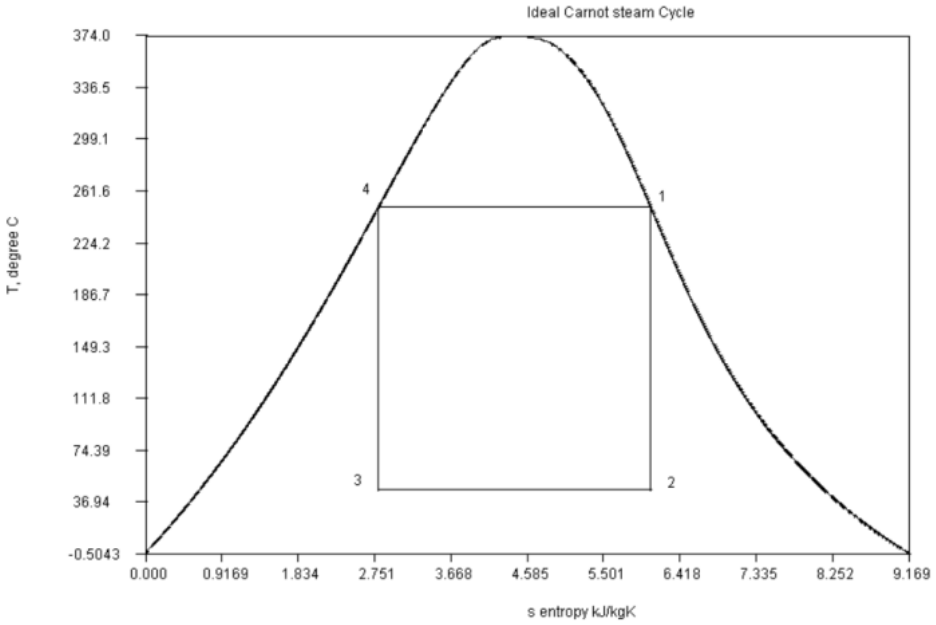
public void plot()
{
 Plot pp=new Plot(TS1());
 double t1[]={T[1],T[2],T[3],T[4],T[1]};
 double s1[]={s[1],s[2],s[3],s[4],s[1]};
 pp.setPlabel("Ideal Carnot steam Cycle");
 pp.setYlabel("T, degree C");
 pp.setXlabel("s entropy kJ/kgK");
 pp.addData(s1,t1);
 pp.plot();
}
}
```

Program 3.1.2 carnottest.java Carnot's cycle with steam test program

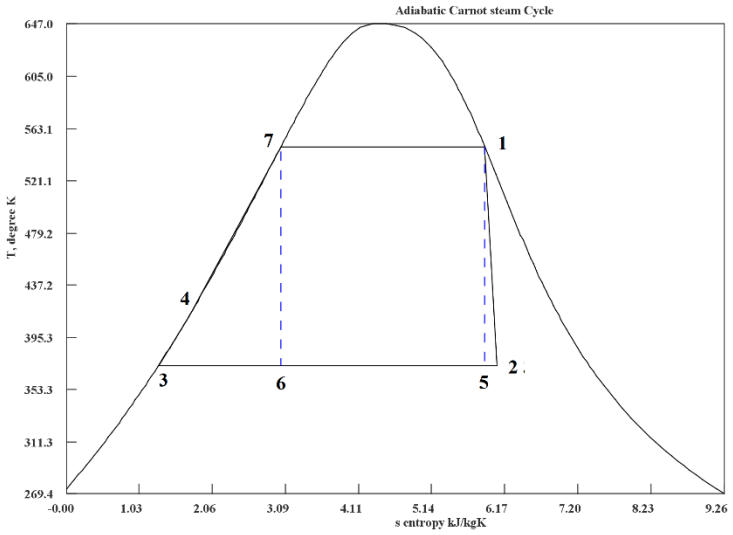
```
import javax.swing.*;
public class carnottest
{public static void main(String arg[])
{double m=50/745.25928;  // kg/s mass flow rate
double P1=40.0;         //  bar turbine inlet pressure
double P2=0.1 ;         //  bar turbine exit pressure
carnot r=new carnot(m,P1,P2);
System.out.println(r.toString());
r.plot();
}}
```

```
---------- Capture Output ----------
> "C:\java\bin\javaw.exe" carnottest
steam carnot cycle
```

cycle efficiency by using eta=1-Tlow/Thigh = 0.39077855463263766
cycle efficiency = 0.3907418195973338
heat energy utilisation rate = 0.6092581804026662
total heat+power efficiency = 1.0
Turbine power output = 59.06120727009282 kW
Net power output = 44.93508288571553 kW
Boiler heat input = 114.9994206712297 kW
Condenser heat = 70.06433778551417 kW
h1 turbine input = 2801.4126393729175 kJ/kg
T1 turbine input = 250.40495633511458 derece C
P1 turbine input = 40.0 bar
s1 turbine input = 6.07012036785386 kJ/kgK
x1 turbine input = 0.0 kgvapor/kgtotal
T2 turbine output = 45.81090722772479 derece C
P2 turbine output = 40.0 bar
s2 turbine output = 6.07012036785386 kJ/kgK
x2 turbine output = 0.0 kgvapor/kgtotal
h3 condenser output = 876.7724246179332 kJ/kg
T3 condenser output = 45.81090722772479 derece C
P3 condenser output = 40.0 bar
s3 condenser output = 2.796412171681691 kJ/kgK
x3 condenser output = 0.28625026940019205 kgvapor/kgtotal
h4 pump output = 1087.3249303757625 kJ/kg
T4 pump output = 250.40495633511458 derece C
s4 pump output = 2.796412171681691 kJ/kgK
x4 pump output = 0.0 kgvapor/kgtotal



If adiabatic processes replaced isentropic turbine and compressor process in Carnot cycle, efficiency will be less than carnot cycle. Let us call this as adiabatic carnot cycle and investigate how processes occurs:



$W_{isentropic\ compressor} = m(h_7 - h_6)$ $\quad s_7 = s_6$

$h_4 = h_3 + (h_7 - h_6)/\eta_{compressor}$

$x_3 = 0$

$W_{adiabtic\ compressor} = m(h_4 - h_3)$

$h_2 = h_1 - \eta_{turbine}(h_1 - h_5)$

$Q_{boiler} = m(h_1 - h_4) = mT_1(s_1 - s_4)$

$Q_{condenser} = m(h_3 - h_2)$

$W_{isentropic\ turbine} = m(h_1 - h_5)$ $\quad s_1 = s_5$

$h_2 = h_1 - \eta_{turbine}(h_1 - h_5)$

$W_{adiabatic\ turbine} = m(h_1 - h_2)$

$$\eta_{cycle} = \frac{W_{turbine} - W_{compressor}}{Q_{boiler}} = \frac{Q_{boiler} - Q_{condenser}}{Q_{boiler}} = \frac{mT_1(s_1 - s_4) - mT_2(s_1 - s_4)}{mT_1(s_1 - s_4)}$$

$$\eta_{isentropic\ cycle} = 1 - \frac{T_2}{T_1}$$

A computer model is developed to analyse this cycle in details

```
/*
M. Turhan Coban
a steam ideal carnot cycle
steam b=new steam();
double a[]=b.property("tp",30.0,1.01325);
input parameters
Thermodynamic set input types
pt tp vt tv rot tro tx px pv pro vp rop ph hp ps sp

output parameters (a)
 a[0] "P, pressure        ";
   a[1] "T, temperature     ";
   a[2] "v, specific volume ";
   a[3] "h, enthalpy        ";
   a[4] "u, internal energy ";
   a[5] "s, entropy         ";
   a[6] "x, quality         ";
   a[7] "Cp, specific heat at constant pressure ";
```

```
   a[8] "Cv, specific heat at constant volume ";
   a[9] ""+'\u03B1'+" isobaric thermal expansion coefficient ";
   a[10] ""+'\u03BA'+" isothermal compressibility" ;
   a[11] ""+'\u03B7'+" Dynamic viscosity" ;
   a[12] "k Thermal conductivity" ;
   a[13] ""+'\u03C3'+" Surface tension" ;
   a[14] "Prandtl number" ;
   a[15] ""+'\u03C1'+" density" ;
   a[16] " speed of sound" ;
*/
public class carnot_adiabatic
{
double T[];
double P[];
double h[];
double s[];
double x[];
double etait,etaip;
double etacycle,etacycle1,etaisi,etatotal;
double m;//kg/s
double Wt,Wp,W;//adiabatic carnot work
double Wt_isent,Wp_isent,W_isent;//adiabatic carnot work
double Qboiler,Qcondenser;
steamIAPWS_IF97 b;
double c3[][];
double c1[][];
double eta_turbine;
double eta_compressor;
public carnot_adiabatic(double mi,double P1, double P2,double eta_turbinei,double
eta_compressori)
{
m=mi;
eta_turbine=eta_turbinei;
eta_compressor=eta_compressori;
b=new steamIAPWS_IF97();
T=new double[9];
P=new double[9];
h=new double[9];
s=new double[9];
x=new double[9];

P[4]=P[1]=P1;//turbine inlet
P[3]=P[2]=P2;//turbine outlet
c1=new double[2][20];
c3=new double[2][20];
}
public void cycle()
{
// Turbine input  1
double a1[]=b.property("px",P[1],1.0);
h[1]=a1[3];
s[1]=a1[5];
T[1]=a1[1];
x[1]=1.0;
// isentropik turbine output 5
```

```
s[5]=s[1];
double a5[]=b.property("ps",P[2],s[5]);
T[5]=a5[1];
h[5]=a5[3];
x[5]=a5[6];
//Adiabatic turbine output 2
h[2]=h[1]-eta_turbine*(h[1]-h[5]);
double a2[]=b.property("ph",P[2],h[2]);
T[2]=a2[1];
s[2]=a2[5];
x[5]=a5[6];
//condenser saturated liquid property 7
double a7[]=b.property("px",P[4],0.0);
h[7]=a7[3];
s[7]=a7[5];
T[7]=a7[1];
// isentropic compresor output 6
s[6]=s[7];
double a6[]=b.property("ps",P[3],s[6]);
h[6]=a6[3];
s[6]=a6[5];
T[6]=a6[1];
x[6]=a6[6];
//Adiabatic compressor input 3
x[3]=0.0;
double a3[]=b.property("px",P[3],x[3]);
h[3]=a3[3];
s[3]=a3[5];
T[3]=a3[1];
//Adiabatic compressor output 4
h[4]=h[3]+(h[7]-h[6])/eta_compressor;
double a4[]=b.property("ph",P[4],h[4]);
s[4]=a4[5];
T[4]=a4[1];
x[4]=a4[6];
// adiabatic carnot
Wt=m*(h[1]-h[2]);
Wp=m*(h[4]-h[3]);
W=Wt-Wp;
//isentropic carnot
Wt=m*(h[1]-h[2]);
Wp=m*(h[4]-h[3]);
W=Wt-Wp;
Qboiler=m*(h[1]-h[4]);
Qcondenser=m*(h[2]-h[3]);
etacycle=W/Qboiler;
etacycle1=1.0-T[2]/T[1];
etaisi=Qcondenser/Qboiler;
etatotal=etacycle+etaisi;
}

public String toString()
{
cycle();
String ss="steam adiabatic carnot cycle\n";
```

```
ss+=" cycle carnot efficiency by using eta=1-Tlow/Thigh = "+etacycle1+"\n";
ss+=" cycle efficiency    = " + etacycle+"\n";
ss+=" heat energy utilisation rate = "+etaisi+"\n";
ss+=" total heat+power efficiency = "+etatotal+"\n";
ss+=" Turbine power output    = " + Wt+" kW\n";
ss+=" Net power output    = " + W +" kW\n";
ss+=" Boiler heat input = " + Qboiler +" kW\n";
ss+=" Condenser heat  = "+Qcondenser +" kW\n";

ss+=" h1 turbine input = "+h[1] +" kJ/kg\n";
ss+=" T1 turbine input = "+T[1] +" derece C\n";
ss+=" P1 turbine input = "+P[1] +" bar \n";
ss+=" s1 turbine input = "+s[1] +" kJ/kgK\n";
ss+=" x1 turbine input = "+x[1] +" kgvapor/kgtotal"+"\n";

ss+=" h2 condenser output = "+h[2] +" kJ/kg\n";
ss+=" T2 turbine output = "+T[2] +" derece C\n";
ss+=" P2 turbine output = "+P[2] +" bar \n";
ss+=" s2 turbine output = "+s[2] +" kJ/kgK\n";
ss+=" x2 turbine output = "+x[2] +" kgvapor/kgtotal"+"\n";

ss+=" h3 condenser output = "+h[3] +" kJ/kg\n";
ss+=" T3 condenser output = "+T[3] +" derece C\n";
ss+=" P3 condenser output = "+P[3] +" bar \n";
ss+=" s3 condenser output = "+s[3] +" kJ/kgK\n";
ss+=" x3 condenser output = "+x[3] +" kgvapor/kgtotal"+"\n";

ss+=" h4 pump output = "+h[4] +" kJ/kg\n";
ss+=" T4 pump output = "+T[4] +" derece C\n";
ss+=" s4 pump output = "+s[4] +" kJ/kgK\n";
ss+=" P4 pump output = "+P[4] +" bar \n";
ss+=" x4 pump output = "+x[4] +" kgvapor/kgtotal"+"\n";
return ss;
}
public double[][] TS1()
{
double a[][]=Text.readDoubleT("steam.txt");
return a;
}

public double[][] TS()
{
double a[][]=new double[2][201];
double tc=b.Tc-273.15;
double dt=(tc-10.0)/100.0;
int i=0;
double t;
for(t=10.0;t<=tc;t+=dt)
{double a1[]=b.property("tx",t,0.0);
 a[1][i]=t;
 a[0][i]=a1[5];
 i++;
}
double a3[]=b.property("tx",tc,1.0);
 a[1][i]=t;
```

```
 a[0][i]=a3[5];
 i++;
for(t=tc;t>=10;t-=dt)
{double a2[]=b.property("tx",t,1.0);
 a[1][i]=t;
 a[0][i]=a2[5];
 i++;
}
return a;
}

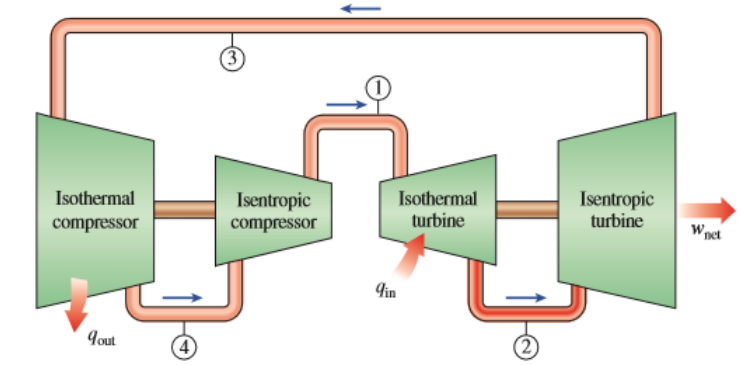
public void plot()
{
 graph2D pp=new graph2D();
 pp.addData(b.ass,b.ats);
 double t1[]={T[1],T[2],T[3],T[4],T[7],T[1]};
 double s1[]={s[1],s[2],s[3],s[4],s[7],s[1]};
 pp.addData(s1,t1);
 double t2[]={T[1],T[5]};
 double s2[]={s[1],s[5]};
  pp.addData(s2,t2,1,"blue");
 double t3[]={T[6],T[7]};
 double s3[]={s[6],s[7]};
 pp.addData(s3,t3,1,"blue");
 pp.setPlabel("Adiabatic Carnot steam Cycle");
 pp.setYlabel("T, degree K");
 pp.setXlabel("s entropy kJ/kgK");

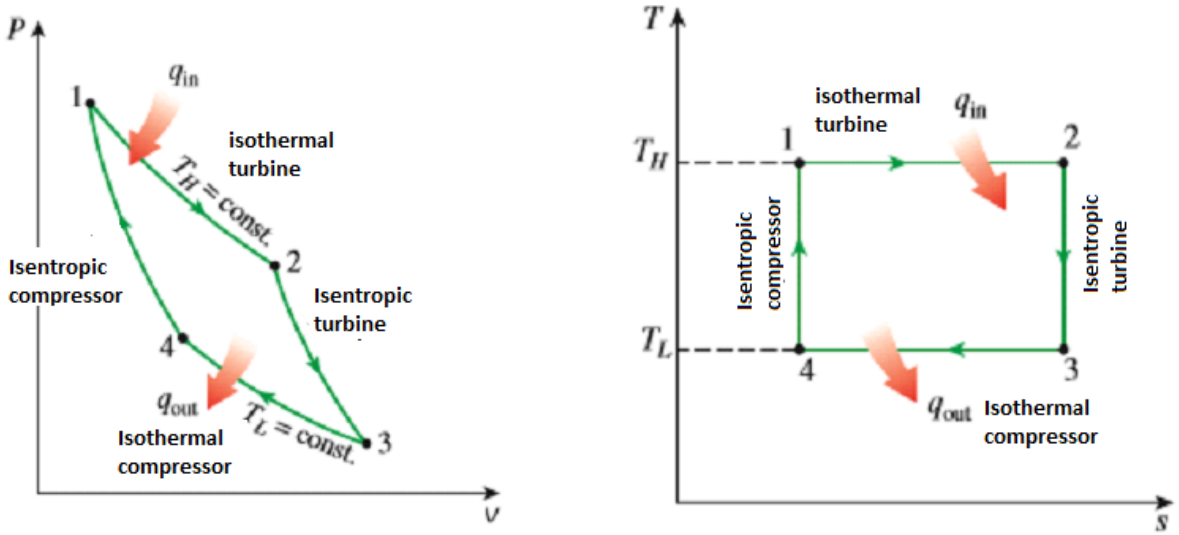
 pp.plot();
}
public static void main(String arg[])
{
double m=1.0;  // kg/s cycle mass flow rate
double P2=100.0;         //  kPa turbine inlet pressure
double P1=6000.0 ;       //  kPa turbine exit pressure
double eta_turbine=0.9;
double eta_compressor=0.9;
carnot_adiabatic r=new carnot_adiabatic(m,P1,P2,eta_turbine,eta_compressor);
System.out.println(r.toString());
r.plot();
}
}
```

```
---------- Capture Output ----------
> "C:\Users\Turhan
Çoban\.p2\pool\plugins\org.eclipse.justj.openjdk.hotspot.jre.full.win32.x86_64_19.0.2.v2
0230129-1123\jre\bin\javaw.exe" carnot_adiabatic
steam adiabatic carnot cycle
 cycle carnot efficiency by using eta=1-Tlow/Thigh = 0.32070132160948717
 cycle efficiency    = 0.1916540037291697
 heat energy utilisation rate = 0.8083459962708304
 total heat+power efficiency = 1.0
 Turbine power output    = 591.3862952259287 kW
 Net power output    = 421.01709060083664 kW
 Boiler heat input = 2196.7560416623737 kW
 Condenser heat  = 1775.7389510615371 kW
```

h1 turbine input = 2784.5617321036975 kJ/kg
T1 turbine input = 548.7364107560489 derece C
P1 turbine input = 6000.0 bar
s1 turbine input = 5.890068009035895 kJ/kgK
x1 turbine input = 1.0 kgvapor/kgtotal
h2 condenser output = 2193.175436877769 kJ/kg
T2 turbine output = 372.7559186113376 derece C
P2 turbine output = 100.0 bar
s2 turbine output = 6.0663475852939435 kJ/kgK
x2 turbine output = 0.0 kgvapor/kgtotal
h3 condenser output = 417.4364858162317 kJ/kg
T3 condenser output = 372.7559186113376 derece C
P3 condenser output = 100.0 bar
s3 condenser output = 1.3025601737745955 kJ/kgK
x3 condenser output = 0.0 kgvapor/kgtotal
h4 pump output = 587.8056904413238 kJ/kg
T4 pump output = 411.9637328740635 derece C
s4 pump output = 1.7211839998921759 kJ/kgK
P4 pump output = 6000.0 bar
x4 pump output = -1.0 kgvapor/kgtotal

Carnot cycle can beconstructed by using air as working fluid by using an isothermal turbine, an isothermal compressor, an isentropic turbine an an isentropic compressor.





Let us assume that minimum and maximum temperatures and minimum and maximum pressures are given.

$$h(T) = h_{ref} + \int_{T_{ref}}^{T} C_p(T)dT$$

$$s(T,P) = s_{ref} + \int_{T_{ref}}^{T} \frac{C_p(T)}{T} dT - Rln\left(\frac{P}{P_{ref}}\right) = s^0(T) - Rln\left(\frac{P}{P_{ref}}\right)$$

Where $s^0(T) = s_{ref} + \int_{T_{ref}}^{T} \frac{C_p(T)}{T} dT$

Reduced pressure in isentropic processes

$$P_r(\mathrm{T}) = \left(\frac{P}{P_{ref}}\right) = \exp\left(\frac{s^0(T) - s^0(T_{ref})}{R}\right)$$

So

$$s_2(T,P) - s_1(T,P) = 0 = s_2^0(T) - s_1^0(T) - Rln\left(\frac{P_2}{P_1}\right)$$

$$\frac{P_2}{P_1} = \exp\left(\frac{s_2^0(T) - s_1^0(T)}{R}\right) = \frac{P_{r2}}{P_{r1}}$$

By using these relations air Carnot cycle can be evaluated

Program 3.1-1 **carnot_air Carnot cycle for air (ideal gas approach)**

```
 public class carnot_air
{
// an airideal carnot cycle
// Gas b=new Gas("air");
// double a[]=g1.property(30.0,1.01325);
// output set
//a[0]=p;
//a[1]=t;
//a[2]=v(t,p);
//a[3]=h(t);
//a[4]=u(t);
//a[5]=s(t,p);
//a[6]=g(t,p);
//a[7]=ht(t);
//a[8]=gt(t,p);
//a[9]=Cp(t);
//a[10]=Cv(t);
//a[11]=gamma(t);
//a[12]=c(t);
//a[13]=vis(t);
//a[14]=k(t);
//a[15]=M;
//a[16]=Prandtl(t);
//a[17]=pr(t);
//a[18]=vr(t);
double T[];
double P[];
double h[];
double s[];
double v[];
double pr[];
double etait,etaip;
double etacevrim,etacevrim1,etaisi,etatoplam;
double m;//kg/s
double Wt_1_2,Wt_2_3,Wc_3_4,Wc_4_1,W;//i?
double Q_1_2,Q_3_4,Q;
Gas b;
double c1[][];
```

```
double c3[][];
public carnot_air(double mi,double P1, double T1,double P3,double T3)
{
m=mi;
b=new Gas("air");
b.mole=false;
T=new double[9];
P=new double[9];
h=new double[9];
s=new double[9];
v=new double[9];
pr=new double[9];
P[1]=P1;
P[3]=P3;
T[1]=T1;
T[3]=T3;
c1=new double[2][20];
c3=new double[2][20];
}
public void cycle()
{
double a1[]=b.property(T[1],P[1]);
h[1]=a1[3];
s[1]=a1[5];
T[1]=a1[1];
v[1]=a1[2];
pr[1]=a1[17];
double a3[]=b.property(T[3],P[3]);
T[3]=a3[1];
h[3]=a3[3];
s[3]=a3[5];
v[3]=a3[2];
pr[3]=a3[17];
//2-3 isentropik process
T[2]=T[1];
pr[2]=pr[1];
s[2]=s[3];
P[2]=P[3]*pr[2]/pr[3];


h[2]=h[1]+T[1]*(s[2]-s[1]);
v[2]=P[1]*v[1]/P[2];
//4-1 isentropik process
pr[4]=pr[3];
T[4]=T[3];
s[4]=s[1];
P[4]=P[1]*pr[4]/pr[1];
h[4]=h[3]-T[3]*(s[3]-s[4]);
v[4]=P[3]*v[3]/P[4];

Q_1_2=m*(h[2]-h[1]);
Q_3_4=m*(h[3]-h[4]);
Q=Q_1_2;
W=Q_1_2-Q_3_4;
etacevrim=W/Q;
etacevrim1=1.0-T[3]/T[2];
}

public String toString()
{
cycle();
String ss="air carnot cycle\n";
ss+=" cycle efficiency by using eta=1-Tlow/Thigh = "+etacevrim1+"\n";
ss+=" cycle efficiency    = " + etacevrim+"\n";
ss+=" Net power output    = " + W +" kW\n";
ss+="heat input = " + Q+" kW\n";
```

```
ss+=" h1 isothermal turbine input = "+h[1] +" kJ/kg\n";
ss+=" T1 isothermal turbine input = "+T[1] +" degree C\n";
ss+=" P1 isothermal turbine input = "+P[1] +" bar \n";
ss+=" s1 isothermal turbine input = "+s[1] +" kJ/kgK\n";
ss+="v1 isothermal turbine input = "+v[1] +" m^3/kg"+"\n";
ss+=" h2 isothermal turbine input = "+h[2] +" kJ/kg\n";
ss+=" T2 isentropic turbine input = "+T[2] +" degree C\n";
ss+=" P2 isentropic turbine input = "+P[2] +" bar \n";
ss+=" s2 isentropic turbine input = "+s[2] +" kJ/kgK\n";
ss+=" v2 isentropic turbine input = "+v[2] +" m^3/kg"+"\n";
ss+=" h3  isothermal compressor input = "+h[3] +" kJ/kg\n";
ss+=" T3 isothermal compressor input = "+T[3] +" degree C\n";
ss+=" P3 isothermal compressor input = "+P[3] +" bar \n";
ss+=" s3 isothermal compressor input = "+s[3] +" kJ/kgK\n";
ss+=" v3 isothermal compressor input = "+v[3] +" m^3/kg"+"\n";
ss+=" h4 isentropic compressor input = "+h[4] +" kJ/kg\n";
ss+=" T4  isentropic compressor input  = "+T[4] +" degree C\n";
ss+=" P4 isentropic compressor input = "+P[4] +" bar \n";
ss+=" s4  isentropic compressor input  = "+s[4] +" kJ/kgK\n";
ss+=" x4  isentropic compressor input  = "+v[4] +" m^3/kg"+"\n";
return ss;
}
public void plot_P_v()
{
double t1[]={T[1],T[2],T[3],T[4],T[1]};
 double p1[]={P[1],P[2],P[3],P[4],P[1]};
 double v1[]={v[1],v[2],v[3],v[4],v[1]};
  double pr1[]={pr[1],pr[2],pr[3],pr[4],pr[1]};
 double p[][]=new double[4][101];
  double v[][]=new double[4][101];
  double t[][]=new double[4][101];
  double pri[][]=new double[4][101];
     int i=0;
     for(int j=0;j<=100;j++)
     {p[i][j]=p1[i]+(p1[i+1]-p1[i])/100*j;
      v[i][j]=p1[i]*v1[i]/p[i][j];
      System.out.println("i="+i+"j="+j+"*"+p[i][j]+" "+v[i][j]);
     }
     i=1;
     double a[];
     for(int j=0;j<=100;j++)
     {t[i][j]=t1[i]+(t1[i+1]-t1[i])/100*j;
      pri[i][j]=b.pr(t[i][j]);
      p[i][j]=p1[i]*pri[i][j]/pr1[i];
      v[i][j]=b. v(t[i][j], p[i][j]);
      System.out.println("i="+i+"j="+j+"* p"+p[i][j]+" v "+v[i][j]+" t "+t[i][j]);
     }
      i=2;
     for(int j=0;j<=100;j++)
     {p[i][j]=p1[i]+(p1[i+1]-p1[i])/100*j;
      v[i][j]=p1[i]*v1[i]/p[i][j];
           System.out.println("i="+i+"j="+j+"*"+p[i][j]+" "+v[i][j]);
     }
     i=3;
     for(int j=0;j<=100;j++)
     {t[i][j]=t1[i]+(t1[i+1]-t1[i])/100*j;
      pri[i][j]=b.pr(t[i][j]);
      p[i][j]=p1[i]*pri[i][j]/pr1[i];
      v[i][j]=b. v(t[i][j], p[i][j]);
           System.out.println("i="+i+"j="+j+"*"+p[i][j]+" "+v[i][j]);
     }
   Plot pp=new Plot(v1,p1);
   pp.setPlotType(0,25)               ;
   pp.setPlabel("Ideal Carnot air Cycle p-v diagram");
   pp.setYlabel("P, bar");
   pp.setXlabel("v specific volume m^3/kg");
```

```
     pp.addData(v[0],p[0]);
    pp.addData(v[1],p[1]);
    pp.addData(v[2],p[2]);
    pp.addData(v[3],p[3]);
    pp.plot();
  }

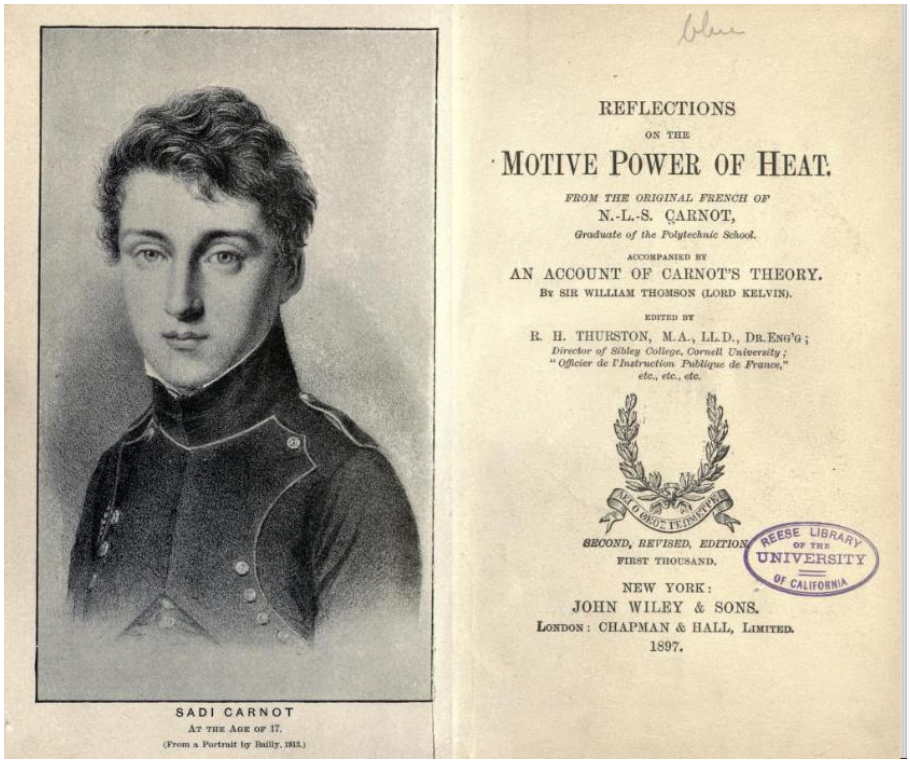

public void plot()
{
 double t1[]={T[1],T[2],T[3],T[4],T[1]};
 double s1[]={s[1],s[2],s[3],s[4],s[1]};
  Plot pp=new Plot(s1,t1);
 pp.setPlabel("Ideal Carnot air Cycle");
 pp.setYlabel("T, degree C");
 pp.setXlabel("s entropy kJ/kgK");
 pp.plot();
}
}
```

```
import javax.swing.*;

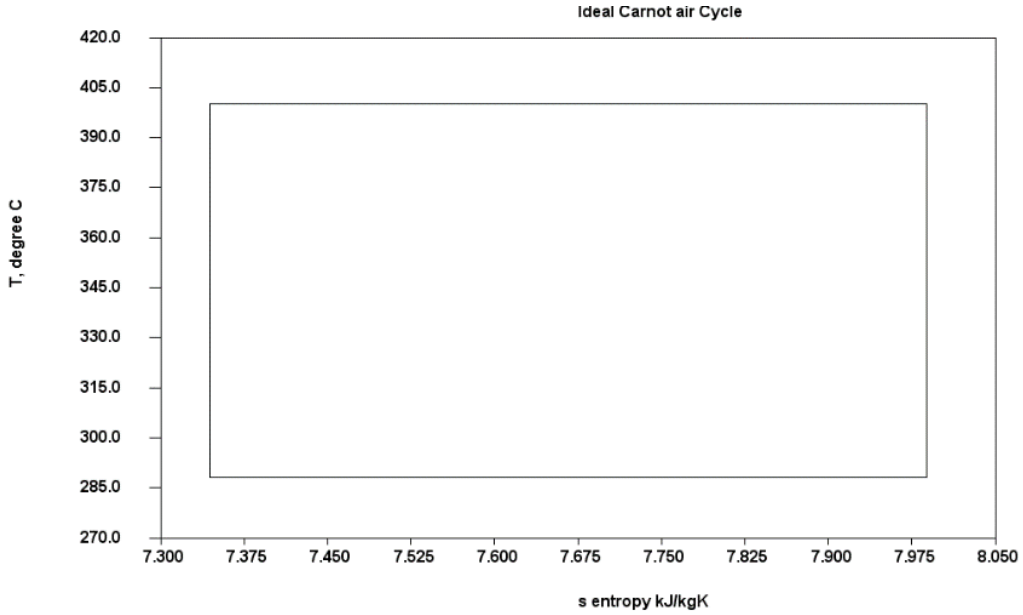
public class carnotairtest
{
public static void main(String arg[])
{
double m=1.0;  // kg/s çevrim R134a debisi
double T1=400; //degree K
double P1=30.0;         //  bar isothermal türbine inlet pressure bar
double P3=1.0 ;         // bar isothermal compressor inlet pressure bar
double T3=288;  //degree K
carnot_air r=new carnot_air(m,P1,T1,P3,T3);
r.cycle();
System.out.println(r);
r.plot();
r.plot_P_v();
}
}
```

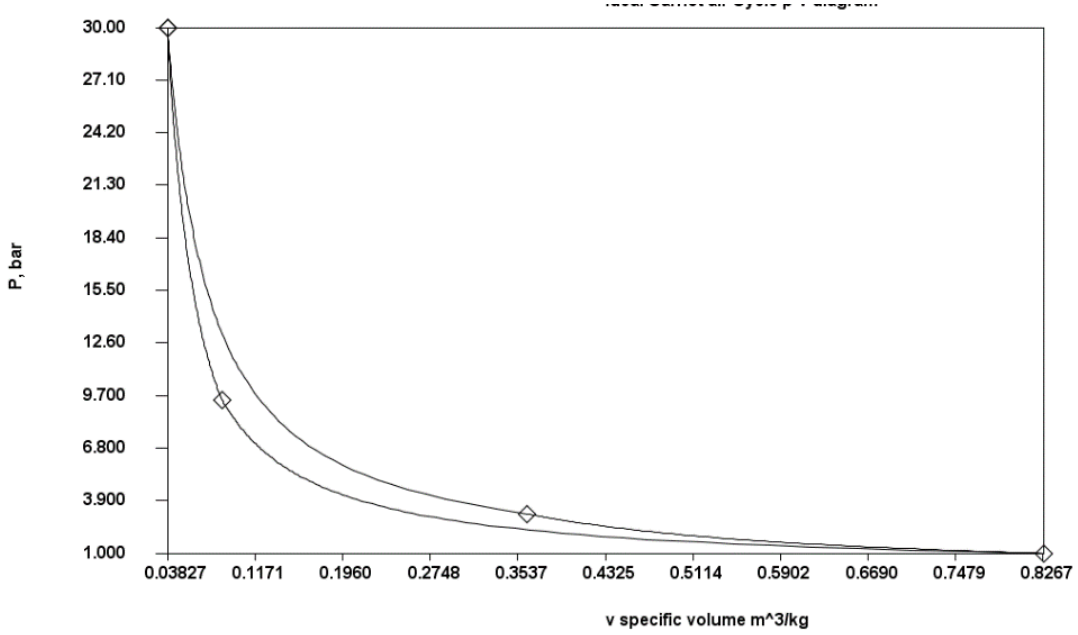
```
---------- Capture Output ----------
> "E:\co\java\bin\java.exe" carnotairtest
air carnot cycle
 cycle efficiency by using eta=1-Tlow/Thigh = 0.28
 cycle efficiency    = 0.28
 Net power output    = 72.2344328794403 kW
heat input = 257.9801174265725 kW
 h1 isothermal turbine input = 621.2095212656864 kJ/kg
 T1 isothermal turbine input = 400.0 degree C
 P1 isothermal turbine input = 30.0 bar
 s1 isothermal turbine input = 7.343821736658354 kJ/kgK
v1 isothermal turbine input = 0.03827483979850585 m^3/kg
 h2 isothermal turbine input = 879.1896386922589 kJ/kg
 T2 isentropic turbine input = 400.0 degree C
 P2 isentropic turbine input = 3.1722852485320434 bar
 s2 isentropic turbine input = 7.988772030224785 kJ/kgK
 v2 isentropic turbine input = 0.3619615211105367 m^3/kg
 h3  isothermal compressor input = 508.1966825499989 kJ/kg
 T3 isothermal compressor input = 288.0 degree C
 P3 isothermal compressor input = 1.0 bar
 s3 isothermal compressor input = 7.988772030224785 kJ/kgK
 v3 isothermal compressor input = 0.8267365396477263 m^3/kg
 h4 isentropic compressor input = 322.4509980028667 kJ/kg
 T4  isentropic compressor input  = 288.0 degree C
 P4 isentropic compressor input = 9.45690492804275 bar
 s4  isentropic compressor input  = 7.343821736658354 kJ/kgK
```

v4 isentropic compressor input = 0.08742147097156364 m^3/kg

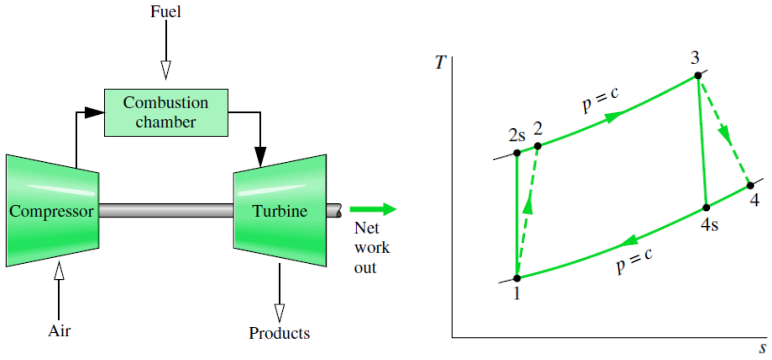


Nicolas Leonard Sadi Carnot





3.2 AIR STANDARD BRAYTON- ERICCSON (GAS TURBINE) CYCLE



	
George Brayton	**John Ericcson**

In Brayton (Gas Turbine) cycle, air is compressed in a compressor, and then in combustion chamber fuel is burned and combustion gases is heated and then these gases is used to turn a tubine and create work output. As an approximation to the cyclei it can be assumed to carry

out all cycle with air. After looking combustion process in detail, the problem will be reinvestigated with combustion gas output.

For air(ideal gas) process:

$W_{compressor} = m(h_2 - h_1)$ (3.2.1)

$W_{isentropiccompressor} = m(h_{2s} - h_1) \qquad s_{2s} = s_1$ (3.2.2)

$\eta_{isentropiccompressor} = \frac{W_{isentropiccompressor}}{W_{compressor}}$ (3.2.3)

$Q_{combustionchamber} = m(h_3 - h_2) \qquad P_2 = P_3$ (3.2.4)

$W_{turbine} = m(h_3 - h_4)$ (3.2.5)

$W_{isentropicturbine} = m(h_3 - h_{4s}) \qquad s_3 = s_{4s} \quad P_4 = P_1$ (3.2.6)

$\eta_{isentropicturbine} = \frac{W_{turbine}}{W_{isentropicturbine}}$ (3.2.7)

$\eta_{cycle} = \frac{W_{turbine} - W_{compressor}}{Q_{combustionchamber}}$ (3.2.8)

Program 3.2.1 Brayton cycle

```
public class brayton
{
Gas g;
double TT1[],ss1[];
double TT2[],ss2[];
double Wturbin,Wkompresor,Wnet,Q,Qisi,eta,etaisi,etatoplam,T1,P1,T2,P2,T3,P3,T4,P4;
public brayton(double m,double T1,double P1, double P2, double T3,double etakompresor,double
etaturbin)
{ TT1=new double[25];ss1=new double[25];
  TT2=new double[2];ss2=new double[2];
g=new Gas("air");
g.mole=false;
// 1
double P3=P2;
double P4=P1;
double a1[]=g.property(T1,P1);
double h1=a1[3];
double s1=a1[5];
TT1[0]=T1;ss1[0]=s1;
double pr1=a1[17];
// 2i
double pr2=P2/P1*pr1;
double T2i=g.Ti(pr2,P2);
double a2i[]=g.property(T2i,P2);
double h2i=a2i[3];
double s2i=a2i[5];
TT1[1]=T2i;ss1[1]=s2i;
double Cp2i=a2i[9];
// 2
double h2=(h2i-h1)/etakompresor+h1;
double T2=(h2-h2i)/Cp2i+T2i;
double a2[]=g.property(T2,P2);
double s2=a2[5];
//3 noktası
double a3[]=g.property(T3,P3);
double h3=a3[3];
double s3=a3[5];
double pr3=a3[17];
double pr4=P1/P2*pr3;
//4i
```

```
double T4i=g.Ti(pr4,P1);
double a4i[]=g.property(T4i,P1);
double h4i=a4i[3];
double s4i=a4i[5];
double Cp4i=a4i[9];
for(int i=0;i<10;i++)
{TT1[i+2]=T2i+i*(T3-T2)/10;
 a3=g.property(TT1[i+2],P3);
 ss1[i+2]=a3[5];
}
TT1[12]=T3;ss1[12]=s3;
TT2[0]=T3;ss2[0]=s3;

//System.out.println("pr3="+g.pr(T3));

TT2[1]=T4i;ss2[1]=s4i;
//System.out.println("t4i="+T4i+"h4i="+h4i+"Cp4i="+Cp4i);
//4 noktası
double h4=h3-etaturbin*(h3-h4i);
double T4=(h4-h4i)/Cp4i+T4i;
double a4[]=g.property(T4,P1);
double s4=a4[5];
for(int i=0;i<10;i++)
{TT1[i+13]=T4-i*(T4-T1)/10;
 a3=g.property(TT1[i+13],P1);
 ss1[i+13]=a3[5];
}
TT1[23]=T1;
ss1[23]=s1;
TT1[24]=T2;
ss1[24]=s2;
System.out.println("t4="+T4+"h4="+h4+"s4="+s4);
System.out.println("t4i="+T4i+"h4i="+h4i+"s4i="+s4i);
//System.out.println("t4="+T4+"h4="+h4+"s4="+s4);
Wturbin=m*(h3-h4);
Wkompresor=m*(h2-h1);
Q=m*(h3-h2);
Qisi=m*(h4-h1);
Wnet=Wturbin-Wkompresor;
eta=Wnet/Q;
etaisi=Qisi/Q;
etatoplam=eta+etaisi;
this.T1=T1;
this.P1=P1;
this.T2=T2;
this.P2=P2;
this.T3=T3;
this.P3=P3;
this.T4=T4;
this.P4=P4;
Plot pp=new Plot(ss1,TT1);
pp.addData(ss2,TT2);
pp.plot();
}
public String toString()
{
String s=" Brayton cycle \n";
s+="Turbine power output W turbine = "+Wturbin+" kW \n";
```

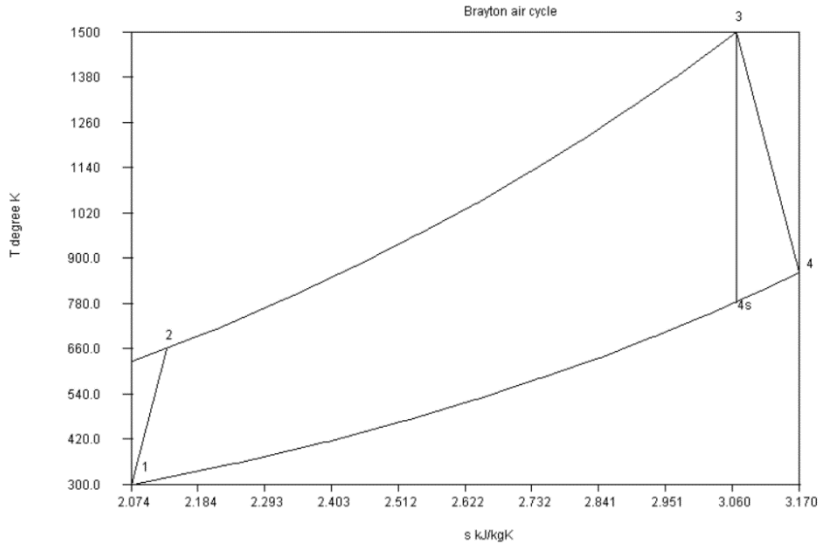
```
s+="Compressor power input W kompresor = "+Wkompresor+" kW\n";
s+="Net power output  Wnet = "+Wnet+" kW\n";
s+="Combustion chamber heat input Q combustion = "+Q+" kW\n";
s+="Evailable exhaust heat energy Q exhaust available = "+Qisi+" kW \n";
s+="Cycle efficiency eta  = "+eta+"\n";
s+="Heat utilisation efficiency eta ısı = "+etaisi+" \n";
s+="Total heat energy utilisation efficiency = "+etatoplam+"\n";
s+="Compressor inlet temperature = "+(T1-273.15)+" degree C\n";
s+="Compressor inlet pressure  = "+P1+" bar\n";
s+="Combustion chamber inlet temperature = "+(T2-273.15)+" degree C\n";
s+="Combustion chamber inlet pressure   = "+P2+" bar\n";
s+="Turbine inlet temperature = "+(T3-273.15)+" degree C\n";
s+="Turbine inlet pressure   = "+P3+" bar\n";
s+="Turbine exit temperature = "+(T4-273.15)+" degree C\n";
s+="Turbine exit pressure   = "+P4+" bar\n";
return s;
}

}
```

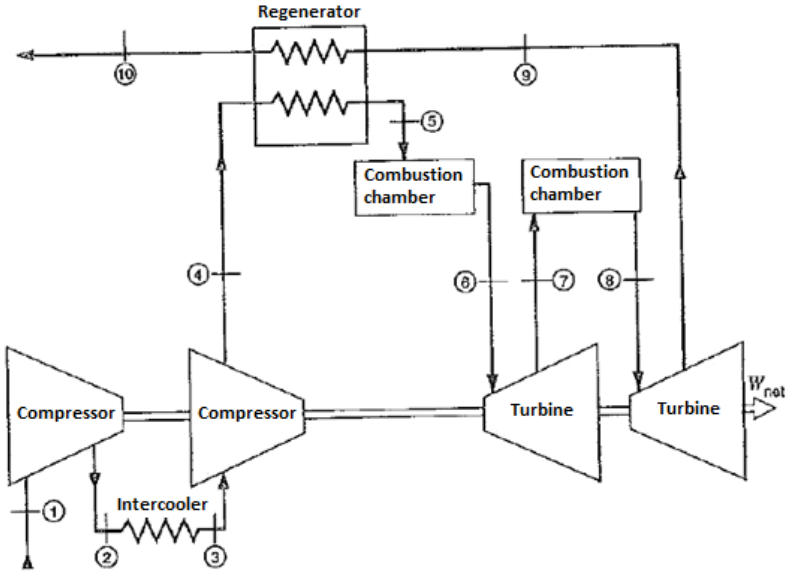
Program 3.2.2 Brayton cycle test program

```
public class braytontest
{
  public static void main(String arg[])
  {
                double mhava=1; //kg/s hava giriş debisi
                double T1=300.0; //derece K hava giriş sıaklığı
                double P1=1.1;// bar hava giriş basıncı
                double P2=15.0;// bar sıkıştırılmış hava basıncı
                double T3=1500.0; //derece K türbin giriş sıcaklığı
                double etakompresor=0.9; //kompresör isntropik verimi
                double etaturbin=0.9; //türbin isentropik verimi
     brayton bry=new brayton(mhava,T1,P1,P2,T3,etakompresor,etaturbin);
     System.out.println(bry.toString());
  }
}
```

```
 ---------- Capture Output ----------
> "C:\java\bin\javaw.exe" braytontest
Brayton cycle
Turbine power output W turbine = 748.794597011468 kW
Compressor power input W kompresor = 370.08310171234666 kW
Net power output  Wnet = 378.71149529912134 kW
Combustion chamber heat input Q combustion = 965.7618454299507 kW
Evailable exhaust heat energy Q exhaust available = 587.0503501308293 kW
Cycle efficiency eta  = 0.392137561751077
Heat utilisation efficiency eta ısı = 0.607862438248923
Total heat energy utilisation efficiency = 1.0
Compressor inlet temperature = 26.850000000000023 degree C
Compressor inlet pressure  = 1.1 bar
Combustion chamber inlet temperature = 386.6815012229864 degree C
Combustion chamber inlet pressure   = 15.0 bar
Turbine inlet temperature = 1226.85 degree C
Turbine inlet pressure   = 15.0 bar
Turbine exit temperature = 586.4004575651818 degree C
Turbine exit pressure   = 1.1 bar
```



If better efficiency is required, Brayton cycle can be improved by adding intercooler and regenerator heat exchangers . Some of the energy required to heat gas can be supplied from turbine exhoust gases so the energy input into combustion camber is reduced. By using compressor intercooler total compresser work input is reduced.



$W_{lowpressurecompressor} = m(h_2 - h_1)$ (3.2.9)

$W_{lowpressureisentropiccompressor} = m(h_{2s} - h_1)$ $s_{2s} = s_1$ (3.2.10)

$W_{highpressurecompressor} = m(h_4 - h_3)$ (3.2.11)

$W_{highpressureisentropiccompressor} = m(h_{4s} - h_3)$ $s_{4s} = s_3$ (3.2.12)

$\eta_{isentropiccompressor} = \frac{W_{isentropiccompressor}}{W_{compressor}}$ (3.2.13)

$Q_{int\,ercooler} = m(h_2 - h_3)$ (3.2.14)

$Q_{regenerator} = m(h_5 - h_4) = m(h_9 - h_{10})$ (3.2.15)

$Q_{firstcombustionchamber} = m(h_6 - h_5)$ $P_6 = P_5 = P_4$ (3.2.16)

$W_{highpressureturbine} = m(h_6 - h_7)$ (3.2.17)

$W_{isentropichighpressureturbine} = m(h_6 - h_{7i})$ (3.2.18)

$Q_{sec\,ondcombustionchamber} = m(h_8 - h_7)$ $P_7 = P_8$ (3.2.19)

$W_{lowpressureturbine} = m(h_8 - h_9)$ (3.2.20)

$W_{isentropiclowpressureturbine} = m(h_8 - h_{9i})$ (3.2.21)

$\eta_{isentropicturbine} = \frac{W_{turbine}}{W_{isentropicturbine}}$ (3.2.22)

$\eta_{cycle} = \frac{W_{turbine} - W_{compressor}}{Q_{combustionchamber}}$ (3.2.23)

Program 3.2.3 Brayton cycle program

```
public class brayton2
{
Gas g;
double PP1[],vv1[];
double PP2[],vv2[];
double TT1[],ss1[];
double TT2[],ss2[];
double
Wturbine1,Wturbine2,Wturbine,Wcompressor1,Wcompressor2,Wcompressor,Wnet,Q1,Q2,Qregenerator,Qi
ntercooler,Qnet,eta;
double T[],P[],h[],s[];
public brayton2(double m,double T1,double P1,double P2,double P4,double T3,double T6,double T8,double
dT,double etacompressor1,double etacompressor2,double etaturbine1,double etaturbine2)
{  T=new double[15];
   P=new double[15];
   h=new double[15];
   s=new double[15];
   TT1=new double[25];ss1=new double[25];
   TT2=new double[2];ss2=new double[2];
   g=new Gas("air");
   g.mole=false;
// 1
P[1]=P1;T[1]=T1;
P[2]=P2;
P[3]=P2;T[3]=T3;
P[4]=P4;
P[5]=P4;
P[6]=P4;T[6]=T6;
P[7]=P2;
P[8]=P2;T[8]=T8;
P[9]=P1;
P[10]=P1;
double a[]=g.property(T[1],P[1]);
h[1]=a[3];
s[1]=a[5];
TT1[0]=T[1];ss1[0]=s[1];
double pr1=a[17];
// 2i = 11
double pr2=P2/P1*pr1;
double T2i=g.Ti(pr2,P2);

a=g.property(T2i,P2);
double h2i=a[3];
double s2i=a[5];
double Cp2i=a[9];
T[11]=T2i;
s[11]=s2i;
TT1[1]=T2i;ss1[1]=s2i;
```

```
// 2
h[2]=(h2i-h[1])/etacompressor1+h[1];
T[2]=(h[2]-h2i)/Cp2i+T2i;
a=g.property(T[2],P[2]);
s[2]=a[5];
// 3
a=g.property(T[3],P[3]);
h[3]=a[3];
s[3]=a[5];
double pr3=a[17];
// 4i = 12 isentropic process
double pr4=P[4]/P[3]*pr3;
double T4i=g.Ti(pr4,P[4]);
a=g.property(T4i,P[4]);
double h4i=a[3];
double s4i=a[5];
double Cp4i=a[9];
T[12]=T4i;
s[12]=s4i;
//4
h[4]=(h4i-h[3])/etacompressor2+h[3];

T[4]=(h[4]-h4i)/Cp4i+T4i;
a=g.property(T[4],P[4]);
s[4]=a[5];
System.out.println("h4i="+h4i+"h4="+h[4]+"T4i="+T4i+"T4="+T[4]+"s4i="+s4i+"s4="+s[4]+"s3="+s[3]);
double Cp4=a[9];
//6
a=g.property(T[6],P[6]);
h[6]=a[3];
s[6]=a[5];
double Cp6=a[9];
double pr6=a[17];
//7i
double pr7=P[7]/P[6]*pr6;
double T7i=g.Ti(pr7,P[7]);
a=g.property(T7i,P[7]);
double h7i=a[3];
double s7i=a[5];
double Cp7i=a[9];
T[13]=T7i;
s[13]=s7i;
//7
h[7]=h[6]-etaturbine1*(h[6]-h7i);
T[7]=(h[7]-h7i)/Cp7i+T7i;
a=g.property(T[7],P[7]);
h[7]=a[3];
s[7]=a[5];
//8
a=g.property(T[8],P[8]);
h[8]=a[3];
s[8]=a[5];
double pr8=a[17];
//9i
double pr9=P[9]/P[8]*pr8;
System.out.println("pr8="+pr8+"pr9="+pr9+"P8="+P[8]+"P[9]="+P[9]);
double T9i=g.Ti(pr9,P[9]);
a=g.property(T9i,P[9]);
```

```
double h9i=a[3];
double s9i=a[5];
double Cp9i=a[9];
T[14]=T9i;
s[14]=s9i;
//9
h[9]=h[8]-etaturbine2*(h[8]-h9i);
T[9]=(h[9]-h9i)/Cp9i+T9i;
a=g.property(T[9],P[9]);
s[9]=a[5];
System.out.println("h9="+h[9]+"h9i="+h9i+"Cp9i="+Cp9i+"t9i="+T9i+"t9="+(T[9]-273.15)+"degree C");

//10
T[10]=T[4]+dT;
a=g.property(T[10],P[10]);
h[10]=a[3];
s[10]=a[5];
//5
h[5]=h[4]+(h[9]-h[10]);
double Cp5=(Cp4+Cp6)/2.0;
double T5e=h[5]/Cp5;

T[5]=g.T('h',h[5],P[5]);
System.out.println("T5e="+T5e+"T5="+T[5]);
a=g.property(T[5],P[5]);
s[5]=a[5];

Wturbine1=m*(h[6]-h[7]);
Wturbine2=m*(h[8]-h[9]);
Wturbine=Wturbine1+Wturbine2;
Wcompressor1=m*(h[2]-h[1]);
Wcompressor2=m*(h[4]-h[3]);
Wcompressor=Wcompressor1+Wcompressor2;
Q1=m*(h[6]-h[5]);
Q2=m*(h[8]-h[7]);
Qregenerator=m*(h[5]-h[4]);
Qintercooler=m*(h[2]-h[3]);
Qnet=Q1+Q2;
System.out.println("Q1="+Q1+"Q2="+Q2+"Qnet="+Qnet);
Wnet=Wturbine-Wcompressor;
eta=Wnet/Qnet;
}
public double[][] line(double T1,double T2,double Pi,int n)
{
double TT1[]=new double[n];
double ss1[]=new double[n];
double aa[];
for(int i=0;i<n;i++)
{TT1[i]=T1+i*(T2-T1)/(n-1);
 aa=g.property(TT1[i],Pi);
 ss1[i]=aa[5];
}
double a[][]={TT1,ss1};
return a;
}
public void plot()
{double a[][]=new double[2][61];
 double c1[][]=new double[2][3];
```

```
double c3[][]=new double[2][3];
double c6[][]=new double[2][3];
double c8[][]=new double[2][3];
c1[0][0]=T[1];
c1[1][0]=s[1];
c1[0][1]=T[11];
c1[1][1]=s[11];
c1[0][2]=T[2];
c1[1][2]=s[2];

c3[0][0]=T[3];
c3[1][0]=s[3];
c3[0][1]=T[12];
c3[1][1]=s[12];
c3[0][2]=T[4];
c3[1][2]=s[4];

c6[0][0]=T[6];
c6[1][0]=s[6];
c6[0][1]=T[13];
c6[1][1]=s[13];
c6[0][2]=T[7];
c6[1][2]=s[7];

c8[0][0]=T[8];
c8[1][0]=s[8];
c8[0][1]=T[14];
c8[1][1]=s[14];
c8[0][2]=T[9];
c8[1][2]=s[9];

int i=0;
a[0][0]=T[1];
a[1][0]=s[1];
double b[][]=line(T[2],T[3],P[2],15);
for(i=0;i<15;i++)
{a[0][i+1]=b[0][i];
 a[1][i+1]=b[1][i];
}
b=line(T[4],T[6],P[4],15);
for(i=0;i<15;i++)
{a[0][i+16]=b[0][i];
 a[1][i+16]=b[1][i];
}
b=line(T[7],T[8],P[7],15);
for(i=0;i<15;i++)
{a[0][i+31]=b[0][i];
 a[1][i+31]=b[1][i];
}
 b=line(T[9],T[10],P[9],15);
for(i=0;i<15;i++)
{a[0][i+46]=b[0][i];
 a[1][i+46]=b[1][i];
}
Plot p=new Plot(a[1],a[0]);
p.addData(c1[1],c1[0],1);
p.addData(c3[1],c3[0],1);
p.addData(c6[1],c6[0],1);
```

```
 p.addData(c8[1],c8[0],1);
 p.addData(s,T,22);
 Text.printT(a);
 p.plot();
}

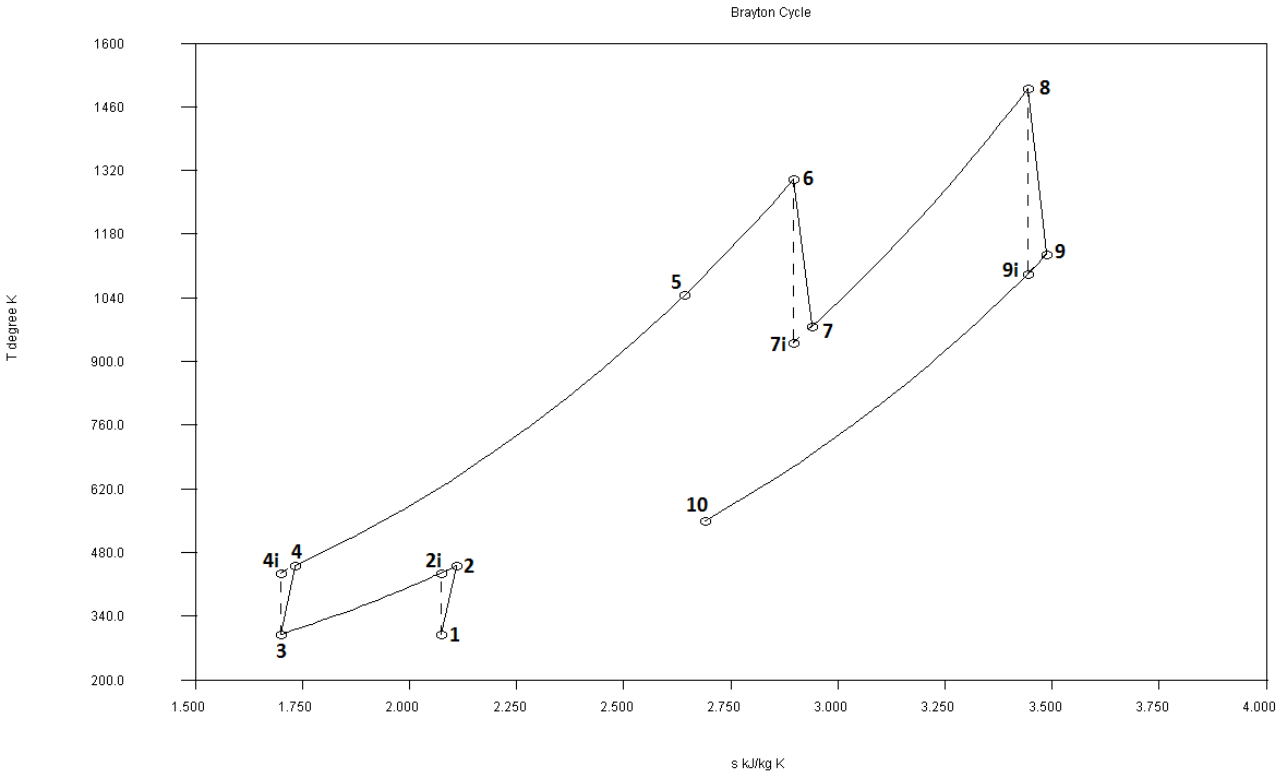
public String toString()
{
String ss=" Brayton cycle \n";
ss+="low pressure turbine power output W turbine = "+Wturbine1+" kW \n";
ss+="high pressure turbine power output W turbine = "+Wturbine2+" kW \n";
ss+="turbine power output W turbine = "+Wturbine+" kW \n";
ss+="low pressure compressor power output W turbine = "+Wcompressor1+" kW \n";
ss+="high pressure compressor power output W turbine = "+Wcompressor2+" kW \n";
ss+="compressor power input W turbine = "+Wcompressor+" kW \n";
ss+="Net power output  Wnet = "+Wnet+" kW\n";
ss+="Combustion chamber heat input Q combustion = "+Qnet+" kW\n";
ss+="Cycle efficiency eta  = "+eta+"\n";
ss+="Equivalent temperature carnot cycle efficiency eta  = "+(1.0-T[1]/T[8])+"\n";
ss+="Regenerator heat transfer   = "+Qregenerator+" kW\n";
ss+="Termodynamic states : \n";
for(int i=1;i<11;i++)
{ss+="P["+i+"] = "+P[i]+"bar T["+i+"] = "+(T[i]-273.15)+" degree C h["+i+"] = "+h[i]+" kJ/kg s["+i+"] =
"+s[i]+" kJ/kgK\n";}
return ss;
}
}
```

Program 3.2.4 Brayton cycle test program

```
public class braytontest2
{
  public static void main(String arg[])
  {
                double mhava=1; //kg/s hava giriş debisi
                double T1=300.0; //derece K hava giriş sıaklığı
                double T3=300.0; //derece K hava giriş sıaklığı
                double P1=1.1;// bar hava giriş basıncı
                double P2=4.062019202;// bar hava giriş basıncı
                double P4=15.0;// bar sıkıştırılmış hava basıncı
                double T6=1300.0; //derece K türbin giriş sıcaklığı
                double T8=1500.0; //derece K türbin giriş sıcaklığı
                double dT=100.0;
                double etakompressor=0.9; //kompresör isntropik verimi
                double etaturbine=0.9; //türbin isentropik verimi
     brayton2 bry=new
brayton2(mhava,T1,P1,P2,P4,T3,T6,T8,dT,etakompressor,etakompressor,etaturbine,etaturbine);
     System.out.println(bry.toString());
     bry.plot();
  }
}
```

```
---------- Capture Output ----------
> "C:\java\bin\javaw.exe" braytontest2
 Brayton cycle
low pressure turbine power output W turbine = 375.24947680294235 kW
high pressure turbine power output W turbine = 434.501864954904 kW
turbine power output W turbine = 809.7513417578464 kW
```

low pressure compressor power output W turbine = 151.43844484359033 kW
high pressure compressor power output W turbine = 151.43844486530872 kW
compressor power input W turbine = 302.87688970889906 kW
Net power output Wnet = 506.8744520489473 kW
Combustion chamber heat input Q combustion = 912.7466282118385 kW
Cycle efficiency eta = 0.555328758696118
Equivalent temperature carnot cycle efficiency eta = 0.8
Regenerator heat transfer = 646.9093508680925 kW
Termodynamic states :
P[1] = 1.1bar T[1] = 26.850000000000023 degree C h[1] = 398.40865520622174 kJ/kg s[1] = 2.0742421489242178 kJ/kgK
P[2] = 4.062019202bar T[2] = 176.63993033451072 degree C h[2] = 549.8471000498121 kJ/kg s[2] = 2.108519067172819 kJ/kgK
P[3] = 4.062019202bar T[3] = 26.850000000000023 degree C h[3] = 398.40865520622174 kJ/kg s[3] = 1.6992338773829843 kJ/kgK
P[4] = 15.0bar T[4] = 176.6399303557949 degree C h[4] = 549.8471000715305 kJ/kg s[4] = 1.733510795634942 kJ/kgK
P[5] = 15.0bar T[5] = 772.5581878590725 degree C h[5] = 1196.756450939623 kJ/kg s[5] = 2.6414669445343124 kJ/kgK
P[6] = 15.0bar T[6] = 1026.85 degree C h[6] = 1494.2417319275119 kJ/kg s[6] = 2.895956514615876 kJ/kgK
P[7] = 4.062019202bar T[7] = 704.5312527586585 degree C h[7] = 1118.9922551245695 kJ/kg s[7] = 2.939585733581961 kJ/kgK
P[8] = 4.062019202bar T[8] = 1226.85 degree C h[8] = 1734.253602348519 kJ/kg s[8] = 3.442655685264856 kJ/kgK
P[9] = 1.1bar T[9] = 861.6963173781736 degree C h[9] = 1299.751737393615 kJ/kg s[9] = 3.486126754877223 kJ/kgK
P[10] = 1.1bar T[10] = 276.63993035579495 degree C h[10] = 652.8423865255226 kJ/kg s[10] = 2.6901974464287823 kJ/kgK



In the previous problem, it is assumed that working fluid is air. That is not entirely true. In compressors, working fluid is air. But Following compression stage, there are combustion chambers, and fuel is combusted and exhaust gases are expanded through turbines.

$W_{lowpressurecompressor} = m_{air}(h_2 - h_1)$ (3.2.9a)

$W_{lowpressureisentropiccompressor} = m(h_{2s} - h_1) \qquad s_{2s} = s_1$ (3.2.10a)

$W_{highpressurecompressor} = m_{air}(h_4 - h_3)$ (3.2.11a)

$W_{highpressureisentropiccompressor} = m_{air}(h_{4s} - h_3) \qquad s_{4s} = s_3$ (3.2.12a)

$\eta_{isentropiccompressor} = \frac{W_{isentropiccompressor}}{W_{compressor}}$ (3.2.13a)

$Q_{intercooler} = m_{air}(h_2 - h_3)$ (3.2.14a)

$Q_{regenerator} = m_{air}(h_5 - h_4) = m_{exhaust\ gases\ 2}(h_9 - h_{10})$ (3.2.15a)

$Q_{firstcombustionchamber} = m_{exhaust\ gases\ 1}(h_6 - h_5) \qquad P_6 = P_5 = P_4$ (3.2.16a)

$W_{highpressureturbine} = m_{exhaust\ gases\ 1}(h_6 - h_7)$ (3.2.17)a

$W_{isentropichighpressureturbine} = m_{exhaust\ gases\ 1}(h_6 - h_{7i})$ (3.2.18a)

$Q_{secondcombustionchamber} = m_{exhaust\ gases\ 2}(h_8 - h_7) \qquad P_7 = P_8$ (3.2.19a)

$W_{lowpressureturbine} = m_{exhaust\ gases\ 2}(h_8 - h_9)$ (3.2.20a)

$W_{isentropiclowpressureturbine} = m_{exhaust\ gases\ 2}(h_8 - h_{9i})$ (3.2.21a)

$\eta_{isentropicturbine} = \frac{W_{turbine}}{W_{isentropicturbine}}$ (3.2.22a)

$\eta_{cycle} = \frac{W_{turbine} - W_{compressor}}{Q_{combustionchamber}}$ (3.2.23a)

Actual exhaust gas combination is due to combustion process. We will investigate combustion in later chapters. Fort his simulation, combustion exhaust gas formulations will be given.

Problem Data:

	CO_2	H_2O	N_2	O2
Exhaust gas 1	0.15	0.1	0.6	0.15
Exhaust gas 2	0.2	0.15	0.6	0.05

$$m_{air} = 1\ kg/s$$
$$m_{fuel\ 1} = 0.1\ kg/s$$
$$m_{fuel\ 2} = 0.1\ kg/s$$
$$T_1 = 300\ K$$
$$T_3 = 300\ K$$
$$P_1 = 110\ kPa$$
$$P_2 = 406.2\ kPa$$
$$P_2 = 1500\ kPa$$
$$T_6 = 1350\ K$$
$$T_8 = 1500\ K$$

```
public class brayton2_exhaust_gas
{
Gmix_PG g1,g2;
air_PG air;
double PP1[],vv1[];
double PP2[],vv2[];
double TT1[],ss1[];
double TT2[],ss2[];
```

```
double m_air,m_fuel1,m_fuel2;
double
Wturbine1,Wturbine2,Wturbine,Wcompressor1,Wcompressor2,Wcompressor,Wnet,Q1,Q
2,Qregenerator,Qintercooler,Qnet,eta;
double T[],P[],h[],s[];
public brayton2_exhaust_gas(String gname[],double gn1[],double gn2[],double
m_airi,double m_fuel1i,double m_fuel2i,double T1,double P1,double P2,double
P4,double T3,double T6,double T8,double dT,double etacompressor1,double
etacompressor2,double etaturbine1,double etaturbine2)
{  m_air=m_airi;
   m_fuel1=m_fuel1i;
   m_fuel2=m_fuel2i;
          T=new double[15];
   P=new double[15];
   h=new double[15];
   s=new double[15];
   TT1=new double[25];ss1=new double[25];
   TT2=new double[2];ss2=new double[2];
   g1=new Gmix_PG("exhaust gas" ,gname,gn1);
   g2=new Gmix_PG("exhaust gas" ,gname,gn2);
   air=new air_PG();
   g1.mole=false; //mass base calculations
   g2.mole=false; //mass base calculations
   //First processes done by air
// 1  compressor 1 inlet Working fluid: air
P[1]=P1;T[1]=T1;
P[2]=P2;
P[3]=P2;T[3]=T3;
P[4]=P4;
P[5]=P4;
P[6]=P4;
T[6]=T6;
P[7]=P2;
P[8]=P2;
T[8]=T8;
P[9]=P1;
P[10]=P1;
double a[]=air.property("tp",T[1],P[1]);
h[1]=a[3];
s[1]=a[5];
TT1[0]=T[1];ss1[0]=s[1];
//2i compressor1 isentropic outlet Working fluid: air
double s2i=s[1];
a=air.property("ps",P2,s2i);
double h2i=a[3];
double T2i=a[1];
T[11]=T2i;
s[11]=s2i;
TT1[1]=T2i;ss1[1]=s2i;
// 2 compressor1 adiabatic outlet Working fluid: air
h[2]=(h2i-h[1])/etacompressor1+h[1];
//T[2]=(h[2]-h2i)/Cp2i+T2i;
a=air.property("ph",P[2],h[2]);
s[2]=a[5];
T[2]=a[1];
```

```
// 3 1  compressor 2 inlet Working fluid: air
a=air.property("tp",T[3],P[3]);
h[3]=a[3];
s[3]=a[5];
//4i compressor2 isentropic outlet Working fluid: air
double s4i=s[3];
a=air.property("ps",P[4],s4i);
double h4i=a[3];
double T4i=a[1];
T[12]=T4i;
s[12]=s4i;
//4 compressor2 adiabatic outlet Working fluid: air
h[4]=(h4i-h[3])/etacompressor2+h[3];

//T[4]=(h[4]-h4i)/Cp4i+T4i;
a=air.property("ph",P[4],h[4]);
s[4]=a[5];
T[4]=a[1];
System.out.println("h4i="+h4i+"h4="+h[4]+"T4i="+T4i+"T4="+T[4]+"s4i="+s4i+"s4="+
s[4]+"s3="+s[3]);

//6 combustion chamber 1 outlet working fluid: exhaust gases1 (air+fuel after combustion)
a=g1.property("tp",T[6],P[6]);
h[6]=a[3];
s[6]=a[5];
//now combustion gases takes place in the cycle
//7i isentropic turbine outlet: working fluid : exhaust gases1
double s7i=s[6];
a=g1.property("ps",P[7],s7i);
double h7i=a[3];
double T7i=a[1];
T[13]=T7i;
s[13]=s7i;
//7 adiabatic turbine outlet: working fluid : exhaust gases1
h[7]=h[6]-etaturbine1*(h[6]-h7i);
//T[7]=(h[7]-h7i)/Cp7i+T7i;
a=g1.property("ph",P[7],h[7]);
T[7]=a[1];
s[7]=a[5];
//8 combustion chamber 2 outlet working fluid exhaust gases 2
a=g2.property("tp",T[8],P[8]);
h[8]=a[3];
s[8]=a[5];
//9i isentropic turbine 2 outlet  working fluid exhaust gases 2
double s9i=s[8];
a=g2.property("ps",P[9],s9i);
double h9i=a[3];
double T9i=a[1];
T[14]=T9i;
s[14]=s9i;
//9 adiabatic turbine 2 outlet  working fluid exhaust gases 2
h[9]=h[8]-etaturbine2*(h[8]-h9i);
a=g2.property("ph",P[9],h[9]);
s[9]=a[5];
T[9]=a[1];
```

```
//10 regenerator (heat exchanger) outlet
T[10]=T[4]+dT;
System.out.println("dT="+dT);
a=g2.property("tp",T[10],P[10]);
h[10]=a[3];
s[10]=a[5];
//5 regenerator (heat exchanger) outlet
h[5]=h[4]+(m_air+m_fuel1+m_fuel2)/m_air*(h[9]-h[10]);
a=air.property("ph",P[5],h[5]);
s[5]=a[5];
T[5]=a[1];
System.out.println("s[5]="+s[5]+"T[5]="+T[5]+"s[6]="+s[6]+"T[6]="+T[6]);
Wturbine1=(m_air+m_fuel1)*(h[6]-h[7]);
Wturbine2=(m_air+m_fuel1+m_fuel2)*(h[8]-h[9]);
Wturbine=Wturbine1+Wturbine2;
Wcompressor1=m_air*(h[2]-h[1]);
Wcompressor2=m_air*(h[4]-h[3]);
Wcompressor=Wcompressor1+Wcompressor2;
Q1= (m_air+m_fuel1)*h[6]-   m_air*h[5];
Q2=(m_air+m_fuel1+m_fuel2)*h[8]-(m_air+m_fuel1)*h[7];
Qregenerator=m_air*(h[5]-h[4]);
Qintercooler=m_air*(h[2]-h[3]);
Qnet=Q1+Q2;
System.out.println("Q1="+Q1+"Q2="+Q2+"Qnet="+Qnet);
Wnet=Wturbine-Wcompressor;
eta=Wnet/Qnet;
}
public double[][] line_air(double T1,double T2,double Pi,int n)
{
double TT1[]=new double[n];
double ss1[]=new double[n];
double aa[];
for(int i=0;i<n;i++)
{TT1[i]=T1+i*(T2-T1)/(n-1);
 aa=air.property("tp",TT1[i],Pi);
 ss1[i]=aa[5];
}
double a[][]={TT1,ss1};
return a;
}
public double[][] line1(double T1,double T2,double Pi,int n)
{
double TT1[]=new double[n];
double ss1[]=new double[n];
double aa[];
for(int i=0;i<n;i++)
{TT1[i]=T1+i*(T2-T1)/(n-1);
 aa=g1.property("tp",TT1[i],Pi);
 ss1[i]=aa[5];
}
double a[][]={TT1,ss1};
return a;
}
public double[][] line2(double T1,double T2,double Pi,int n)
{
```

```
double TT1[]=new double[n];
double ss1[]=new double[n];
double aa[];
for(int i=0;i<n;i++)
{TT1[i]=T1+i*(T2-T1)/(n-1);
 aa=g2.property("tp",TT1[i],Pi);
 ss1[i]=aa[5];
}
double a[][]={TT1,ss1};
return a;
}
public void plot()
{double a[][]=new double[2][61];
 double c1[][]=new double[2][3];
 double c3[][]=new double[2][3];
 double c6[][]=new double[2][3];
 double c8[][]=new double[2][3];
 c1[0][0]=T[1];
 c1[1][0]=s[1];
 c1[0][1]=T[11];
 c1[1][1]=s[11];
 c1[0][2]=T[2];
 c1[1][2]=s[2];

 c3[0][0]=T[3];
 c3[1][0]=s[3];
 c3[0][1]=T[12];
 c3[1][1]=s[12];
 c3[0][2]=T[4];
 c3[1][2]=s[4];

 c6[0][0]=T[6];
 c6[1][0]=s[6];
 c6[0][1]=T[13];
 c6[1][1]=s[13];
 c6[0][2]=T[7];
 c6[1][2]=s[7];

 c8[0][0]=T[8];
 c8[1][0]=s[8];
 c8[0][1]=T[14];
 c8[1][1]=s[14];
 c8[0][2]=T[9];
 c8[1][2]=s[9];


 int i=0;
 a[0][0]=T[1];
 a[1][0]=s[1];
 double b[][]=line_air(T[2],T[3],P[2],15);
 for(i=0;i<15;i++)
 {a[0][i+1]=b[0][i];
  a[1][i+1]=b[1][i];
 }
 b=line_air(T[4],T[5],P[4],14);
```

```
for(i=0;i<14;i++)
{a[0][i+16]=b[0][i];
 a[1][i+16]=b[1][i];
}
 b=line1(T[5],T[6],P[4],1);
for(i=0;i<1;i++)
{a[0][i+30]=b[0][i];
 a[1][i+30]=b[1][i];
}
b=line1(T[7],T[8],P[7],15);
for(i=0;i<15;i++)
{a[0][i+31]=b[0][i];
 a[1][i+31]=b[1][i];
}
 b=line2(T[9],T[10],P[9],15);
for(i=0;i<15;i++)
{a[0][i+46]=b[0][i];
 a[1][i+46]=b[1][i];
}
double TT1[]={T[1],T[2]};
double ss1[]={s[1],s[2]};

graph2D p=new graph2D();
p.addData(ss1,TT1);
double TT2[]={T[2],T[3]};
double ss2[]={s[2],s[3]};
 p.addData(ss2,TT2);
 double TT3[]={T[3],T[4]};
double ss3[]={s[3],s[4]};
 p.addData(ss3,TT3);
 double a1[][]=line_air(T[4],T[5],P[5],20);
 p.addData(a1[1],a1[0]);
 double TT4[]={T[5],T[6]};
double ss4[]={s[5],s[6]};
 p.addData(ss4,TT4);

double TT5[]={T[6],T[7]};
double ss5[]={s[6],s[7]};
 p.addData(ss5,TT5);

double TT6[]={T[7],T[8]};
double ss6[]={s[7],s[8]};
 p.addData(ss6,TT6);

double TT7[]={T[8],T[9]};
double ss7[]={s[8],s[9]};
 p.addData(ss7,TT7);

 double a2[][]=line2(T[9],T[10],P[10],20);
 p.addData(a2[1],a2[0]);

p.addData(c1[1],c1[0],1);
p.addData(c3[1],c3[0],1);
p.addData(c6[1],c6[0],1);
p.addData(c8[1],c8[0],1);
```

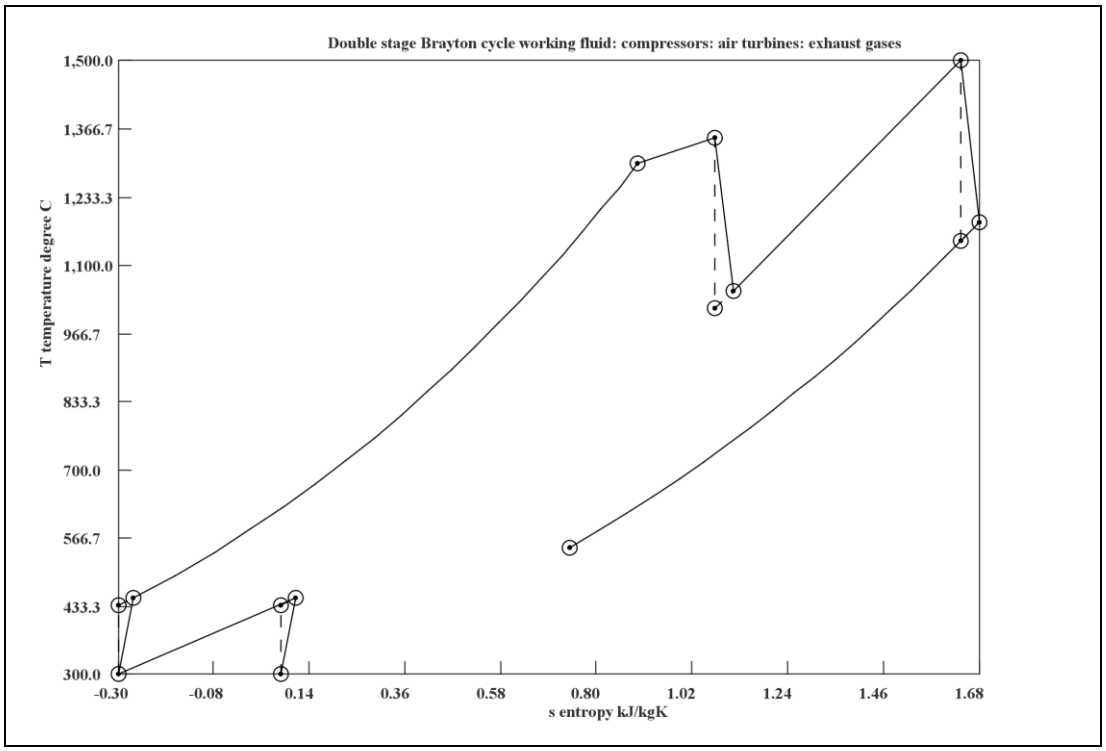
```
 double as[]=new double[14];
 double aT[]=new double[14];
 for(int ii=0;ii<14;ii++)
 {as[ii]=s[ii+1];
  aT[ii]=T[ii+1];
 }
 p.addPoint(as,aT);
 p.setPlabel(0.3,"Double stage Brayton cycle working fluid: compressors: air turbines:
exhaust gases");
 p.setXlabel("s entropy kJ/kgK");
 p.setYlabel("T temperature degree C");
 p.plot();
}

public String toString()
{
String ss=" Brayton cycle \n";
ss+="low pressure turbine power output W turbine = "+Wturbine1+" kW \n";
ss+="high pressure turbine power output W turbine = "+Wturbine2+" kW \n";
ss+="turbine power output W turbine = "+Wturbine+" kW \n";
ss+="low pressure compressor power output W turbine = "+Wcompressor1+" kW \n";
ss+="high pressure compressor power output W turbine = "+Wcompressor2+" kW \n";
ss+="compressor power input W turbine = "+Wcompressor+" kW \n";
ss+="Net power output  Wnet = "+Wnet+" kW\n";
ss+="Combustion chamber heat input Q combustion = "+Qnet+" kW\n";
ss+="Cycle efficiency eta  = "+eta+"\n";
ss+="Equivalent temperature carnot cycle efficiency eta  = "+(1.0-T[1]/T[8])+"\n";
ss+="Regenerator heat transfer   = "+Qregenerator+" kW\n";
ss+="Termodynamic states : \n";
for(int i=1;i<11;i++)
{ss+="P["+i+"] = "+P[i]+"bar T["+i+"] = "+(T[i]-273.15)+" degree C h["+i+"] = "+h[i]+"
kJ/kg s["+i+"] = "+s[i]+" kJ/kgK\n";}
return ss;
}
}
```

```
public class braytontest2_exhaust_gas
{
  public static void main(String arg[])
  {
            double m_air=1; // kg/s air inlet mass flow rate kg/s
            double m_fuel1=0.1; //kg/s inlet fuel to first  combuster
             double m_fuel2=0.1; //kg/s inlet fuel to first  combuster
            double T1=300.0; // derece K air inlet temperature
            double T3=300.0; //derece K air exit temperature
            double P1=110;// kPa air inlet pressure bar
            double P2=406.2019202;// bar primary stage pressurized air pressure bar
            double P4=1500;// bar secondary stage pressurized air pressure bar
            double T6=1350.0; //derece K turbine inlet pressure
            double T8=1500.0; //derece K turbine inlet temperature
            double dT=100.0;
            double etakompressor=0.9; //compressor isentropic efficiency
            double etaturbine=0.9; //turbine isentropic efficiency
            String name[]={"CO2","H2O","N2","O2"};
     double n1[]={0.15,0.1,0.60,0.15};
      double n2[]={0.2,0.15,0.60,0.05};
```

```
    brayton2_exhaust_gas bry=new
brayton2_exhaust_gas(name,n1,n2,m_air,m_fuel1,m_fuel2,T1,P1,P2,P4,T3,T6,T8,dT,etak
ompressor,etakompressor,etaturbine,etaturbine);
    System.out.println(bry.toString());
    bry.plot();
  }
}
```

```
---------- Capture Output ----------
> "C:\Users\Turhan
Çoban\.p2\pool\plugins\org.eclipse.justj.openjdk.hotspot.jre.full.win32.x86_64_19.0.2.v2
0230129-1123\jre\bin\javaw.exe" braytontest2_exhaust_gas
h4i=163.26154512563105h4=178.4005969467496T4i=434.7283145491478T4=449.5721
1717957273s4i=-0.304252768272461s4=-0.27001015319389177s3=-
0.304252768272461
dT=100.0
s[5]=0.8926798252859475T[5]=1299.6580461687797s[6]=1.0715478598607389T[6]=13
50.0
Q1=271.7491712925096Q2=849.3405073773072Qnet=1121.0896786698167
 Brayton cycle
low pressure turbine power output W turbine = 421.59426279539053 kW
high pressure turbine power output W turbine = 516.5596714358694 kW
turbine power output W turbine = 938.1539342312599 kW
low pressure compressor power output W turbine = 151.3905180044329 kW
high pressure compressor power output W turbine = 151.39051821118565 kW
compressor power input W turbine = 302.78103621561854 kW
Net power output  Wnet = 635.3728980156413 kW
Combustion chamber heat input Q combustion = 1121.0896786698167 kW
Cycle efficiency eta  = 0.5667458278355726
Equivalent temperature carnot cycle efficiency eta  = 0.8
Regenerator heat transfer   = 944.2024492680446 kW
Termodynamic states :
P[1] = 110.0bar T[1] = 26.850000000000023 degree C h[1] = 27.01007873556395 kJ/kg
s[1] = 0.07073960252330289 kJ/kgK
P[2] = 406.2019202bar T[2] = 176.42211699829045 degree C h[2] =
178.40059673999684 kJ/kg s[2] = 0.1049822172350971 kJ/kgK
P[3] = 406.2019202bar T[3] = 26.850000000000023 degree C h[3] = 27.01007873556395
kJ/kg s[3] = -0.304252768272461 kJ/kgK
P[4] = 1500.0bar T[4] = 176.42211717957275 degree C h[4] = 178.4005969467496 kJ/kg
s[4] = -0.27001015319389177 kJ/kgK
P[5] = 1500.0bar T[5] = 1026.5080461687799 degree C h[5] = 1122.6030462147942
kJ/kg s[5] = 0.8926798252859475 kJ/kgK
P[6] = 1500.0bar T[6] = 1076.85 degree C h[6] = 1267.5929250066397 kJ/kg s[6] =
1.0715478598607389 kJ/kgK
P[7] = 406.2019202bar T[7] = 777.2201008582625 degree C h[7] = 884.3254133744665
kJ/kg s[7] = 1.1127651937486809 kJ/kgK
P[8] = 406.2019202bar T[8] = 1226.85 degree C h[8] = 1518.41538507435 kJ/kg s[8] =
1.6365398453684004 kJ/kgK
P[9] = 110.0bar T[9] = 910.1821019932348 degree C h[9] = 1087.9489922111256 kJ/kg
s[9] = 1.6775881201112475 kJ/kgK
P[10] = 110.0bar T[10] = 276.42211717957275 degree C h[10] = 301.11361782108867
kJ/kg s[10] = 0.7348356279007722 kJ/kgK
```

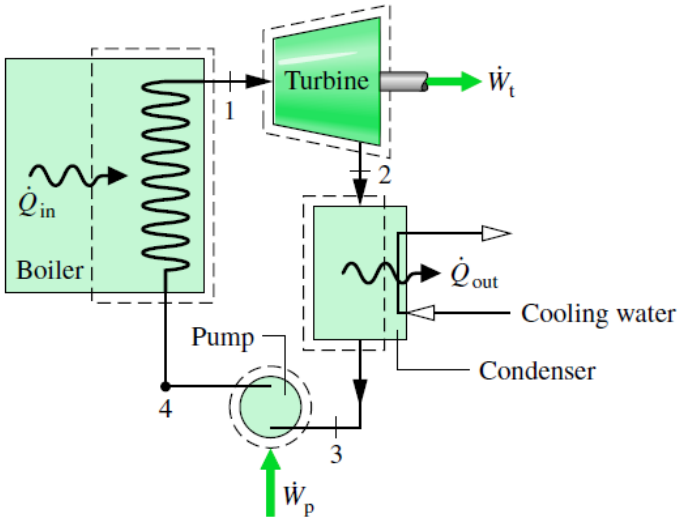


3.3 RANKINE CYCLE

In simple rankine cycle pressurised water heated up in boiler, then expands in a turbine and gives power output. Expanded steam condensed in a condenser and repressurised by using a water pump.



Willian John Rankine



$W_{pump} = m(h_4 - h_3)$ (3.3.1)

$W_{isentropicpump} = m(h_{4s} - h_3)$ $s_{4s} = s_3$ (3.3.2)

$\eta_{isentropicpump} = \frac{W_{isentropicpump}}{W_{pump}}$ (3.3.3)

$Q_{boiler} = m(h_1 - h_4)$ (3.3.4)

$W_{turbine} = m(h_1 - h_2)$ (3.3.5)

$W_{isentropicturbine} = m(h_1 - h_{2s})$ $s_1 = s_{2s}$ (3.3.6)

$\eta_{isentropic\text{-}turbine} = \frac{W_{turbine}}{W_{isentropicturbine}}$ (3.3.7)

Program 3.3.1 Simple Rankine cycle

```
public class rankine
{
// steam sınıfını kullanır
// çağrı
// steam b=new steam();
// double a[]=b.property("tp",30.0,1.01325);
// input parameters
// thermodynamic sets "tv" "tp" "th" "tu" "ts"tx"
//                "pv" "pt" "ph" "pu" "ps" "px"
// output parameters(a)
// a[0]=P a[1]=T a[2]=V a[3]=h
// a[4]u a[5]=s a[6]=x
double T[];
double P[];
double h[];
double s[];
double x[];
double etait,etaip;
double etacevrim,etaisi,etatoplam;
double m;//kg/s
double Wt,Wp,W;//iş
double Qkazan,Qyogusturucu;
steam b;
double c1[][];
double c3[][];
public rankine(double mi,double t1,double P1, double P2, double etaiti, double etaipi)
{
m=mi;
b=new steam();
```

```
T=new double[9];
P=new double[9];
h=new double[9];
s=new double[9];
x=new double[9];
T[1]=t1;//türbin girişi
P[1]=P1;
P[2]=P2;//türbin çıkış basıncı
etait=etaiti;//türbin isentropik verimi
etaip=etaipi;//pompa isentropik verimi
c1=new double[2][20];
c3=new double[2][20];
}
public void cevrimhesabi()
{
// Türbin girişi
double a1[]=b.property("tp",T[1],P[1]);
h[1]=a1[3];
s[1]=a1[5];
// isentropik Türbin çıkışı
s[0]=s[1];
P[0]=P[2];
double a0[]=b.property("ps",P[0],s[0]);
T[0]=a0[1];
h[0]=a0[3];
// gerçek türbin çıkışı
h[2]=h[1]-etait*(h[1]-h[0]);
double a2[]=b.property("ph",P[2],h[2]);
T[2]=a2[1];
x[2]=a2[6];
s[2]=a2[5];
// yoğuşturucu çıkışı
P[3]=P[2];
x[3]=0.0;
double a3[]=b.property("px",P[3],x[3]);
T[3]=a3[1];
h[3]=a3[3];
s[3]=a3[5];

// isentropik pompa
P[5]=P[1];
s[5]=s[3];
double a5[]=b.property("ps",P[5],s[5]);
h[5]=a5[3];
s[5]=a5[5];
// gerçek pompa
h[4]=h[3]+(h[5]-h[3])/etaip;
P[4]=P[1];
double a4[]=b.property("ph",P[4],h[4]);
T[4]=a4[1];
s[4]=a4[5];
// kazan kaynama  bölgesi girişi
P[6]=P[1];
x[6]=0.0;
double a6[]=b.property("px",P[6],x[6]);
T[6]=a6[1];
s[6]=a6[5];
// kazan kızgın buhar bölgesi girişi
```

```
P[7]=P[1];
x[7]=1.0;
double a7[]=b.property("px",P[7],x[7]);
T[7]=a7[1];
s[7]=a7[5];
// yoğuşturucu kızgın buhar bölgesi çıkışı
P[8]=P[2];
x[8]=1.0;
double a8[]=b.property("px",P[8],x[8]);
T[8]=a8[1];
s[8]=a8[5];

if(s[2]>s[8])
{double dt1=(T[2]-T[8])/19;
double t=T[8];
for(int i=0;i<20;i++)
{double b8[]=b.property("tp",t,P[8]);
c3[0][i]=b8[5];c3[1][i]=b8[1];t+=dt1;
}
}
double dt=(T[1]-T[7])/19;
double t=T[7];
for(int i=0;i<20;i++)
{double b7[]=b.property("tp",t,P[7]);
c1[0][i]=b7[5];c1[1][i]=b7[1];t+=dt;
}
Wt=m*(h[1]-h[2]);
Wp=m*(h[4]-h[3]);
W=Wt-Wp;
Qkazan=m*(h[1]-h[4]);
Qyogusturucu=m*(h[2]-h[3]);
etacevrim=W/Qkazan;
etaisi=Qyogusturucu/Qkazan;
etatoplam=etacevrim+etaisi;
}

public String cikti()
{
cevrimhesabi();
String s="Simple Rankine cycle\n";
s+=" cycle efficiency    = " + etacevrim+"\n";
s+=" heat energy utilisation rate = "+etaisi+"\n";
s+=" total heat+power efficiency = "+etatoplam+"\n";
s+=" Turbine power output    = " + Wt+" kW\n";
s+=" Pump power input    = " + Wp+" kW\n";
s+=" Net power output    = " + W +" kW\n";
s+=" Boiler heat input = " + Qkazan +" kW\n";
s+=" Condenser heat  = "+Qyogusturucu +" kW\n";
s+=" h1 turbine input = "+h[1] +" kJ/kg\n";
s+=" T1 turbine input = "+T[1] +" derece C\n";
s+=" P1 turbine input = "+P[1] +" bar \n";
s+=" h2 turbine output = "+h[2] +" kJ/kg\n";
s+=" T2 turbine output = "+T[3] +" derece C\n";
s+=" P2 turbine output = "+P[3] +" bar \n";
s+=" h3 condenser output = "+h[3] +" kJ/kg\n";
s+=" T3 condenser output = "+T[3] +" derece C\n";
s+=" P3 condenser output = "+P[3] +" bar \n";
s+=" h4 pump output = "+h[4] +" kJ/kg\n";
```

```
s+=" T4 pump output = "+T[4] +" derece C\n";
s+=" P4 pump output = "+P[4] +" bar \n";
return s;
}

public double[][] TS1()
{
double a[][]=Text.readDoubleT("steam.txt");
return a;
}

public double[][] TS()
{
double a[][]=new double[2][201];
double tc=b.Tc-273.15;
double dt=(tc-10.0)/100.0;
int i=0;
double t;
for(t=10.0;t<=tc;t+=dt)
{double a1[]=b.property("tx",t,0.0);
 a[1][i]=t;
 a[0][i]=a1[5];
 i++;
}
double a3[]=b.property("tx",tc,1.0);
 a[1][i]=t;
 a[0][i]=a3[5];
 i++;
for(t=tc;t>=10;t-=dt)
{double a2[]=b.property("tx",t,1.0);
 a[1][i]=t;
 a[0][i]=a2[5];
 i++;
}
return a;
}

public void plot()
{
 Plot pp=new Plot(TS1());
 double t1[]={T[1],T[2]};
 double t2[]={T[8],T[3],T[4],T[6],T[7]};
 double s1[]={s[1],s[2]};
 double s2[]={s[8],s[3],s[4],s[6],s[7]};

 pp.addData(s1,t1);
 pp.addData(c1[0],c1[1]);
 pp.addData(s2,t2);
 if(s[2]>s[8])
 { pp.addData(c3[0],c3[1]);
 }

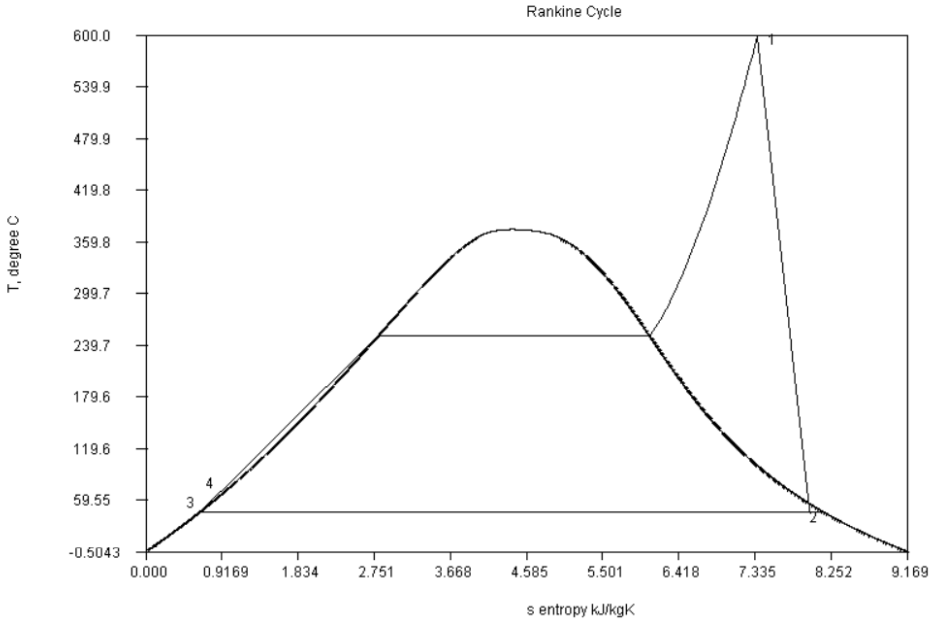
 pp.setPlabel("Rankine Cycle");
 pp.setYlabel("T, degree C");
 pp.setXlabel("s entropy kJ/kgK");
 pp.plot();
}
```

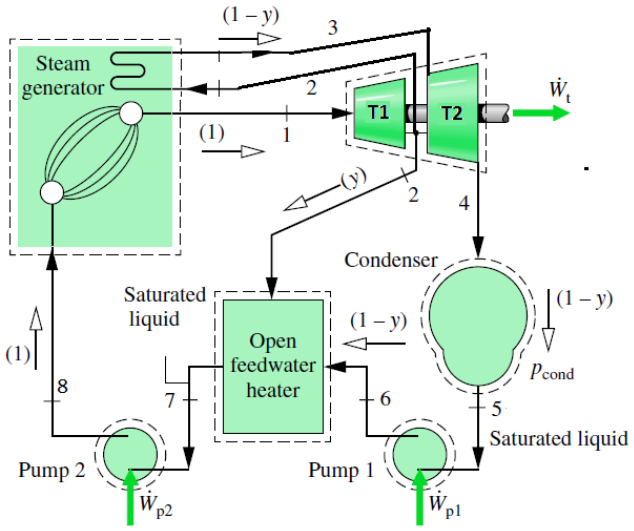
```
}
```

Program 3.3.2 Simple Rankine cycle test program

```
import javax.swing.*;

public class rankinetest
{
public static void main(String arg[])
{
double m=50/745.25928;  // kg/s çevrim su debisi
double t1=600.0;        //  derece C türbin giriş sıcaklığı
double P1=40.0;         //  bar türbin giriş basıncı
double P2=0.1 ;         //  bar türbin çıkış basıncı
double etaiturbin=0.85; //türbin isentropik verimi
double etaipompa=0.85;  //pompa isentropik verimi
rankine r=new rankine(m,t1,P1,P2,etaiturbin,etaipompa);
System.out.println(r.cikti());
r.plot();
}
}
```





In order to improve efficiency of rankine cycle, reheating can be applied. Another process to improve efficiency is feed water heating. An open water heating (mixing process) is given in the example problem

$y = \frac{m_2}{m} = \frac{m_{openfeedwaterinlet}}{m}$ (3.3.8)

$W_{pump\rightleftarrows 1} = m(1-y)(h_6 - h_5)$ (3.3.9)

$W_{isentropicpump1} = m(1-y)(h_{6i} - h_5)$ (3.3.10)

$\eta_{isentropicpump1} = \frac{W_{isentropicpump1}}{W_{pump1}}$ (3.3.11)

$W_{pump\rightleftarrows 2} = m(h_8 - h_7)$ (3.3.12)

$W_{isentropicpump2} = m(h_{8i} - h_7)$ (3.3.13)

$\eta_{isentropicpump2} = \frac{W_{isentropicpump2}}{W_{pump2}}$ (3.3.14)

open feed water heater energy balance (exit must be saturated water)

$m(1-y)h_6 + myh_2 = mh_7$ (3.3.15)

$y = \frac{h_7 - h_6}{h_2 - h_6}$ (3.3.16)

$Q_{boiler} = m(h_1 - h_8)$ (3.3.17)

$W_{turbine1} = m(h_1 - h_2)$ (3.3.18)

$W_{turbine2} = m(1-y)(h_3 - h_4)$ (3.3.19)

$W_{isentropicturbine1} = m(h_1 - h_{2i})$ (3.3.20)

$W_{isentropicturbine2} = m(h_{31} - h_{4i})$ (3.3.21)

$\eta_{isentropic\rightleftarrows turbine} = \frac{W_{turbine}}{W_{isentropicturbine}}$ (3.3.22)

$W_{net1} = W_{turbine1} + W_{turbine2} - W_{pump1} + W_{tpump2}$ (3.3.23)

$\eta_{cycle} = \frac{W_{net}}{Q_{boiler}}$ (3.3.24)

Program 3.3.3 reheated Rankine cycle with open feedwater heater

```
public class rankine2
{
// use class steam
// steam b=new steam();
// double a[]=b.property("tp",30.0,1.01325);
// inlet parameters
// termodynamic couple "tv" "tp" "th" "tu" "ts"tx"
//                "pv" "pt" "ph" "pu" "ps" "px"
```

```
// exit parameters(a) a[0]=P a[1]=T a[2]=V a[3]=h
// a[4]=u a[5]=s a[6]=x
double T[];
double P[];
double h[];
double s[];
double x[];
double etait1,etait2;
double etaip1,etaip2;
double etacevrim;
double m;
double Wt,Wt1,Wt2,Wp,Wp1,Wp2,W;
double Qkazan,Qkaynama,Qsuperheat,Qyogusturucu;
double y;//türbinden ara ısıtıcıya gönderilen buhar yüzdesi
steam b;

public rankine2(double mi,double T1,double P1, double T3,double P3,double P4,double etait1i, double
etait2i,double etaip1i,double etaip2i)
{
m=mi;
b=new steam();
T=new double[16];
P=new double[16];
h=new double[16];
s=new double[16];
x=new double[16];
//Turbine 1 inlet girişi
T[1]=T1;
P[1]=P1;
T[3]=T3;
P[3]=P3;
P[2]=P3;
P[0]=P3;//2i
P[4]=P4;
P[11]=P4;//4i
P[9]=P4;
P[5]=P4;
P[6]=P[2];
P[7]=P[6];
P[12]=P[6];//6i
P[10]=P1;
P[8]=P1;
P[13]=P1;//8i
P[14]=P1;
etait1=etait1i;
etait2=etait2i;
etaip1=etaip1i;
etaip2=etaip2i;
}
public void cevrimhesabi()
{
// turbine 1 inlet
double a1[]=b.property("tp",T[1],P[1]);
h[1]=a1[3];
s[1]=a1[5];
// isentropic turbine1 exit
s[0]=s[1];
//P[0]=P[2];
```

```
double a0[]=b.property("ps",P[0],s[0]);
T[0]=a0[1];
h[0]=a0[3];
s[0]=a0[5];
x[0]=a0[6];
// turbine1 exit
h[2]=h[1]-etait1*(h[1]-h[0]);
double a2[]=b.property("ph",P[2],h[2]);
T[2]=a2[1];
x[2]=a2[6];
s[2]=a2[5];
// turbine1 exit saturation point
x[15]=1.0;
P[15]=P[2];
double a15[]=b.property("px",P[15],x[15]);
T[15]=a15[1];
h[15]=a15[3];
s[15]=a15[5];
// Türbin 2 inlet
double a3[]=b.property("tp",T[3],P[3]);
h[3]=a3[3];
s[3]=a3[5];
// 11 : Turbine2 isentropic
s[11]=s[3];
//P[11]=P[4];
double a11[]=b.property("ps",P[11],s[11]);
T[11]=a11[1];
h[11]=a11[3];
x[11]=a11[6];
// gerçek türbin 2 çıkışı
h[4]=h[3]-etait2*(h[3]-h[11]);
double a4[]=b.property("ph",P[4],h[4]);
T[4]=a4[1];
x[4]=a4[6];
s[4]=a4[5];

// yoğuşturucu doyma buhar noktası
x[9]=1.0;
double a9[]=b.property("px",P[9],x[9]);
T[9]=a9[1];
h[9]=a9[3];
s[9]=a9[5];
// yoğuşturucu çıkışı
x[5]=0.0;
double a5[]=b.property("px",P[5],x[5]);
T[5]=a5[1];
h[5]=a5[3];
s[5]=a5[5];

// isentropik pompa 1 (12=6i)
s[12]=s[5];
double a12[]=b.property("ps",P[12],s[12]);
h[12]=a12[3];
s[12]=a12[5];
// gerçek pompa 1
h[6]=h[5]+(h[12]-h[5])/etaip1;
double a6[]=b.property("ph",P[6],h[6]);
T[6]=a6[1];
```

```
s[6]=a6[5];
//ara kızdırıcı çıkışı
x[7]=0.0;
double a7[]=b.property("px",P[7],x[7]);
T[7]=a7[1];
h[7]=a7[3];
s[7]=a7[5];

// isentropik pompa 2 (13=8i)
s[13]=s[7];
double a13[]=b.property("ps",P[13],s[13]);
h[13]=a13[3];
s[13]=a13[5];
// gerçek pompa 2
h[8]=h[7]+(h[13]-h[7])/etaip2;
double a8[]=b.property("ph",P[8],h[8]);
T[8]=a8[1];
s[8]=a8[5];

// kazan kaynama  bölgesi girişi
x[10]=0.0;
double a10[]=b.property("px",P[10],x[10]);
T[10]=a10[1];
h[10]=a10[3];
s[10]=a10[5];

// kazan kızgın buhar bölgesi çıkışı

x[14]=1.0;
double a14[]=b.property("px",P[14],x[14]);
T[14]=a14[1];
h[14]=a14[3];
s[14]=a14[5];
//
/*
if(s[2]>s[8])
{double dt1=(T[2]-T[8])/19;
double t=T[8];
for(int i=0;i<20;i++)
{double b8[]=b.property("tp",t,P[8]);
c3[0][i]=b8[5];c3[1][i]=b8[1];t+=dt1;
}
}
double dt=(T[1]-T[7])/19;
double t=T[7];
for(int i=0;i<20;i++)
{double b7[]=b.property("tp",t,P[7]);
c1[0][i]=b7[5];c1[1][i]=b7[1];t+=dt;
}
*/
y=(h[7]-h[6])/(h[2]-h[6]);
Wt1=m*(h[1]-h[2]);
Wt2=m*(1-y)*(h[3]-h[4]);
Wt=Wt1+Wt2;
Wp1=m*(1-y)*(h[6]-h[5]);
Wp2=m*(h[8]-h[7]);
Wp=Wp1+Wp2;
W=Wt-Wp;
```

```
Qkaynama=m*(h[1]-h[8]);
Qsuperheat=m*(1-y)*(h[3]-h[2]);
Qkazan=Qkaynama+Qsuperheat;
Qyogusturucu=m*(1-y)*(h[4]-h[5]);
etacevrim=W/Qkazan;
}

public String cikti()
{
cevrimhesabi();
String s="Rankine cycle with an open water heater and two stage turbine with reheating\n";
s+="percantage of steam going into water heater="+y+"\n";
s+="percantage of steam going into second turbine="+(1-y)+"\n";
s+=" system efficiency    = "+etacevrim+"\n";
s+=" Turbine 1 power exit    = "+Wt1+" kW\n";
s+=" Turbine 2 power exit  = "+Wt2+" kW\n";
s+=" Total turbine  1 power exit    = "+Wt+" kW\n";

s+=" Pump 1 inlet     = "+Wp1+" kW\n";
s+=" Pump 2 inlet      = "+Wp2+" kW\n";
s+=" Total pumppower inlet= "+Wp+" kW\n";
s+=" Net power output    = "+W +" kW\n";
s+=" Boiler heat input = "+Qkaynama +" kW\n";
s+=" Reheater heat input = "+Qsuperheat +" kW\n";
s+=" Boiler-superheater total heat input = "+Qkazan +" kW\n";
s+=" Condenser heat output = "+Qyogusturucu +" kW\n";
return s;
}

public double[][] TS1()
{
double a[][]=Text.readDoubleT("steam.txt");
return a;
}

public double[][] TS()
{
double a[][]=new double[2][201];
double tc=b.Tc-273.15;
double dt=(tc-10.0)/100.0;
int i=0;
double t;
for(t=10.0;t<=tc;t+=dt)
{double a1[]=b.property("tx",t,0.0);
 a[1][i]=t;
 a[0][i]=a1[5];
 i++;
}
double a3[]=b.property("tx",tc,1.0);
 a[1][i]=t;
 a[0][i]=a3[5];
 i++;
for(t=tc;t>=10;t-=dt)
{double a2[]=b.property("tx",t,1.0);
 a[1][i]=t;
 a[0][i]=a2[5];
 i++;
}
```

```
return a;
}

public void plot()
{
 Plot pp=new Plot(TS1());
 double TT1[],ss1[];
 double TT2[],ss2[];
 double TT3[],ss3[];
  double TT4[],ss4[];
 TT1=new double[10];
 TT2=new double[10];
 TT3=new double[10];
 TT4=new double[10];
 ss1=new double[10];
 ss2=new double[10];
 ss3=new double[10];
 ss4=new double[10];
 double a3[];
 //For more realistic TS diagram
 //constant pressure lines draw in details
 for(int i=0;i<10;i++)
 {TT1[i]=T[14]+i*(T[1]-T[14])/10;
 a3=b.property("tp",TT1[i],P[1]);
  ss1[i]=a3[5];
 }
 for(int i=0;i<10;i++)
 {TT2[i]=T[2]+i*(T[3]-T[2])/10;
 a3=b.property("tp",TT2[i],P[2]);
  ss2[i]=a3[5];
 }
 for(int i=0;i<10;i++)
 {TT3[i]=T[15]+i*(T[2]-T[15])/10;
 a3=b.property("tp",TT3[i],P[2]);
  ss3[i]=a3[5];
 }
 for(int i=0;i<10;i++)
 {TT4[i]=T[4]+i*(T[9]-T[4])/10;
 a3=b.property("tp",TT4[i],P[4]);
  ss4[i]=a3[5];
 }
 double
t1[]={T[1],TT2[0],TT2[1],TT2[2],TT2[3],TT2[4],TT2[5],TT2[6],TT2[7],TT2[8],TT2[9],T[3],T[4],TT4
[0],TT4[1],TT4[2],TT4[3],TT4[4],TT4[5],TT4[6],TT4[7],TT4[8],TT4[9],T[9],T[5],T[6],T[7],T[8],T[10
],TT1[0],TT1[1],TT1[2],TT1[3],TT1[4],TT1[5],TT1[6],TT1[7],TT1[8],TT1[9],T[1]};
 double
s1[]={s[1],ss2[0],ss2[1],ss2[2],ss2[3],ss2[4],ss2[5],ss2[6],ss2[7],ss2[8],ss2[9],s[3],s[4],ss4[0],ss4[1],ss4
[2],ss4[3],ss4[4],ss4[5],ss4[6],ss4[7],ss4[8],ss4[9],s[9],s[5],s[6],s[7],s[8],s[10],ss1[0],ss1[1],ss1[2],ss1[
3],ss1[4],ss1[5],ss1[6],ss1[7],ss1[8],ss1[9],s[1]};
 double
t2[]={T[7],T[15],TT3[0],TT3[1],TT3[2],TT3[3],TT3[4],TT3[5],TT3[6],TT3[7],TT3[8],TT3[9],T[2]};
 double s2[]={s[7],s[15],ss3[0],ss3[1],ss3[2],ss3[3],ss3[4],ss3[5],ss3[6],ss3[7],ss3[8],ss3[9],s[2]};
 double t3[]={T[1],T[0]};
 double s3[]={s[1],s[0]};
  double t4[]={T[3],T[11]};
 double s4[]={s[3],s[11]};
 pp.addData(s1,t1);
 pp.addData(s2,t2);
```

```
 pp.addData(s3,t3,1);
 pp.addData(s4,t4,1);
 pp.setPlabel("Rankine cycle");
 pp.setYlabel("T, degree C");
 pp.setXlabel("s entropy kJ/kgK");
 pp.plot();
}
}
```

Program 3.3.4 reheated Rankine cycle with open feedwater heater test program

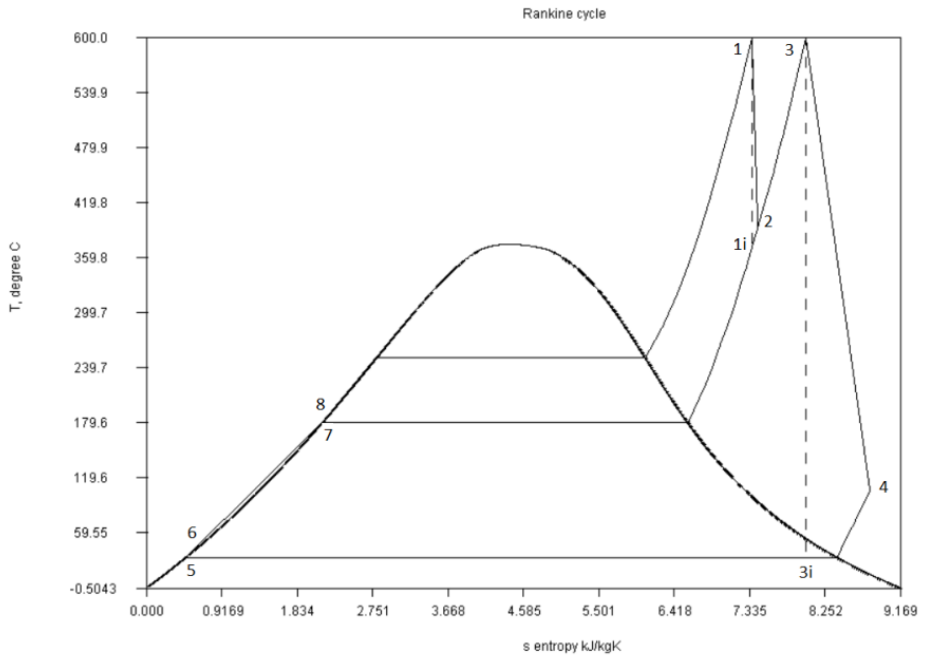
```
 import javax.swing.*;

public class rankine2test
{
public static void main(String arg[])
{
double m=1.0; //cycle mass flow rate kg/s
double T1=600.0; //first stage turbine inlet temperature C
double P1=40.0; //first stage turbine inlet pressure bar
double T2=600.0; //second stage turbine inlet temperature C
double P2=10.0; //second stage turbine inlet pressure bar
double P4=0.05; //second stage turbine exit pressure bar
double etat1=0.9; //first stage turbine isentropic efficiency
double etat2=0.8; //second stage turbine isentropic efficiency
double etap1=0.9; //first stage pump isentropic efficiency
double etap2=0.9; //second stage turbine isentropic efficiency

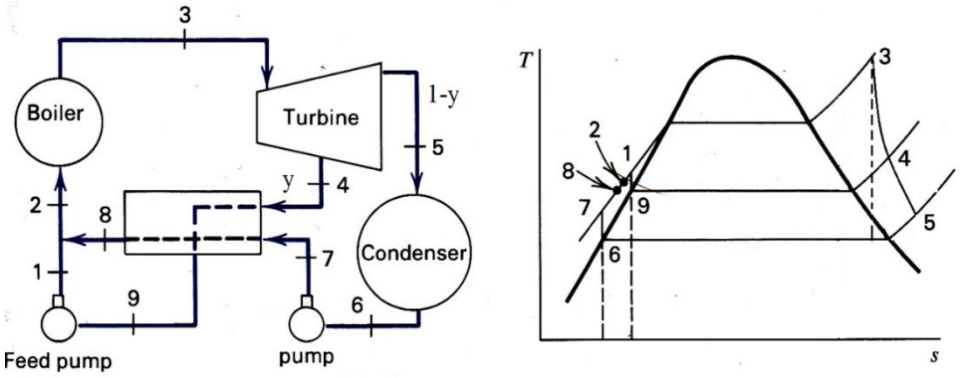
rankine2 r=new rankine2(m,T1,P1,T2,P2,P4,etat1,etat2,etap1,etap2);

System.out.println(r.cikti());
r.plot();
}
}
```

```
---------- Capture Output ----------
> "C:\java\bin\javaw.exe" rankine2test
Rankine cycle with an open water heater and two stage turbine with reheating
percantage of steam going into water heater=0.20067141771197267
percantage of steam going into second turbine=0.7993285822880274
 system efficiency    = 0.37343522490931674
 Turbine 1 power exit    = 426.48738298556054 kW
 Turbine 2 power exit  = 798.3493677724169 kW
 Total turbine  1 power exit    = 1224.8367507579774 kW
 Pump 1 inlet     = 0.8881486448834859 kW
 Pump 2 inlet      = 3.7486610800126527 kW
 Total pumppower inlet= 4.636809724896139 kW
 Net power output    = 1220.1999410330811 kW
 Boiler heat input = 2907.886599195056 kW
 Reheater heat input = 359.61432102313904 kW
 Boiler-superheater total heat input = 3267.500920218195 kW
 Condenser heat output = 2047.3009791851143 kW
```



In the next example a rankine cycle with a closed feedwater heater will be investigated.



$y = \frac{m_4}{m} = \frac{m_{extract}}{m_{boiler}}$ (3.3.25) Note that point 9 (closed feed water heater exit) should be saturated water in order to pump with the feed pump.

$W_{pump\rightleftarrows 1} = m(1-y)(h_7 - h_6)$ (3.3.26)

$W_{isentropicpump1} = m(1-y)(h_{7i} - h_6)$ (3.3.27)

$\eta_{isentropicpump1} = \frac{W_{isentropicpump1}}{W_{pump1}}$ (3.3.28)

$W_{pump\rightleftarrows 2} = m(h_1 - h_9)$ (3.3.29)

$W_{isentropicpump2} = m(h_{1i} - h_9)$ (3.3.30)

$\eta_{isentropicpump2} = \frac{W_{isentropicpump2}}{W_{pump2}}$ (3.3.31)

closed feed water heater energy balance (exit , point 9 must be saturated water)

$m(y)(h_4 - h_9) = m(1-y)(h_8 - h_7)$ (3.3.32)

Stream mix energy balance:

$(1-y)h_1 + yh_8 = h_2$ (3.3.33)

$Q_{boiler} = m(h_3 - h_2)$ (3.3.34)

$W_{turbine1} = m(h_1 - h_4)$ (3.3.35)

$W_{turbine2} = m(1-y)(h_4 - h_5)$ (3.3.36)

$W_{isentropicturbine1} = m(h_1 - h_{4i})$ (3.3.37)

$W_{isentropicturbine2} = m(h_4 - h_{5i})$ (3.3.38)

$\eta_{isentropic\text{-}turbine} = \frac{W_{turbine}}{W_{isentropicturbine}}$ (3.3.39)

$W_{net1} = W_{turbine1} + W_{turbine2} - W_{pump1} - W_{pump2}$ (3.3.40)

$\eta_{cycle} = \frac{W_{net}}{Q_{boiler}}$ (3.3.41)

Program 3.3.5 Rankine cycle with closed feedwater heater

```
public class rankine3
{
//Closed feed water heater
// use class steam
// steam b=new steam();
// double a[]=b.property("tp",30.0,1.01325);
// inlet parameters
// termodynamic couple "tv" "tp" "th" "tu" "ts"tx"
//                 "pv" "pt" "ph" "pu" "ps" "px"
// exit parameters(a) a[0]=P a[1]=T a[2]=V a[3]=h
// a[4]=u a[5]=s a[6]=x
double T[];
double P[];
double h[];
double s[];
double x[];
double etait1;
double etaip1,etaip2;
double etacycle;
double m;
double Wt,Wt1,Wt2,Wp,Wp1,Wp2,W;
double Qboiler,Qcondenser;
double y;//percent of turbine extract
double dT;
steam b;

public rankine3(double mi,double T3,double P3,double P4,double P5,double etait1i,double etaip1i,double
etaip2i,double dTi)
{ // T3,P3 turbine inlet
  // P4 clodes feed water steam extract pressure
  // P5 condenser inlet presure
  // isentropic points 1i=0 4i=10 5i=11 7i=12
  // saturation points
  // boiler saturated water: 15
  // boiler saturated steam: 16
  // condenser saturated water: 13
  // condenser saturated steam: 14
  // turbine extract saturated steam: 17
  // dT temperature difference at closefeedwater heater
m=mi;
dT=dTi;
b=new steam();
T=new double[19];
P=new double[19];
h=new double[19];
s=new double[19];
x=new double[19];
//Turbine 3 inlet
T[3]=T3;
P[3]=P[1]=P[2]=P[8]=P[7]=P[12]=P[15]=P[16]=P[0]=P3;
```

```
P[4]=P[9]=P[10]=P[17]=P4;
P[5]=P[6]=P[11]=P[13]=P[14]=P5;
etait1=etait1i;
etaip1=etaip1i;
etaip2=etaip2i;
cycle();
}

public void cycle()
{
// turbine 1 inlet
double a1[]=b.property("tp",T[3],P[3]);
h[3]=a1[3];
s[3]=a1[5];
// isentropic turbine exit 4i=10
s[10]=s[3];
a1=b.property("ps",P[10],s[10]);
T[10]=a1[1];
h[10]=a1[3];
s[10]=a1[5];
x[10]=a1[6];
// turbine exit 4
h[4]=h[3]-etait1*(h[3]-h[10]);
a1=b.property("ph",P[4],h[4]);
T[4]=a1[1];
x[4]=a1[6];
s[4]=a1[5];
// isentropic turbine exit 5i=11
s[11]=s[3];
a1=b.property("ps",P[11],s[11]);
T[11]=a1[1];
h[11]=a1[3];
s[11]=a1[5];
x[11]=a1[6];
// turbine exit 5
h[5]=h[4]-etait1*(h[4]-h[11]);
a1=b.property("ph",P[5],h[5]);
T[5]=a1[1];
x[5]=a1[6];
s[5]=a1[5];
//point 18
s[18]=s[4];
P[18]=P[5];
a1=b.property("ps",P[18],s[18]);
T[18]=a1[1];
h[18]=a1[3];
x[18]=a1[6];
// turbine extract saturated steam point 17
x[17]=1.0;
a1=b.property("px",P[17],x[17]);
T[17]=a1[1];
h[17]=a1[3];
s[17]=a1[5];
// closed feedwater exit saturated water point 9
x[9]=0.0;
a1=b.property("px",P[9],x[9]);
T[9]=a1[1];
h[9]=a1[3];
```

```
s[9]=a1[5];
// condenser saturated steam point 14
x[14]=1.0;
a1=b.property("px",P[14],x[14]);
T[14]=a1[1];
h[14]=a1[3];
s[14]=a1[5];
// condenser saturated water point 6 (condenser exit)
x[6]=0.0;
a1=b.property("px",P[6],x[6]);
T[6]=a1[1];
h[6]=a1[3];
s[6]=a1[5];
// boiler saturated steam point 16
x[16]=1.0;
a1=b.property("px",P[16],x[16]);
T[16]=a1[1];
h[16]=a1[3];
s[16]=a1[5];
// boiler saturated water point 15
x[15]=0.0;
a1=b.property("px",P[15],x[15]);
T[15]=a1[1];
h[15]=a1[3];
s[15]=a1[5];

// isentropik pompa 1 (12=7i)
s[12]=s[6];
a1=b.property("ps",P[12],s[12]);
h[12]=a1[3];
s[12]=a1[5];
// gerçek pompa 1
h[7]=h[6]+(h[12]-h[6])/etaip1;
a1=b.property("ph",P[7],h[7]);
T[7]=a1[1];
s[7]=a1[5];
// isentropik pompa 2 feedpump  (0=1i)
s[0]=s[9];
a1=b.property("ps",P[0],s[0]);
h[0]=a1[3];
s[0]=a1[5];
// real pump 1
h[1]=h[9]+(h[0]-h[9])/etaip1;
a1=b.property("ph",P[1],h[1]);
T[1]=a1[1];
s[1]=a1[5];
//State 8 is assumed to be compressed liquid
// at dT C colder than state 9
T[8]=T[9]-dT;
a1=b.property("tp",T[8],P[8]);
h[8]=a1[3];
s[8]=a1[5];
//we have enough information to solve y now
y=(h[8]-h[7])/((h[4]-h[9])+(h[8]-h[7]));
//boiler inlet condition
h[2]=h[8]*(1-y)+h[1]*y;
a1=b.property("ph",P[2],h[2]);
T[2]=a1[1];
```

```
s[2]=a1[5];

Wt1=m*(h[3]-h[4]);
Wt2=m*(1-y)*(h[4]-h[5]);
Wt=Wt1+Wt2;
Wp1=m*(1-y)*(h[7]-h[6]);
Wp2=m*y*(h[1]-h[9]);
Wp=Wp1+Wp2;
W=Wt-Wp;
Qboiler=m*(h[3]-h[2]);;
Qcondenser=m*(1-y)*(h[4]-h[5]);
etacycle=W/Qboiler;
}

public String toString()
{
cycle();
String s="Rankine cycle with a closed water heater \n";
s+="percantage of steam extracted for closed water heater="+y+"\n";
s+=" system efficiency    = "+etacycle+"\n";
s+=" Turbine 1 power exit    = "+Wt1+" kW\n";
s+=" Turbine 2 power exit  = "+Wt2+" kW\n";
s+=" Total turbine  1 power exit    = "+Wt+" kW\n";

s+=" Pump 1 inlet     = "+Wp1+" kW\n";
s+=" Pump 2 inlet      = "+Wp2+" kW\n";
s+=" Total pump power inlet= "+Wp+" kW\n";
s+=" Net power output    = "+W +" kW\n";
s+=" Boiler heat input = "+Qboiler +" kW\n";
s+=" Condenser heat output = "+Qcondenser +" kW\n";
return s;
}

public double[][] TS1()
{
double a[][]=Text.readDoubleT("steam.txt");
return a;
}

public double[][] TS()
{
double a[][]=new double[2][201];
double tc=b.Tc-273.15;
double dt=(tc-10.0)/100.0;
int i=0;
double t;
for(t=10.0;t<=tc;t+=dt)
{double a1[]=b.property("tx",t,0.0);
 a[1][i]=t;
 a[0][i]=a1[5];
 i++;
}
double a3[]=b.property("tx",tc,1.0);
 a[1][i]=t;
 a[0][i]=a3[5];
 i++;
for(t=tc;t>=10;t-=dt)
{double a2[]=b.property("tx",t,1.0);
```

```
 a[1][i]=t;
 a[0][i]=a2[5];
 i++;
 }
return a;
 }

public void plot()
{
 Plot pp=new Plot(TS1());
 double TT1[],ss1[];
 double TT2[],ss2[];
 double TT3[],ss3[];
  double TT4[],ss4[];
 TT1=new double[10];
 TT2=new double[10];
 TT3=new double[10];
 TT4=new double[10];
 ss1=new double[10];
 ss2=new double[10];
 ss3=new double[10];
 ss4=new double[10];
 double a3[];
 //For more realistic TS diagram
 //constant pressure lines draw in details

 for(int i=0;i<10;i++)
 {TT1[i]=T[16]+i*(T[3]-T[16])/9;
  a3=b.property("tp",TT1[i],P[3]);
  ss1[i]=a3[5];
 }
 for(int i=0;i<10;i++)
 {TT2[i]=T[17]+i*(T[4]-T[17])/9;
 a3=b.property("tp",TT2[i],P[4]);
  ss2[i]=a3[5];
 }
 for(int i=0;i<10;i++)
 {TT3[i]=T[14]+i*(T[5]-T[14])/9;
  a3=b.property("tp",TT3[i],P[5]);
  ss3[i]=a3[5];
 }
 double t1[]={T[3],T[4],T[5]};
 double s1[]={s[3],s[4],s[5]};
  double t2[]={T[15],T[16]};
  double s2[]={s[15],s[16]};
  double t3[]={T[9],T[17]};
  double s3[]={s[9],s[17]};
  double t4[]={T[6],T[14]};
  double s4[]={s[6],s[14]};
  double t5[]={T[6],T[7],T[8],T[2],T[1],T[15]};
  double s5[]={s[6],s[7],s[8],s[2],s[1],s[15]};
  double t6[]={T[3],T[11]};
  double s6[]={s[3],s[11]};
  double t7[]={T[4],T[18]};
  double s7[]={s[4],s[18]};
  pp.addData(ss1,TT1);
  pp.addData(ss2,TT2);
  pp.addData(ss3,TT3);
```

```
  pp.addData(s1,t1);
  pp.addData(s2,t2);
  pp.addData(s3,t3);
  pp.addData(s4,t4);
  pp.addData(s5,t5);
  pp.addData(s6,t6);
  pp.addData(s7,t7);
  pp.setPlotType(10,4);
  pp.setPlotType(9,4);
  pp.setPlabel("Rankine cycle with closed feedwater heater");
  pp.setYlabel("T, degree C");
  pp.setXlabel("s entropy kJ/kgK");
  pp.plot();
}

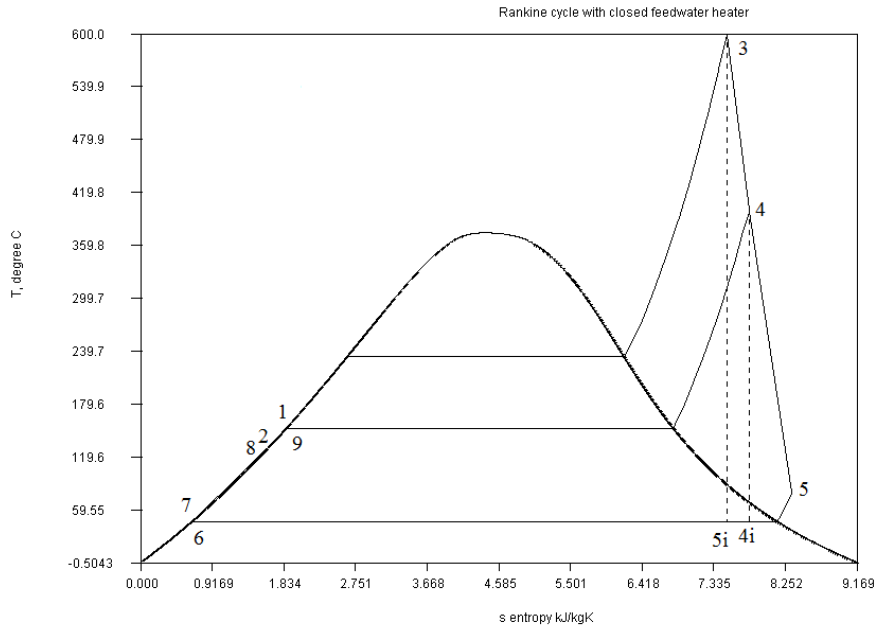
}
```

Program 3.3.6 Rankine cycle with closed feedwater heater test program

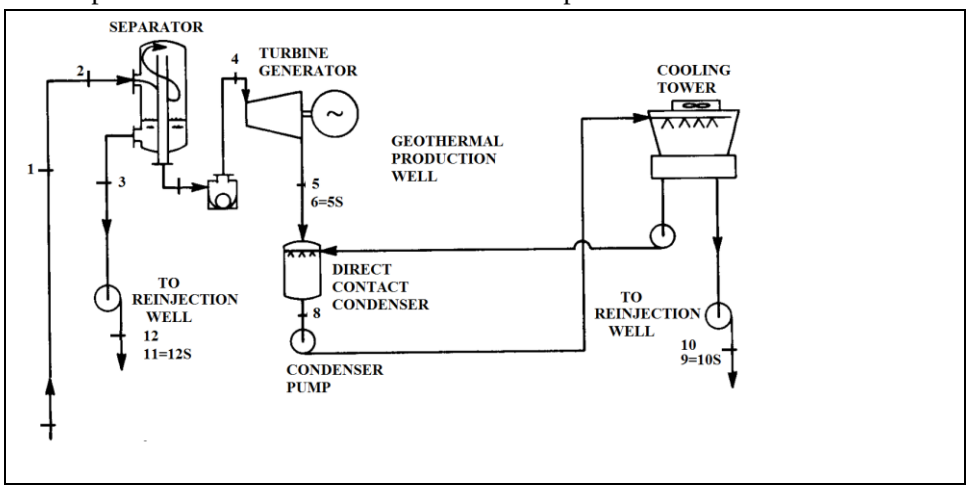
```
import javax.swing.*;

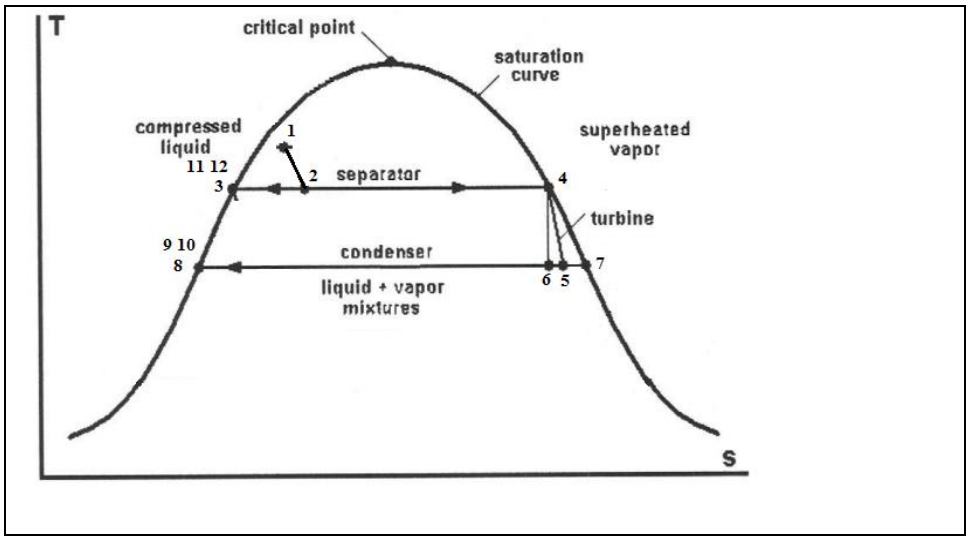
public class rankine3test
{
public static void main(String arg[])
{
double m=1.0; //cycle mass flow rate kg/s
double T3=600.0; //first stage turbine inlet temperature C
double P3=30.0; //first stage turbine inlet pressure bar
double P4=5.0; //second stage turbine inlet pressure bar
double P5=0.1; //second stage turbine exit pressure bar
double etat1=0.7; //first stage turbine isentropic efficiency
double etap1=0.8; //first stage pump isentropic efficiency
double etap2=0.8; //second stage turbine isentropic efficiency
double dT=5.0;
rankine3 r=new rankine3(m,T3,P3,P4,P5,etat1,etap1,etap2,dT);
System.out.println(r.toString());
r.plot();
}
}
```

```
---------- Capture Output ----------
> "C:\java\bin\java.exe" rankine3test
Rankine cycle with a closed water heater
percantage of steam extracted for closed water heater=0.13905546051502635
 system efficiency    = 0.3090315973727121
 Turbine 1 power exit    = 412.9618177734387 kW
 Turbine 2 power exit  = 536.0413853480288 kW
 Total turbine  1 power exit    = 949.0032031214675 kW
 Pump 1 inlet     = 3.248672415293518 kW
 Pump 2 inlet      = 0.4738381237412723 kW
 Total pump power inlet= 3.72251053903479 kW
 Net power output    = 945.2806925824327 kW
 Boiler heat input = 3058.8480291947726 kW
 Condenser heat output = 536.0413853480288 kW
```



Low enthalpy rankine cycles are challanging processes due to low efficiency. For example let us investigate a flash type wet steam geothermal power plant. This is an open cycle as far as working fluid is concerned. Geothermal wet steam is found as high pressure , medium temperature source from the production wells. This wet steam is sent to a separator. Steam and water is separated and steam sent to a turbine to expand and create work.





Wet steam inlet condition: P_1, x_1 (Could be in liquid region as well)

Separator inlet $P_2, h_2 = h_1$ (3.3.42)

$h_2 = (1 - x_2)h_3 + x_2 h_4$ (3.3.43)

$W_{turbine} = mx_2(h_4 - h_5)$ (3.3.44)

$W_{isentropicturbine} = mx(h_4 - h_6)$ (3.3.45)

$\eta_{isentropicturbine} = \frac{W_{turbine}}{W_{isentropicturbine}}$ (3.3.46)

$W_{net} = W_{turbine} - W_{reinjectionpump} - W_{condenspump} - W_{coolingtowerpump}$ (3.3.47)

$W_{reinjectionpump} = m(1 - x_2) * (h_{12} - h_3) + mx_2(h_8 - h_{10})$ (3.3.48)

Since the system is an open system, classical system efficiency have no significant meaning in here, so we can use exergetic efficiency here. System exergy (Maximum available work potential)

$W[(h_1 - h_0) - T_0(s_1 - s_0)]_{max}$ (3.3.49)

$\eta_{exergy} = \frac{W_{net}}{W_{max}}$ (3.3.50)

Of course a virtual boiler can be considered to define inlet heat transfer and so a virtual system efficiency can be described

$Q_{virtualboiler} = m[(h_1 - h_8)]$ (3.3.51)

$\eta_{virtualsystem} = \frac{W_{net}}{Q_{virtualboiler}}$ (3.3.52)

Program 3.3.7 Flash geothermal power plant

```
public class flash1
{
// steam sınıfını kullanır
// çağrı
// steam b=new steam();
// double a[]=b.property("tp",30.0,1.01325);
// girdi parametreleri
// termodinamik set "tv" "tp" "th" "tu" "ts"tx"
//                  "pv" "pt" "ph" "pu" "ps" "px"
// çıkış parametreleri(a)
// a[0]=P a[1]=T a[2]=V a[3]=h
// a[4]u a[5]=s a[6]=x
double T[];
double P[];
double v[];
```

```
double h[];
double s[];
double x[];
double etait,etaip;
double etacycle,etaexergy;
double m;//kg/s well production
double Wmax,Wt,Wp,Wp1,Wp2,W;//work
double Qboiler,Qcondenser;
steam b;
double c1[][];
double c3[][];
public flash1(double mi,double xi,double P1, double P2, double P3,double etaiti, double etaipi,double
T0,double Preinjection)
{
m=mi;
b=new steam();
T=new double[13];
P=new double[13];
h=new double[13];
s=new double[13];
x=new double[13];
v=new double[13];
x[1]=xi;//uürbine inlet
P[10]=Preinjection;
P[1]=P1;
P[2]=P[3]=P[4]=P2;
P[5]=P[6]=P[7]=P[8]=P3;
T[0]=T0;
etait=etaiti;//turbine isentropic efficiency
etaip=etaipi;//pompa isentropc efficiency
}
public void cycle()
{
//geothermal production well exit Point 1
double a[]=b.property("px",P[1],x[1]);
T[1]=a[1];
h[1]=a[3];
s[1]=a[5];
//separator inlet point 2
h[2]=h[1];
a=b.property("ph",P[2],h[2]);
T[2]=a[1];
x[2]=a[6];
s[2]=a[5];
// Turbine inlet 4
T[4]=T[2];
x[4]=1.0;
a=b.property("tx",T[4],x[4]);
h[4]=a[3];
s[4]=a[5];
// isentropic Turbine exit 6
s[6]=s[4];
a=b.property("ps",P[6],s[6]);
T[6]=a[1];
h[6]=a[3];
// real turbine exit 5
h[5]=h[4]-etait*(h[4]-h[6]);
a=b.property("ph",P[5],h[5]);
```

```
T[5]=a[1];
x[5]=a[6];
s[5]=a[5];
// condenser exit 8
x[8]=0.0;
a=b.property("px",P[8],x[8]);
T[8]=a[1];
h[8]=a[3];
s[8]=a[5];
v[8]=a[2];
// condenser saturated steam point 7
x[7]=1.0;
a=b.property("px",P[7],x[7]);
T[7]=a[1];
h[7]=a[3];
s[7]=a[5];
//Point 9=10s
// P8-Preinjection isentropic reinjection pump
P[9]=P[10];
if(P[9]<P[1]) System.out.println("Reinjection pressure is not sufficient");
s[9]=s[8];
a=b.property("ps",P[9],s[9]);
h[9]=a[3];
T[9]=a[1];
s[9]=a[5];
//h[9]=h[8]+v[8]*(P[9]-P[8]);
//Point 10
// P9-Preinjection reinjection pump exit
h[10]=h[8]+(h[9]-h[8])/etaip;
a=b.property("ph",P[10],h[10]);
h[10]=a[3];
T[10]=a[1];
s[10]=a[5];
// virtual boiler saturated water 3
P[3]=P[2];
x[3]=0.0;
a=b.property("px",P[3],x[3]);
T[3]=a[1];
h[3]=a[3];
s[3]=a[5];
v[3]=a[2];
//Point 11
// P3-Preinjection isentropic reinjection pump
P[11]=P[10];
if(P[11]<P[1]) System.out.println("Reinjection pressure is not sufficient");
s[11]=s[3];
a=b.property("ps",P[11],s[11]);
h[11]=h[3]+v[3]*(P[11]-P[3]);
//h[11]=a[3];
//Point 12
// P3-Preinjection reinjection pump exit
h[12]=h[3]+(h[11]-h[3])/etaip;
P[12]=P[10];
a=b.property("ph",P[12],h[12]);
T[12]=a[1];
s[12]=a[5];
//Point 0 environmental condition
P[0]=1.01325; //bar
```

```
a=b.property("tp",T[0],P[0]);
h[0]=a[3];
s[0]=a[5];
Wt=m*x[2]*(h[4]-h[5]);
Wp1=m*x[2]*(h[10]-h[8]);
Wp2=m*(1.0-x[2])*(h[12]-h[3]);
Wp=Wp1+Wp2;
W=Wt-Wp;
Qboiler=m*(h[1]-h[8]);
Qcondenser=m*x[2]*(h[5]-h[8]);
etacycle=W/Qboiler;
Wmax=m*((h[1]-h[0])-(T[0]+273.15)*(s[1]-s[0]));
etaexergy=W/Wmax;
}

public String toString()
{
cycle();
String s="single flash geothermal power plant\n";
s+=" cycle efficiency    = " + etacycle+"\n";
s+=" cycle exergy efficiency    = " + etaexergy+"\n";
s+=" Exergy = "+Wmax+"kW\n";
s+=" Turbine power output    = " + Wt+" kW\n";
s+=" Reenjection pump 1 power input    = " + Wp1+" kW\n";
s+=" Reenjection pump 2 power input    = " + Wp2+" kW\n";
s+=" Reenjection pump power input    = " + Wp+" kW\n";
s+=" Net power output    = " + W +" kW\n";
s+=" Virtual boiler heat input = " + Qboiler +" kW\n";
s+=" Condenser heat removal  = "+Qcondenser +" kW\n";
s+=" h1  well head = "+h[1] +" kJ/kg\n";
s+=" T1  well head = "+T[1] +" derece C\n";
s+=" P1  well head = "+P[1] +" bar \n";
s+=" x1  well head = "+x[1] +" kg vapor/kg \n";
s+=" h2 separator inlet = "+h[2] +" kJ/kg\n";
s+=" T2 separator inlet = "+T[3] +" derece C\n";
s+=" P2 separator inlet = "+P[3] +" bar \n";
s+=" x2  separator inlet = "+x[2] +" kg vapor/kg \n";
s+=" h3 separator exit (saturated water) = "+h[3] +" kJ/kg\n";
s+=" T3 separator exit (saturated water) ="+T[3] +" derece C\n";
s+=" P3 separator exit (saturated water) ="+P[3] +" bar \n";
s+=" x3   separator exit = "+x[3] +" kg vapor/kg \n";
s+=" h4 turbine inlet ="+h[4] +" kJ/kg\n";
s+=" T4 turbine inlet = "+T[4] +" derece C\n";
s+=" P4 turbine inlet = "+P[4] +" bar \n";
s+=" x4 turbine inlet = "+x[4] +" kg vapor/kg \n";
s+=" h5 turbine exit ="+h[5] +" kJ/kg\n";
s+=" T5 turbine exit = "+T[5] +" derece C\n";
s+=" P5 turbine exit = "+P[5] +" bar \n";
s+=" x5 turbine inlet = "+x[4] +" kg vapor/kg \n";
s+=" h6 isentropic turbine exit ="+h[5] +" kJ/kg\n";
s+=" T6 isentropic turbine exit = "+T[5] +" derece C\n";
s+=" P6 isentropic turbine exit = "+P[5] +" bar \n";
s+=" x6 isentropic turbine inlet = "+x[4] +" kg vapor/kg \n";
s+=" h7 condenser saturated vapor ="+h[5] +" kJ/kg\n";
s+=" T7 condenser saturated vapor = "+T[5] +" derece C\n";
s+=" P7 condenser saturated vapor = "+P[5] +" bar \n";
s+=" x7 condenser saturated vapor = "+x[4] +" kg vapor/kg \n";
s+=" h8 condenser saturated liquid ="+h[8] +" kJ/kg\n";
```

```
s+=" T8 condenser saturated liquid = "+T[8] +" derece C\n";
s+=" P8 condenser saturated liquid = "+P[8] +" bar \n";
s+=" x8 condenser saturated liquid = "+x[8] +" kg vapor/kg \n";
s+=" h9 isentropic reinjection pump exit ="+h[9] +" kJ/kg\n";
s+=" T9 isentropic reinjection pump exit = "+T[9] +" derece C\n";
s+=" P9 isentropic reinjection pump exit = "+P[9] +" bar \n";
s+=" x9 isentropic reinjection pump exit = "+x[9] +" kg vapor/kg \n";
s+=" h10  reinjection pump exit ="+h[10] +" kJ/kg\n";
s+=" T10  reinjection pump exit = "+T[10] +" derece C\n";
s+=" P10  reinjection pump exit = "+P[10] +" bar \n";
s+=" x10  reinjection pump exit = "+x[10] +" kg vapor/kg \n";
s+=" h11 isentropic reinjection pump exit ="+h[11] +" kJ/kg\n";
s+=" T11 isentropic reinjection pump exit = "+T[11] +" derece C\n";
s+=" P11 isentropic reinjection pump exit = "+P[11] +" bar \n";
s+=" x11 isentropic reinjection pump exit = "+x[11] +" kg vapor/kg \n";
s+=" h12  reinjection pump exit ="+h[12] +" kJ/kg\n";
s+=" T12  reinjection pump exit = "+T[12] +" derece C\n";
s+=" P12  reinjection pump exit = "+P[12] +" bar \n";
s+=" x12  reinjection pump exit = "+x[12] +" kg vapor/kg \n";
return s;
}

public double[][] TS1()
{
double a[][]=Text.readDoubleT("steam.txt");
return a;
}

public double[][] TS()
{
double a[][]=new double[2][201];
double tc=b.Tc-273.15;
double dt=(tc-10.0)/100.0;
int i=0;
double t;
for(t=10.0;t<=tc;t+=dt)
{double a1[]=b.property("tx",t,0.0);
 a[1][i]=t;
 a[0][i]=a1[5];
 i++;
}
double a3[]=b.property("tx",tc,1.0);
 a[1][i]=t;
 a[0][i]=a3[5];
 i++;
for(t=tc;t>=10;t-=dt)
{double a2[]=b.property("tx",t,1.0);
 a[1][i]=t;
 a[0][i]=a2[5];
 i++;
}
return a;
}

public void plot()
{
 Plot pp=new Plot(TS1());
 double t1[]={T[1],T[2],T[4],T[5],T[8]};
```

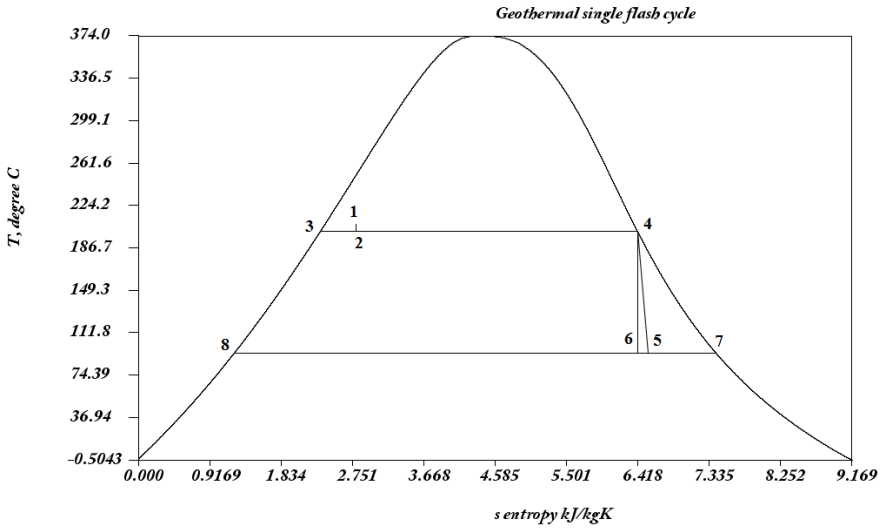
```
 double s1[]={s[1],s[2],s[4],s[5],s[8]};
 double t2[]={T[5],T[7]};
 double s2[]={s[5],s[7]};
 double t3[]={T[3],T[2]};
 double s3[]={s[3],s[2]};
 double t4[]={T[4],T[6]};
 double s4[]={s[4],s[6]};
 pp.addData(s1,t1);
 pp.addData(s2,t2);
 pp.addData(s3,t3);
 pp.addData(s4,t4);
 pp.setPlabel("Geothermal single flash cycle");
 pp.setYlabel("T, degree C");
 pp.setXlabel("s entropy kJ/kgK");
 pp.plot();
 }

 }
```

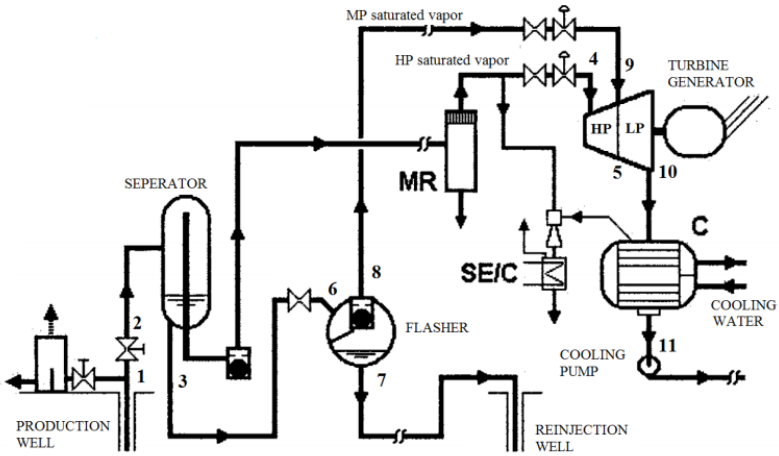
```
public class flash1_cevrimi_test
{
public static void main(String arg[])
{ //(double mi,double xi,double P1, double P2, double P3,double etaiti, double etaipi,double T0,double
Preinjection)
flash1 rc=new flash1(20.0,0.0,19.0,14.0,0.80,0.9,0.9,25.0,20.0);
System.out.println(rc.toString());
rc.plot();
}
}
```

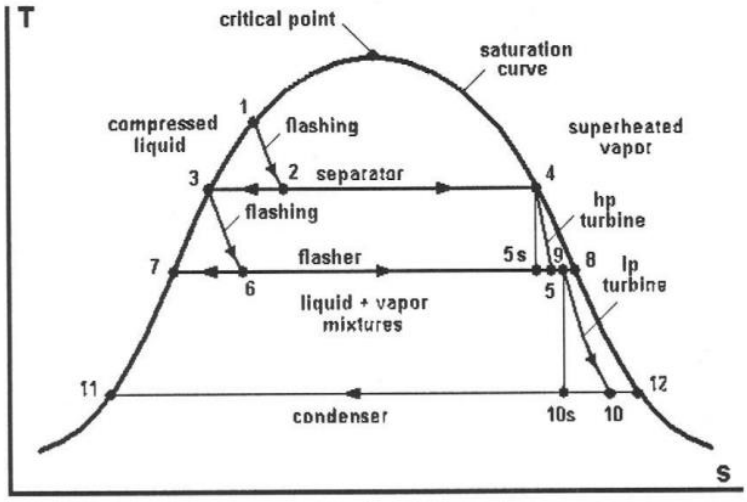
```
---------- Capture Output ----------
> "C:\java\bin\java.exe" flash1_cevrimi_test
single flash geothermal power plant
 cycle efficiency    = 0.07339140495014446
 cycle exergy efficiency    = 0.20344173788400155
 Exergy = 4937.7658660852485kW
 Turbine power output    = 1009.611247905253 kW
 Reenjection pump 1 power input    = 4.972145722760613 kW
 Reenjection pump 2 power input    = 0.0914331218072619 kW
 Reenjection pump power input    = 5.063578844567875 kW
 Net power output    = 1004.547669060685 kW
 Virtual boiler heat input = 13687.538339715456 kW
 Condenser heat removal  = 4383.921267972627 kW
 h1  well head = 1076.0391527789507 kJ/kg
 T1  well head = 207.15135070759823 derece C
 P1  well head = 18.0 bar
 x1  well head = 0.1 kg vapor/kg
 h2 separator inlet = 1076.0391527789507 kJ/kg
 T2 separator inlet = 201.40793020719082 derece C
 P2 separator inlet = 16.0 bar
 x2  separator inlet = 0.11225388414973327 kg vapor/kg
 h3 separator exit (saturated water) = 858.8006256859758 kJ/kg
 T3 separator exit (saturated water) =201.40793020719082 derece C
 P3 separator exit (saturated water) =16.0 bar
 x3   separator exit = 0.0 kg vapor/kg
 h4 turbine inlet =2794.0434793159193 kJ/kg
 T4 turbine inlet = 201.40793020719082 derece C
```

P4 turbine inlet = 16.0 bar
x4 turbine inlet = 1.0 kg vapor/kg
h5 turbine exit =2344.343562225054 kJ/kg
T5 turbine exit = 93.50435536185492 derece C
P5 turbine exit = 0.8 bar
x5 turbine inlet = 1.0 kg vapor/kg
h6 isentropic turbine exit =2344.343562225054 kJ/kg
T6 isentropic turbine exit = 93.50435536185492 derece C
P6 isentropic turbine exit = 0.8 bar
x6 isentropic turbine inlet = 1.0 kg vapor/kg
h7 condenser saturated vapor =2344.343562225054 kJ/kg
T7 condenser saturated vapor = 93.50435536185492 derece C
P7 condenser saturated vapor = 0.8 bar
x7 condenser saturated vapor = 1.0 kg vapor/kg
h8 condenser saturated liquid =391.66223579317784 kJ/kg
T8 condenser saturated liquid = 93.50435536185492 derece C
P8 condenser saturated liquid = 0.8 bar
x8 condenser saturated liquid = 0.0 kg vapor/kg
h9 isentropic reinjection pump exit =393.6554546400712 kJ/kg
T9 isentropic reinjection pump exit = 93.62822347328785 derece C
P9 isentropic reinjection pump exit = 20.0 bar
x9 isentropic reinjection pump exit = 0.0 kg vapor/kg
h10 reinjection pump exit =393.8769234008371 kJ/kg
T10 reinjection pump exit = 93.68090134423034 derece C
P10 reinjection pump exit = 20.0 bar
x10 reinjection pump exit = 0.0 kg vapor/kg
h11 isentropic reinjection pump exit =858.8052604463063 kJ/kg
T11 isentropic reinjection pump exit = 0.0 derece C
P11 isentropic reinjection pump exit = 20.0 bar
x11 isentropic reinjection pump exit = 0.0 kg vapor/kg
h12 reinjection pump exit =858.8057754196764 kJ/kg
T12 reinjection pump exit = 200.74087440542496 derece C
P12 reinjection pump exit = 20.0 bar
x12 reinjection pump exit = 0.0 kg vapor/kg



As it is seen from the results, cycle efficiencies are not very impressive, they are relatively low. To improve cycle efficiencies of geothermal flash cycle a little bit, two stage flash cycle can be utilized.





If geothermal fluid mass flow rate is m, mass flow rates in systems can be given as:

$mx_2 = m_4 = m_5$ (3.3.53)

$m(1 - x_2) = m_3 = m_6$ (3.3.54)

$m_8 = (1 - x_2)x_6 m$ (3.3.55)

$m_7 = (1 - x_2)(1 - x_6)m$ (3.3.56)

$m_9 = m(x_2 + (1 - x_2)x_6)$ (3.3.57)

Wet steam inlet condition: P_1, x_1 (Could be in liquid region as well)

Separator inlet $P_2, h_2 = h_1$ (3.3.58)

$h_2 = (1 - x_2)h_3 + x_2 h_4$ (3.3.59)

Flasher point 6: $P_6, h_3 = h_6$ (3.3.60)

$h_6 = (1 - x_6)h_7 + x_6 h_8$ (3.3.61)

$W_{turbineHP} = mx_2(h_4 - h_5)$ (3.3.62)

$W_{isentropicturbine\rightleftarrows\rightleftarrows\rightleftarrows HP} = mx_2(h_4 - h_{17})$ (3.3.63)

$\eta_{isentropicturbine\rightleftarrows\rightleftarrows HP} = \frac{W_{turbine\rightleftarrows HP}}{W_{isentropicturbine\rightleftarrows\rightleftarrows HP}}$ (3.3.64)

Mixing process:

$m_5h_5 + m_8h_8 = m_9h_9 = (m_5 + m_8)h_9$ (3.3.65)
$mx_2h_5 + m(1 - x_2)x_6h_8 = m(x_2 + (1 - x_2)x_6)h_9$ (3.3.66)
$W_{turbineLP} = m_9(h_9 - h_{10})$ (3.3.67)
$W_{isentropicturbine\rightleftarrows\rightleftarrows\rightleftarrows LP} = m_9(h_9 - h_{18})$ (3.3.68)
$\eta_{isentropicturbine\rightleftarrows\rightleftarrows LP} = \frac{W_{turbineLP\rightleftarrows}}{W_{isentropicturbine\rightleftarrows\rightleftarrows LP}}$ (3.3.69)

$W_{net} = W_{turbine} - W_{reinjectionpump} - W_{condenspump} - W_{coolingtowerpump}$ (3.3.70)
$W_{reinjectionpump} = m_9(h_{13} - h_{11}) + m_7(h_{15} - h_7)$ (3.3.71)

Since the system is an open system, classical system efficiency have no significant meaning in here, so we can use exergetic efficiency here. System exergy (Maximum available work potential)
$W[(h_1 - h_0) - T_0(s_1 - s_0)]_{max}$ (3.3.72)
$\eta_{exergy} = \frac{W_{net}}{W_{max}}$ (3.3.73)
Of course a virtual boiler can be considered to define inlet heat transfer and so a virtual system efficiency can be described
$Q_{virtualboiler} = m[(h_1 - h_{11})]$ (3.3.74)
$\eta_{virtualsystem} = \frac{W_{net}}{Q_{virtualboiler}}$ (3.3.75)

Program 3.3.8 Two stage flash geothermal power plant

```
public class flash2
{
// steam sınıfını kullanır
// çağrı
// steam b=new steam();
// double a[]=b.property("tp",30.0,1.01325);
// girdi parametreleri
// termodinamik set "tv" "tp" "th" "tu" "ts"tx"
//                  "pv" "pt" "ph" "pu" "ps" "px"
// çıkış parametreleri(a)
// a[0]=P a[1]=T a[2]=V a[3]=h
// a[4]u a[5]=s a[6]=x
double T[];
double P[];
double v[];
double h[];
double s[];
double x[];
double etait1,etait2,etaip;
double etacycle,etaexergy;
double m;//kg/s well production
double Wmax,Wt,Wt1,Wt2,Wp,Wp1,Wp2,W;//work
double Qboiler,Qcondenser;
steam b;
double c1[][];
double c3[][];
double m4,m5,m3,m6,m7,m8,m9;
public flash2(double mi,double xi,double P1, double P2, double P3,double P4,double etait1i, double
etait2i,double etaipi,double T0,double Preinjection)
{
m=mi;
b=new steam();
T=new double[19];
P=new double[19];
```

```
h=new double[19];
s=new double[19];
x=new double[19];
v=new double[19];
x[1]=xi;//turbine inlet
P[1]=P1;
P[2]=P[3]=P[4]=P2;
P[5]=P[6]=P[7]=P[8]=P[9]=P[17]=P3;
P[10]=P[11]=P[12]=P[18]=P4;
T[0]=T0;
P[13]=P[14]=P[15]=P[16]=Preinjection;
etait1=etait1i;//turbine isentropic efficiency
etait2=etait2i;
etaip=etaipi;//pump isentropic efficiency
}
public void cycle()
{
//geothermal production well exit Point 1
double a[]=b.property("px",P[1],x[1]);
T[1]=a[1];
h[1]=a[3];
s[1]=a[5];
//separator inlet point 2
h[2]=h[1];
a=b.property("ph",P[2],h[2]);
T[2]=a[1];
x[2]=a[6];
s[2]=a[5];
// virtual boiler saturated water 3
x[3]=0.0;
a=b.property("px",P[3],x[3]);
T[3]=a[1];
h[3]=a[3];
s[3]=a[5];
v[3]=a[2];
// Turbine inlet 4
T[4]=T[2];
x[4]=1.0;
a=b.property("tx",T[4],x[4]);
h[4]=a[3];
s[4]=a[5];
// isentropic Turbine exit 17
s[17]=s[4];
a=b.property("ps",P[17],s[17]);
T[17]=a[1];
h[17]=a[3];
// real turbine exit 5
h[5]=h[4]-etait1*(h[4]-h[17]);
a=b.property("ph",P[5],h[5]);
T[5]=a[1];
x[5]=a[6];
s[5]=a[5];
// flash point 6
h[6]=h[3];
a=b.property("ph",P[6],h[6]);
T[6]=a[1];
x[6]=a[6];
s[6]=a[5];
```

```
m4=m5=m*x[2];
m3=m6=m*(1-x[2]);
m8=m*(1-x[2])*x[6];
m9=m*(x[2]+(1-x[2])*x[6]);
//point 7 saturated water
x[7]=0.0;
a=b.property("px",P[7],x[7]);
T[7]=a[1];
s[7]=a[5];
//point 8 saturated steam
x[8]=1.0;
a=b.property("px",P[8],x[8]);
T[8]=a[1];
s[8]=a[5];
h[8]=a[3];
//point 9 LP turbine inlet point
h[9]=(m5*h[5]+m8*h[8])/(m4+m8);
a=b.property("ph",P[9],h[9]);
T[9]=a[1];
s[9]=a[5];
// isentropic Turbine exit 18=10s
s[18]=s[9];
a=b.property("ps",P[18],s[18]);
T[18]=a[1];
h[18]=a[3];
// real turbine exit 10
h[10]=h[9]-etait2*(h[9]-h[18]);
a=b.property("ph",P[10],h[10]);
T[10]=a[1];
x[10]=a[6];
s[10]=a[5];
// condenser exit 11
x[11]=0.0;
a=b.property("px",P[11],x[11]);
T[11]=a[1];
h[11]=a[3];
s[11]=a[5];
v[11]=a[2];
// condenser saturated steam point 12
x[12]=1.0;
a=b.property("px",P[12],x[12]);
T[12]=a[1];
h[12]=a[3];
s[12]=a[5];
// P11-Preinjection isentropic reinjection pump
if(P[13]<P[1]) System.out.println("Reinjection pressure is not sufficient");
s[13]=s[11];
a=b.property("ps",P[13],s[13]);
h[13]=a[3];
T[13]=a[1];
s[13]=a[5];
//h[9]=h[8]+v[8]*(P[9]-P[8]);
//Point 14
// P7-Preinjection reinjection pump exit
h[14]=h[11]+(h[13]-h[11])/etaip;
a=b.property("ph",P[14],h[14]);
h[14]=a[3];
T[14]=a[1];
```

```
s[14]=a[5];

//Point 15
// P7-Preinjection isentropic reinjection pump
if(P[15]<P[1]) System.out.println("Point 15 Reinjection pressure is not sufficient");
s[15]=s[7];
a=b.property("ps",P[15],s[15]);
h[15]=h[15]+v[11]*(P[14]-P[11]);
//h[15]=a[3];

//Point 12
// P3-Preinjection reinjection pump exit
h[16]=h[7]+(h[15]-h[7])/etaip;

a=b.property("ph",P[16],h[16]);
T[16]=a[1];
s[16]=a[5];
//Point 0 environmental condition
P[0]=1.01325; //bar
a=b.property("tp",T[0],P[0]);
h[0]=a[3];
s[0]=a[5];
Wt1=m4*(h[4]-h[5]);
Wt2=m9*(h[9]-h[10]);
Wp1=m9*(h[14]-h[11]);
Wp2=m4*(h[15]-h[7]);
Wt=Wt1+Wt2;
Wp=Wp1+Wp2;
W=Wt-Wp;
Qboiler=m*(h[1]-h[11]);
Qcondenser=m9*(h[10]-h[11]);
etacycle=W/Qboiler;
Wmax=m*((h[1]-h[0])-(T[0]+273.15)*(s[1]-s[0]));
etaexergy=W/Wmax;
}

public String toString()
{
cycle();
String s="double flash geothermal power plant\n";
s+=" cycle efficiency    = " + etacycle+"\n";
s+=" cycle exergy efficiency    = " + etaexergy+"\n";
s+=" Exergy = "+Wmax+"kW\n";
s+=" HP Turbine power output    = " + Wt1+" kW\n";
s+=" LP Turbine power output    = " + Wt2+" kW\n";
s+=" Turbine power output    = " + Wt+" kW\n";
s+=" Reenjection pump 1 power input    = " + Wp1+" kW\n";
s+=" Reenjection pump 2 power input    = " + Wp2+" kW\n";
s+=" Reenjection pump power input    = " + Wp+" kW\n";
s+=" Net power output    = " + W +" kW\n";
s+=" Virtual boiler heat input = " + Qboiler +" kW\n";
s+=" Condenser heat removal  = "+Qcondenser +" kW\n";
s+=" h1  well head = "+h[1] +" kJ/kg\n";
s+=" T1  well head = "+T[1] +" derece C\n";
s+=" P1  well head = "+P[1] +" bar \n";
s+=" x1  well head = "+x[1] +" kg vapor/kg \n";
s+=" h2 separator inlet = "+h[2] +" kJ/kg\n";
s+=" T2 separator inlet = "+T[3] +" derece C\n";
```

```
s+=" P2 separator inlet = "+P[3] +" bar \n";
s+=" x2  separator inlet = "+x[2] +" kg vapor/kg \n";
s+=" h3 separator exit (saturated water) = "+h[3] +" kJ/kg\n";
s+=" T3 separator exit (saturated water) ="+T[3] +" derece C\n";
s+=" P3 separator exit (saturated water) ="+P[3] +" bar \n";
s+=" x3   separator exit = "+x[3] +" kg vapor/kg \n";
s+=" h4 turbine inlet ="+h[4] +" kJ/kg\n";
s+=" T4 turbine inlet = "+T[4] +" derece C\n";
s+=" P4 turbine inlet = "+P[4] +" bar \n";
s+=" x4 HP turbine inlet = "+x[4] +" kg vapor/kg \n";
s+=" h5 HP turbine exit ="+h[5] +" kJ/kg\n";
s+=" T5 HP turbine exit = "+T[5] +" derece C\n";
s+=" P5 HP turbine exit = "+P[5] +" bar \n";
s+=" x5 HP turbine inlet = "+x[4] +" kg vapor/kg \n";
s+=" h17 HP isentropic turbine exit ="+h[5] +" kJ/kg\n";
s+=" T17 HP isentropic turbine exit = "+T[5] +" derece C\n";
s+=" P17 HP isentropic turbine exit = "+P[5] +" bar \n";
s+=" x17 isentropic turbine exit = "+x[4] +" kg vapor/kg \n";
s+=" h7 condenser saturated water ="+h[7] +" kJ/kg\n";
s+=" T7 condenser saturated water = "+T[7] +" derece C\n";
s+=" P7 condenser saturated water = "+P[7] +" bar \n";
s+=" x7 condenser saturated water = "+x[7] +" kg vapor/kg \n";
s+=" h8 condenser saturated vapor ="+h[8] +" kJ/kg\n";
s+=" T8 condenser saturated vapor = "+T[8] +" derece C\n";
s+=" P8 condenser saturated vapor = "+P[8] +" bar \n";
s+=" x8 condenser saturated vapor = "+x[8] +" kg vapor/kg \n";
s+=" h9 LP turbine inlet ="+h[9] +" kJ/kg\n";
s+=" T9 LP turbine inlet = "+T[9] +" derece C\n";
s+=" P9 LP turbine inlet = "+P[9] +" bar \n";
s+=" x9 LP turbine inlet = "+x[9] +" kg vapor/kg \n";
s+=" h10  LP turbine exit ="+h[10] +" kJ/kg\n";
s+=" T10  LP turbine exit = "+T[10] +" derece C\n";
s+=" P10  LP turbine exit = "+P[10] +" bar \n";
s+=" x10  LP turbine exit = "+x[10] +" kg vapor/kg \n";
s+=" h18  LP isentropic turbine exit ="+h[18] +" kJ/kg\n";
s+=" T18  LP isentropic turbine exit = "+T[18] +" derece C\n";
s+=" P18  LP isentropic turbine exit = "+P[18] +" bar \n";
s+=" x18  LP isentropic turbine exit = "+x[18] +" kg vapor/kg \n";
s+=" h13 isentropic reinjection pump exit ="+h[13] +" kJ/kg\n";
s+=" T13 isentropic reinjection pump exit = "+T[13] +" derece C\n";
s+=" P13 isentropic reinjection pump exit = "+P[13] +" bar \n";
s+=" x13 isentropic reinjection pump exit = "+x[13] +" kg vapor/kg \n";
s+=" h14  reinjection pump exit ="+h[14] +" kJ/kg\n";
s+=" T14  reinjection pump exit = "+T[14] +" derece C\n";
s+=" P14  reinjection pump exit = "+P[14] +" bar \n";
s+=" x14  reinjection pump exit = "+x[14] +" kg vapor/kg \n";
s+=" h15 isentropic reinjection pump exit ="+h[15] +" kJ/kg\n";
s+=" T15 isentropic reinjection pump exit = "+T[15] +" derece C\n";
s+=" P15 isentropic reinjection pump exit = "+P[15] +" bar \n";
s+=" x15 isentropic reinjection pump exit = "+x[15] +" kg vapor/kg \n";
s+=" h16  reinjection pump exit ="+h[16] +" kJ/kg\n";
s+=" T16  reinjection pump exit = "+T[16] +" derece C\n";
s+=" P16  reinjection pump exit = "+P[16] +" bar \n";
s+=" x16  reinjection pump exit = "+x[16] +" kg vapor/kg \n";
return s;
}

public double[][] TS1()
```

```
{
double a[][]=Text.readDoubleT("steam.txt");
return a;
}

public double[][] TS()
{
double a[][]=new double[2][201];
double tc=b.Tc-273.15;
double dt=(tc-10.0)/100.0;
int i=0;
double t;
for(t=10.0;t<=tc;t+=dt)
{double a1[]=b.property("tx",t,0.0);
 a[1][i]=t;
 a[0][i]=a1[5];
 i++;
}
double a3[]=b.property("tx",tc,1.0);
 a[1][i]=t;
 a[0][i]=a3[5];
 i++;
for(t=tc;t>=10;t-=dt)
{double a2[]=b.property("tx",t,1.0);
 a[1][i]=t;
 a[0][i]=a2[5];
 i++;
}
return a;
}

public void plot()
{
 Plot pp=new Plot(TS1());
 double t1[]={T[1],T[2],T[3],T[4],T[5],T[8],T[7],T[9],T[10],T[12],T[11]};
 double s1[]={s[1],s[2],s[3],s[4],s[5],s[8],s[7],s[9],s[10],s[12],s[11]};
 double t2[]={T[4],T[17]};
 double s2[]={s[4],s[17]};
 double t3[]={T[9],T[18]};
 double s3[]={s[9],s[18]};
double t4[]={T[3],T[6]};
 double s4[]={s[3],s[6]};
 pp.addData(s1,t1);
 pp.addData(s2,t2);
 pp.addData(s3,t3);
 pp.addData(s4,t4);
 pp.setPlabel("Geothermal double flash cycle");
 pp.setYlabel("T, degree C");
 pp.setXlabel("s entropy kJ/kgK");
 pp.plot();
}

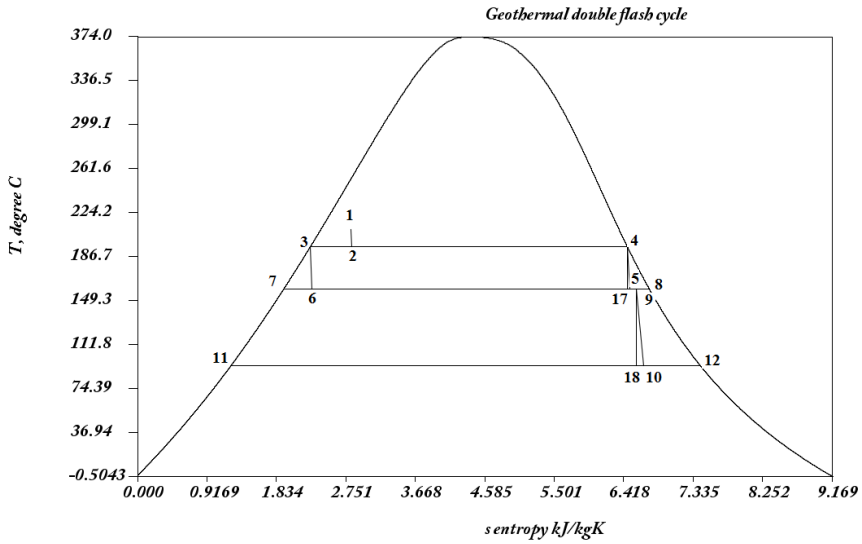
}
```

```
public class flash2_cevrimi_test
{
public static void main(String arg[])
```

```
{ //(double mi,double xi,double P1, double P2, double P3,double P4,double etait1i, double etait2i,double
etaipi,double T0,double Preinjection)
flash2 rc=new flash2(20.0,0.1,19.0,14.0,6.0,0.80,0.9,0.9,0.9,25.0,20.0);
System.out.println(rc.toString());
rc.plot();
}
}
```

```
---------- Capture Output ----------
> "C:\java\bin\java.exe" flash2_cevrimi_test
double flash geothermal power plant
cycle efficiency    = 0.11002348627748726
cycle exergy efficiency    = 0.30382860987800914
Exergy = 5037.194295158879kW
HP Turbine power output    = 374.72917422610067 kW
LP Turbine power output    = 1164.5194311642715 kW
Turbine power output    = 1539.2486053903722 kW
Reenjection pump 1 power input    = 8.752593539089982 kW
Reenjection pump 2 power input    = 0.05227146772186603 kW
Reenjection pump power input    = 8.804865006811848 kW
Net power output    = 1530.4437403835602 kW
Virtual boiler heat input = 13910.154932953765 kW
Condenser heat removal  = 7895.057313914444 kW
h1  well head = 1087.169982440866 kJ/kg
T1  well head = 209.83868256665156 derece C
P1  well head = 19.0 bar
x1  well head = 0.1 kg vapor/kg
h2 separator inlet = 1087.169982440866 kJ/kg
T2 separator inlet = 195.07483020345808 derece C
P2 separator inlet = 14.0 bar
x2  separator inlet = 0.13106771301772813 kg vapor/kg
h3 separator exit (saturated water) = 830.3150734915121 kJ/kg
T3 separator exit (saturated water) =195.07483020345808 derece C
P3 separator exit (saturated water) =14.0 bar
x3   separator exit = 0.0 kg vapor/kg
h4 turbine inlet =2790.0266075945906 kJ/kg
T4 turbine inlet = 195.07483020345808 derece C
P4 turbine inlet = 14.0 bar
x4 HP turbine inlet = 1.0 kg vapor/kg
h5 HP turbine exit =2647.0740964084775 kJ/kg
T5 HP turbine exit = 158.85419273697445 derece C
P5 HP turbine exit = 6.0 bar
x5 HP turbine inlet = 1.0 kg vapor/kg
h17 HP isentropic turbine exit =2647.0740964084775 kJ/kg
T17 HP isentropic turbine exit = 158.85419273697445 derece C
P17 HP isentropic turbine exit = 6.0 bar
x17 isentropic turbine exit = 1.0 kg vapor/kg
h7 condenser saturated water =0.0 kJ/kg
T7 condenser saturated water = 158.85419273697445 derece C
P7 condenser saturated water = 6.0 bar
x7 condenser saturated water = 0.0 kg vapor/kg
h8 condenser saturated vapor =2756.8184791029785 kJ/kg
T8 condenser saturated vapor = 158.85419273697445 derece C
P8 condenser saturated vapor = 6.0 bar
x8 condenser saturated vapor = 1.0 kg vapor/kg
h9 LP turbine inlet =2684.0264657046 kJ/kg
T9 LP turbine inlet = 158.85419273697445 derece C
P9 LP turbine inlet = 6.0 bar
```

x9 LP turbine inlet = 0.0 kg vapor/kg
h10 LP turbine exit =2389.3656041485806 kJ/kg
T10 LP turbine exit = 93.50435536185492 derece C
P10 LP turbine exit = 0.8 bar
x10 LP turbine exit = 0.8784630763964493 kg vapor/kg
h18 LP isentropic turbine exit =2356.625508420134 kJ/kg
T18 LP isentropic turbine exit = 93.50435536185492 derece C
P18 LP isentropic turbine exit = 0.8 bar
x18 LP isentropic turbine exit = 0.0 kg vapor/kg
h13 isentropic reinjection pump exit =393.6554546400712 kJ/kg
T13 isentropic reinjection pump exit = 93.62822347328785 derece C
P13 isentropic reinjection pump exit = 20.0 bar
x13 isentropic reinjection pump exit = 0.0 kg vapor/kg
h14 reinjection pump exit =393.8769234008371 kJ/kg
T14 reinjection pump exit = 93.68090134423034 derece C
P14 reinjection pump exit = 20.0 bar
x14 reinjection pump exit = 0.0 kg vapor/kg
h15 isentropic reinjection pump exit =0.01994063469879719 kJ/kg
T15 isentropic reinjection pump exit = 0.0 derece C
P15 isentropic reinjection pump exit = 20.0 bar
x15 isentropic reinjection pump exit = 0.0 kg vapor/kg
h16 reinjection pump exit =0.02215626077644132 kJ/kg
T16 reinjection pump exit = 212.4183856142063 derece C
P16 reinjection pump exit = 20.0 bar
x16 reinjection pump exit = 0.0 kg vapor/kg



Let us now consider a binary geothermal power plant. Binary geothermal power plants operates with standard rankine cycle, with a working fluid lower boiling temperature compare to water so that it will evaporate in lower temperatures. In the next example R134a is taken as cycle working fluid. Temperature of the turbine inlet is taken as 200 C, which is a normal temperature for such a power plant. System pressures can be adjusted, since it is a closed cycle. Heat from geothermal fluid is transferred to the working fluid through a heat exchanger. The binary cycle is considered both as supercritical and sub-critical. In case of supercritical cycle, better cycle efficiencies are obtained, and heat exchanger designed to transfer geothermal heat to the secondary fluid is smaller due to almost linear temperature profile. In case of sub-critical cycle pinch points arise in heat exchanger due to boiling process in the secondary fluid. Bu pressure of the first cycle has higher pressure, therefore material cost might be higher due to high turbine inlet and pumpoperation pressure.

Program 3.3.9 Binary geothermal power plant, with R134a as working fluid

```
 public class rankine_R134a
{
// steam sınıfını kullanır
// çağrı
// refISO b=new refISO("R134a");
// double a[]=b.property("tp",303.0,1.01325);
// girdi parametreleri
// termodinamik set "tv" "tp" "th" "tu" "ts"tx"
//                  "pv" "pt" "ph" "pu" "ps" "px"
// çıkış parametreleri(a)
// a[0]=P a[1]=T a[2]=V a[3]=h
// a[4]u a[5]=s a[6]=x
double T[];
double P[];
double h[];
double s[];
double x[];
double etait,etaip;
double etacevrim,etaisi,etatoplam;
double m;//kg/s
double N;
double Wt,Wp,W;//iş
double Qkazan,Qyogusturucu;
public refISO b;
double c1[][];
double c2[][];
double c3[][];
double c4[][];
public boolean supercritical=false;
public void print(int i)
{System.out.println("T["+i+"] = "+T[i]+" P["+i+"] = "+P[i]+" h["+i+"] = "+h[i]+"
s["+i+"] = "+s[i]+" x["+i+"] = "+x[i]);}

public rankine_R134a(double mi,double t1,double P1, double P2, double etaiti,
double etaipi)
{
m=mi;
b=new refISO("R134a");
N=m/b.M;
//b.setMass();
T=new double[9];
P=new double[9];
h=new double[9];
s=new double[9];
x=new double[9];
T[1]=t1;//turbin girişi
```

```
P[1]=P1;
P[2]=P2;//turbin çıkış basıncı
etait=etaiti;//turbin isentropik verimi
etaip=etaipi;//pompa isentropik verimi
c1=new double[2][20];
c2=new double[2][20];
c3=new double[2][20];
c4=new double[2][3];
}
public void cevrimhesabi()
{
// Turbin girişi
double a1[]=b.property("tp",T[1],P[1]);
h[1]=a1[3];
s[1]=a1[5];
print(1);
if(P[1]>b.Pc) supercritical=true;
// isentropik Turbin çıkışı 0
s[0]=s[1];
P[0]=P[2];
double a0[]=b.property("ps",P[0],s[0]);
T[0]=a0[1];
h[0]=a0[3];
print(0);
// gerçek turbin çıkışı
h[2]=h[1]-etait*(h[1]-h[0]);
double a2[]=b.property("ph",P[2],h[2]);
T[2]=a2[1];
x[2]=a2[6];
s[2]=a2[5];
print(2);
// yoğuşturucu çıkışı
P[3]=P[2];
x[3]=0.0;
double a3[]=b.property("px",P[3],x[3]);
T[3]=a3[1];
h[3]=a3[3];
s[3]=a3[5];
print(3);
// isentropik pompa
P[5]=P[1];
s[5]=s[3];
double a5[]=b.property("ps",P[5],s[5]);
T[5]=a5[1];
h[5]=a5[3];
x[5]=a5[6];
print(5);
```

```
// gerçek pompa
h[4]=h[3]+(h[5]-h[3])/etaip;
P[4]=P[1];
double a4[]=b.property("ph",P[4],h[4]);
T[4]=a4[1];
s[4]=a4[5];
x[4]=a4[6];
print(4);
if(!supercritical)
{
// kazan kaynama  bölgesi girişi
P[6]=P[1];
x[6]=0.0;
double a6[]=b.property("px",P[6],x[6]);
T[6]=a6[1];
s[6]=a6[5];
print(6);
// kazan kızgın buhar bölgesi girişi
P[7]=P[1];
x[7]=1.0;
double a7[]=b.property("px",P[7],x[7]);
T[7]=a7[1];
s[7]=a7[5];
print(7);
}

// yoğuşturucu kızgın buhar bölgesi çıkışı
P[8]=P[2];
x[8]=1.0;
double a8[]=b.property("px",P[8],x[8]);
T[8]=a8[1];
s[8]=a8[5];
print(8);
Wt=m*(h[1]-h[2]);
Wp=m*(h[4]-h[3]);
W=Wt-Wp;
Qkazan=m*(h[1]-h[4]);
Qyogusturucu=m*(h[2]-h[3]);
etacevrim=W/Qkazan;
etaisi=Qyogusturucu/Qkazan;
etatoplam=etacevrim+etaisi;
if(s[2]>s[8])
{double dt1=(T[2]-T[8])/19;
double t=T[8];
for(int i=0;i<20;i++)
{double b8[]=b.property("tp",t,P[8]);
c3[0][i]=b8[5]/b.M;c3[1][i]=b8[1];t+=dt1;
```

```
}
}
if(!supercritical)
{
double a[][]=line1(T[1],T[7],P[1],P[1],19);c1[0]=a[1];c1[1]=a[0];
double b1[]={T[7],T[6],T[4]};
c4[0]=b1;
double b2[]={s[7]/b.M,s[6]/b.M,s[4]/b.M};
c4[1]=b2;
}
else
{double a[][]=line1(T[1],T[4],P[1],P[1],19);c1[0]=a[1];c1[1]=a[0];}

}
public double[][] line1(double T1,double T2,double Pi,double Po,int n)
{
double TT1[]=new double[n];
double ss1[]=new double[n];
double aa[];
double P;
for(int i=0;i<n;i++)
{TT1[i]=T1+i*(T2-T1)/(n-1);
 P=Pi+i*(Po-Pi)/(n-1);
 aa=b.property("tp",TT1[i],P);
 ss1[i]=aa[5]/b.M;
}
double a[][]={TT1,ss1};
return a;
}
public String cikti()
{
cevrimhesabi();
String s="Simple Rankine cycle\n";
s+=" cycle efficiency    = " + etacevrim+"\n";
s+=" heat energy utilisation rate = "+etaisi+"\n";
s+=" total heat+power efficiency = "+etatoplam+"\n";
s+=" Turbine power output    = " + Wt+" kW\n";
s+=" Pump power input    = " + Wp+" kW\n";
s+=" Net power output    = " + W +" kW\n";
s+=" Boiler heat input = " + Qkazan +" kW\n";
s+=" Condenser heat  = "+Qyogusturucu +" kW\n";
s+=" h1 turbine input = "+h[1]/b.M +" kJ/kg\n";
s+=" T1 turbine input = "+(T[1]-273.15) +" derece C\n";
s+=" P1 turbine input = "+P[1] +" kPA \n";
s+=" h2 turbine output = "+h[2]/b.M +" kJ/kg\n";
s+=" T2 turbine output = "+(T[2]-273.15) +" derece C\n";
s+=" P2 turbine output = "+P[3] +" kPA \n";
```

```
s+=" h3 condenser output = "+h[3]/b.M +" kJ/kg\n";
s+=" T3 condenser output = "+(T[3]-273.15) +" derece C\n";
s+=" P3 condenser output = "+P[3] +" kPA \n";
s+=" h4 pump output = "+h[4]/b.M +" kJ/kg\n";
s+=" T4 pump output = "+(T[4]-273.15) +" derece C\n";
s+=" P4 pump output = "+P[4] +" kPA \n";
return s;
}

public void plot()
{  double a[][]=b.TS_saturation();
   for(int i=0;i<a[0].length;i++) {a[1][i]=a[1][i]/b.M;}

 Plot pp=new Plot(a[1],a[0]);
 double t1[]={T[1],T[2]};
 double t2[]={T[8],T[3],T[4]};
 double s1[]={s[1]/b.M,s[2]/b.M};
 double s2[]={s[8]/b.M,s[3]/b.M,s[4]/b.M};
 double t3[]={T[4],T[6],T[4]};
 double s3[]={s[8]/b.M,s[3]/b.M,s[4]/b.M};
 pp.addData(s1,t1);
 if(!supercritical)
 {pp.addData(c4[1],c4[0]);}
 pp.addData(c1[0],c1[1]);
 pp.addData(s2,t2);
 if(s[2]>s[8]) { pp.addData(c3[0],c3[1]);}
 pp.setPlabel("Rankine Cycle");
 pp.setYlabel("T, degree K");
 pp.setXlabel("s entropy kJ/kgK");
 pp.plot();
}

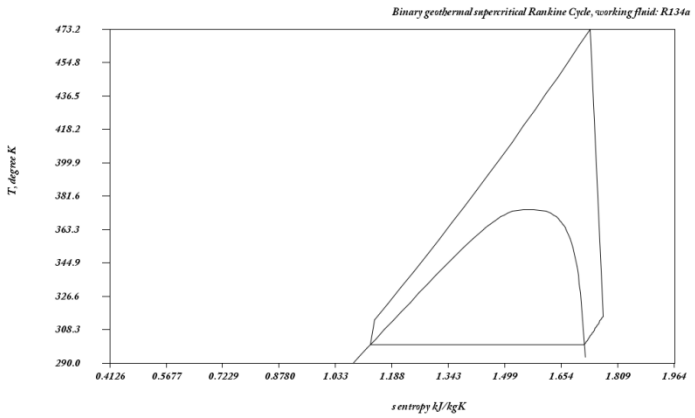
}
```

```
 import javax.swing.*;

public class rankinetest
{
public static void main(String arg[])
{
double m=1.0;    //  kg/s
double t1=200.0+273.15;          //  derece K turbine inlet temperature
double P1=25000.0;          //  kPa turbine inlet pressure
double P2=700.0;           //  kPa turbine exit pressure
double etaiturbin=0.85; //    turbine isentropic efficiency
double etaipompa=0.85;  //    pump isentropic efficicency
rankine_R134a r=new rankine_R134a(m,t1,P1,P2,etaiturbin,etaipompa);
```

```
System.out.println(r.cikti());
r.plot();
}
}
```

```
---------- Capture Output ----------
> "C:\java\bin\javaw.exe" rankinetest
Simple Rankine cycle
cycle efficiency    = 0.16949916732858386
heat energy utilisation rate = 0.8305008326714162
total heat+power efficiency = 1.0
Turbine power output    = 6353.192255496513 kW
Pump power input    = 2353.2846691457053 kW
Net power output    = 3999.907586350808 kW
Boiler heat input = 23598.390773193343 kW
Condenser heat  = 19598.483186842535 kW
h1 turbine input = 491.34365083693024 kJ/kg
T1 turbine input = 200.0 derece C
P1 turbine input = 25000.0 kPA
h2 turbine output = 429.07698689329965 kJ/kg
T2 turbine output = 42.35717901485077 degree C
P2 turbine output = 700.0 kPA
h3 condenser output = 236.99525580067643 kJ/kg
T3 condenser output = 26.713375870973778 degree C
P3 condenser output = 700.0 kPA
h4 pump output = 260.0594383036726 kJ/kg
T4 pump output = 40.44716388848133 degree C
P4 pump output = 25000.0 kPA
```



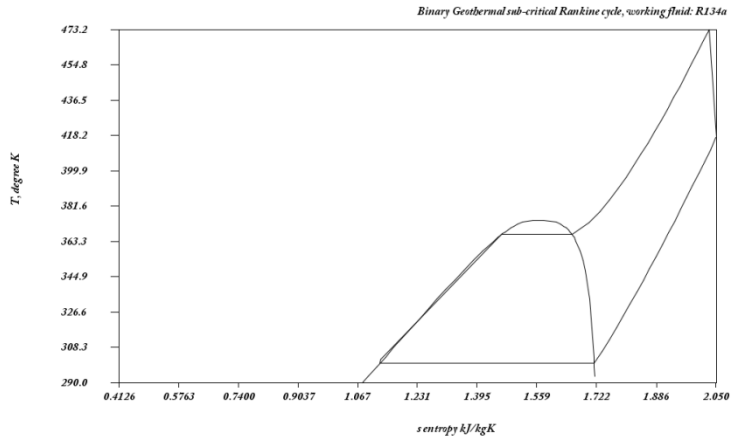
If the same geothermal plants of Tinlet=200 C operates as sub-critical rankine cycle (lower pressure):

```
import javax.swing.*;

public class rankinetest
{
public static void main(String arg[])
{
```

```
double m=1.0;    //  kg/s
double t1=200.0+273.15;          //  derece K turbine inlet temperature
double P1=3500.0;          //  kPa turbine inlet pressure
double P2=700.0;           //  kPa turbine exit pressure
double etaiturbin=0.85; //    turbine isentropic efficiency
double etaipompa=0.85;  //    pump isentropic efficicency
rankine_R134a r=new rankine_R134a(m,t1,P1,P2,etaiturbin,etaipompa);
System.out.println(r.cikti());
r.plot();
}}
```

```
---------- Capture Output ----------
> "C:\java\bin\javaw.exe" rankinetest
Simple Rankine cycle
 cycle efficiency    = 0.12538631819352664
 heat energy utilisation rate = 0.8746136818064734
 total heat+power efficiency = 1.0
 Turbine power output    = 4593.428533216946 kW
 Pump power input    = 278.16462421843244 kW
 Net power output    = 4315.263908998513 kW
 Boiler heat input = 34415.74783572598 kW
 Condenser heat  = 30100.48392672747 kW
 h1 turbine input = 577.0249764759981 kJ/kg
 T1 turbine input = 200.0 derece C
 P1 turbine input = 3500.0 kPA
 h2 turbine output = 532.0054871665957 kJ/kg
 T2 turbine output = 143.7643080291872 derece C
 P2 turbine output = 700.0 kPA
 h3 condenser output = 236.99525580067643 kJ/kg
 T3 condenser output = 26.713375870973778 derece C
 P3 condenser output = 700.0 kPA
 h4 pump output = 239.721504665919 kJ/kg
 T4 pump output = 28.556914587844688 derece C
 P4 pump output = 3500.0 kPA
```

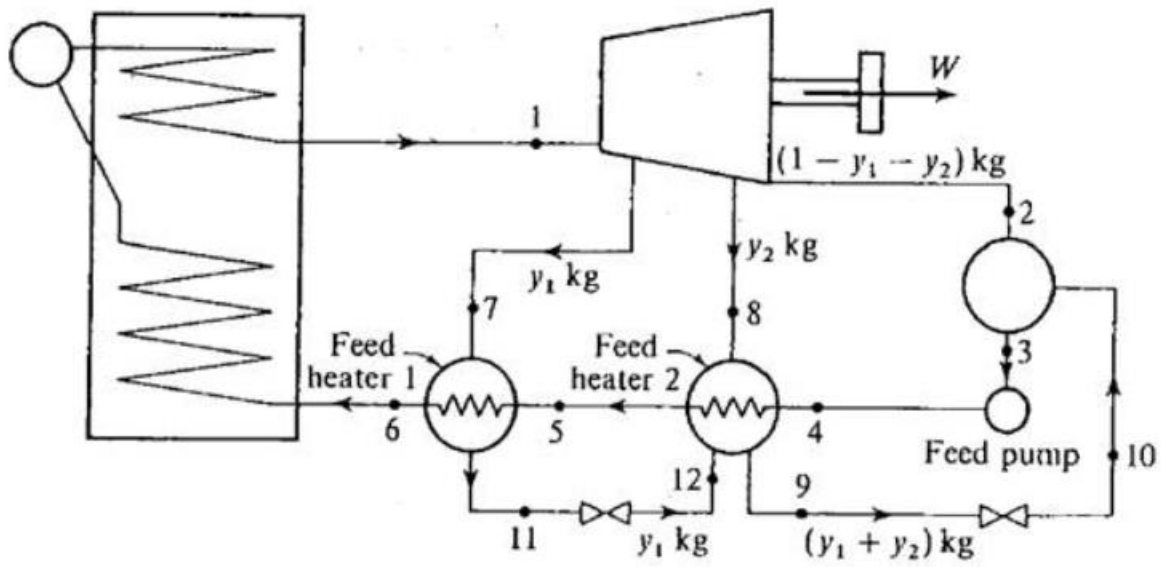


PROBLEM

In a regenerative rankine cycle employing two closed-feed heaters the steam is supplied to the turbine at 4000 kPa and 500 C and is exhausted to the condenser at 3.5 kPa. The intermediate bleed pressures are obtained such that the stauration temperature intervals are approximately equal, giving pressures of 1000 kPa and 1100 kPa. Claculate the steam

blade at each stage, the work output of the plant per kg of boiler steam and cycle efficiency of the plant. Assume all isentropic efficiencies are 0.9

HAND SOLUTION



In hand solution I am using steamTableIF97.java GUI program to read thermodynamic properties

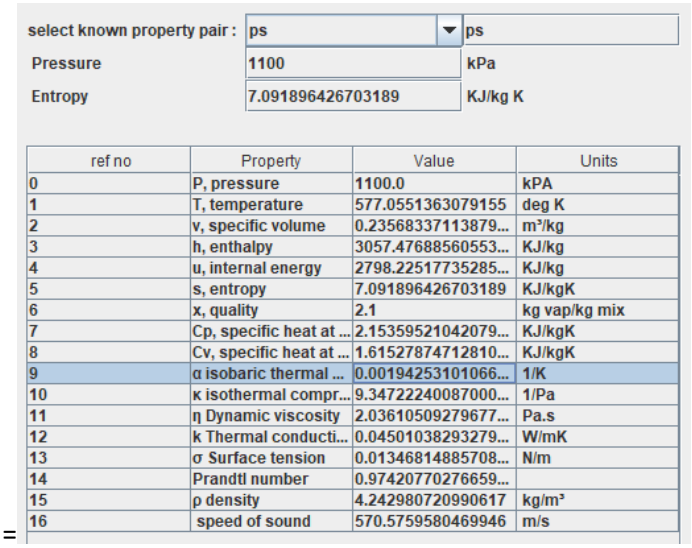
Properties of water and steam
International Association for the Properties of Water and Steam (IAPWS)
Industrial formulation 1997
Dr. M. Turhan Çoban,
Ege University, School of Engineering, Mechanical Eng. Dept., Izmir, Turkey
email : turhan.coban@turhancoban.tr
web : www.turhancoban.com
Reference: International Steam Tables, Wolfgang Wagner, Hans-Joachim Kretzschamar
ISBN 978-3-540-21419-9, Springer 2008
273.15 K <= T <= 1073.15 K && 0< P <= 100 MPa
1073.15 K <= T <= 2273.15 K && 0< p <= 50 MPa

select known property pair :	tp	tp
Temperature	500	degree C
Pressure	4000	kPa

ref no	Property	Value	Units
0	P, pressure	4000.0	kPA
1	T, temperature	773.15	deg K
2	v, specific volume	0.08644100541263...	m³/kg
3	h, enthalpy	3445.83740674479...	KJ/kg
4	u, internal energy	3100.073385094268	KJ/kg
5	s, entropy	7.091896426703189	KJ/kgK
6	x, quality	2.0	kg vap/kg mix
7	Cp, specific heat at ...	2.28919954452732...	KJ/kgK
8	Cv, specific heat at ...	1.73577290411892...	KJ/kgK
9	α isobaric thermal ...	0.00196859496135...	1/K
10	κ isothermal compr...	2.58190826990513...	1/Pa
11	η Dynamic viscosity	2.86655204566728...	Pa.s
12	k Thermal conducti...	0.06974024279050...	W/mK
13	σ Surface tension	NaN	N/m
14	Prandtl number	0.94093587500373...	
15	ρ density	11.56858362794877	kg/m³
16	speed of sound	618.6759720879483	m/s

Point 1: T_1=500 C P_1=4000 kPa h_1=3445.8374067447944 kJ/kg s_1=7.091896426703189

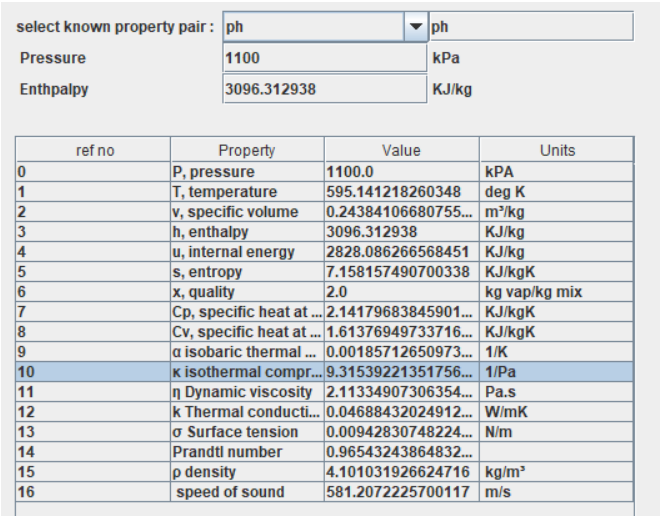
Point 7i=13 P13=1100 kPa T13=577 $s_1=s_{13}$ isentropic process



ref no	Property	Value	Units
0	P, pressure	1100.0	kPA
1	T, temperature	577.0551363079155	deg K
2	v, specific volume	0.23568337113879...	m³/kg
3	h, enthalpy	3057.47688560553...	KJ/kg
4	u, internal energy	2798.22517735285...	KJ/kg
5	s, entropy	7.091896426703189	KJ/kgK
6	x, quality	2.1	kg vap/kg mix
7	Cp, specific heat at ...	2.15359521042079...	KJ/kgK
8	Cv, specific heat at ...	1.61527874712810...	KJ/kgK
9	α isobaric thermal ...	0.00194253101066...	1/K
10	κ isothermal compr...	9.34722240087000...	1/Pa
11	η Dynamic viscosity	2.03610509279677...	Pa.s
12	k Thermal conducti...	0.04501038293279...	W/mK
13	σ Surface tension	0.01346814885708...	N/m
14	Prandtl number	0.97420770276659...	
15	ρ density	4.242980720990617	kg/m³
16	speed of sound	570.5759580469946	m/s

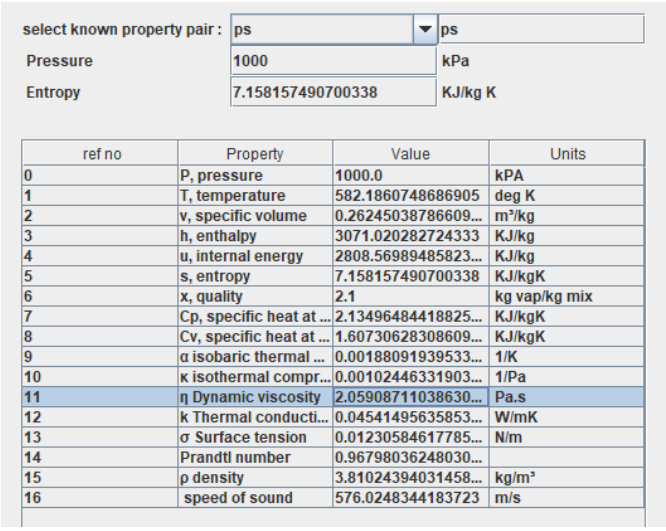
=

Point 7 $h_7 = h_1 - \eta_{isentropic}(h_{13} - h_1)$



ref no	Property	Value	Units
0	P, pressure	1100.0	kPA
1	T, temperature	595.141218260348	deg K
2	v, specific volume	0.24384106680755...	m³/kg
3	h, enthalpy	3096.312938	KJ/kg
4	u, internal energy	2828.086266568451	KJ/kg
5	s, entropy	7.158157490700338	KJ/kgK
6	x, quality	2.0	kg vap/kg mix
7	Cp, specific heat at ...	2.14179683845901...	KJ/kgK
8	Cv, specific heat at ...	1.61376949733716...	KJ/kgK
9	α isobaric thermal ...	0.00185712650973...	1/K
10	κ isothermal compr...	9.31539221351756...	1/Pa
11	η Dynamic viscosity	2.11334907306354...	Pa.s
12	k Thermal conducti...	0.04688432024912...	W/mK
13	σ Surface tension	0.00942830748224...	N/m
14	Prandtl number	0.96543243864832...	
15	ρ density	4.101031926624716	kg/m³
16	speed of sound	581.2072225700117	m/s

Point 14=8i P=1000 kPa s_7=s_{13} İsentropic process



ref no	Property	Value	Units
0	P, pressure	1000.0	kPA
1	T, temperature	582.1860748686905	deg K
2	v, specific volume	0.26245038786609...	m³/kg
3	h, enthalpy	3071.020282724333	KJ/kg
4	u, internal energy	2808.56989485823...	KJ/kg
5	s, entropy	7.158157490700338	KJ/kgK
6	x, quality	2.1	kg vap/kg mix
7	Cp, specific heat at ...	2.13496484418825...	KJ/kgK
8	Cv, specific heat at ...	1.60730628308609...	KJ/kgK
9	α isobaric thermal ...	0.00188091939533...	1/K
10	κ isothermal compr...	0.00102446331903...	1/Pa
11	η Dynamic viscosity	2.05908711038630...	Pa.s
12	k Thermal conducti...	0.04541495635853...	W/mK
13	σ Surface tension	0.01230584617785...	N/m
14	Prandtl number	0.96798036248030...	
15	ρ density	3.81024394031458...	kg/m³
16	speed of sound	576.0248344183723	m/s

Point 8 $h_8 = h_7 - \eta_{isentropic}(h_{14} - h_7)$

select known property pair :	ph	ph
Pressure	1000	kPa
Enthpalpy	3073.549548	KJ/kg

ref no	Property	Value	Units
0	P, pressure	1000.0	kPA
1	T, temperature	583.3697934732463	deg K
2	v, specific volume	0.26303455438403...	m³/kg
3	h, enthalpy	3073.549548	KJ/kg
4	u, internal energy	2810.51253175589...	KJ/kg
5	s, entropy	7.162488862529842	KJ/kgK
6	x, quality	2.0	kg vap/kg mix
7	Cp, specific heat at ...	2.134302718750179	KJ/kgK
8	Cv, specific heat at ...	1.60726305211847...	KJ/kgK
9	α isobaric thermal ...	0.00187562811792...	1/K
10	κ isothermal compr...	0.00102425220895...	1/Pa
11	η Dynamic viscosity	2.06413052363282...	Pa.s
12	k Thermal conducti...	0.04553777247249...	W/mK
13	σ Surface tension	0.01203931115244...	N/m
14	Prandtl number	0.96743409904504...	
15	ρ density	3.80178186984500...	kg/m³
16	speed of sound	576.7069638688953	m/s

Point 15=2i P=3.5 kPa s_8=s_{15} İsentropic process

select known property pair :	ps	ps
Pressure	3.5	kPa
Entropy	7.162488862529842	KJ/kg K

ref no	Property	Value	Units
0	P, pressure	3.5	kPA
1	T, temperature	299.8231926726793	deg K
2	v, specific volume	32.87218681425655	m³/kg
3	h, enthalpy	2142.182534619141	KJ/kg
4	u, internal energy	2027.12988076924...	KJ/kg
5	s, entropy	7.162488862529842	KJ/kgK
6	x, quality	0.83288166963885...	kg vap/kg mix
7	Cp, specific heat at ...	2.29268906925524...	KJ/kgK
8	Cv, specific heat at ...	1.891444584282974	KJ/kgK
9	α isobaric thermal ...	0.0	1/K
10	κ isothermal compr...	0.0	1/Pa
11	η Dynamic viscosity	1.51363489756431...	Pa.s
12	k Thermal conducti...	0.11731472083405...	W/mK
13	σ Surface tension	0.07171340250282...	N/m
14	Prandtl number	2.95810633125730...	
15	ρ density	0.03042085412967...	kg/m³
16	speed of sound	607.3857062933289	m/s

Point 2: $h_2 = h_8 - \eta_{isentropic}(h_{15} - h_8)$

select known property pair : ph | ph

Pressure 3.5 kPa

Enthpalpy 2235.319236 KJ/kg

ref no	Property	Value	Units
0	P, pressure	3.5	kPA
1	T, temperature	299.8231926726793	deg K
2	v, specific volume	34.3800639250166...	m³/kg
3	h, enthalpy	2235.319236	KJ/kg
4	u, internal energy	2114.98901226244...	KJ/kg
5	s, entropy	7.473128361842154	KJ/kgK
6	x, quality	0.87108787571551...	kg vap/kg mix
7	Cp, specific heat at ...	2.206047491919798	KJ/kgK
8	Cv, specific heat at ...	1.78867733293145...	KJ/kgK
9	α isobaric thermal ...	0.00294061434249...	1/K
10	κ isothermal compr...	0.24934263540528...	1/Pa
11	η Dynamic viscosity	1.18989046360791...	Pa.s
12	k Thermal conducti...	0.09471323211063...	W/mK
13	σ Surface tension	0.07171340250282...	N/m
14	Prandtl number	2.77147639712618...	
15	ρ density	0.02908662421864...	kg/m³
16	speed of sound	566.3241879321604	m/s

Point 10: condenser exit x=0

select known property pair : px | px

Pressure 3.5 kPa

Quality 0 kg vapor/kg mix

ref no	Property	Value	Units
0	P, pressure	3.5	kPA
1	T, temperature	299.8231926726793	deg K
2	v, specific volume	0.00100344943026...	m³/kg
3	h, enthalpy	111.8356534054894	KJ/kg
4	u, internal energy	111.832141332483...	KJ/kg
5	s, entropy	0.39065853936422...	KJ/kgK
6	x, quality	0.0	kg vap/kg mix
7	Cp, specific heat at ...	4.181444562184937	KJ/kgK
8	Cv, specific heat at ...	4.13173402098117	KJ/kgK
9	α isobaric thermal ...	0.0	1/K
10	κ isothermal compr...	0.0	1/Pa
11	η Dynamic viscosity	8.57114812326609...	Pa.s
12	k Thermal conducti...	0.61001911620159...	W/mK
13	σ Surface tension	0.07171340250282...	N/m
14	Prandtl number	5.87518977026097	
15	ρ density	996.5624273991448	kg/m³
16	speed of sound	1502.51216564235...	m/s

Point 16=4i isentropic process $s_{16}=s_{10}$

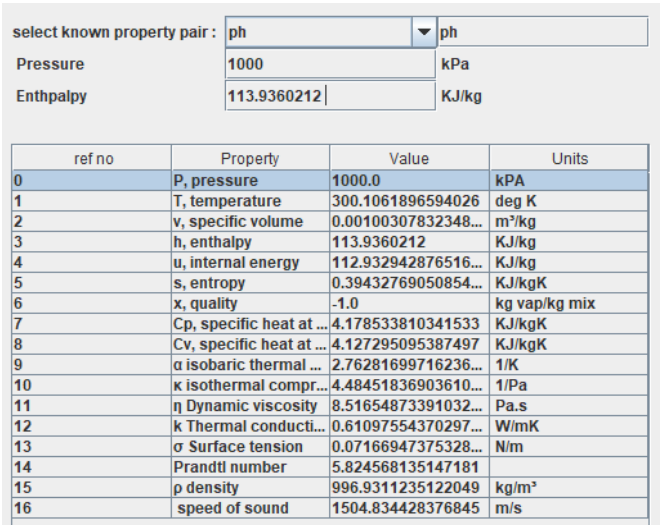
select known property pair : ps | ps

Pressure 1000 kPa

Entropy 0.39065853936422884 KJ/kg K

ref no	Property	Value	Units
0	P, pressure	1000.0	kPA
1	T, temperature	299.8416365758884	deg K
2	v, specific volume	0.00100300533388...	m³/kg
3	h, enthalpy	112.830564460735...	KJ/kg
4	u, internal energy	111.827559126854...	KJ/kg
5	s, entropy	0.39065853936422...	KJ/kgK
6	x, quality	-1.0	kg vap/kg mix
7	Cp, specific heat at ...	4.178630496439059	KJ/kgK
8	Cv, specific heat at ...	4.12837258337789...	KJ/kgK
9	α isobaric thermal ...	2.73838593796472...	1/K
10	κ isothermal compr...	4.48724769773073...	1/Pa
11	η Dynamic viscosity	8.56670171556009...	Pa.s
12	k Thermal conducti...	0.61057601557087...	W/mK
13	σ Surface tension	0.07171054088509...	N/m
14	Prandtl number	5.862837735128983	
15	ρ density	997.0036710882368	kg/m³
16	speed of sound	1504.14305123447...	m/s

Point 4: $h_4 = h_{10} + \frac{(h_{16} - h_{10})}{\eta_{isentropic}}$



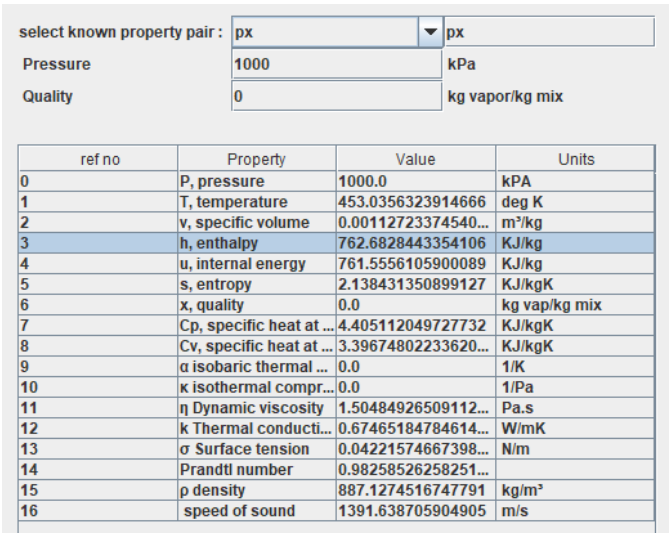
select known property pair : ph | ph

Pressure 1000 kPa

Enthpalpy 113.9360212 KJ/kg

ref no	Property	Value	Units
0	P, pressure	1000.0	kPA
1	T, temperature	300.1061896594026	deg K
2	v, specific volume	0.00100307832348...	m³/kg
3	h, enthalpy	113.9360212	KJ/kg
4	u, internal energy	112.932942876516...	KJ/kg
5	s, entropy	0.39432769050854...	KJ/kgK
6	x, quality	-1.0	kg vap/kg mix
7	Cp, specific heat at ...	4.178533810341533	KJ/kgK
8	Cv, specific heat at ...	4.127295095387497	KJ/kgK
9	α isobaric thermal ...	2.76281699716236...	1/K
10	κ isothermal compr...	4.48451836903610...	1/Pa
11	η Dynamic viscosity	8.51654873391032...	Pa.s
12	k Thermal conducti...	0.61097554370297...	W/mK
13	σ Surface tension	0.07166947375328...	N/m
14	Prandtl number	5.824568135147181	
15	ρ density	996.9311235122049	kg/m³
16	speed of sound	1504.834428376845	m/s

Point 9: P=1000 kPa x=0 saturated liquid



select known property pair : px | px

Pressure 1000 kPa

Quality 0 kg vapor/kg mix

ref no	Property	Value	Units
0	P, pressure	1000.0	kPA
1	T, temperature	453.0356323914666	deg K
2	v, specific volume	0.00112723374540...	m³/kg
3	h, enthalpy	762.6828443354106	KJ/kg
4	u, internal energy	761.5556105900089	KJ/kg
5	s, entropy	2.138431350899127	KJ/kgK
6	x, quality	0.0	kg vap/kg mix
7	Cp, specific heat at ...	4.405112049727732	KJ/kgK
8	Cv, specific heat at ...	3.39674802233620...	KJ/kgK
9	α isobaric thermal ...	0.0	1/K
10	κ isothermal compr...	0.0	1/Pa
11	η Dynamic viscosity	1.50484926509112...	Pa.s
12	k Thermal conducti...	0.67465184784614...	W/mK
13	σ Surface tension	0.04221574667398...	N/m
14	Prandtl number	0.98258526258251...	
15	ρ density	887.1274516747791	kg/m³
16	speed of sound	1391.638705904905	m/s

Point 11 P=1100 x=0 saturated liquid

select known property pair : px | px

Pressure: 1100 kPa

Quality: 0 kg vapor/kg mix

ref no	Property	Value	Units
0	P, pressure	1100.0	kPA
1	T, temperature	457.219675677945...	deg K
2	v, specific volume	0.00113299184155...	m³/kg
3	h, enthalpy	781.1977431114818	KJ/kg
4	u, internal energy	779.9514520857758	KJ/kg
5	s, entropy	2.17886389157392...	KJ/kgK
6	x, quality	0.0	kg vap/kg mix
7	Cp, specific heat at ...	4.42166284493527...	KJ/kgK
8	Cv, specific heat at ...	3.37962277219957...	KJ/kgK
9	α isobaric thermal ...	0.0	1/K
10	κ isothermal compr...	0.0	1/Pa
11	η Dynamic viscosity	1.46873755970868...	Pa.s
12	k Thermal conducti...	0.672639194154109	W/mK
13	σ Surface tension	0.04127999924160...	N/m
14	Prandtl number	0.96548972363880...	
15	ρ density	882.6188886154331	kg/m³
16	speed of sound	1379.78064324425...	m/s

Point 12 throttling process $h_{11}=h_{12}$

select known property pair : ph | ph

Pressure: 1000 kPa

Enthpalpy: 781.1977431 KJ/kg

ref no	Property	Value	Units
0	P, pressure	1000.0	kPA
1	T, temperature	453.0356323914666	deg K
2	v, specific volume	0.00290315418582...	m³/kg
3	h, enthalpy	781.1977431	KJ/kg
4	u, internal energy	778.2945889141777	KJ/kg
5	s, entropy	2.17930003631341...	KJ/kgK
6	x, quality	0.00919110480151...	kg vap/kg mix
7	Cp, specific heat at ...	4.389577913010558	KJ/kgK
8	Cv, specific heat at ...	3.38327009559130...	KJ/kgK
9	α isobaric thermal ...	0.00123826956006...	1/K
10	κ isothermal compr...	1.07615484848193...	1/Pa
11	η Dynamic viscosity	1.49239498625986...	Pa.s
12	k Thermal conducti...	0.66877636091491...	W/mK
13	σ Surface tension	0.04221574667398...	N/m
14	Prandtl number	0.97954779086569...	
15	ρ density	344.4529418669954	kg/m³
16	speed of sound	1383.24613904197...	m/s

Point 10: throttling process $h_9=h_{10}$

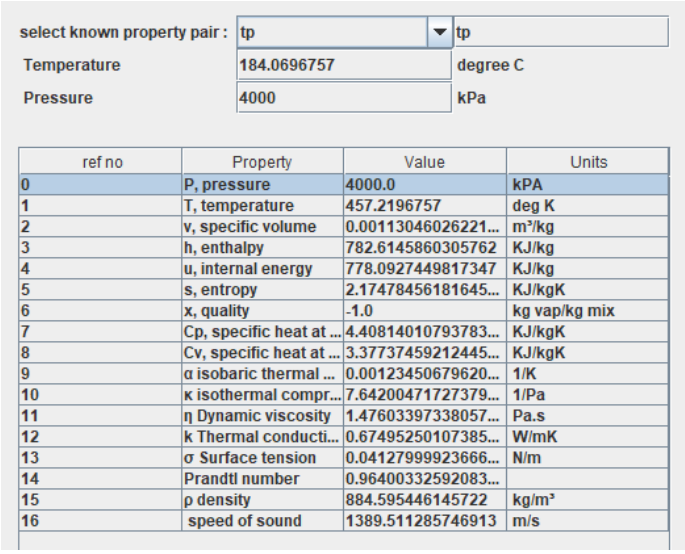
select known property pair : ph | ph

Pressure: 3.5 kPa

Enthpalpy: 762.6828443 KJ/kg

ref no	Property	Value	Units
0	P, pressure	3.5	kPA
1	T, temperature	299.8231926726793	deg K
2	v, specific volume	10.5381769503382...	m³/kg
3	h, enthalpy	762.6828443	KJ/kg
4	u, internal energy	725.7992249738163	KJ/kg
5	s, entropy	2.56143378167752...	KJ/kgK
6	x, quality	0.26698821765271...	kg vap/kg mix
7	Cp, specific heat at ...	3.575985850555792	KJ/kgK
8	Cv, specific heat at ...	3.413587727775308	KJ/kgK
9	α isobaric thermal ...	0.00109036522663...	1/K
10	κ isothermal compr...	0.07642376781139...	1/Pa
11	η Dynamic viscosity	6.30879448303336...	Pa.s
12	k Thermal conducti...	0.45207799065626...	W/mK
13	σ Surface tension	0.07171340250282...	N/m
14	Prandtl number	4.990324738579317	
15	ρ density	0.09489307350906...	kg/m³
16	speed of sound	1215.57078288985...	m/s

Point 6 : $T_6=T_{11}$



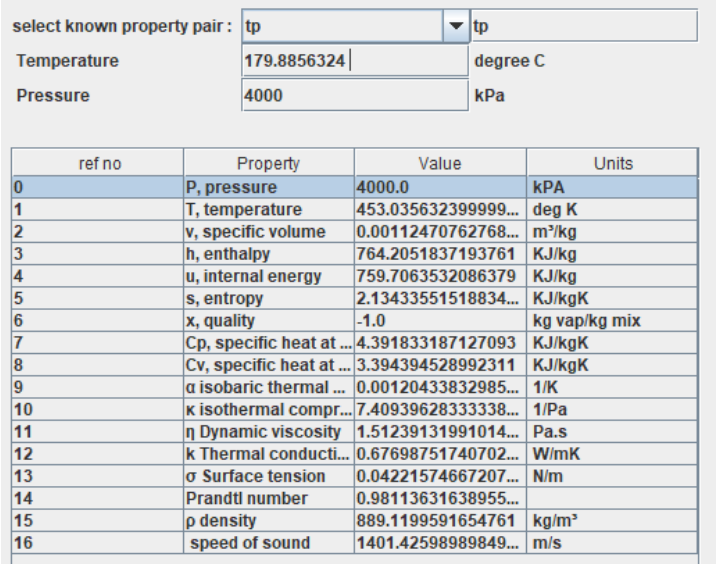
select known property pair : tp | tp

Temperature 184.0696757 degree C

Pressure 4000 kPa

ref no	Property	Value	Units
0	P, pressure	4000.0	kPA
1	T, temperature	457.2196757	deg K
2	v, specific volume	0.00113046026221...	m³/kg
3	h, enthalpy	782.6145860305762	KJ/kg
4	u, internal energy	778.0927449817347	KJ/kg
5	s, entropy	2.17478456181645...	KJ/kgK
6	x, quality	-1.0	kg vap/kg mix
7	Cp, specific heat at ...	4.40814010793783...	KJ/kgK
8	Cv, specific heat at ...	3.37737459212445...	KJ/kgK
9	α isobaric thermal ...	0.00123450679620...	1/K
10	κ isothermal compr...	7.64200471727379...	1/Pa
11	η Dynamic viscosity	1.47603397338057...	Pa.s
12	k Thermal conducti...	0.67495250107385...	W/mK
13	σ Surface tension	0.04127999923666...	N/m
14	Prandtl number	0.96400332592083...	
15	ρ density	884.595446145722	kg/m³
16	speed of sound	1389.511285746913	m/s

Point 5: $T_2=T_9$



select known property pair : tp | tp

Temperature 179.8856324 degree C

Pressure 4000 kPa

ref no	Property	Value	Units
0	P, pressure	4000.0	kPA
1	T, temperature	453.035632399999...	deg K
2	v, specific volume	0.00112470762768...	m³/kg
3	h, enthalpy	764.2051837193761	KJ/kg
4	u, internal energy	759.7063532086379	KJ/kg
5	s, entropy	2.13433551518834...	KJ/kgK
6	x, quality	-1.0	kg vap/kg mix
7	Cp, specific heat at ...	4.391833187127093	KJ/kgK
8	Cv, specific heat at ...	3.394394528992311	KJ/kgK
9	α isobaric thermal ...	0.00120433832985...	1/K
10	κ isothermal compr...	7.40939628333338...	1/Pa
11	η Dynamic viscosity	1.51239131991014...	Pa.s
12	k Thermal conducti...	0.67698751740702...	W/mK
13	σ Surface tension	0.04221574667207...	N/m
14	Prandtl number	0.98113631638955...	
15	ρ density	889.1199591654761	kg/m³
16	speed of sound	1401.42598989849...	m/s

Point	P kPa	T K	T C	h	s	x	**η**	**0.9**
1	4000	773.15	500	3445.837	7.091896	2		
13=7i	1100	577.0551	303.9051	3057.477	7.091896	2	s13=S1	isentropic process
7	1100	595.1412	321.9912	3096.313	7.158157	2	h7=h1-η(h1-h13)	
14=8i	1000	582.1861	309.0361	3071.02	7.158157	2	s14=s7	isentropic process
8	1000	583.4816	310.3316	3073.55	7.162489	2	h8=h7-η(H7-h14)	
15=2i	3.5	299.8232	26.67319	2142.183	7.162489	0.832882	s15=s8	isentropic process
2	3.5	299.8232	26.67319	2235.319	7.473128	0.871088	h2=h8-η(h8-h15)	
3	3.5	299.8232	26.67319	111.8357	0.390659	0	saturated liquid	
16=4i	4000	299.901	26.751	115.8368	0.390659	-1	s16=s4	isentropic process
4	4000	300.8993	27.74932	120.2826	2.796653	-1	h4=h3+(h15-h3)/η	
9	1000	453.0356	179.8856	762.6828	2.138431	0	x=0	saturated liquid
11	1100	457.2197	184.0697	781.1977	2.178864	0	x=0	saturated liquid
12	1000	453.0356	179.8856	781.1977	2.1793	0.009191	h11=h12	throttling process
10	3.5	299.8232	26.67319	762.6828	2.561434	0.266988	h9=h10	throttling process

6	4000	457.2197	184.0697	782.6146	2.174785	-1	T6=T11
5	4000	453.0356	179.8856	764.2052	2.134336	-1	T5=T9

y1=(h6-h5)/(h7-h11)		0.007952
y2=((h5-h4)-y1*(h12-h9))/(h8-h)		0.276376
1-y1-y2		0.715673
W1	h1-h7	349.5245
W2	(1-y1)*(h7-h8)	22.58238
W3	(1-y1-y2)*(h8-h2)	599.8985
W		972.0053
Q	h1-h6	2663.223
ηcycle		0.364973

COMPUTER SOLUTION

```
public class rankine_2_close_feed_heater
{
            public static void main(String arg[])
  {double T[]=new double[20];
   double P[]=new double[20];
   double h[]=new double[20];
   double s[]=new double[20];
   double x[]=new double[20];
   double y1,y2;
   double eta=0.9;
   //a[0]=P;a[1]=T;a[2]=1.0/ro;a[3]=h2(P,T);a[4]=u2(P,T);a[5]=s2(P,T);
   //a[6]=2.0;a[7]=Cp2(P,T);a[8]=Cv2(P,T);a[9]=alpha2(P,T);a[10]=kappa2(P,T);
   steamIAPWS_IF97 st=new steamIAPWS_IF97();

P[1]=4000;P[13]=1100;P[7]=1100;P[4]=1000;P[8]=1000;P[15]=3.5;P[2]=3.5;P[3]=3.5;P[16]=4000;P[4]=4000;P[9]=1000;
P[11]=1100;
   P[12]=1000;P[10]=3.5;P[6]=4000;P[5]=4000;P[14]=1000;

   T[1]=500.0; //degree C
   double a1[]=st.propertyC("tp",T[1],P[1]);
   h[1]=a1[3];s[1]=a1[5];x[1]=a1[6];
   int i=1;
   System.out.println("P["+i+"] = "+P[i]+"T["+i+"] = "+T[i]+"h["+i+"] = "+h[i]+"s["+i+"] = "+s[i]);

   //13=i7 isentropic proses
   s[13]=s[1];
   double a13[]=st.propertyC("ps",P[13],s[13]);
   h[13]=a13[3];T[13]=a13[1];x[13]=a13[6];
   i=13;
   System.out.println("P["+i+"] = "+P[i]+"T["+i+"] = "+T[i]+"h["+i+"] = "+h[i]+"s["+i+"] = "+s[i]);

   //7
   h[7]=h[1]-eta*(h[1]-h[13]);
   double a7[]=st.propertyC("ph",P[7],h[7]);
   T[7]=a7[1];s[7]=a7[5];x[7]=a7[6];
   i=7;
   System.out.println("P["+i+"] = "+P[i]+"T["+i+"] = "+T[i]+"h["+i+"] = "+h[i]+"s["+i+"] = "+s[i]);


   //14=i8 isentropic proses
   s[14]=s[7];
   double a14[]=st.propertyC("ps",P[14],s[14]);
   h[14]=a14[3];T[14]=a14[1];x[14]=a14[6];
   i=14;
   System.out.println("P["+i+"] = "+P[i]+"T["+i+"] = "+T[i]+"h["+i+"] = "+h[i]+"s["+i+"] = "+s[i]);
```

```
//8
h[8]=h[7]-eta*(h[7]-h[14]);
double a8[]=st.propertyC("ph",P[8],h[8]);
T[8]=a8[1];s[8]=a8[5];x[8]=a8[6];
i=8;
System.out.println("P["+i+"] = "+P[i]+"T["+i+"] = "+T[i]+"h["+i+"] = "+h[i]+"s["+i+"] = "+s[i]);


//15=i2 isentropic proses
s[15]=s[8];
double a15[]=st.propertyC("ps",P[15],s[15]);
h[15]=a15[3];T[15]=a15[1];x[15]=a15[6];
i=15;
System.out.println("P["+i+"] = "+P[i]+"T["+i+"] = "+T[i]+"h["+i+"] = "+h[i]+"s["+i+"] = "+s[i]);


//2
h[2]=h[8]-eta*(h[8]-h[15]);
double a2[]=st.propertyC("ph",P[2],h[2]);
T[2]=a2[1];s[2]=a2[5];x[2]=a2[6];
i=2;
System.out.println("P["+i+"] = "+P[i]+"T["+i+"] = "+T[i]+"h["+i+"] = "+h[i]+"s["+i+"] = "+s[i]);

//3 saturated liquid
x[3]=0.0;
double a3[]=st.propertyC("px",P[3],x[3]);
T[3]=a3[1];s[3]=a3[5];h[3]=a3[3];
i=3;
System.out.println("P["+i+"] = "+P[i]+"T["+i+"] = "+T[i]+"h["+i+"] = "+h[i]+"s["+i+"] = "+s[i]);

//16=4i
s[16]=s[3];
double a16[]=st.propertyC("ps",P[16],s[16]);
T[16]=a3[1];x[16]=a3[6];h[16]=a16[3];
i=16;
System.out.println("P["+i+"] = "+P[i]+"T["+i+"] = "+T[i]+"h["+i+"] = "+h[i]+"s["+i+"] = "+s[i]);

//4
h[4]=h[16]+(h[16]-h[3])/eta;
double a4[]=st.propertyC("ph",P[4],h[4]);
T[4]=a4[1];s[4]=a4[5];x[4]=a4[6];
i=4;
System.out.println("P["+i+"] = "+P[i]+"T["+i+"] = "+T[i]+"h["+i+"] = "+h[i]+"s["+i+"] = "+s[i]);

//9 saturated liquid x=0
x[9]=0.0;
double a9[]=st.propertyC("px",P[9],x[9]);
T[9]=a9[1];s[9]=a9[5];h[9]=a9[3];
i=9;
System.out.println("P["+i+"] = "+P[i]+"T["+i+"] = "+T[i]+"h["+i+"] = "+h[i]+"s["+i+"] = "+s[i]);

//11 saturated liquid x=0
x[11]=0.0;
double a11[]=st.propertyC("px",P[11],x[11]);
T[11]=a11[1];s[11]=a11[5];h[11]=a11[3];
i=11;
System.out.println("P["+i+"] = "+P[i]+"T["+i+"] = "+T[i]+"h["+i+"] = "+h[i]+"s["+i+"] = "+s[i]);

//12 throttling process
h[12]=h[11];
 double a12[]=st.propertyC("ph",P[12],h[12]);
T[12]=a12[1];s[12]=a12[5];x[12]=a12[6];
i=12;
   System.out.println("P["+i+"] = "+P[i]+"T["+i+"] = "+T[i]+"h["+i+"] = "+h[i]+"s["+i+"] = "+s[i]);

//10 throttling process
```

```
    h[10]=h[9];
     double a10[]=st.propertyC("ph",P[10],h[10]);
    T[10]=a10[1];s[10]=a10[5];x[10]=a10[6];
    i=10;
       System.out.println("P["+i+"] = "+P[i]+"T["+i+"] = "+T[i]+"h["+i+"] = "+h[i]+"s["+i+"] = "+s[i]);

    //6  T6=T11
    T[6]=T[11];
    double a6[]=st.propertyC("tp",T[6]-273.15,P[6]);
    h[6]=a6[3];s[6]=a6[5];x[6]=a6[6];
    i=6;
    System.out.println("P["+i+"] = "+P[i]+"T["+i+"] = "+T[i]+"h["+i+"] = "+h[i]+"s["+i+"] = "+s[i]);

    //5  T5=T9
    T[5]=T[9];
    double a5[]=st.propertyC("tp",T[5]-273.15,P[5]);
    h[5]=a5[3];s[5]=a5[5];x[5]=a5[6];
    i=5;   System.out.println("P["+i+"] = "+P[i]+"T["+i+"] = "+T[i]+"h["+i+"] = "+h[i]+"s["+i+"] = "+s[i]);

     y1=(h[6]-h[5])/(h[7]-h[11]);
     y2=((h[5]-h[4])-y1*(h[12]-h[9]))/(h[8]-h[9]);
     double W1=h[1]-h[7];
     double W2=(1.0-y1)*(h[7]-h[8]);
     double W3=(1.0-y1-y2)*(h[8]-h[2]);
     double W=W1+W2+W3;
     double Q=h[1]-h[6];
     double etacycle=W/Q;
     System.out.println("y1="+y1+"y2="+y2);
     System.out.println("W1="+W1+"W2="+W2+"W3="+W3);
     System.out.println("W="+W+"Q="+Q+"eta="+etacycle);




   }

}
```

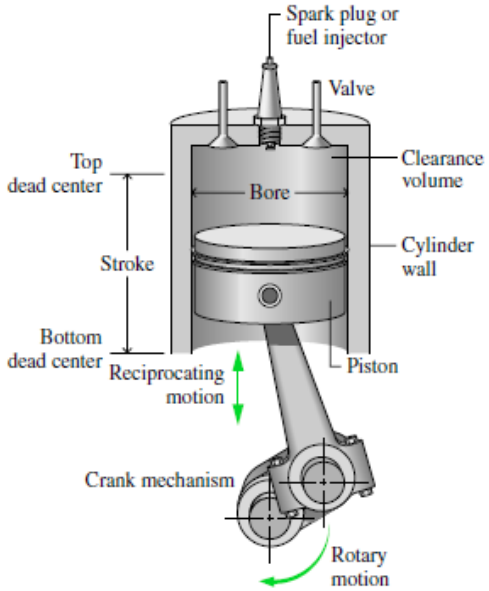
```
---------- Capture Output ----------
> "E:\co\java\bin\javaw.exe" rankine_2_close_feed_heater
P[1] = 4000.0T[1] = 500.0h[1] = 3445.8374067447944s[1] = 7.091896426703189
P[13] = 1100.0T[13] = 577.0551363079155h[13] = 3057.4768856055343s[13] = 7.091896426703189
P[7] = 1100.0T[7] = 595.1412181293588h[7] = 3096.3129377194605s[7] = 7.158157490228933
P[14] = 1000.0T[14] = 582.1860747401407h[14] = 3071.020282449884s[14] = 7.158157490228933
P[8] = 1000.0T[8] = 583.369793462395h[8] = 3073.5495479768415s[8] = 7.162488862490142
P[15] = 3.5T[15] = 299.8231926726793h[15] = 2142.182534607238s[15] = 7.162488862490142
P[2] = 3.5T[2] = 299.8231926726793h[2] = 2235.319235944198s[2] = 7.473128361656038
P[3] = 3.5T[3] = 299.8231926726793h[3] = 111.8356534054894s[3] = 0.39065853936422884
P[16] = 4000.0T[16] = 299.8231926726793h[16] = 115.83682515355919s[16] = 0.39065853936422884
P[4] = 4000.0T[4] = 300.9670871706007h[4] = 120.28257154030341s[4] = 0.4054377391765746
P[9] = 1000.0T[9] = 453.0356323914666h[9] = 762.6828443354106s[9] = 2.138431350899127
P[11] = 1100.0T[11] = 457.21967567794565h[11] = 781.1977431114818s[11] = 2.1788638915739202
P[12] = 1000.0T[12] = 453.0356323914666h[12] = 781.1977431114818s[12] = 2.179300036338758
P[10] = 3.5T[10] = 299.8231926726793h[10] = 762.6828443354106s[10] = 2.561433781795628
P[6] = 4000.0T[6] = 457.21967567794565h[6] = 782.6145859333573s[6] = 2.174784561603826
P[5] = 4000.0T[5] = 453.0356323914666h[5] = 764.2051836818985s[5] = 2.1343355151056147

y1=0.007951829910811885y2=0.27858611827383195

W1=349.52446902533393W2=22.582379139192156W3=598.0455183166359
W=970.1523664811621Q=2663.222820811437eta=0.364277580869322
```

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

3.4 OTTO CYCLE



Air standard Otto cycle can be considered as an approximation to internal combustion engine operation cycle. As mentioned in the Brayton cycle actual working fluid composition will be changed during the process. In this approximation, it is taken as air. Note that , in this process cycle is considered as a closed system process.

Program 3.4.1 ideal air Otto cycle

```
public class otto
{
Gas g;
double TT1[],ss1[];
double TT2[],ss2[];
double W12,W34,Wnet,Q23,Q41,eta,T1,P1,T2,P2,T3,P3,T4,P4;
public otto(double m,double T1,double P1, double compressionratio, double T3,double
etacompression,double etaexpansion)
{ TT1=new double[24];ss1=new double[24];
  TT2=new double[3];ss2=new double[3];
g=new Gas("air");
g.mole=false;
// 1
double a1[]=g.property(T1,P1);
double h1=a1[3];
double u1=a1[4];
double s1=a1[5];
double v1=a1[2];
double vr1=a1[18];
System.out.println("p1="+P1+"t1="+T1+"h1="+h1+"u1="+u1+"s1="+s1+"vr1="+vr1);
double vr2=vr1/compressionratio;
double v2=v1/compressionratio;
double T2i=g.Tvi(vr2,v2);
//double P2=P1*T2i/T1*v1/v2;
double P2=g.P(T2i,v2);
double a2[]=g.property(T2i,P2);
double h2i=a2[3];
double u2i=a2[4];
```

```
double s2i=a2[5];
double Cp2i=a2[9];
System.out.println("p2="+P2+"t2="+T2i+"h2="+h2i+"u2i="+u2i+"s2="+s2i+"vr2="+vr2);

double pr1=a1[17];
double h2=h1+(h2i-h1)/etacompression;
double T2=T2i+(h2-h2i)/Cp2i;
a2=g.property(T2,P2);
double s2=a2[5];
double u2=a2[4];
double P3=P2*T3/T2;
TT1[0]=T1;ss1[0]=s1;
TT1[1]=T2;ss1[1]=s2;
System.out.println("p2="+P2+"t2="+T2+"h2="+h2+"u2="+u2+"s2="+s2+"vr2="+vr2);

double a3[]=g.property(T3,P3);
double vr3=a3[18];
double s3=a3[5];
double h3=a3[3];
double v3=a3[2];
double u3=a3[4];
System.out.println("p3="+P3+"t3="+T3+"h3="+h3+"u3="+u3+"s3="+s3+"vr2="+vr3);
double vr4=vr3*compressionratio;
double T4i=g.Tvi(vr4,v1);

double P4i=P1*T4i/T1;
double a4[]=g.property(T4i,P4i);
double v4i=a4[2];
double s4i=a4[5];
double h4i=a4[3];
double u4i=a4[4];
double Cp4i=a4[9];
System.out.println("p4i="+P4i+"t4i="+T4i+"h4i="+h4i+"u4i="+u4i+"s4i="+s4i);

double h4=h3-(h3-h4i)*etaexpansion;
double T4=T4i+(h4-h4i)/Cp4i;
double P4=P1*T4/T1;
a4=g.property(T4,P4);
double v4=a4[2];
double s4=a4[5];
double u4=a4[4];
System.out.println("p4="+P4+"t4="+T4+"h4="+h4+"u4="+u4+"s4="+s4+"vr4="+vr4);

double PP=0.0;
for(int i=0;i<10;i++)
{TT1[i+2]=T2+i*(T3-T2)/10;
 PP=P2*TT1[i+2]/T2;
 a3=g.property(TT1[i+2],PP);
 ss1[i+2]=a3[5];
}
TT1[12]=T3;ss1[12]=s3;
TT1[13]=T4;ss1[13]=s4;

TT2[0]=T1;ss2[0]=s1;
TT2[1]=T2i;ss2[1]=s2i;
TT2[2]=T2;ss2[2]=s2;

for(int i=0;i<10;i++)
```

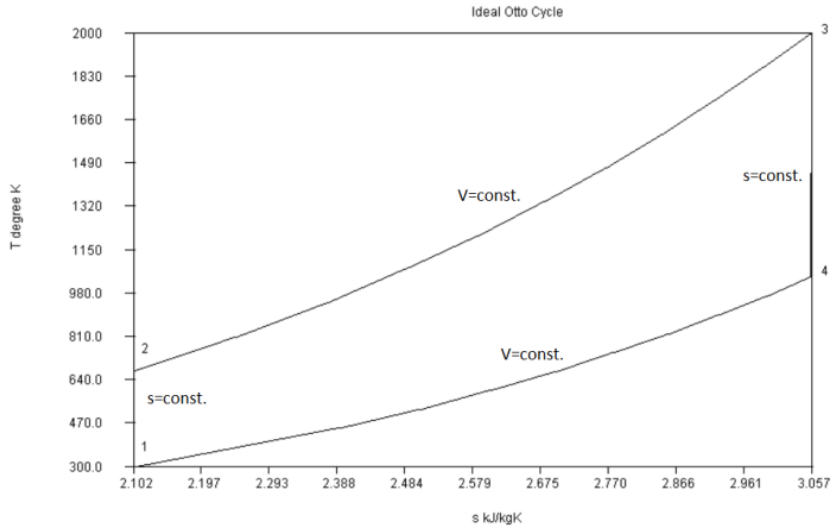
```
{TT1[i+14]=T4-i*(T4-T1)/10;
 PP=P4*TT1[i+14]/T4;
 a3=g.property(TT1[i+14],PP);
 ss1[i+14]=a3[5];
}
TT1[23]=T1;
ss1[23]=s1;
W12=m*(u2-u1);
W34=m*(u3-u4);
Wnet=W34-W12;
Q23=m*(u3-u2);
Q41=m*(u4-u1);
Wnet=W34-W12;
eta=Wnet/Q23;
this.T1=T1;
this.P1=P1;
this.T2=T2;
this.P2=P2;
this.T3=T3;
this.P3=P3;
this.T4=T4;
this.P4=P4;
double aa[][]={ss1,TT1};
Text.printT(aa);
Plot pp=new Plot(ss1,TT1);
//pp.addData(ss2,TT2);
pp.plot();
}
public String toString()
{
String s=" Otto cycle \n";
s+="W12 = "+W12+" kJ \n";
s+="W34 = "+W34+" kJ\n";
s+="Wnet = "+Wnet+" kW\n";
s+="Q23 = "+Q23+" kJ\n";
s+="Q41 = "+Q41+" kJ \n";
s+="Cycle efficiency eta  = "+eta+"\n";
return s;
}

}
```

Program 3.4.2 ideal air Otto cycle test

```
public class ottotest
{
  public static void main(String arg[])
  {
                double mhava=6.5e-4;  // kg air in the cylinder
                double T1=300.0; // air inlet temperature degree K
                double P1=1.0;   // air inlet pressure bar
                double CR=8.0;   // compression ratio V2/V1
                double T3=2000.0; //degree K maximum temperature
                double etaexpansion=1.0; //compression isentropic efficiency
                double etacompression=1.0; //expansion isentropic efficien
     otto o=new otto(mhava,T1,P1,CR,T3,etacompression,etaexpansion);
     System.out.println(o.toString());
  }
}
```

```
---------- Capture Output ----------
> "C:\java\bin\javaw.exe" ottotest
p1=1.0t1=300.0h1=398.40865520622174u1=312.29026565958355s1=2.1016020128980477
vr1=620.6296072019526
p2=17.94464819940428t2=672.9243074776605h2=782.6029059036684u2i=589.4323804144589
s2=2.1016020124781805vr2=77.57870090024407
p2=17.94464819940428t2=672.9243074776605h2=782.6029059036684u2=589.4323804144589
s2=2.1016020124781805vr2=77.57870090024407
p3=53.333333333333336t3=2000.0h3=2348.910601482316u3=1774.7880045047282s3=3.0569417279
203743
vr2=2.7823444902218544
p4i=3.4717396455816925t4i=1041.5218936745077h4i=1191.946317933364u4i=892.9656907306523
s4i=3.056941727667531
p4=3.4717396455816925t4=1041.5218936745077h4=1191.946317933364u4=892.9656907306523
s4=3.056941727667531vr4=22.258755921774835
 Otto cycle
W12 = 0.18014237459066895 kJ
W34 = 0.5731845039531492 kJ
Wnet = 0.3930421293624803 kJ
Q23 = 0.7704811556586751 kJ
Q41 = 0.37743902629619464 kJ
Cycle efficiency eta  = 0.5101255578749012
```

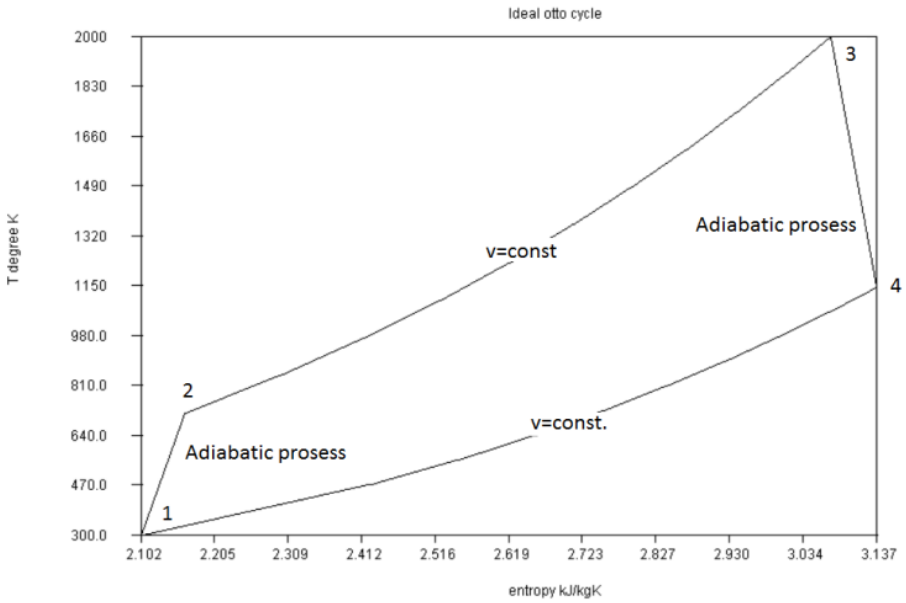


Program 3.4.2 adiabatic air Otto cycle test

```
public class ottotest
{
  public static void main(String arg[])
  {            double mhava=6.5e-4;  // kg air in the cylinder
                double T1=300.0; // air inlet temperature degree K
                double P1=1.0;   // air inlet pressure bar
                double CR=8.0;   // compression ratio V2/V1
                double T3=2000.0; //degree K maximum temperature
                double etaexpansion=0.9; //compression isentropic efficiency
                double etacompression=0.9; //expansion isentropic efficiency
                otto o=new otto(mhava,T1,P1,CR,T3,etacompression,etaexpansion);
                System.out.println(o.toString());
  }
}
```

```
---------- Capture Output ----------
> "C:\java\bin\javaw.exe" ottotest
p1=1.0t1=300.0h1=398.40865520622174u1=312.29026565958355s1=2.1016020128980477
vr1=620.6296072019526
p2=17.94464819940428t2=672.9243074776605h2=782.6029059036684u2i=589.4323804144589
s2=2.1016020124781805vr2=77.57870090024407
p2=17.94464819940428t2=712.8832847545507h2=825.2911559811625u2=620.8421544326077
s2=2.1635016449112885vr2=77.57870090024407
p3=50.3438601610156t3=2000.0h3=2348.910601482316u3=1774.7880045047282s3=3.073500818165091
vr2=2.7823444902218544
p4i=3.4717396455816925t4i=1041.5218936745077h4i=1191.946317933364u4i=892.9656907306523
s4i=3.056941727667531
p4=3.807487638328895t4=1142.2462914986686h4=1307.6427462882593u4=980.6221163316686
s4=3.137266639144191vr4=22.258755921774835
 Otto cycle
W12 = 0.2005587277024657 kJ
W34 = 0.5162078273124887 kJ
Wnet = 0.315649099610023 kJ
Q23 = 0.7500648025468782 kJ
Q41 = 0.4344157029368552 kJ
Cycle efficiency eta  = 0.42082910508295085

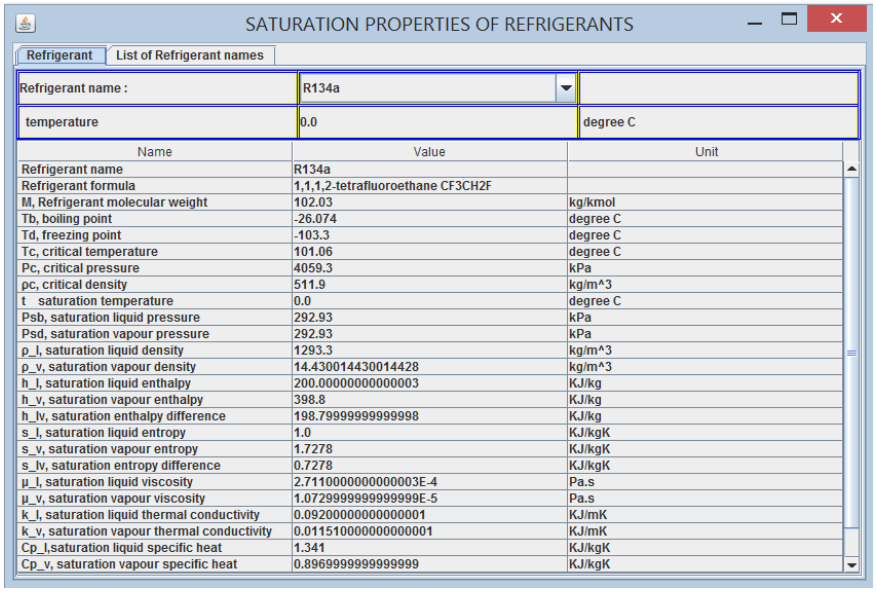
> Terminated with exit code 0.
```



Power cycles will be re-investigated after definition of combustion processes.

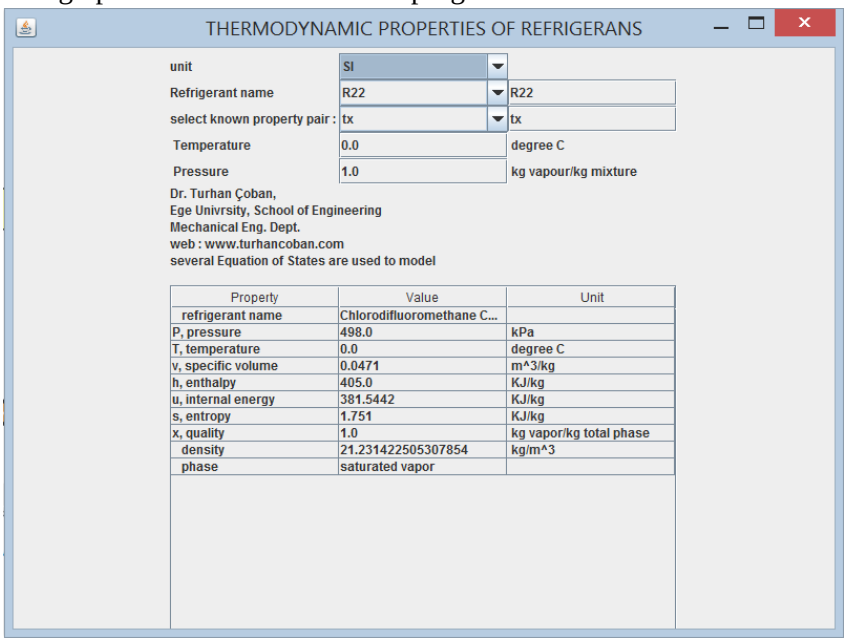
4. REFRIGERATION & HEAT PUMP CYCLES

Refrigeration & Heat pump cycles are basic cycles to use for heating and cooling spaces (heat transfer). Ususally work supply for the process and heat energy output is taken. A large range of refrigerants might use in refrigeration cycle. Several EOS for refrigerants are collected in one class, refrigerant.java. the class code is listed in A5. In addition to that, saturation properties of the refrigerants are evaluated in **ref_CS3EN.java** class. This program is basically a cubic spline curve fitting of the saturation $P_s(T_s)$ data. One advantage of such a curve fitting is simple creation of inverse curve fittin so function $T_s(P_s)$ can be evaluated as easily. Ref_CS3EN is also have thermophysical properties of refrigerants such as viscosity, thermal conductivity, surface tension. These packages are utilized in heat transfer applications as well as thermodynamic applications. Graphic user output of **ref_CS3EN** class is as follows:



Name	Value	Unit
Refrigerant name	R134a	
Refrigerant formula	1,1,1,2-tetrafluoroethane CF3CH2F	
M, Refrigerant molecular weight	102.03	kg/kmol
Tb, boiling point	-26.074	degree C
Td, freezing point	-103.3	degree C
Tc, critical temperature	101.06	degree C
Pc, critical pressure	4059.3	kPa
ρc, critical density	511.9	kg/m^3
t saturation temperature	0.0	degree C
Psb, saturation liquid pressure	292.93	kPa
Psd, saturation vapour pressure	292.93	kPa
ρ_l, saturation liquid density	1293.3	kg/m^3
ρ_v, saturation vapour density	14.430014430014428	kg/m^3
h_l, saturation liquid enthalpy	200.00000000000003	KJ/kg
h_v, saturation vapour enthalpy	398.8	KJ/kg
h_lv, saturation enthalpy difference	198.79999999999998	KJ/kg
s_l, saturation liquid entropy	1.0	KJ/kgK
s_v, saturation vapour entropy	1.7278	KJ/kgK
s_lv, saturation entropy difference	0.7278	KJ/kgK
μ_l, saturation liquid viscosity	2.7110000000000003E-4	Pa.s
μ_v, saturation vapour viscosity	1.0729999999999999E-5	Pa.s
k_l, saturation liquid thermal conductivity	0.09200000000000001	KJ/mK
k_v, saturation vapour thermal conductivity	0.011510000000000001	KJ/mK
Cp_l,saturation liquid specific heat	1.341	KJ/kgK
Cp_v, saturation vapour specific heat	0.8969999999999999	KJ/kgK

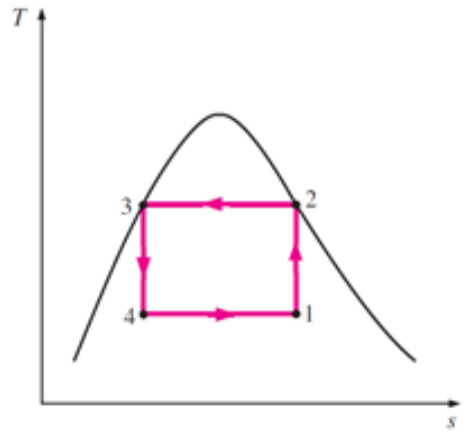
refrigerantEN is the equation of states for thermodynamic property calculations.
refTableEN is graphic user interface for this program.

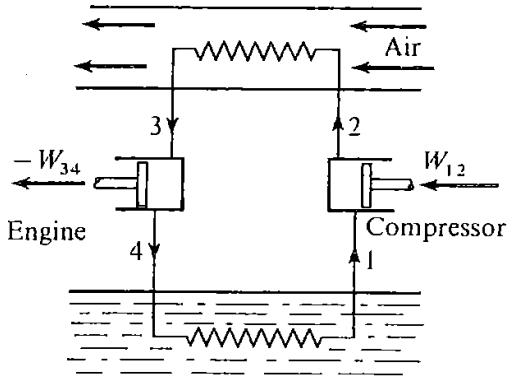


Property	Value	Unit
refrigerant name	Chlorodifluoromethane C...	
P, pressure	498.0	kPa
T, temperature	0.0	degree C
v, specific volume	0.0471	m^3/kg
h, enthalpy	405.0	KJ/kg
u, internal energy	381.5442	KJ/kg
s, entropy	1.751	KJ/kg
x, quality	1.0	kg vapor/kg total phase
density	21.231422505307854	kg/m^3
phase	saturated vapor	

By using this class we can able to program refrigerant cycles.

4.1 CARNOT CYCLE





Carnot cycle is an ideal cycle and it is important as a reference cycle . It consists of two isothermal and two isentropic processes. If a carnot cycle constructed by using an isentropic compressor , a condenser , an isentropic turbine and a boiler (evaporator), and, the cycle can be calculated by the following equations:

$W_{isentropiccompressor} = m(h_2 - h_1)$ $\quad s_2 = s_1$ (4.1.1)

$Q_{condenser} = m(h_2 - h_3) = mT_2(s_2 - s_3) = mT_2(s_1 - s_4)$ (4.1.2)

$W_{isentropicturbine} = m(h_3 - h_4)$ $\quad s_3 = s_4$ (4.1.3)

$Q_{evaporator} = m(h_1 - h_4) = mT_1(s_1 - s_4)$ (4.1.4)

$W_{netin} = W_{isentropiccompressor} - W_{isentropicturbine} = Q_{condenser} - Q_{boiler}$ (4.1.5)

$W_{netin} = mT_2(s_1 - s_4) - mT_1(s_1 - s_4) = m(T_2 - T_1)(s_1 - s_4)$ (4.1.6)

$Coefficient of Performance for evaporator = \frac{Q_{evaporator}}{W_{compressor} - W_{turbine}} = \frac{mT_1(s_1 - s_4)}{m(T_2 - T_1)(s_1 - s_4)} = \frac{T_1}{(T_2 - T_1)}$ (4.1.7)

$Coefficient of Performance for condenser = \frac{Q_{condenser}}{W_{compressor} - W_{turbine}} = \frac{mT_2(s_1 - s_4)}{m(T_2 - T_1)(s_1 - s_4)} = \frac{T_2}{(T_2 - T_1)}$ (4.1.8)

A computer model is developed to analyse this cycle in details

Program 4.1.1 ideal refrigeration Carnot cycle

```
public class carnot1
{
// a R134 ideal carnot refrigeration cycle
// refrigerantEN b=new refrigerantEN("R134a");
// double a[]=b.property("tp",30.0,1.01325);
///---- String s values and meanings --------
//    tv=temperature-specific volume
//    tp=temperature-pressure
//    th=temperature-enthalpy
//    tu=temperature-internal energy
//    ts=temperature-entropy
//    tx=temperature quality
//    pv=pressure-specific volume
//    pt=pressure-temperature
//    ph=pressure-enthapy
//    pu=pressure-internal energy
//    ps=pressure-entropy
//    px=pressure-quality
//    vp=specific volume-pressure
```

```
//    vt=specific volume-temperature
//---- output values (method property) ---------
// r[0] P Pressure kPa
// r[1] t temperature degree C
// r[2] v specific volume m^3/kg
// r[3] h enthapy KJ/kg
// r[4] u internal energy KJ/kg
// r[5] s entropy KJ/kgK
// r[6] x quality kg vapour/kg total
// r[7] ro density kg/m^3

double T[];
double P[];
double h[];
double s[];
double x[];
double COP_condenser,COP_evaporator,COP_evap_carnot,COP_condenser_carnot;
double m;//kg/s
double Wt,Wp,W;//work
double Qevaporator,Qcondenser;
refrigerantEN b;
double c1[][];
double c3[][];
public carnot1(double mi,double P1, double P2)
{
m=mi;
b=new refrigerantEN("R134a");
T=new double[9];
P=new double[9];
h=new double[9];
s=new double[9];
x=new double[9];

P[1]=P1;//compressor inlet
P[2]=P2;//compressor outlet
c1=new double[2][20];
c3=new double[2][20];
}
public void cycle()
{
// isentropic compressor output
double a1[]=b.property("px",P[2],1.0);
h[2]=a1[3];
s[2]=a1[5];
T[2]=a1[1];
// isentropik compressor input
s[1]=s[2];
a1=b.property("ps",P[1],s[1]);
T[1]=a1[1];
h[1]=a1[3];
x[1]=a1[6];

//isentropic turbine input
P[3]=P[2];
a1=b.property("px",P[3],0.0);
h[3]=a1[3];
s[3]=a1[5];
T[3]=a1[1];
```

```
// //isentropic turbine output
P[4]=P[1];
s[4]=s[3];
a1=b.property("ps",P[4],s[4]);
h[4]=a1[3];
s[4]=a1[5];
T[4]=a1[1];
x[4]=a1[6];


// isentropik compressor
Wp=m*(h[2]-h[1]);
Wt=m*(h[3]-h[4]);
W=Wp-Wt;
Qevaporator=m*(h[1]-h[4]);
Qcondenser=m*(h[2]-h[3]);
COP_evaporator=Qevaporator/W;
COP_condenser_carnot=(T[2]+273.15)/(T[2]-T[1]);
COP_condenser=Qcondenser/W;
COP_evap_carnot=(T[1]+273.15)/(T[2]-T[1]);
}

public String toString()
{
cycle();
String ss="R134a carnot refrigeration cycle\n";
ss+=" compressor work = "+Wp+" kW\n";
ss+=" evaporator work = "+Wt+" kW\n";
ss+=" net work input= "+W+" kW\n";
ss+=" condenser heat output = "+Qcondenser+" kW\n";
ss+=" evaporator heat input = "+Qevaporator+" kW\n";
ss+=" COP evaporator    = " + COP_evaporator+" kW\n";
ss+=" COP condenser     = " + COP_condenser+" kW\n";
ss+=" COP evaporator  carnot = " + COP_evap_carnot+" kW\n";
ss+=" COP condenser   carnot = " + COP_condenser_carnot+" kW\n";
ss+=" h1 compressor input = "+h[1] +" kJ/kg\n";
ss+=" T1 compressor input = "+T[1] +" derece C\n";
ss+=" P1 compressor input = "+P[1] +" kPa \n";
ss+=" s1 compressor input = "+s[1] +" kJ/kgK\n";
ss+=" x1 compressor input = "+x[1] +" kgvapor/kgtotal"+"\n";
ss+=" T2 compressor output = "+T[3] +" derece C\n";
ss+=" P2 compressor output = "+P[3] +" kPa \n";
ss+=" s2 compressor output = "+s[2] +" kJ/kgK\n";
ss+=" x2 compressor output = "+x[2] +" kgvapor/kgtotal"+"\n";
ss+=" h3 turbine input = "+h[3] +" kJ/kg\n";
ss+=" T3 turbine input = "+T[3] +" derece C\n";
ss+=" P3 turbine input = "+P[3] +" kPa \n";
ss+=" s3 turbine input = "+s[3] +" kJ/kgK\n";
ss+=" x3 turbine input = "+x[3] +" kgvapor/kgtotal"+"\n";
ss+=" h4 turbine output = "+h[4] +" kJ/kg\n";
ss+=" T4 turbine output = "+T[4] +" derece C\n";
ss+=" s4 turbine output = "+s[4] +" kJ/kgK\n";
ss+=" x4 turbine output = "+x[4] +" kgvapor/kgtotal"+"\n";
return ss;
}
public double[][] TS1()
{
double a[][]=Text.readDoubleT("R134a_Ps.txt");
```

```
return a;
}

public double[][] TS()
{
double a[][]=new double[2][421];
double tc=b.r.Tc;
System.out.println("tc="+tc);
double dt=(tc-10.0)/100.0;
int i=0;
double t;
for(t=-90.0;t<=tc;t+=dt)
{double a1[]=b.property("tx",t,0.0);
 a[1][i]=t;
 a[0][i]=a1[5];
 i++;
}
double a3[]=b.property("tx",tc,1.0);
 a[1][i]=t;
 a[0][i]=a3[5];
 i++;
for(t=tc;t>=-90;t-=dt)
{double a2[]=b.property("tx",t,1.0);
 a[1][i]=t;
 a[0][i]=a2[5];
 i++;
}
return a;
}

public void plot()
{
 Plot pp=new Plot(TS());
 double t1[]={T[1],T[2],T[3],T[4],T[1]};
 double s1[]={s[1],s[2],s[3],s[4],s[1]};
 pp.setPlabel("Ideal Carnot refrigeration R134a Cycle");
 pp.setYlabel("T, degree C");
 pp.setXlabel("s entropy kJ/kgK");
 pp.addData(s1,t1);
 pp.plot();
}
}
```

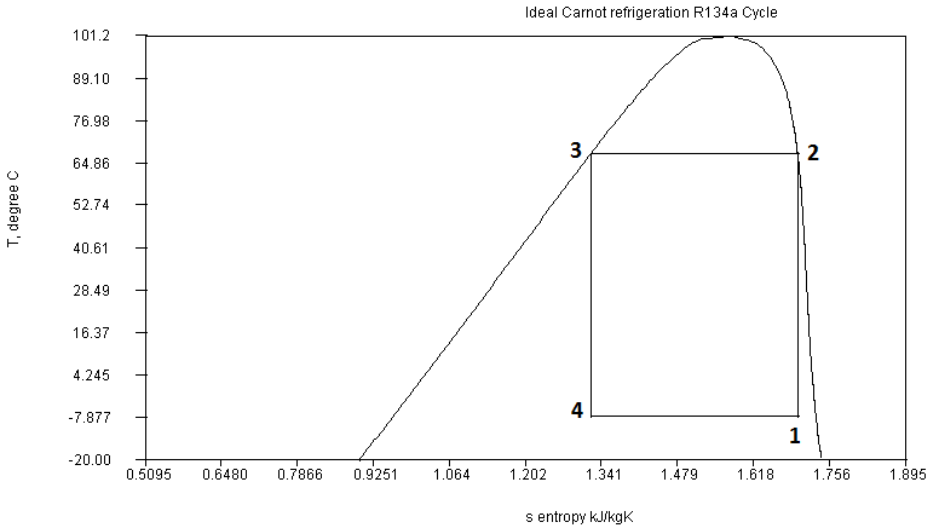
Program 4.2.1 ideal refrigeration Carnot cycle

```
import javax.swing.*;

public class carnottest
{
public static void main(String arg[])
{
double m=50/745.25928;  // kg/s çevrim R134a debisi
double P2=2000.0;         //  bar turbin giriş basıncı
double P1=220.0 ;         //  bar turbin çıkış basıncı
carnot1 r=new carnot1(m,P1,P2);
System.out.println(r.toString());
r.plot();
}
}
```

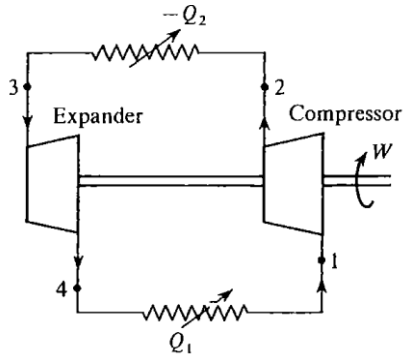
```
---------- Capture Output ----------
> "C:\java\bin\javaw.exe" carnottest
R134a carnot refrigeration cycle
 compressor work = 2.9194665890306633 kW
 evaporator work = 1.019143155011926 kW
 net work input= 1.9003234340187372 kW
 condenser heat output = 8.611038292271756 kW
 evaporator heat input = 6.710714858253018 kW
 COP evaporator    = 3.5313540516949975 kW
 COP condenser     = 4.531354051694998 kW
 COP evaporator  carnot = 3.5351212616738557 kW
 COP condenser   carnot = 4.535121261673855 kW
 h1 compressor input = 385.2301064454953 kJ/kg
 T1 compressor input = -7.63976989535311 derece C
 P1 compressor input = 220.0 kPa
 s1 compressor input = 1.6991352552692158 kJ/kgK
 x1 compressor input = 0.9566957691465375 kgvapor/kgtotal
 T2 compressor output = 67.46662970207672 derece C
 P2 compressor output = 2000.0 kPa
 s2 compressor output = 1.6991352552692158 kJ/kgK
 x2 compressor output = 0.0 kgvapor/kgtotal
 h3 turbine input = 300.39617385297873 kJ/kg
 T3 turbine input = 67.46662970207672 derece C
 P3 turbine input = 2000.0 kPa
 s3 turbine input = 1.3222852352261514 kJ/kgK
 x3 turbine input = 0.0 kgvapor/kgtotal
 h4 turbine output = 285.2056559745564 kJ/kg
 T4 turbine output = -7.63976989535311 derece C
 s4 turbine output = 1.3222852352261514 kJ/kgK
 x4 turbine output = 0.46733390915649975 kgvapor/kgtotal

tc=101.06
```



4.3 BRAYTON-ERICSSON CYCLE

Brayton Cycle was investigated as a power cycle in the previous chapter. If cycle operates in inverse direction a gas cycle for refrigeration is defined. Brayton cycle used in airplanes as air conditining cycle.



$W_{compressor} = \dot{m}(h_2 - h_1)$ (4.3.1)

$W_{isentropic\ compressor} = \dot{m}(h_{2s} - h_1)$ $s_{2s} = s_1$ (4.3.2)

$\eta_{isentropic\ compressor} = \frac{W_{compressor}}{W_{isentropic\ compressor}}$ (4.3.3)

$Q_{heat\ exchanger} = \dot{m}(h_2 - h_3)$ $P_2 = P_3$ (4.3.4)

$W_{turbine} = \dot{m}(h_3 - h_4)$ (4.3.5)

$W_{isentropic\ turbine} = \dot{m}(h_3 - h_{4s})$ $s_{4s} = s_3$ $P_4 = P_1$ (4.3.6)

$Q_{refrigeration} = \dot{m}(h_1 - h_4)$ (4.3.7)

$\eta_{isentropic\ turbine} = \frac{W_{turbine}}{W_{isentropic\ turbine}}$ (4.3.8)

$COP_{heat\ exchanger} = \frac{Q_{heat\ exchanger}}{W_{compressor} - W_{turbine}}$ (4.3.9)

$COP_{refrigeration} = \frac{Q_{refrigeration}}{W_{compressor} - W_{turbine}}$ (4.3.10)

In order to test the cycle 3 different program is developed by using 3 different air model. The first one is ideal gas EOS (class gas.java is used for this purpose). For the ideal gas, isentropic process is defined as:

$s(T,P) = s_0 + \int_{T_0}^{T} \frac{C_p(T)}{T} dT - Rln\frac{P}{P_o} = s^0(T) - Rln\frac{P}{P_o}$ (4.3.11)

$s_2(T_2,P_2) - s_1(T_1,P_1) = s_2^0(T_2) - s_1^0(T_1) - Rln\frac{P_2}{P_1}$ (4.3.12)

$\frac{P_2}{P_1} = \exp\left(\frac{s_2^0(T_2) - s_1^0(T_1)}{R}\right) = \frac{\exp\left(\frac{s_2^0(T_2)}{R}\right)}{\exp\left(\frac{s_1^0(T_1)}{R}\right)} = \frac{P_{r2}}{P_{r1}}$ (4.3.13)

$P_r = \exp\left(\frac{s^0(T)}{R}\right)$ (4.3.13) is called reduced pressure and used as a termodynamic variable to determine isentropic process.

In the real processes Equation of state is given as a helmholtz energy, gibbs energy or pressure equation form as a function of density(specific volume) and temperature. So isentropic process should be solved by using numerical root finding methods. If T and s is the given pair, equation to be solved is:

$s(T_{given}, \rho) - s_{given} = 0$ (4.3.14)

If P and s is given pair equation to be solved is:

$s(T,\rho) - s_{given} = 0$ (4.3.15a)

$P(T,\rho) - P = 0$ (4.3.15b)

In this case a system of equation should be solved. As an example Brayton cycle is created by using ideal gas class (Gas.java) and two real gas EOS air_PS(using Peng-Robinson EOS) and air_NIST(using experimental measurement based full EOS for air). Please follow above differences in the given programs. Please also note small differences between the programs (exactly same input conditions are taken)

Program 4.3.1 brayton cycle test program

```
//Brayton-Ericsson refrigeration cycle
public class brayton3
```

```
{
Gas g;
double T[];
double P[];
double s[];
double h[];
double Cp[];
double etakompressor;
double etaturbin;
double Wturbin,Wkompresor,Wnet,Q,Qref,COP,COPref,T1,P1,T2,P2,T3,P3,T4,P4;
public brayton3(double m,double T1,double P1, double P2, double T3,double etakompressori,double etaturbini)
{ set(m,T1,P1,P2,T3,etakompressori,etaturbini);}

public double set(double m,double T1,double P1, double P2, double T3,double etakompressori,double etaturbini)
{
  T=new double[6];
  P=new double[6];
  s=new double[6];
  h=new double[6];
  Cp=new double[6];
  g=new Gas("air");
  g.mole=false;
  P[1]=P1;
  P[2]=P2;
  P[4]=P1;
  P[3]=P2;
  T[1]=T1;
  T[3]=T3;
  etakompressor=etakompressori;
  etaturbin=etaturbini;
// 1
double a1[]=g.property(T[1],P[1]);
h[1]=a1[3];
s[1]=a1[5];
double pr1=a1[17];
// 2i = 0 isentropic process
double pr2=P[2]/P[1]*pr1;
T[0]=g.Ti(pr2,P[2]);
a1=g.property(T[0],P[2]);
h[0]=a1[3];
s[0]=a1[5];
Cp[0]=a1[9];
// 2
h[2]=(h[0]-h[1])/etakompressor+h[1];
T[2]=(h[2]-h[0])/Cp[0]+T[0];
a1=g.property(T[2],P[2]);
s[2]=a1[5];
//3
a1=g.property(T[3],P[3]);
h[3]=a1[3];
s[3]=a1[5];
double pr3=a1[17];
double pr4=P1/P2*pr3;
//4i = 5
T[5]=g.Ti(pr4,P[1]);
a1=g.property(T[5],P[1]);
h[5]=a1[3];
s[5]=a1[5];
Cp[5]=a1[9];
//4
h[4]=h[3]-etaturbin*(h[3]-h[5]);
T[4]=(h[4]-h[5])/Cp[5]+T[5];
a1=g.property(T[4],P[4]);
s[4]=a1[5];
Wturbin=m*(h[3]-h[4]);
Wkompresor=m*(h[2]-h[1]);
Q=m*(h[2]-h[3]);
Qref=m*(h[1]-h[4]);
Wnet=Wkompresor-Wturbin;
COP=Q/Wnet;
COPref=Qref/Wnet;
return T[4];
}
public double[][] line1(double T1,double T2,double P,int n)
{
```

```
double TT1[]=new double[n];
double ss1[]=new double[n];
double aa[];

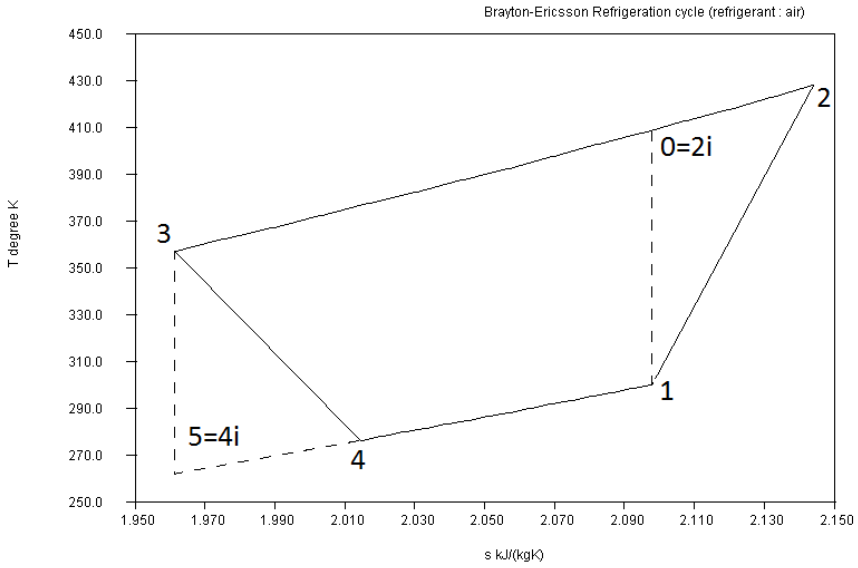
for(int i=0;i<n;i++)
{TT1[i]=T1+i*(T2-T1)/(n-1);
 aa=g.property(TT1[i],P);
 ss1[i]=aa[5];
}
double a[][]={TT1,ss1};
return a;
}
public String toString()
{
String s=" Brayton cycle \n";
s+="Turbine power output W turbine = "+Wturbin+" kW \n";
s+="Compressor power input W kompresor = "+Wkompresor+" kW\n";
s+="Net power output  Wnet = "+Wnet+" kW\n";
s+="Heat exchanger heat removal Q  = "+Q+" kW\n";
s+="refrigeration heat removal Qref = "+Qref+" kW \n";
s+="COP refrigeration  = "+COP+"\n";
s+="COP refrigeration  = "+COP+"\n";
s+="Compressor inlet temperature = "+(T[1]-273.15)+" degree C\n";
s+="Compressor inlet pressure  = "+P[1]+" kPa\n";
s+="Compressor exit temperature = "+(T[2]-273.15)+" degree C\n";
s+="Compressor exit  pressure   = "+P[2]+" kPa\n";
s+="Turbine inlet temperature = "+(T[3]-273.15)+" degree C\n";
s+="Turbine inlet pressure   = "+P[3]+" kPA\n";
s+="Turbine exit temperature = "+(T[4]-273.15)+" degree C\n";
s+="Turbine exit pressure   = "+P[4]+" kPa\n";
return s;
}
public void plot()
{
 double a23[][]=line1(T[2],T[3],P[2],15);

 double tt[]=a23[0];
 double ss[]=a23[1];
 Plot pp=new Plot(ss,tt);
 double a41[][]=line1(T[4],T[1],P[1],15);
 tt=a41[0];
 ss=a41[1];
 pp.addData(ss,tt);
 double t1[]={T[1],T[2]};
 double s1[]={s[1],s[2]};
 pp.addData(s1,t1);
 double t2[]={T[1],T[0]};
 double s2[]={s[1],s[0]};
 pp.addData(s2,t2,1);
 double t3[]={T[3],T[4]};
 double s3[]={s[3],s[4]};
 pp.addData(s3,t3);
 double t4[]={T[3],T[5],T[4]};
 double s4[]={s[3],s[5],s[4]};
 pp.addData(s4,t4,1);
 pp.plot();
}}
```

```
 public class brayton3test
{
  public static void main(String arg[])
  {
                double mhava=0.124; //kg/s air inlet mass flow rate
                double T1=300.0; //derece K air inlet temperature
                double P1=1.01325;// bar air inlet pressure
                double P2=3;// bar compressor exit pressure
                double T3=357.0; //derece K turbine inlet temperature
                double etakompresor=0.85; //compressor isentropic efficiency
                double etaturbin=0.85; ///turbine isentropic efficiency
     brayton3 bry=new brayton3(mhava,T1,P1,P2,T3,etakompresor,etaturbin);
     System.out.println(bry.toString());
     bry.plot();
```

```
  }
}
```

```
---------- Capture Output ----------
> "C:\java\bin\javaw.exe" brayton3test
 Brayton cycle
Turbine power output W turbine = 10.082410266562684 kW
Compressor power input W kompresor = 15.982194445826448 kW
Net power output  Wnet = 5.899784179263763 kW
Heat exchanger heat removal Q  = 8.8694836816727 kW
refrigeration heat removal Qref = 2.969699502408937 kW
COP refrigeration  = 1.5033573114160128
COP refrigeration  = 1.5033573114160128
Compressor inlet temperature = 26.850000000000023 degree C
Compressor inlet pressure  = 1.01325 kPa
Compressor exit temperature = 154.517442534128 degree C
Compressor exit  pressure   = 3.0 kPa
Turbine inlet temperature = 83.85000000000002 degree C
Turbine inlet pressure   = 3.0 kPA
Turbine exit temperature = 2.9618911756290345 degree C
Turbine exit pressure   = 1.01325 kPa
```




```
public class air_PR extends air
{  // Peng-Robinson Equation of State for air
   // Assumed gas mixture
   // Reference: Robert C. Reid , John M. Prausnitz ; The Properties of Gases & Liquids ,
   // Mc-Graw Hill, Fourth edition, ISBN 0-07-0511799-1
   // Pseudocritical method
   String name="air_PR";
   public double M;
   public String option;
public double RL=8314.51;  //J/kmolK


public double T0=100; //degree K
public double h0=6494.1;  //  /kJ/kmol
public double u0=5680; // kJ/kmol
public double s0=80.463;//kJ/kmolK
public double ro0=0.12283; //kmol/m^3
public double v0=1/ro0;  //m^3/kmol
public double Ts=132.4336;
public double Ps=37.56068e2;//kPa
public double ros=341.1674;
public double Vs=1.0/ros;
public double ws=0.033761;
public double Zs=0.289081;
public double Tc[]={126.19,154.58,150.69,304.13,44.492,5.19,190.56,209.48,33.145,289.73};
public double Pc[]={33.958,50.43,48.63,73.773,26.786,22.7,45.997,55,25.9,12.964,58.42}; //bar
```

```
public double Zc[]={0.289388,0.287879,0.289493,0.274586,0.303196,3.0655,0.286299,0.292363,0.303345,0.288687};
public double Vc[]=new double[Tc.length];
public double w[]={0.0372,0.0222,-0.00219,0.22394,-0.0387,-0.365,0.01142,-0.00089,-0.219,0.00363};
public double Pcij[][]=new double[Tc.length][Tc.length];
public double Tcij[][]=new double[Tc.length][Tc.length];
public double Vcij[][]=new double[Tc.length][Tc.length];
public double Zcij[][]=new double[Tc.length][Tc.length];
public double wij[][]=new double[Tc.length][Tc.length];
public double kij[][]=new double[Tc.length][Tc.length];
public double fwij[][]=new double[Tc.length][Tc.length];
public double aij[][]=new double[Tc.length][Tc.length];
public double bi[]=new double[Tc.length];
public double a,b,da,d2a;
public double Trij[][]=new double[Tc.length][Tc.length];
public double y[]={78.07914146e-2,20.9446967e-2,0.933941885e-2,0.03969753e-2,0.001817887e-2,0.000523967e-
2,0.000178989e-2,9.99938e-09,4.99969e-09,8.99944e-10};
public double Mi[]={28.013,31.999,39.948,44.01,20.18,4.003,16.42,83.798,2.016,131.293};
public double N2[][]={{29.4086307829066,-2.251447032687441,-
0.012473218602163025,4.520888618827954,100.0,350.0},
{27.646169006870817,0.8823555268073686,0.7700742080972421,4.764422867499932,350.0,700.0},
{21.6017064499508,14.8784143146306,3.8128084888605867,-4.165466950629217,700.0,1200.0},
{29.83076594551716,5.421560790726098,-15.043096021474726,-1.089613826823155,1200.0,1700.0},
{35.47674151217584,0.9735825946463841,-42.54762748746501,-0.09746644007894256,1700.0,2200.0},
{34.92820280433006,1.3194039652973688,-38.18419194507587,-0.1599114820430245,2200.0,2700.0},
{36.26252563953146,0.581500103295118,-50.89836208050225,-0.04573113128151122,2700.0,3200.0},
{35.6573409828445,0.7661686026531045,-34.66593634159328,-0.05981705212108953,3200.0,3700.0},
{36.418045420513636,0.4325957723438321,-44.18470620127133,-0.02015217267109989,3700.0,4200.0},
{38.077688052800745,-0.15296039735610514,-80.31180751006634,0.036793817165442805,4200.0,4700.0},
{37.76028438906843,-0.049949299917673017,-73.10115599104877,0.027768544193751826,4700.0,5200.0},
{39.97385521775111,-0.8545553354515152,-77.57593768915008,0.10125346647512108,5200.0,6000.0}};
public double O2[][]={{30.436049777710164,-11.223759902204828,-
0.047090482245673554,26.321488811822327,100.0,350.0},
{21.0013128463894,23.612414099939112,2.0465414904607973,-10.135175753703894,350.0,700.0},
{29.742598604094326,7.97891089460452,-6.123333275436695,-2.2403149499236052,700.0,1200.0},
{36.312768266381276,0.05081402171843047,-19.655239086792907,0.45704197252067635,1200.0,1700.0},
{33.34540380166893,2.3286053031351015,-4.526644218591393,-0.03714805197262758,1700.0,2200.0},
{31.03636259554282,3.8048102299936915,11.95127296669745,-0.30141413192161576,2200.0,2700.0},
{29.75055507663274,4.5049829955864,24.833038220280205,-0.4085902583861984,2700.0,3200.0},
{32.146855060672344,3.466534913579155,-12.259898720486945,-0.2827184121559496,3200.0,3700.0},
{43.527006041753204,-0.6697614318250725,-268.7947582132037,0.14080601236499463,3700.0,4200.0},
{54.768226681468356,-4.218495669713744,-604.9056827788098,0.4565002641241911,4200.0,4700.0},
{63.26081572875727,-6.726400138455676,-894.0814520229812,0.6649044162599882,4700.0,5200.0},
{71.31479957693472,-8.893656534014266,-1220.2917324495174,0.8284435689613064,5200.0,6000.0}};
public double Ar[][]={{20.786,0,0,0,100.0,350.0},
{20.786,0,0,0,350.0,700.0},
{20.786,0,0,0,700.0,1200.0},
{20.786,0,0,0,1200.0,1700.0},
{20.786,0,0,0,1700.0,2200.0},
{20.786,0,0,0,2200.0,2700.0},
{20.786,0,0,0,2700.0,3200.0},
{20.786,0,0,0,3200.0,3700.0},
{20.786,0,0,0,3700.0,4200.0},
{20.786,0,0,0,4200.0,4700.0},
{20.786,0,0,0,4700.0,5200.0},
{20.786,0,0,0,5200.0,6000.0}};
public double Ne[][]={{20.786,0,0,0,100.0,350.0},
{20.786,0,0,0,350.0,700.0},
{20.786,0,0,0,700.0,1200.0},
{20.786,0,0,0,1200.0,1700.0},
{20.786,0,0,0,1700.0,2200.0},
{20.786,0,0,0,2200.0,2700.0},
{20.786,0,0,0,2700.0,3200.0},
{20.786,0,0,0,3200.0,3700.0},
{20.786,0,0,0,3700.0,4200.0},
{20.786,0,0,0,4200.0,4700.0},
{20.786,0,0,0,4700.0,5200.0},
{20.786,0,0,0,5200.0,6000.0}};
public double He[][]={{20.786,0,0,0,100.0,350.0},
{20.786,0,0,0,350.0,700.0},
{20.786,0,0,0,700.0,1200.0},
{20.786,0,0,0,1200.0,1700.0},
{20.786,0,0,0,1700.0,2200.0},
{20.786,0,0,0,2200.0,2700.0},
{20.786,0,0,0,2700.0,3200.0},
{20.786,0,0,0,3200.0,3700.0},
{20.786,0,0,0,3700.0,4200.0},
```

```
{20.786,0,0,0,4200.0,4700.0},
{20.786,0,0,0,4700.0,5200.0},
{20.786,0,0,0,5200.0,6000.0}};
public double CO2[][]={{17.300873699905434,80.30037265388783,0.4395943896443757,-
52.053796318493056,100.0,500.0},
{32.82713012987489,33.27479124145044,-5.039475525678202,-11.292594189876088,500.0,1000.0},
{47.392905334573165,12.630493080860983,-27.138160476940687,-3.002235351466388,1000.0,1500.0},
{58.83520997915058,1.9824455333482274,-67.52352061782346,-0.19014534560113247,1500.0,2000.0},
{60.89376216048979,0.9095624409942658,-86.51551985502067,-0.049975648956251735,2000.0,2500.0},
{70.41645837659225,-3.8184039507822227,-201.41621112385792,0.6118115953072466,2500.0,3000.0},
{56.10341642565405,2.640738216832377,16.60136887360514,-0.2201228164776734,3000.0,3500.0},
{54.28390190479408,3.1146833735671464,80.13275747980263,-0.2493339658694632,3500.0,4000.0},
{61.79621452397064,0.5716447960137 22,-116.8521744294146,-0.006160617823509 5324,4000.0,4500.0},
{139.30048071121965,-21.379600631537595,-2977.620803457743,1.7421559071731059,4500.0,5000.0},
{-39.70675713613505,25.809754875975532,4655.537698036879,-1.7567411037897742,5000.0,5500.0},
{54.71006515941323,1.9884447696572416,271.9708219962126,-0.06775324710984956,5500.0,6000.0}};
public double CH4[][]={{38.91237365038245,-55.38748721420229,-
0.16243781154650505,150.90531471518315,100.0,350.0},
{0.2017001920942126,103.14997063247195,6.824405432334339,-32.664701932534165,350.0,700.0},
{11.546938187448935,82.19569329154288,-3.4532853747849552,-21.604988532068983,700.0,1200.0},
{50.57262867843102,38.28672483389998,-96.53220573246487,-7.625508294088279,1200.0,1700.0},
{79.68125733676185,14.814185315851725,-230.3204126171157,-2.2882055069086182,1700.0,2200.0},
{94.20352251055668,5.840286388869979,-343.8735028312377,-0.724872349501491,2200.0,2700.0},
{104.35726881638747,0.9375681236754352,-473.8206804592937,-0.05735825649099603,2700.0,3200.0},
{103.63017172019306,1.3552795586146313,-469.7217838104155,-0.12079244525350849,3200.0,3700.0},
{105.45811322278479,0.6836699074162705,-509.361916797209,-0.05164204343214277,3700.0,4200.0},
{111.01788820096512,-1.011448107760356,-685.2166188469638,0.09329341035170628,4200.0,4700.0},
{111.68722446905167,-1.05072384596387,-747.2766372898087,0.0840648676898491,4700.0,5200.0},
{104.2567574929384,0.8702029903725688,-413.470500528235,-0.05620390320104198,5200.0,6000.0}};
public double Kr[][]={{20.786,0,0,0,100.0,350.0},
{20.786,0,0,0,350.0,700.0},
{20.786,0,0,0,700.0,1200.0},
{20.786,0,0,0,1200.0,1700.0},
{20.786,0,0,0,1700.0,2200.0},
{20.786,0,0,0,2200.0,2700.0},
{20.786,0,0,0,2700.0,3200.0},
{20.786,0,0,0,3200.0,3700.0},
{20.786,0,0,0,3700.0,4200.0},
{20.786,0,0,0,4200.0,4700.0},
{20.786,0,0,0,4700.0,5200.0},
{20.786,0,0,0,5200.0,6000.0}};
public double H2[][]={{14.746635916306719,75.98814041267556,0.6855650913365514,-
104.78427257655125,100.0,350.0},
{31.87738315272732,-7.019570781101979,-1.2522956170328061,5.578010465060371,350.0,700.0},
{27.00990995073016,1.4614189874718833,3.602636119336523,1.3744817930200472,700.0,1200.0},
{20.04974364199245,9.89520983707123,17.774588216836854,-1.5043855020756678,1200.0,1700.0},
{24.04843252848654,6.810225792306218,-2.300503485911463,-0.8329576524132585,1700.0,2200.0},
{29.21564799070763,3.650895060350098,-43.35797560782945,-0.2892229316928226,2200.0,2700.0},
{32.31424383733543,2.0694485310000306,-78.1959957855126,-0.06296407156822922,2700.0,3200.0},
{35.68314889318526,0.6762347249769796,-136.55016578297088,0.09907036168920383,3200.0,3700.0},
{35.56302783331784,0.8914164593950742,-156.48978650154623,0.060326987381420026,3700.0,4200.0},
{21.727270639022365,5.373281895434687,234.06266006928675,-0.3479505267822607,4200.0,4700.0},
{13.019905614959066,7.872396496868688,547.1226571751322,-0.549658082637511,4700.0,5200.0},
{6.7479064642451005,9.543911192630823,806.6543827682647,-0.6746394177226044,5200.0,6000.0}};
public double Xe[][]={{20.786,0,0,0,100.0,350.0},
{20.786,0,0,0,350.0,700.0},
{20.786,0,0,0,700.0,1200.0},
{20.786,0,0,0,1200.0,1700.0},
{20.786,0,0,0,1700.0,2200.0},
{20.786,0,0,0,2200.0,2700.0},
{20.786,0,0,0,2700.0,3200.0},
{20.786,0,0,0,3200.0,3700.0},
{20.786,0,0,0,3700.0,4200.0},
{20.786,0,0,0,4200.0,4700.0},
{20.786,0,0,0,4700.0,5200.0},
{20.786,0,0,0,5200.0,6000.0}};
public double bb[][][]={N2,O2,Ar,CO2,Ne,He,CH4,Kr,H2,Xe};
Gas g=new Gas("air");

air_vis_k1 avk=new air_vis_k1();
double ds[]={273.1551877698175,
270.91415329178733,
139.09162499504905,
20.00253131105914,
57.00962377288705,
-67.7836965657238,
```

```
40.10812247077545,
-11.790148544605461,
1.5416756743038067,
-5.526242418853896E-4,
-0.01377998370015372};
double dh[]={281.75954013189823,
0.7734988091769495,
0.0013056859113692863,
-3.4307455915799553E-6,
4.351694425964074E-9,
-3.1574762255452993E-12,
1.3850919856396723E-15,
-3.7159262496573843E-19,
5.935681999785052E-23,
-5.171030846130408E-27,
1.8887123996405406E-31};
//consructor
public air_PR()
{input("english");}
public air_PR(String optioni)
{input(optioni);}
public void input(String optioni)
{  option=optioni;
                g.mole=false;
    M=0;
   Ts=0;Ps=0;ros=0;Zs=0;
   double yt=0;
   for(int i=0;i<Tc.length;i++)
   {
     Vc[i]=Zc[i]*RL*Tc[i]/(Pc[i]*1.0e5);
     Ts+=y[i]*Tc[i];
     Ps+=y[i]*Pc[i];
     Zs+=y[i]*Zc[i];
     Vs+=y[i]*Vc[i];
     bi[i]=0.0778*RL*Tc[i]/(Pc[i]*1e5);
     M+=y[i]*Mi[i];
   }
   System.out.println("M="+M);
   ros=1.0/Vs;
 for(int i=0;i<Tc.length;i++)
 {
   for(int j=0;j<Tc.length;j++)
   {if(i==j)
     { Tcij[i][j]=Tc[i];
       Pcij[i][j]=Pc[i];
       Zcij[i][j]=Tc[i];
       Vcij[i][j]=Vc[i];
       wij[i][j]=w[i];
       kij[i][j]=0.0;
       fwij[i][j]=0.37464+1.54226*wij[i][j]-0.269992*wij[i][j]*wij[i][j];
     }
    else
     { double xx=Math.cbrt(Vc[i])+Math.cbrt(Vc[j]);
       Vcij[i][j]=xx*xx*xx/8.0;
       Zcij[i][j]=(Zc[i]+Zc[j])/2.0;
       kij[i][j]=1-Math.sqrt(Vc[i]*Vc[j])/Vcij[i][j];
                 Tcij[i][j]=Math.sqrt(Tc[i]*Tc[j])*(1-kij[i][j]);
       Pcij[i][j]=Zcij[i][j]*RL*Tcij[i][j]/Vcij[i][j]*1e-5;
       wij[i][j]=(w[i]+w[j])/2.0;
       fwij[i][j]=0.37464+1.54226*wij[i][j]-0.269992*wij[i][j]*wij[i][j];
     }
  }
 }
 b=0;
 for(int i=0;i<Tc.length;i++)
 {b+=y[i]*bi[i];}
}
//P(T,v)
double P_Tv(double T,double v)
{ // P kPa v m^3/kg T degree K
calc_a(T);
double v1=v*M;
double P1=RL*T/(v1-b)-a/(v1*v1+2.0*b*v1-b*b);
return P1*1e-3;
}
//P(T,ro)
```

```
double P_Tro(double T,double ro)
{ // P kPa v m^3/kg T degree K
calc_a(T);
double v=1.0/ro;
double v1=v*M;
double P1=RL*T/(v1-b)-a/(v1*v1+2.0*b*v1-b*b);
return P1*1e-3;
}

//Cp kJ/kmolK for individual gases
public double integCv(double aa[][],double T)
{ double iCv=0;
  int m=1;
  double TH=0;
  double TL=0;
  for(int i=0;i<12;i++)
  {if(T>=aa[i][4] && T<=aa[i][5]) {m=i;break;}}
  int i=0;
  for(i=0;i<m;i++)
  {   TH=aa[i][5];TL=aa[i][4];
                iCv+=(aa[i][0]-RL*1e-3)*(TH-TL)+aa[i][1]/2.0*1e-3*(TH*TH-TL*TL)-aa[i][2]*1e5*(1.0/TH-
1.0/TL)+aa[i][3]*1e-6/3.0*(TH*TH*TH-TL*TL*TL);
  }
  TL=aa[m][4];
  iCv+=(aa[m][0]-RL*1e-3)*(T-TL)+aa[m][1]/2.0*1e-3*(T*T-TL*TL)-aa[m][2]*1e5*(1.0/T-1.0/TL)+aa[m][3]*1e-
6/3.0*(T*T*T-TL*TL*TL);
  return iCv;
}
//for air integral(Cv(T))dT
public double integCv(double T)
{ double iCv=0;
  calc_da(T);
  double x=0;
  for(int i=0;i<bb.length;i++)
    {   x=integCv(bb[i],T);
                  iCv+=y[i]*x;
                  }
  return iCv/M;
}
//integral(Cv(T)/T)dT for specific gases
public double integCv_T(double aa[][],double T)
{ double iCv=0;
  int m=0;
  double TH=0;
  double TL=0;
  for(int i=0;i<12;i++)
  {if(T>=aa[i][4] && T<=aa[i][5]) {m=i;break;}}
  for(int i=0;i<m;i++)
  {   TH=aa[i][5];TL=aa[i][4];
                iCv+=(aa[i][0]-RL*1e-3)*Math.log(TH/TL)+aa[i][1]*1e-3*(TH-TL)-aa[i][2]*1e5/2.0*(1.0/(TH*TH)-
1.0/(TL*TL))+aa[i][3]*1e-6/2.0*(TH*TH-TL*TL);
  }
  TL=aa[m][4];
  iCv+=(aa[m][0]-RL*1e-3)*Math.log(T/TL)+aa[m][1]*1e-3*(T-TL)-aa[m][2]*1e5/2.0*(1.0/(T*T)-
1.0/(TL*TL))+aa[m][3]*1e-6/2.0*(T*T-TL*TL);
  return iCv;
}
//for air   integral(Cv(T)/T)dT
public double integCv_T(double T)
{ double iCv=0;
  for(int i=0;i<bb.length;i++)
    {iCv+=y[i]*integCv_T(bb[i],T);}
  return iCv/M;
}
//Cp for individual gases kJ/kgK Specific heat at constant pressure
//  isochoric specific heat
public double Cp(double aa[][],double T)
{ double Cp=0;
  for(int i=0;i<12;i++)
  {if(T>=aa[i][4] && T<=aa[i][5])
   {Cp=aa[i][0]+aa[i][1]*1e-3*T+aa[i][2]*1e5/(T*T)+aa[i][3]*1e-6*T*T;break;}
  }
  return Cp;
}
//Cp(T) kJ/kgK Specific heat at constant pressure
//  isochoric specific heat
```

```
public double Cp(double T)
{ double Cp=0;
  double xx=0;
  for(int i=0;i<bb.length;i++)
    {   xx=Cp(bb[i],T);
        //System.out.println("i="+i+"Cp="+xx);
                Cp+=y[i]*xx;}
                //System.out.println("air="+"Cp="+Cp);
  return Cp/M;
}
//Cv(T) kJ/kgK Specific heat at constant volume
//  isovolumetric specific heat
public double Cv(double T)
{ return Cp(T)-RL*1e-3/M;}
// a(T)  Peng-Robinson coefficient
void calc_a(double T)
{double xx=0;
for(int i=0;i<Tc.length;i++)
{for(int j=0;j<Tc.length;j++)
{Trij[i][j]=T/Tcij[i][j];
 xx=1.0+fwij[i][j]*(1.0-Math.sqrt(Trij[i][j]));
 aij[i][j]=0.45724*RL*RL*Tcij[i][j]*Tcij[i][j]/(Pcij[i][j]*1.0e5)*xx*xx;
}}
a=0;
for(int i=0;i<Tc.length;i++)
{ for(int j=0;j<Tc.length;j++) { a+=y[i]*y[j]*aij[i][j];}}
}
// da/dT(T)
void calc_da(double T)
{double xx1=0;
 double xx2=0;
 calc_a(T);
 double total=0;
for(int i=0;i<Tc.length;i++)
{for(int j=0;j<Tc.length;j++)
{Trij[i][j]=T/Tcij[i][j];
 xx1=fwij[i][j]*Math.sqrt(Trij[i][j]);
 xx2=1.0+fwij[i][j]*(1.0-Math.sqrt(Trij[i][j]));
 total+=y[i]*y[j]*aij[i][j]*xx1/xx2;
}}
da=-1.0/T*total;
}
//d2a/dT^2(T)
void calc_d2a(double T)
{double xx1=0;
 double xx2=0;
 calc_a(T);
 double total=0;
for(int i=0;i<Tc.length;i++)
{for(int j=0;j<Tc.length;j++)
{Trij[i][j]=T/Tcij[i][j];
 xx1=(1.0-fwij[i][j])*Math.sqrt(Trij[i][j])*Tcij[i][j]/(Pcij[i][j]*1.0e5);
 total+=y[i]*y[j]*aij[i][j]*xx1;
}}
d2a=0.457236*RL*RL/(2.0*T)*total;
}
//dP/dT(T,v)
public double dP_dT(double T,double v)
{calc_da(T);
double v1=v*M;
double dP=RL/(v1-b)-1.0/(v1*v1+2.0*b*v1-b*b)*da;
return dP;
}
//v(T,P) m^3/kg
double v(double T,double P)
{ double v1=(RL/M)*T/(P*1e3);
  if_x f1=(x)->P_Tv(T,x)-P;
  double v=secant(f1,v1);
  return v;
}
//v(T,P) m^3/kg
double v1(double T,double P)
{   //P kPa T degree K v m^3/kg
double P1=P*1e3; //Pa
//calculate coefficient a for peng_Robinson EOS
calc_a(T);
```

```
   double AY=a*P1/(RL*RL*T*T);
   double BY=b*P1/(RL*T);
   double u=2.0;double w=-1.0;
   double A=-(1+BY-2.0*BY);
   double B=AY+w*BY*BY-u*BY-u*BY*BY;
   double C=-AY*BY-w*BY*BY-w*BY*BY*BY;
   double cof[]=new double[4];
   cof[0]=1;
   cof[1]=A;
   cof[2]=B;
   cof[3]=C;
   double zz1[][]=cubic_roots(cof);
   c zc[]=cubic_rootsC(cof);
   double zz=0;
   for(int i=0;i<3;i++)
    {
    if((zc[i].nr>=0.0 && zc[i].nr<=2.0) &&
       (zc[i].ni>=0.0 && zc[i].ni<1e-4))
       {
       zz=zc[i].nr;
       break;
       }
    }
   double v1=zz*RL*T/(P*1e3)/M;
   //v1 : m^3/kg
   return v1;
}
//ro(T,P)  kg/m^3
double ro(double T,double P)
{ return 1.0/v(T,P);}
// dro/dP(T,P)
double d_ro_dP(double T,double P)
{ if_x f1=x->ro(T,x);
 return f1.dfunc(P);
}
// dro/dT(T,P)
double d_ro_dT(double T,double P)
{ if_x f1=x->ro(x,P);
 return f1.dfunc(T);}

// Thermal expansion coefficient beta(T,P) 1/K
double beta(double T,double P)
{double ro=ro(T,P);
 double beta=-1/ro*d_ro_dT(T,P);
 return beta;
}
//u(T,v) internal energy kJ/kg
public double u_Tv(double T,double v)
{int m=N2.length;
 int n=0;
 double P2=100; //kPa
 double v1=v*M;
 double T2=273.15;
 double v2=v(T2,P2)*M;
 double x0=(1.0-Math.sqrt(2))*b;
 double x1=(1.0+Math.sqrt(2))*b;
 double u=integCv(T)-integCv(T2);
 double x2,x3;
 x2=(a-T*da)*1.0/Math.sqrt(8.0*b*b)*Math.log((v1+x0)/(v1+x1));
 x2-=(a-T*da)*1.0/Math.sqrt(8.0*b*b)*Math.log((v0+x0)/(v0+x1));
 x2*=1e-3/M;
 u+=x2;
 return u;
}
//u(T,ro) kJ/kg
public double u_Tro(double T,double ro)
{int m=N2.length;
 double v=1.0/ro;
 return u_Tv(T,v);
}
//u(T,P) internal energy kJ/kg
public double u(double T,double P)
{ double v=v(T,P);
  return u_Tv(T,v);
}
//s(T,ro) entropy kJ/kgK
```

```
public double s_Tro(double T,double ro)
{ double v=1.0/ro;
  return s_Tv(T,v);
}
//s(T,v) entropy kJ/kgK
public double s_Tv(double T,double v)
{
 int n=0;
 double v1=v*M;
 double x0=Math.sqrt(2);
 double P2=100;  //kPa
 double T2=273.15;
 double v2=v(T2,P2)*M;
 double s=integCv_T(T)-integCv_T(T2);
 double x1,x2;
  calc_da(T);
 x1=(RL*Math.log(v1/v0)-da/Math.sqrt(8.0*b*b)*(Math.log((v1+(1.0-x0)*b)/(v1+(1.0+x0)*b))-Math.log((v0+(1.0-
x0)*b)/(v0+(1.0+x0)*b))))*1e-3/M;
 s+=x1;
 x2=(RL*Math.log(v2/v0)-da/Math.sqrt(8.0*b*b)*(Math.log((v2+(1.0-x0)*b)/(v2+(1.0+x0)*b))-Math.log((v0+(1.0-
x0)*b)/(v0+(1.0+x0)*b))))*1e-3/M;
 s-=x2;
 return s;
}
//s(T,P) entropy kJ/kgK
public double s(double T,double P)
{ double v=v(T,P);
  return s_Tv(T,v);
}
//h(T,v) kJ/kg
public double h_Tv(double T,double v)
{ double P=P_Tv(T,v);
  return h(T,P);
}
//h(T,ro) enthalpy kJ/kg
public double h_Tro(double T,double ro)
{ double P=P_Tro(T,ro);
  return h(T,P);
}
//h(T,P) enthalpy kJ/kg
public double h1(double T,double P)
{ double v=v(T,P);
  double T2=100.0;
  double P2=100.0;
  double v2=v(T2,P2);
  double u=u(T,P);
  double h=u(T,P)+P*v;
  return h;
}
//h(T,P) enthalpy kJ/kg
public double h(double T,double P)
{  double T2=273.15;
   double P2=100.0;
  return h1(T,P)-h1(T2,P2);
}
//T(P,ro) temperature K
public double T_Pro(double P,double ro)
{if_x f1=x->P_Tro(x,ro)-P;
 double T1=ro*RL*1e3/(P*1e2); //case of ideal gas
 double T=secant(f1,T1);
 return T;
}
//double (carrying complex) cubic roots
public static double[][] cubic_roots(double d[])
{
// roots of a degree 3 polynomial
// P(x)=d[0]*x^3+d[1]*x^2+d[2]*x+d[3]=0;
// assuming d is all real;
// if value of d will be entered bigger than d[3]
// remaining terms will be ignored
double x[][]=new double[2][3];
double a,b,c;
a=d[1]/d[0];
b=d[2]/d[0];
c=d[3]/d[0];
double Q,R,theta;
```

```
Q=(a*a-3.0*b)/9.0;
double x1,x2,x3,x4;
x1=2.0*a*a*a;
x2=-9*a*b;
x3=27*c;
x4=x1+x2+x3;
R=x4/54.0;
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][2]=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][0]=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][2]=(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][0]=-0.5*(A+B)-a/3;
x[1][0]=-Math.sqrt(3.0)/2.0*(A-B);
}
return x;
}
// complex cubic roots
public static c[] cubic_rootsC(double c[])
{
// roots of a degree 3 polynomial
//return 3 c 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;
}
//T(v,P)
public double T_vP(double v,double P)
{ if_x pp=(x)->P_Tv(x,v)-P;
double v1=v*M;
  double T1=P*1e3*v1/RL;
  double T2=secant(pp,T1);
  return T2;
}
//T(ro,P)
public double T_roP(double ro,double P)
{ double v=1.0/ro;
  return T_vP(v,P);
}
// dT/dv(v,P)
public double T_vP_dv(double v,double P)
{  if_x f=(x)->T_vP(x,P);
   double xx=f.dfunc(v);
   //double xx=(T_vP(v+0.001,P)-T_vP(v-0.001,P))/0.002;
   return xx;
}
//viscosity
public double vis(double T,double P)
{return avk.vis1(T,P)*1e-6;}
public double k(double T,double P)
{return avk.k1(T,P)*1e-3;}
//Prandtl number
public double Pr_Tro(double T,double ro)
{  double P=P_Tro(T,ro);
   double mu=vis(T,P);
   double k=k(T,ro);
   double Cp=Cp(T);
   return Cp*1e3*mu/k;
```

```
}
// dP/dv(T,v)
public double dP_dv_Tv(double T,double v)
{double v1=v*M;
double x=v1*v1+2.0*b*v1-b*b;
 double y=-RL*T/((v1-b)*(v1-b))-a*2.0*(v+b)/(x*x);
 return y;
}
// dP/dv(T,ro)
public double dP_dv_Tro(double T,double ro)
{double v=1.0/ro;
              double v1=v*M;
double x=v1*v1+2.0*b*v1-b*b;
 double y=-RL*T/((v1-b)*(v1-b))-a*2.0*(v+b)/(x*x);
 return y;
}
// speed of sound m/s
public double a_Tro(double T,double ro)
{double y1=dP_dv_Tro(T,ro);
 double v=1.0/ro;
 double v1=v*M;
 double y=-v1*v1*Cp(T)/(Cv(T)*M)*y1;
 return Math.sqrt(y);
}
// speed of sound m/s
public double a_TP(double T,double P)
{ double v=v(T,P);
 double ro=1.0/v;
 double y1=dP_dv_Tro(T,ro);
 double v1=v*M;
 double y=-v1*v1*Cp(T)/(Cv(T)*M)*y1;
 return Math.sqrt(y);
}
//Isothermal compresibility
public double kappaS_Tro(double T,double ro)
{  double y1=dP_dv_Tro(T,ro);
   double v=1.0/ro;
   double v1=v*M;
   double y=-1.0/y1/v1;
  return y;
}
//Isothermal compresibility
public double kappaS_TP(double T,double P)
{  double v=v(T,P);
   double ro=1.0/v;
   double y1=dP_dv_Tro(T,ro);
   double v1=v*M;
   double y=-1.0/y1/v1;
  return y;
}
// temperature and density as a function of pressure and enthalpy
public double[] Tro_Ph(double P,double h)
{
//Lambda variable
                ifi_xj f2=(double x[])->
                {  double ff[]=new double[2];
        ff[0]=P_Tro(x[0],x[1])-P;
        ff[1]=h_Tro(x[0],x[1])-h;
        return ff;
      };
      poly ph=new poly(dh);
      double T1=ph.func(h);
      double ro1=ro(T1,P);
      double [] x0={T1,ro1};
      //Find double roots by secant(newton?) method
      double c[]=secant2(f2,x0);//NA30Ci.continuationRK4(f2,x0,4);//
      return c;
}
// temperature and density as a function of pressure and entropy
public double[] Tro_Ps(double P,double s)
{
//Lambda variable
                ifi_xj f2=(double x[])->
                {  double ff[]=new double[2];
                  ff[0]=P_Tro(x[0],x[1])-P;
                  ff[1]=s_Tro(x[0],x[1])-s;
```

```
        return ff;
      };
      poly ps=new poly(ds);
      double T1=ps.func(s);
      double ro1=ro(T1,P);
      double [] x0={T1,ro1};
      //Find double roots by secant(newton?) method
      double c[]=secant2(f2,x0);//NA30Ci.continuationRK4(f2,x0,4);//
      return c;
}
//Pressure as a function of temperature and entropy
public double P_Ts(double T,double s)
{if_x f1=x->s(T,x)-s;
 double P1=10.0;
 double P2=10000.0;
 double P=bisection(f1,P1,P2);
 return P;
}
// PROPERY CALCULATIONS =======
public double[] property(String s,double v1,double v2)
{ if(s.equals("pt")) return property_TP(v2,v1);
  else if(s.equals("tp"))  {return property_TP(v1,v2);}
  else if(s.equals("vt"))  {double ro=1.0/v1;return property_Tro(v2,ro);}
  else if(s.equals("tv"))  {double ro=1.0/v2;return property_Tro(v1,ro);}
  else if(s.equals("rot")) {return property_Tro(v2,v1);}
  else if(s.equals("tro")) {return property_Tro(v1,v2);}
  else if(s.equals("pv"))  return property_Pro(v1,1.0/v2);
  else if(s.equals("pro")) return property_Pro(v1,v2);
  else if(s.equals("vp"))  return property_Pro(v2,1.0/v1);
  else if(s.equals("rop")) return property_Pro(v2,v1);
  else if(s.equals("ph"))  return property_Ph(v1,v2);
  else if(s.equals("hp"))  return property_Ph(v2,v1);
  else if(s.equals("ps"))  return property_Ps(v1,v2);
  else if(s.equals("sp"))  return property_Ps(v2,v1);
  else if(s.equals("ts"))  return property_Ts(v1,v2);
  else if(s.equals("st"))  return property_Ts(v2,v1);
  else return property_TP(v1,v2);
}
public double[] propertyC(String s,double v1,double v2)
{ double t=273.15;
  if(s.equals("pt")) return property_TP(v2,(v1+t));
  else if(s.equals("tp"))  {return property_TP((v1+t),v2);}
  else if(s.equals("vt"))  {double ro=1.0/v1;return property_Tro((v2+t),ro);}
  else if(s.equals("tv"))  {double ro=1.0/v2;return property_Tro((v1+t),ro);}
  else if(s.equals("rot")) {return property_Tro((v2+t),v1);}
  else if(s.equals("tro")) {return property_Tro((v1+t),v2);}
  else if(s.equals("pv"))  return property_Pro(v1,1.0/v2);
  else if(s.equals("pro")) return property_Pro(v1,v2);
  else if(s.equals("vp"))  return property_Pro(v2,1.0/v1);
  else if(s.equals("rop")) return property_Pro(v2,v1);
  else if(s.equals("ph"))  return property_Ph(v1,v2);
  else if(s.equals("hp"))  return property_Ph(v2,v1);
  else if(s.equals("ps"))  return property_Ps(v1,v2);
  else if(s.equals("sp"))  return property_Ps(v2,v1);
   else if(s.equals("ts"))  return property_Ts((v1+t),v2);
  else if(s.equals("st"))  return property_Ts((v2+t),v1);
  else return property_TP((v1+t),v2);
}
public double[] property_Tro(double T,double ro)
{ double a[]=new double[17];
 double P=a[0]=P_Tro(T,ro);
 a[1]=T;
 a[2]=1.0/ro;
 a[3]=h_Tro(T,ro);
 a[4]=u_Tro(T,ro);
 a[5]=s_Tro(T,ro);
 a[6]=2.0;
 a[7]=Cp(T);
 a[8]=Cv(ro);
 a[9]=beta(T,P);
 a[10]=kappaS_Tro(T,ro);
 a[11]=vis(T,P);
 a[12]=k(T,P);
 a[13]=0.0;
 a[14]=Pr_Tro(T,ro);;
 a[15]=ro;
```

```
 a[16]=a_Tro(T,ro);;
 return a;
}
public double[] property_TP(double T,double P)
{double a[]=new double[17];
 double ro=ro(T,P);
 a[0]=P;
 a[1]=T;
 a[2]=1.0/ro;
 a[3]=h_Tro(T,ro);
 a[4]=u_Tro(T,ro);
 a[5]=s(T,P);
 a[6]=2.0;
 a[7]=Cp(T);
 a[8]=Cv(T);
 a[9]=beta(T,P);
 a[10]=kappaS_Tro(T,ro);
 a[11]=vis(T,ro);
 a[12]=k(T,P);
 a[13]=0.0;
 a[14]=Pr_Tro(T,ro);
 a[15]=ro;
 a[16]=a_Tro(T,ro);
 return a;
}
public double[] property_Pro(double P,double ro)
{double a[]=new double[17];
 double T=T_Pro(P,ro);
 a[0]=P;
 a[1]=T;
 a[2]=1.0/ro;
 a[3]=h_Tro(T,ro);
 a[4]=u_Tro(T,ro);
 a[5]=s(T,P);
 a[6]=2.0;
 a[7]=Cp(T);
 a[8]=Cv(T);
 a[9]=beta(T,P);
 a[10]=kappaS_Tro(T,ro);
 a[11]=vis(T,ro);
 a[12]=k(T,P);
 a[13]=0.0;
 a[14]=Pr_Tro(T,ro);
 a[15]=ro;
 a[16]=a_Tro(T,ro);
 return a;
}
public double[] property_Ts(double T,double s)
{ double P=P_Ts(T,s);
  return property_TP(T,P);
}
public double[] property_Ph(double P,double h)
{   double a[]=new double[17];
                double c[]=Tro_Ph(P,h);
     double T=c[0];
     double ro=c[1];
 a[0]=P;
 a[1]=T;
 a[2]=1.0/ro;
 a[3]=h;
 a[4]=u_Tro(T,ro);
 a[5]=s(T,P);
 a[6]=2.0;
 a[7]=Cp(T);
 a[8]=Cv(T);
 a[9]=beta(T,P);
 a[10]=kappaS_Tro(T,ro);
 a[11]=vis(T,ro);
 a[12]=k(T,P);
 a[13]=0.0;
 a[14]=Pr_Tro(T,ro);
 a[15]=ro;
 a[16]=a_Tro(T,ro);
 return a;
}
public double[] property_Ps(double P,double s)
```

```
{   double a[]=new double[17];
                double c[]=Tro_Ps(P,s);
     double T=c[0];
     double ro=c[1];
 a[0]=P;
 a[1]=T;
 a[2]=1.0/ro;
 a[3]=h_Tro(T,ro);
 a[4]=u_Tro(T,ro);
 a[5]=s;
 a[6]=2.0;
 a[7]=Cp(T);
 a[8]=Cv(T);
 a[9]=beta(T,P);
 a[10]=kappaS_Tro(T,ro);
 a[11]=vis(T,ro);
 a[12]=k(T,P);
 a[13]=0.0;
 a[14]=Pr_Tro(T,ro);
 a[15]=ro;
 a[16]=a_Tro(T,ro);
 return a;
}


//================
//Bisection root finding method
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 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.func(x);
   dy=f.dfunc(x);
   i++;
  }
  if(i>=miter)
    System.out.println("i="+i+"result may not be correct");
  return x;
}
public static double[] secant2(ifi_xj f,double x[])
{
// Roots of nonlinear system of equations secant method
//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=20;
double tolerance=1.0e-5;
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) System.out.println("Warning : Maximum number of iterations are exceeded \nResults may not be valid");
return x;
}
public String[][] toString(String s, double v1, double v2)
{  String s1[][]=new String[17][4];
   if(option.equals("Turkish"))
   {s1[0][1]="P, basınç        ";
   s1[1][1]="T, sıcaklık     ";
   s1[2][1]="v, özgül hacim ";
   s1[3][1]="h, enthalpi        ";
   s1[4][1]="u, iç enerji ";
   s1[5][1]="s, entropi         ";
   s1[6][1]="x, doygunluk derecesi          ";
   s1[7][1]="Cp, sabit basınçta özgül ısı ";
   s1[8][1]="Cv, sabit hacimde özgül ısı";
   s1[9][1]=""+'\u03B2'+" sabit basınçta ısıl genleşme katsayısı";
   s1[10][1]=""+'\u03BA'+" izotermal sıkıştırılabilirlik" ;
   s1[11][1]=""+'\u03B7'+" Dinamik vizkozite" ;
   s1[12][1]="k ısıl iletim sabiti" ;
   s1[13][1]=""+'\u03C3'+" yüzey gerilimi" ;
   s1[14][1]="Prandtl sayısı" ;
   s1[15][1]=""+'\u03C1'+" yoğunluk" ;
   s1[16][1]=" ses hızı" ;
   s1[0][3]=" kPA          ";
   s1[1][3]=" deg K         ";
   s1[2][3]=" m"+'\u00B3'+"/kg        ";
   s1[3][3]=" KJ/kg         ";
   s1[4][3]=" KJ/kg         ";
   s1[5][3]=" KJ/kgK         ";
   s1[6][3]=" kg buhar/kg karışım ";
   s1[7][3]=" KJ/kgK";
   s1[8][3]=" KJ/kgK";
   s1[9][3]=" 1/K";
   s1[10][3]=" kPa";
   s1[11][3]=" Pa.s";
   s1[12][3]=" W/mK";
   s1[13][3]=" N/m";
   s1[14][3]=" ";
   s1[15][3]=" kg/m^3";
   s1[16][3]=" m/s";
   }
   else{
   s1[0][1]="P, pressure        ";
   s1[1][1]="T, temperature     ";
   s1[2][1]="v, specific volume ";
   s1[3][1]="h, enthalpy        ";
   s1[4][1]="u, internal energy ";
   s1[5][1]="s, entropy         ";
   s1[6][1]="x, quality         ";
   s1[7][1]="Cp, specific heat at constant pressure ";
   s1[8][1]="Cv, specific heat at constant volume ";
   s1[9][1]=""+'\u03B2'+" isobaric thermal expansion coefficient ";
   s1[10][1]=""+'\u03BA'+" isothermal compressibility" ;
   s1[11][1]=""+'\u03B7'+" Dynamic viscosity" ;
   s1[12][1]="k Thermal conductivity" ;
   s1[13][1]=""+'\u03C3'+" Surface tension" ;
   s1[14][1]="Prandtl number" ;
   s1[15][1]=""+'\u03C1'+" density" ;
   s1[16][1]=" speed of sound" ;
   s1[0][3]=" kPA          ";
   s1[1][3]=" deg K         ";
   s1[2][3]=" m"+'\u00B3'+"/kg        ";
   s1[3][3]=" KJ/kg         ";
   s1[4][3]=" KJ/kg         ";
   s1[5][3]=" KJ/kgK         ";
   s1[6][3]=" kg vap/kg mix ";
   s1[7][3]=" KJ/kgK";
   s1[8][3]=" KJ/kgK";
   s1[9][3]=" 1/K";
   s1[10][3]=" kPa";
   s1[11][3]=" Pa.s";
```

```
   s1[12][3]=" W/mK";
   s1[13][3]=" N/m";
   s1[14][3]=" ";
   s1[15][3]=" kg/m^3";
   s1[16][3]=" m/s";
   }
double pp[]=property(s,v1,v2);
for(int i=0;i<17;i++)
   {s1[i][2]=""+pp[i];}
    //s1[6][2]=phase(pp[6]);
for(int i=0;i<17;i++)
{s1[i][0]=""+i;}
return s1;
}

public String[][] toStringC(String s, double v1, double v2)
{  String s1[][]=new String[17][4];
     if(option.equals("Turkish"))
   {s1[0][1]="P, basınç        ";
   s1[1][1]="T, sıcaklık     ";
   s1[2][1]="v, özgül hacim ";
   s1[3][1]="h, enthalpi        ";
   s1[4][1]="u, iç enerji ";
   s1[5][1]="s, entropi         ";
   s1[6][1]="x, doygunluk derecesi         ";
   s1[7][1]="Cp, sabit basınçta özgül ısı ";
   s1[8][1]="Cv, sabit hacimde özgül ısı";
   s1[9][1]=""+'\u03B1'+" sabit basınçta ısıl genleşme katsayısı";
   s1[10][1]=""+'\u03BA'+" izotermal sıkıştırılabilirlik" ;
   s1[11][1]=""+'\u03B7'+" Dinamik vizkozite" ;
   s1[12][1]="k ısıl iletim sabiti" ;
   s1[13][1]=""+'\u03C3'+" yüzey gerilimi" ;
   s1[14][1]="Prandtl sayısı" ;
   s1[15][1]=""+'\u03C1'+" yoğunluk" ;
   s1[16][1]=" ses hızı" ;
   s1[0][3]=" kPA          ";
   s1[1][3]=" deg K         ";
   s1[2][3]=" m"+'\u00B3'+"/kg        ";
   s1[3][3]=" KJ/kg         ";
   s1[4][3]=" KJ/kg         ";
   s1[5][3]=" KJ/kgK         ";
   s1[6][3]=" kg buhar/kg karışım ";
   s1[7][3]=" KJ/kgK";
   s1[8][3]=" KJ/kgK";
   s1[9][3]=" 1/K";
   s1[10][3]=" kPa";
   s1[11][3]=" Pa.s";
   s1[12][3]=" W/mK";
   s1[13][3]=" N/m";
   s1[14][3]=" ";
   s1[15][3]=" kg/m^3";
   s1[16][3]=" m/s";
   }
   else{
   s1[0][1]="P, pressure        ";
   s1[1][1]="T, temperature     ";
   s1[2][1]="v, specific volume ";
   s1[3][1]="h, enthalpy        ";
   s1[4][1]="u, internal energy ";
   s1[5][1]="s, entropy         ";
   s1[6][1]="x, quality         ";
   s1[7][1]="Cp, specific heat at constant pressure ";
   s1[8][1]="Cv, specific heat at constant volume ";
   s1[9][1]=""+'\u03B1'+" isobaric thermal expansion coefficient ";
   s1[10][1]=""+'\u03BA'+" isothermal compressibility" ;
   s1[11][1]=""+'\u03B7'+" Dynamic viscosity" ;
   s1[12][1]="k Thermal conductivity" ;
   s1[13][1]=""+'\u03C3'+" Surface tension" ;
   s1[14][1]="Prandtl number" ;
   s1[15][1]=""+'\u03C1'+" density" ;
   s1[16][1]=" speed of sound" ;
   s1[0][3]=" kPA          ";
   s1[1][3]=" deg K         ";
   s1[2][3]=" m"+'\u00B3'+"/kg        ";
   s1[3][3]=" KJ/kg         ";
   s1[4][3]=" KJ/kg         ";
```

```
   s1[5][3]=" KJ/kgK         ";
   s1[6][3]=" kg vap/kg mix ";
   s1[7][3]=" KJ/kgK";
   s1[8][3]=" KJ/kgK";
   s1[9][3]=" 1/K";
   s1[10][3]=" kPa";
   s1[11][3]=" Pa.s";
   s1[12][3]=" W/mK";
   s1[13][3]=" N/m";
   s1[14][3]=" ";
   s1[15][3]=" kg/m^3";
   s1[16][3]=" m/s";
   }
  double pp[]=propertyC(s,v1,v2);
for(int i=0;i<17;i++)
   {s1[i][2]=""+pp[i];}
    //s1[6][2]=phase(pp[6]);
for(int i=0;i<17;i++)
{s1[i][0]=""+i;}
return s1;
}
public static void main(String arg[])
{   air_PR air=new air_PR();}
}
```

```
//Brayton-Ericsson refrigeration cycle
public class brayton3_PR
{
air_PR g;
double T[];
double P[];
double s[];
double h[];
double Cp[];
double etacompressor;
double etaturbin;
double Wturbin,Wkompresor,Wnet,Q,Qref,COP,COPref,T1,P1,T2,P2,T3,P3,T4,P4;
public brayton3_PR(double m,double T1,double P1, double P2, double T3,double etacompressori,double etaturbini)
{ set(m,T1,P1,P2,T3,etacompressori,etaturbini);}

public double set(double m,double T1,double P1, double P2, double T3,double etacompressori,double etaturbini)
{
  T=new double[6];
  P=new double[6];
  s=new double[6];
  h=new double[6];
  Cp=new double[6];
  g=new air_PR();
  P[0]=P2;
  P[1]=P1;
  P[2]=P2;
  P[4]=P1;
  P[3]=P2;
  T[1]=T1;
  T[3]=T3;
  P[5]=P1;
  etacompressor=etacompressori;
  etaturbin=etaturbini;
// 1
double a1[]=g.property("tp",T[1],P[1]);
h[1]=a1[3];
s[1]=a1[5];
s[0]=s[1];
// 0=2i  isentropic process
double a0[]=g.property("ps",P[0],s[0]);
T[0]=a0[1];
h[0]=a0[3];

// 2
h[2]=(h[0]-h[1])/etacompressor+h[1];
double a2[]=g.property("ph",P[2],h[2]);
s[2]=a2[5];
T[2]=a2[1];
//3
double a3[]=g.property("tp",T[3],P[3]);
```

```
h[3]=a3[3];
s[3]=a3[5];
//4i = 5 isentropic process
s[5]=s[3];
double a5[]=g.property("ps",P[5],s[5]);
T[5]=a5[1];
h[5]=a5[3];
a1=g.property("tp",T[5],P[1]);
//4
h[4]=h[3]-etaturbin*(h[3]-h[5]);
double a4[]=g.property("ph",P[4],h[4]);
T[4]=a4[1];
s[4]=a4[5];
Wturbin=m*(h[3]-h[4]);
Wkompresor=m*(h[2]-h[1]);
Q=m*(h[2]-h[3]);
Qref=m*(h[1]-h[4]);
Wnet=Wkompresor-Wturbin;
COP=Q/Wnet;
COPref=Qref/Wnet;
return T[4];
}
public double[][] line1(double T1,double T2,double P,int n)
{
double TT1[]=new double[n];
double ss1[]=new double[n];
double aa[];

for(int i=0;i<n;i++)
{TT1[i]=T1+i*(T2-T1)/(n-1);
 aa=g.property("tp",TT1[i],P);
 ss1[i]=aa[5];
}
double a[][]={TT1,ss1};
return a;
}
public String toString()
{
String s=" Brayton cycle \n";
s+="Turbine power output W turbine = "+Wturbin+" kW \n";
s+="Compressor power input W kompresor = "+Wkompresor+" kW\n";
s+="Net power output  Wnet = "+Wnet+" kW\n";
s+="Heat exchanger heat removal Q  = "+Q+" kW\n";
s+="refrigeration heat removal Qref = "+Qref+" kW \n";
s+="COP refrigeration  = "+COP+"\n";
s+="COP refrigeration  = "+COP+"\n";
s+="Compressor inlet temperature = "+(T[1]-273.15)+" degree C\n";
s+="Compressor inlet pressure  = "+P[1]+" kPa\n";
s+="Compressor exit temperature = "+(T[2]-273.15)+" degree C\n";
s+="Compressor exit  pressure   = "+P[2]+" kPa\n";
s+="Turbine inlet temperature = "+(T[3]-273.15)+" degree C\n";
s+="Turbine inlet pressure   = "+P[3]+" kPA\n";
s+="Turbine exit temperature = "+(T[4]-273.15)+" degree C\n";
s+="Turbine exit pressure   = "+P[4]+" kPa\n";
return s;
}
public void plot()
{
 double a23[][]=line1(T[2],T[3],P[2],15);

 double tt[]=a23[0];
 double ss[]=a23[1];
 Plot pp=new Plot(ss,tt);
 double a41[][]=line1(T[4],T[1],P[1],15);

 tt=a41[0];
 ss=a41[1];
 pp.addData(ss,tt);
 double t1[]={T[1],T[2]};
 double s1[]={s[1],s[2]};
 pp.addData(s1,t1);
 double t2[]={T[1],T[0]};
 double s2[]={s[1],s[0]};
 pp.addData(s2,t2,1);
 double t3[]={T[3],T[4]};
 double s3[]={s[3],s[4]};
```

```
 pp.addData(s3,t3);
 double t4[]={T[3],T[5],T[4]};
 double s4[]={s[3],s[5],s[4]};
 pp.addData(s4,t4,1);
 pp.plot();
 }
 }
```

```
public class brayton3_PRtest
{
  public static void main(String arg[])
  {              double mhava=0.124; //kg/s air inlet mass flow rate
                 double T1=300.0; //derece K air inlet temperature
                 double P1=101.325;// kPa air inlet pressure
                 double P2=300;// kPa compressor exit pressure
                 double T3=357.0; //derece K turbine inlet temperature
                 double etakompresor=0.85; //compressor isentropic efficiency
                 double etaturbin=0.85; ///turbine isentropic efficiency
     brayton3_PR bry=new brayton3_PR(mhava,T1,P1,P2,T3,etakompresor,etaturbin);
     System.out.println(bry.toString());
     bry.plot();
  }
}
```

```
 ---------- Capture Output ----------
> "E:\co\java\bin\javaw.exe" brayton3_NISTtest
T1s=298.8449784556932
T1=429.07239190547887
T1s=260.766599603997
T1=272.95709504427793
 Brayton cycle
Turbine power output W turbine = 10.079212103560213 kW
Compressor power input W kompresor = 15.98840950516935 kW
Net power output  Wnet = 5.909197401609138 kW
Heat exchanger heat removal Q  = 8.90089769748395 kW
refrigeration heat removal Qref = 2.9917002958748133 kW
COP refrigeration  = 1.5062786183213546
COP refrigeration  = 1.5062786183213546
Compressor inlet temperature = 26.850000000000023 degree C
Compressor inlet pressure  = 101.325 kPa
Compressor exit temperature = 154.54630141284434 degree C
Compressor exit  pressure   = 300.0 kPa
Turbine inlet temperature = 83.85000000000002 degree C
Turbine inlet pressure   = 300.0 kPA
Turbine exit temperature = 2.873314653946238 degree C
Turbine exit pressure   = 101.325 kPa
```

```
public class air_NIST
{  public String option;
   public boolean mole=false;
   public double Tj=132.6312; //degree K Maxcondentherm
   public double Pj=3785.02;  //kPa      Maxcondentherm
   public double roj=10.4477; //kmol/m^3 Maxcondentherm
   public double Tc=132.5306; //Critical temperature
   public double Pc=3786;     //kPa critical pressure
   public double roc=11.8308; //kmol/m^3 critical density
   public double T0=273.15;
   public double P0=101.325;  //kPa reference pressure
   public double h0=7914.254398937379;  //kJ/kmol reference enthalpy
   public double s0=196.22372197083126;    //kJ/(kmolK) refernece entropy
   public double u0=5644.494709060624;
   public double M=28.9586;   // molar mass of dry air kg/kmol
   //public double M=28.96546;
   public double R=8.314510;  //kJ/kmol universal gas constant
   //public double RM=0.287117125; //kJ/kg  air specific gas constant
   public double Ps=5.265; //kPa Freezing reference pressure
   public double Ts=59.75;  //degree K freezing reference temp.
   public double ro0=0.0440575275197657; //kg/m^3 (T=273.15 P=100 kPa)
   public double v0=1.0/0.0440575275197657;  //m^3/kg

   air_IAPWS aIAPWS=new air_IAPWS();
   air_vis_k1 avk=new air_vis_k1();
//bubble and dew point properties
```

```
public double NPB[]={0.2260724,-7.080499,5.700283,-12.44017,17.81926,-10.81364};
public double NPD[]={-0.1567266,-5.539635,0,0,0.7567212,0,0,-3.514322};
public double NroB[]={44.3413,-240.073,285.139,-88.3366,-0.892181};
public double NroD[]={-2.0466,-4.7520,-13.259,-47.652};
public double NCP[]={3.4908880320E+00,
2.3955255830E-06,
7.1721112480E-09,
-3.1154131010E-13,
2.2380668800E-01,
7.9130950900E-01,
2.1223676800E-01,
1.9793890400E-01,
3.3640110000E+03,
2.2424500000E+03,
1.1158040000E+05,
};
public double N0[]={6.0571940000E-08,
-2.1027476900E-05,
-1.5886071600E-04,
-1.3841928076E+01,
1.7275266575E+01,
-1.9536342000E-04,
2.4908880320E+00,
7.9130950900E-01,
2.1223676800E-01,
-1.9793890400E-01,
2.5363650000E+01,
1.6907410000E+01,
8.7312790000E+01};
public double Nk[]={1.18160747229E-01,
7.13116392079E-01,
-1.61824192067E+00,
7.14140178971E-02,
-8.65421396646E-02,
1.34211176704E-01,
1.12626704218E-02,
-4.20533228842E-02,
3.49008431982E-02,
1.64957183186E-04,
-1.01365037912E-01,
-1.73813690970E-01,
-4.72103183731E-02,
-1.22523554253E-02,
-1.46629609713E-01,
-3.16055879821E-02,
2.33594806142E-04,
1.48287891978E-02,
-9.38782884667E-03};
public int ik[]={1,1,1,2,3,3,4,4,4,6,1,3,5,6,1,3,11,1,3};
public double jk[]={0,0.33,1.01,0,0,0.15,0,0.2,0.35,1.35,1.6,0.8,0.95,1.25,3.6,6,3.25,3.5,15};
public int lk[]={0,0,0,0,0,0,0,0,0,0,1,1,1,1,2,2,2,3,3};
double ds[]={273.1310048792371,
9.372035682011568,
0.1607990845018552,
0.0017521546222909953,
6.240982605845459E-6,
-2.600513304826448E-7,
-2.4587722508900994E-10,
1.0081660255713587E-10,
-3.857700254529247E-13,
-1.0915261987184568E-14,
7.783758584201146E-17};
double dh[]={270.00790775698164,
0.03525873115719365,
2.4624682282936277E-7,
-8.376744547776616E-11,
7.453959406764931E-15,
-3.4643553429821037E-19,
9.398947951010273E-24,
-1.5395836918710113E-28,
1.4995068941203076E-33,
-7.995678763739156E-39,
1.797427338632586E-44};

public void setMole(Boolean imole) {mole=imole;}
```

```
public air_NIST()
{input("english");}
public air_NIST(String optioni)
{input(optioni);}
public void input(String optioni) {
option=optioni;}//T0=273.15;P0=100;ro0=ro(T0,P0);v0=1.0/ro0;u0=u_Tro1(T0,ro0);h0=h_Tro1(T0,ro0);s0=s_Tro1(T0,ro0
);}
public double pow(double x,int n)
{double y=1;
 if(n>=0) { for(int i=0;i<n;i++) {y*=x;}
          return y;}
 else      { int n1=-n;for(int i=0;i<n1;i++) {y*=x;}
           return 1.0/y;}
}
public double pow(double x,double n)
{double y=1;
 if(n==0) return 1.0;
 else return Math.pow(x,n);
}
//Buble point-dew point calc
//buble point pressure
public double PB(double T)
{double t0=Tj/T;
double tetha=1.0-T/Tj;
double y=0;
for(int i=0;i<NPB.length;i++)
{y+=NPB[i]*Math.pow(tetha,(double)(i+1.0)/2.0);}
double PB=Pj*Math.exp(t0*y);
return PB;
}
//dew point pressure
public double PD(double T)
{double t0=Tj/T;
double tetha=1.0-T/Tj;
double y=0;
double x=0;
for(int i=0;i<NPD.length;i++)
{  y+=NPD[i]*Math.pow(tetha,(double)(i+1.0)/2.0);}
double PD=Pj*Math.exp(t0*y);
return PD;
}
//Buble point density
public double roB(double T)
{   double tetha=1-T/Tj;
              double y=1.0+NroB[0]*pow(tetha,0.65)+NroB[1]*pow(tetha,0.85)+NroB[2]*pow(tetha,0.95)+
              NroB[3]*pow(tetha,1.1)+NroB[4]*Math.log(T/Tj);
              return roj*y;
}
//Dew point density
public double roD(double T)
{   double tetha=1-T/Tj;
              double y=NroD[0]*pow(tetha,0.41)+NroD[1]*tetha+NroD[2]*pow(tetha,2.8)+NroD[3]*pow(tetha,6.5);
   double roD=roj*Math.exp(y);
   return roD;
}
//Freezing point pressure
public double PF(double T)
{double PF=Ps*(1.0-3549.5*(pow((T/Ts),1.78693)-1.0));
 return PF;
}
public double alpha0(double delta,double to)
{double alpha0=Math.log(delta);
 alpha0+=N0[0]/(to*to*to)+N0[1]/(to*to)+N0[2]/to+N0[3]+N0[4]*to;
 alpha0+=N0[5]*Math.pow(to,1.5)+N0[6]*Math.log(to);
 alpha0+=N0[7]*Math.log(1.0-Math.exp(-N0[10]*to))+N0[8]*Math.log(1.0-Math.exp(-N0[11]*to));
 alpha0+=N0[9]*Math.log(2.0/3.0+Math.exp(N0[12]*to));
 return alpha0;
}
public double alphar(double delta,double to)
{ double alphar=0;
  for(int k=0;k<10;k++)
  {alphar+=Nk[k]*pow(delta,ik[k])*pow(to,jk[k]);}
    for(int k=10;k<19;k++)
  {alphar+=Nk[k]*pow(delta,ik[k])*pow(to,jk[k])*Math.exp(-pow(delta,lk[k]));}
  return alphar;
}
```

```
public double alpha(double delta,double to)
{return alpha0(delta,to)+alphar(delta,to);}

public double d_alpha0_to(double delta,double to)
{//double alpha0=-3.0*N0[0]/(to*to*to*to)-2.0*N0[1]/(to*to*to)-N0[2]/(to*to)+N0[4];
 double alpha0=0;
 for(int i=0;i<5;i++) {alpha0+=N0[i]*(i-3.0)*pow(to,(i-4));}
 alpha0+=1.5*N0[5]*pow(to,0.5)+N0[6]/to+N0[7]*N0[10]/(Math.exp(N0[10]*to)-1.0);
 alpha0+=N0[8]*N0[11]/(Math.exp(N0[11]*to)-1.0);
 alpha0+=N0[9]*N0[12]/((2.0/3.0)*Math.exp(-N0[12]*to)+1.0);
 return alpha0;
}
public double d2_alpha0_toto(double delta,double to)
{//double alpha0=12.0*N0[0]/(to*to*to*to*to)+6.0*N0[1]/(to*to*to*to)+2.0*N0[2]/(to*to*to);
double alpha0=0;
 for(int i=0;i<5;i++) {alpha0+=N0[i]*(i-3.0)*(i-4.0)*pow(to,(i-5));}
 double xx10=Math.exp(N0[10]*to)-1.0;
 double xx11=Math.exp(N0[11]*to)-1.0;
 double xx12=(2.0/3.0)*Math.exp(-N0[12]*to)+1.0;
 alpha0+=0.75*N0[5]/Math.sqrt(to)-N0[6]/(to*to);
 alpha0+=-N0[7]*N0[10]*N0[10]*Math.exp(N0[10]*to)/(xx10*xx10);
 alpha0+=-N0[8]*N0[11]*N0[11]*Math.exp(N0[11]*to)/(xx11*xx11);
 alpha0+=(2.0/3.0)*N0[9]*N0[12]*N0[12]*Math.exp(-N0[12]*to)/(xx12*xx12);
 return alpha0;
}
public double d_alphar_de(double delta,double to)
{double alphar=0;
double xx=1;
 for(int k=0;k<10;k++)
 {alphar+=Nk[k]*ik[k]*pow(delta,(ik[k]-1))*pow(to,jk[k]);}
 for(int k=10;k<19;k++)
 {   xx=Math.pow(delta,lk[k]);
             alphar+=Nk[k]*pow(delta,(ik[k]-1))*pow(to,jk[k])*Math.exp(-xx)*(ik[k]-lk[k]*xx);}
 return alphar;
}
public double d2_alphar_dede(double delta,double to)
{double alphar=0;
double xx=1;
 for(int k=0;k<10;k++)
 {alphar+=Nk[k]*ik[k]*(ik[k]-1.0)*pow(delta,(ik[k]-2))*pow(to,jk[k]);}
 for(int k=10;k<19;k++)
 {   xx=Math.pow(delta,lk[k]);
             alphar+=Nk[k]*pow(delta,(ik[k]-2))*pow(to,jk[k])*Math.exp(-xx)*((ik[k]-lk[k]*xx)*(ik[k]-1.0-lk[k]*xx)-
lk[k]*lk[k]*xx);}
 return alphar;
}
public double d3_alphar_dedede(double delta,double to)
{double alphar=0;
double xx=1;
 for(int k=0;k<10;k++)
 {alphar+=Nk[k]*ik[k]*(ik[k]-1.0)*(ik[k]-2.0)*pow(delta,(ik[k]-3))*pow(to,jk[k]);}
 for(int k=10;k<19;k++)
 {   xx=Math.pow(delta,lk[k]);
             alphar+=Nk[k]*pow(delta,(ik[k]-3))*pow(to,jk[k])*Math.exp(-xx)*((ik[k]*(ik[k]-1)*(ik[k]-2))+
             xx*(-2.0*lk[k]+6.0*ik[k]*lk[k]-3.0-ik[k]*ik[k]*lk[k]-3.0*ik[k]*lk[k]*lk[k]+3.0*lk[k]*lk[k]*lk[k]*lk[k]*lk[k])+
             xx*xx*(3.0*ik[k]*lk[k]*lk[k]-3.0*lk[k]*lk[k]+3.0*lk[k]*lk[k]*lk[k])-
             xx*xx*xx*lk[k]*lk[k]*lk[k]);}
 return alphar;
}
public double d_alphar_to(double delta,double to)
{double alphar=0;
double xx=1;
 for(int k=0;k<10;k++)
 {alphar+=jk[k]*Nk[k]*pow(delta,ik[k])*pow(to,(jk[k]-1));}
 for(int k=10;k<19;k++)
 {   xx=Math.pow(delta,lk[k]);
             alphar+=jk[k]*Nk[k]*pow(delta,ik[k])*pow(to,(jk[k]-1))*Math.exp(-xx);}
 return alphar;
}
public double d2_alphar_toto(double delta,double to)
{double alphar=0;
double xx=1;
 for(int k=0;k<10;k++)
 {alphar+=jk[k]*(jk[k]-1)*Nk[k]*pow(delta,ik[k])*pow(to,(jk[k]-2));}
 for(int k=10;k<19;k++)
 {   xx=Math.pow(delta,lk[k]);
```

```
                alphar+=jk[k]*(jk[k]-1)*Nk[k]*pow(delta,ik[k])*pow(to,(jk[k]-2))*Math.exp(-xx);}
 return alphar;
}
public double d2_alphar_tode(double delta,double to)
{double alphar=0;
double xx=1;
 for(int k=0;k<10;k++)
 {alphar+=jk[k]*ik[k]*Nk[k]*pow(delta,(ik[k]-1))*pow(to,(jk[k]-1));}
 for(int k=10;k<19;k++)
 {   xx=Math.pow(delta,lk[k]);
                alphar+=jk[k]*Nk[k]*pow(delta,(ik[k]-1))*pow(to,(jk[k]-1))*Math.exp(-xx)*(ik[k]-lk[k]*xx);}
 return alphar;
}
//compressibility factor
//ro kmol/m^3 T degree K
public double Z(double T,double ro)
{double delta=ro/roj; double to=Tj/T;
 double Z=1.0+delta*d_alphar_de(delta,to);
 return Z;
}
//ro1 kg/m^3 T degree K ro kmol/m^3
public double Zm(double T,double ro1)
{double ro=ro1/M;return Z(T,ro);}

//Pressure kPa ro kmol/m^3 T degree K
public double P(double T,double ro)
{double delta=ro/roj; double to=Tj/T;
 double P=ro*R*T*Z(T,ro);
 return P;
}
public double Pm(double T,double ro1)
{double ro=ro1/M;return P(T,ro);}

public double P_Tro(double T,double ro)
{double delta=ro/roj; double to=Tj/T;
 double P=ro*R*T*Z(T,ro);
 return P;
}
public double P_Trom(double T,double ro1)
{double ro=ro1/M;return P_Tro(T,ro);}
public double P_Tv(double T,double v)
{double ro=1.0/v;return P_Tro(T,ro);
}
public double P_Tvm(double T,double v1)
{double v=v1*M;return P_Tv(T,v);}

//density ro kmol/m^3 rom kg/m^3
public double ro(double T,double P)
{if_x f=x->P(T,x)-P;
 double ro1,ro=0;
 if(T<Tj)
 {    double roD=PD(T);
      double roB=PB(T);
      double PD=PD(T);
      double PB=PB(T);
      double x=0;
                if(P<PD) {//System.out.println("gas");
                ro1=aIAPWS.ro(T,P);ro=secant(f,ro1);}
                else if(P>=PD && P<=PB) {//System.out.println("saturated");
                x=(P-PD)/(PB-PD);ro1=roB*(1-x)+roD*x;ro=ro1;}
                else {//System.out.println("liquid");ro1=roB;
                ro=bisection(f,roB,100);}
 }
 else
 {//System.out.println("super maxcondentherm");
 ro1=aIAPWS.ro(T,P);ro=secant(f,ro1);}
 return ro;}

public double rom(double T,double P)
{return  ro(T,P)*M;}

//specific volume
public double v(double T,double P)
{ return 1.0/ro(T,P);}
public double vm(double T,double P)
{ return 1.0/rom(T,P);}
```

```
//internal energy kJ/kmol
public double u_Tro1(double T,double ro)
{double delta=ro/roj; double to=Tj/T;
 double u=R*T*to*(d_alpha0_to(delta,to)+d_alphar_to(delta,to));
 return u;}

public double u_Tro(double T,double ro)
{ return u_Tro1(T,ro)-u0;}
public double u_Trom(double T,double ro1)
{ double ro=ro1/M;return u_Tro(T,ro)/M;}

public double u_Tv(double T,double v)
{    double ro=1.0/v;
   return u_Tro1(T,ro);}

public double u_Tvm(double T,double v1)
{double v=v1*M;return u_Tv(T,v)/M; }

//u(T,P) internal energy kJ/kg
public double u(double T,double P)
{ double v=v(T,P);
  return u_Tv(T,v);     }
public double um(double T,double P)
{ double v=v(T,P);
  return u_Tv(T,v)/M; }

//enthalpy kJ/kmol
public double h_Tro1(double T,double ro)
{double delta=ro/roj; double to=Tj/T;
 double h=R*T*(to*(d_alpha0_to(delta,to)+d_alphar_to(delta,to))+delta*d_alphar_de(delta,to)+1.0);
 return h;}

public double h_Tro(double T,double ro)
{return h_Tro1(T,ro)-h0;}

public double h_Trom(double T,double ro1)
{double ro=ro1/M;return (h_Tro1(T,ro)-h0)/M;}

public double h_Tv(double T,double v)
{double ro=1.0/v;
 return h_Tro(T,ro);}

public double h_Tvm(double T,double v1)
{double ro=1.0/(v1*M);
 return h_Tro(T,ro)/M;}

public double h(double T,double P)
{double ro=ro(T,P);
 return h_Tro(T,ro);}
 public double h_TP(double T,double P)
{double ro=ro(T,P);
 return h_Tro(T,ro);}

public double hm(double T,double P)
{double ro=ro(T,P);
 return h_Tro(T,ro)/M;}
public double hm_TP(double T,double P)
{double ro=ro(T,P);
 return h_Tro(T,ro)/M;}
//entropy kJ/kmol
public double s_Tro1(double T,double ro)
{double delta=ro/roj; double to=Tj/T;
 double s=R*(to*(d_alpha0_to(delta,to)+d_alphar_to(delta,to))-alpha0(delta,to)-alphar(delta,to));
 return s;}

 public double s_Tro(double T,double ro)
{return s_Tro1(T,ro)-s0;}

public double s_Trom(double T,double ro1)
{double ro=ro1/M;return (s_Tro1(T,ro)-s0)/M;}

public double s_Tv(double T,double v)
{double ro=1.0/v;return s_Tro(T,ro);}

public double s_Tvm(double T,double v1)
```

```
{double ro=1.0/(v1*M);return s_Tro(T,ro)/M;}

public double s(double T,double P)
{double ro=ro(T,P);return s_Tro(T,ro);}

public double sm(double T,double P)
{double ro=ro(T,P);return s_Tro(T,ro)/M;}

// Helmholtz energy
public double a_Tro(double T,double ro)
{double delta=ro/roj; double to=Tj/T;
 double a=R*T*(alpha0(delta,to)+alphar(delta,to));
 return a;
}

public double a_Trom(double T,double ro1)
{double ro=ro1/M;double delta=ro/roj; double to=Tj/T;
 double a=R*T*(alpha0(delta,to)+alphar(delta,to));
 return a/M;}

public double a_Tv(double T,double v)
{double ro=1.0/v;return a_Tro(T,ro);}

public double a_Tvm(double T,double v1)
{double ro=1.0/(v1*M);return a_Tro(T,ro);}

public double a(double T,double P)
{double ro=ro(T,P);return a_Tro(T,ro);}

public double am(double T,double P)
{double ro=ro(T,P);return a_Tro(T,ro)/M;}

//gibbs energy
public double g_Tro(double T,double ro)
{double delta=ro/roj; double to=Tj/T;
 double g=R*T*(1.0+alpha0(delta,to)+alphar(delta,to)+delta*d_alphar_de(delta,to));
 return g;}

public double g_Trom(double T,double ro1)
{ double ro=ro1/M;
  return g_Tro(T,ro)/M;}

public double g_Tv(double T,double v)
{double ro=1.0/v;return g_Tro(T,ro);}

public double g_Tvm(double T,double v1)
{double v=v1*M;return g_Tv(T,v)/M;}

public double g(double T,double P)
{double ro=ro(T,P);return g_Tro(T,ro);}

public double gm(double T,double P)
{double ro=ro(T,P);return g_Tro(T,ro)/M;}

//Cv specific heat at constant volume isovolumetric specific heat kJ/kmol K
public double Cv_Tro(double T,double ro)
{double delta=ro/roj; double to=Tj/T;
 double Cv=-R*to*to*(d2_alpha0_toto(delta,to)+d2_alphar_toto(delta,to));
 return Cv;}

public double Cv_Trom(double T,double ro1)
{double ro=ro1/M;return Cv_Tro(T,ro)/M;}

public double Cv(double T,double P)
{double ro=ro(T,P);return Cv_Tro(T,ro);}

public double Cvm(double T,double P)
{double ro=ro(T,P);return Cv_Tro(T,ro)/M;}

public double Cv(double T)
{double ro=ro(T,100.0);return Cv_Tro(T,ro);}

public double Cvm(double T)
{double ro=ro(T,100.0);return Cv_Tro(T,ro)/M;}

public double Cv_Tv(double T,double v)
```

```
{double ro=1.0/v;return Cv_Tro(T,ro);}

public double Cv_Tvm(double T,double v1)
{ double ro=1.0/(v1*M);
  return Cv_Tro(T,ro)/M;}

//Cp specific heat at constant pressure isochoric specific heat kJ/kmolK
public double Cp_Tro(double T,double ro)
{double delta=ro/roj; double to=Tj/T;
 double x1=1.0+delta*d_alphar_de(delta,to)-delta*to*d2_alphar_tode(delta,to);
 double x2=1.0+2.0*delta*d_alphar_de(delta,to)+delta*delta*d2_alphar_dede(delta,to);
 double Cp=Cv_Tro(T,ro)+R*x1*x1/x2;
 return Cp;
}

public double Cp_Trom(double T,double ro1)
{double ro=ro1/M;
  return Cp_Tro(T,ro)/M;}

public double Cp(double T,double P)
{double ro=ro(T,P);return Cp_Tro(T,ro);}

public double Cpm(double T,double P)
{double ro=ro(T,P);return Cp_Tro(T,ro)/M;}

public double Cp(double T)
{double ro=ro(T,100.0);return Cp_Tro(T,ro);}

public double Cpm(double T)
{double ro=ro(T,100.0);return Cp_Tro(T,ro)/M;}

public double Cp_Tv(double T,double v)
{double ro=1.0/v;return Cv_Tro(T,ro);}

public double Cp_Tvm(double T,double v1)
{double ro=1.0/(v1*M);
  return Cp_Tro(T,ro)/M;}

//speed of sound m/s

public double w_Tro(double T,double ro)
{double delta=ro/roj; double to=Tj/T;
 double
x1=R/M*1.0e3*T*Cp_Tro(T,ro)/Cv_Tro(T,ro)*(1.0+2.0*delta*d_alphar_de(delta,to)+delta*delta*d2_alphar_dede(delta,to)
);
 double w=Math.sqrt(x1);
 return w;
}
public double w_Trom(double T,double ro1)
{double ro=ro1/M; return w_Tro(T,ro);}

public double w(double T,double P)
{double ro=ro(T,P);return w_Tro(T,ro);}

public double w(double T)
{double ro=ro(T,100.0);return Cp_Tro(T,ro);}

public double w_Tv(double T,double v)
{double ro=1.0/v;return w_Tro(T,ro);}

public double w_Tvm(double T,double v1)
{double ro=1.0/(v1*M);return w_Tro(T,ro);}

//other derivatives
//(dp/dro)T
double dP_dro_T(double T,double ro)
{double delta=ro/roj; double to=Tj/T;
 double dP=R*T*(1.0+2.0*delta*d_alphar_de(delta,to)+delta*delta*d2_alphar_dede(delta,to));
 return dP;
}

double dP_drom_T(double T,double ro1)
{double ro=ro1/M;return dP_drom_T(T,ro);}

//(d2p/dro2)T
double d2P_droro_T(double T,double ro)
```

```
{double delta=ro/roj; double to=Tj/T;
 double
dP=R*T/ro*(2.0*delta*d_alphar_de(delta,to)+4.0*delta*delta*d2_alphar_dede(delta,to)+delta*delta*delta*d3_alphar_dede
de(delta,to));
 return dP;
}
double d2P_dromrom_T(double T,double ro1)
{ double ro=ro1/M;return d2P_droro_T(T,ro);}

//(dp/dT)ro
double dP_dT_ro(double T,double ro)
{double delta=ro/roj; double to=Tj/T;
 double dP=R*ro*(1.0+delta*d_alphar_de(delta,to)-delta*to*d2_alphar_tode(delta,to));
 return dP;
}
double dP_dT_rom(double T,double ro1)
{double ro=ro1/M;return dP_dT_ro(T,ro);}

//beta isobaric thermal expansion coefficient volume expansivity 1/K
double beta(double T,double ro)
{double delta=ro/roj; double to=Tj/T;
 double beta=1.0/ro*dP_dT_ro(T,ro)/dP_dro_T(T,ro);
 return beta;
}

double betam(double T,double ro1)
{double ro=ro1/M;return beta(T,ro);}

//Joule-Thomson coefficient
double mu(double T,double ro)
{double delta=ro/roj; double to=Tj/T;
 double mu=(T*beta(T,ro)-1.0)/(ro*Cp_Tro(T,ro));
 return mu;
}

double mum(double T,double ro1)
{double ro=ro1/M;return mu(T,ro);}

//Isothermal compressibility
double kappaS_Tro(double T,double ro)
{double delta=ro/roj; double to=Tj/T;
 double k=ro*dP_dro_T(T,ro);
 return 1.0/k*1e-3;
}

double kappaS_Trom(double T,double ro1)
{double ro=ro1/M;return kappaS_Tro(T,ro);}

// temperature and density as a function of pressure and enthalpy
public double[] Tro_Ph(double P,double h)
{
//Lambda variable
                ifi_xj f2=(double x[])->
                {  double ff[]=new double[2];
      ff[0]=P_Tro(x[0],x[1])-P;
      ff[1]=h_Tro(x[0],x[1])-h;
      return ff;
    };
    poly ph=new poly(dh);
    double T1=ph.func(h);
    System.out.println("T1="+T1);
    double ro1=ro(T1,P);
    double [] x0={T1,ro1};
    //Find double roots by secant(newton?) method
    double c[]=secant2(f2,x0);//NA30Ci.continuationRK4(f2,x0,4);//
    return c;
}
public double[] Trom_Phm(double P,double h1)
{ double h=h1*M;
            double a[]=Tro_Ph(P,h);
 double c[]={a[0],a[1]*M};return c;
}
// temperature and density as a function of pressure and entropy
public double[] Tro_Ps(double P,double s)
{
//Lambda variable
```

```
                ifi_xj f2=(double x[])->
                {  double ff[]=new double[2];
        ff[0]=P_Tro(x[0],x[1])-P;
        ff[1]=s_Tro(x[0],x[1])-s;
        return ff;
      };
      poly ps=new poly(ds);
      double T1=ps.func(s);
           System.out.println("T1s="+T1);
      double ro1=ro(T1,P);
      double [] x0={T1,ro1};
      //Find double roots by secant(newton?) method
      double c[]=secant2(f2,x0);//NA30Ci.continuationRK4(f2,x0,4);//
      return c;
}
public double[] Trom_Psm(double P,double s1)
{   double s=s1*M;
              double a[]=Tro_Ps(P,s);
 double c[]={a[0],a[1]*M};return c;
}
//Pressure as a function of temperature and entropy
public double P_Ts(double T,double s)
{if_x f1=x->s(T,x)-s;
 double P1=10.0;
 double P2=10000.0;
 double P=bisection(f1,P1,P2);
 return P;
}
public double P_Tsm(double T,double s1)
{ double s=s1*M;return P_Ts(T,s);
}
//root finding
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.func(x);
   dy=f.dfunc(x);
   i++;
  }
  if(i>=miter)
    System.out.println("i="+i+"result may not be correct");
  return x;
}
public static double[] secant2(ifi_xj f,double x[])
{
// Roots of nonlinear system of equations secant method
//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=50;
double tolerance=1.0e-5;
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) System.out.println("Warning : Maximum number of iterations are exceeded \nResults may not be valid");
return x;
}
//Bisection root finding method
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;
  }
  //viscosity
public double vis(double T,double P)
{return avk.vis1(T,P)*1e-6;}
public double k(double T,double P)
{return avk.k1(T,P)*1e-3;}
//Prandtl number
public double Pr_Tro(double T,double ro)
{  double P=P_Tro(T,ro);
   double mu=vis(T,P);
   double k=k(T,ro);
   double Cp=Cpm(T);
   return Cp*1e3*mu/k;
}
public double Pr_Trom(double T,double ro1)
{  double P=P_Trom(T,ro1);
   double mu=vis(T,P);
   double k=k(T,ro1);
   double Cp=Cpm(T);
   return Cp*1e3*mu/k;
}
public double T_Pro(double P,double ro)
{if_x f1=x->P_Tro(x,ro)-P;
 double T1=aIAPWS.T_Pro(P,ro); //case of ideal gas
 double T=secant(f1,T1);
 return T;
}
public double T_Prom(double P,double ro1)
{double ro=ro1/M;return T_Pro(P,ro);}

// PROPERTY CALCULATIONS =======
//mole basis properties:
public double[] property(String s,double v1,double v2)
{ if(mole)
  {
  if(s.equals("pt")) return property_TP(v2,v1);
  else if(s.equals("tp"))  {return property_TP(v1,v2);}
  else if(s.equals("vt"))  {double ro=1.0/v1;return property_Tro(v2,ro);}
  else if(s.equals("tv"))  {double ro=1.0/v2;return property_Tro(v1,ro);}
  else if(s.equals("rot")) {return property_Tro(v2,v1);}
  else if(s.equals("tro")) {return property_Tro(v1,v2);}
  else if(s.equals("pv"))  return property_Pro(v1,1.0/v2);
  else if(s.equals("pro")) return property_Pro(v1,v2);
  else if(s.equals("vp"))  return property_Pro(v2,1.0/v1);
  else if(s.equals("rop")) return property_Pro(v2,v1);
  else if(s.equals("ph"))  return property_Ph(v1,v2);
  else if(s.equals("hp"))  return property_Ph(v2,v1);
  else if(s.equals("ps"))  return property_Ps(v1,v2);
  else if(s.equals("sp"))  return property_Ps(v2,v1);
  else if(s.equals("ts"))  return property_Ts(v1,v2);
  else if(s.equals("st"))  return property_Ts(v2,v1);
  else return property_TP(v1,v2);
  }
  else
  {  if(s.equals("pt")) return propertym_TP(v2,v1);
  else if(s.equals("tp"))  {return propertym_TP(v1,v2);}
  else if(s.equals("vt"))  {double ro=1.0/v1;return propertym_Tro(v2,ro);}
  else if(s.equals("tv"))  {double ro=1.0/v2;return propertym_Tro(v1,ro);}
  else if(s.equals("rot")) {return propertym_Tro(v2,v1);}
  else if(s.equals("tro")) {return propertym_Tro(v1,v2);}
  else if(s.equals("pv"))  return propertym_Pro(v1,1.0/v2);
  else if(s.equals("pro")) return propertym_Pro(v1,v2);
  else if(s.equals("vp"))  return propertym_Pro(v2,1.0/v1);
```

```
  else if(s.equals("rop")) return propertym_Pro(v2,v1);
  else if(s.equals("ph"))  return propertym_Ph(v1,v2);
  else if(s.equals("hp"))  return propertym_Ph(v2,v1);
  else if(s.equals("ps"))  return propertym_Ps(v1,v2);
  else if(s.equals("sp"))  return propertym_Ps(v2,v1);
  else if(s.equals("ts"))  return propertym_Ts(v1,v2);
  else if(s.equals("st"))  return propertym_Ts(v2,v1);
  else return propertym_TP(v1,v2);
  }
}
public double[] propertyC(String s,double v1,double v2)
{ double t=273.15;
  if(mole)
  {
  if(s.equals("pt")) return property_TP(v2,(v1+t));
  else if(s.equals("tp"))  {return property_TP((v1+t),v2);}
  else if(s.equals("vt"))  {double ro=1.0/v1;return property_Tro((v2+t),ro);}
  else if(s.equals("tv"))  {double ro=1.0/v2;return property_Tro((v1+t),ro);}
  else if(s.equals("rot")) {return property_Tro((v2+t),v1);}
  else if(s.equals("tro")) {return property_Tro((v1+t),v2);}
  else if(s.equals("pv"))  return property_Pro(v1,1.0/v2);
  else if(s.equals("pro")) return property_Pro(v1,v2);
  else if(s.equals("vp"))  return property_Pro(v2,1.0/v1);
  else if(s.equals("rop")) return property_Pro(v2,v1);
  else if(s.equals("ph"))  return property_Ph(v1,v2);
  else if(s.equals("hp"))  return property_Ph(v2,v1);
  else if(s.equals("ps"))  return property_Ps(v1,v2);
  else if(s.equals("sp"))  return property_Ps(v2,v1);
  else if(s.equals("ts"))  return property_Ts((v1+t),v2);
  else if(s.equals("st"))  return property_Ts((v2+t),v1);
  else return property_TP((v1+t),v2);
  }
  else
  {if(s.equals("pt")) return propertym_TP(v2,(v1+t));
  else if(s.equals("tp"))  {return propertym_TP((v1+t),v2);}
  else if(s.equals("vt"))  {double ro=1.0/v1;return propertym_Tro((v2+t),ro);}
  else if(s.equals("tv"))  {double ro=1.0/v2;return propertym_Tro((v1+t),ro);}
  else if(s.equals("rot")) {return propertym_Tro((v2+t),v1);}
  else if(s.equals("tro")) {return propertym_Tro((v1+t),v2);}
  else if(s.equals("pv"))  return propertym_Pro(v1,1.0/v2);
  else if(s.equals("pro")) return propertym_Pro(v1,v2);
  else if(s.equals("vp"))  return propertym_Pro(v2,1.0/v1);
  else if(s.equals("rop")) return propertym_Pro(v2,v1);
  else if(s.equals("ph"))  return propertym_Ph(v1,v2);
  else if(s.equals("hp"))  return propertym_Ph(v2,v1);
  else if(s.equals("ps"))  return propertym_Ps(v1,v2);
  else if(s.equals("sp"))  return propertym_Ps(v2,v1);
  else if(s.equals("ts"))  return propertym_Ts((v1+t),v2);
  else if(s.equals("st"))  return propertym_Ts((v2+t),v1);
  else return propertym_TP((v1+t),v2);
  }
}
public double[] property_Tro(double T,double ro)
{ double a[]=new double[17];
 double P=a[0]=P_Tro(T,ro);
 a[1]=T;
 a[2]=1.0/ro;
 a[3]=h_Tro(T,ro);
 a[4]=u_Tro(T,ro);
 a[5]=s_Tro(T,ro);
 a[6]=2.0;
 a[7]=Cp(T);
 a[8]=Cv(ro);
 a[9]=beta(T,ro);
 a[10]=kappaS_Tro(T,ro);
 a[11]=vis(T,P);
 a[12]=k(T,P);
 a[13]=0.0;
 a[14]=Pr_Tro(T,ro);;
 a[15]=ro;
 a[16]=w_Tro(T,ro);;
 return a;
}
public double[] property_TP(double T,double P)
{double a[]=new double[17];
 double ro=ro(T,P);
```

```
 a[0]=P;
 a[1]=T;
 a[2]=1.0/ro;
 a[3]=h_Tro(T,ro);
 a[4]=u_Tro(T,ro);
 a[5]=s(T,P);
 a[6]=2.0;
 a[7]=Cp(T);
 a[8]=Cv(T);
 a[9]=beta(T,ro);
 a[10]=kappaS_Tro(T,ro);
 a[11]=vis(T,ro);
 a[12]=k(T,P);
 a[13]=0.0;
 a[14]=Pr_Tro(T,ro);
 a[15]=ro;
 a[16]=w_Tro(T,ro);
 return a;
}
public double[] property_Pro(double P,double ro)
{double a[]=new double[17];
 double T=T_Pro(P,ro);
 a[0]=P;
 a[1]=T;
 a[2]=1.0/ro;
 a[3]=h_Tro(T,ro);
 a[4]=u_Tro(T,ro);
 a[5]=s(T,P);
 a[6]=2.0;
 a[7]=Cp(T);
 a[8]=Cv(T);
 a[9]=beta(T,ro);
 a[10]=kappaS_Tro(T,ro);
 a[11]=vis(T,ro);
 a[12]=k(T,P);
 a[13]=0.0;
 a[14]=Pr_Tro(T,ro);
 a[15]=ro;
 a[16]=w_Tro(T,ro);
 return a;
}
public double[] property_Ts(double T,double s)
{ double P=P_Ts(T,s);
  return property_TP(T,P);
}
public double[] property_Ph(double P,double h)
{   double a[]=new double[17];
                double c[]=Tro_Ph(P,h);
     double T=c[0];
     double ro=c[1];
 a[0]=P;
 a[1]=T;
 a[2]=1.0/ro;
 a[3]=h;
 a[4]=u_Tro(T,ro);
 a[5]=s(T,P);
 a[6]=2.0;
 a[7]=Cp(T);
 a[8]=Cv(T);
 a[9]=beta(T,ro);
 a[10]=kappaS_Tro(T,ro);
 a[11]=vis(T,ro);
 a[12]=k(T,P);
 a[13]=0.0;
 a[14]=Pr_Tro(T,ro);
 a[15]=ro;
 a[16]=w_Tro(T,ro);
 return a;
}
public double[] property_Ps(double P,double s)
{   double a[]=new double[17];
                double c[]=Tro_Ps(P,s);
     double T=c[0];
     double ro=c[1];
 a[0]=P;
 a[1]=T;
```

```
 a[2]=1.0/ro;
 a[3]=h_Tro(T,ro);
 a[4]=u_Tro(T,ro);
 a[5]=s;
 a[6]=2.0;
 a[7]=Cp(T);
 a[8]=Cv(T);
 a[9]=beta(T,ro);
 a[10]=kappaS_Tro(T,ro);
 a[11]=vis(T,ro);
 a[12]=k(T,P);
 a[13]=0.0;
 a[14]=Pr_Tro(T,ro);
 a[15]=ro;
 a[16]=w_Tro(T,ro);
 return a;
}
//==============================
//Mass basis properties

public double[] propertym_Tro(double T,double ro1)
{ double a[]=new double[17];
 double P=a[0]=P_Trom(T,ro1);
 a[1]=T;
 a[2]=1.0/ro1;
 a[3]=h_Trom(T,ro1);
 a[4]=u_Trom(T,ro1);
 a[5]=s_Trom(T,ro1);
 a[6]=2.0;
 a[7]=Cpm(T);
 a[8]=Cvm(T);
 a[9]=betam(T,ro1);
 a[10]=kappaS_Trom(T,ro1);
 a[11]=vis(T,P);
 a[12]=k(T,P);
 a[13]=0.0;
 a[14]=Pr_Trom(T,ro1);;
 a[15]=ro1;
 a[16]=w_Trom(T,ro1);;
 return a;
}
public double[] propertym_TP(double T,double P)
{double a[]=new double[17];
 double ro1=rom(T,P);
 a[0]=P;
 a[1]=T;
 a[2]=1.0/ro1;
 a[3]=h_Trom(T,ro1);
 a[4]=u_Trom(T,ro1);
 a[5]=sm(T,P);
 a[6]=2.0;
 a[7]=Cpm(T);
 a[8]=Cvm(T);
 a[9]=betam(T,ro1);
 a[10]=kappaS_Trom(T,ro1);
 a[11]=vis(T,ro1);
 a[12]=k(T,P);
 a[13]=0.0;
 a[14]=Pr_Trom(T,ro1);
 a[15]=ro1;
 a[16]=w_Trom(T,ro1);
 return a;
}
public double[] propertym_Pro(double P,double ro1)
{double a[]=new double[17];
 double T=T_Prom(P,ro1);
 a[0]=P;
 a[1]=T;
 a[2]=1.0/ro1;
 a[3]=h_Trom(T,ro1);
 a[4]=u_Trom(T,ro1);
 a[5]=sm(T,P);
 a[6]=2.0;
 a[7]=Cpm(T);
 a[8]=Cvm(T);
 a[9]=betam(T,ro1);
```

```
 a[10]=kappaS_Trom(T,ro1);
 a[11]=vis(T,ro1);
 a[12]=k(T,P);
 a[13]=0.0;
 a[14]=Pr_Trom(T,ro1);
 a[15]=ro1;
 a[16]=w_Trom(T,ro1);
 return a;
 }
public double[] propertym_Ts(double T,double s1)
{ double P=P_Tsm(T,s1);
  return propertym_TP(T,P);
}
public double[] propertym_Ph(double P,double h1)
{   double a[]=new double[17];
                double c[]=Trom_Phm(P,h1);
     double T=c[0];
     double ro1=c[1];
 a[0]=P;
 a[1]=T;
 a[2]=1.0/ro1;
 a[3]=h1;
 a[4]=u_Trom(T,ro1);
 a[5]=sm(T,P);
 a[6]=2.0;
 a[7]=Cpm(T);
 a[8]=Cvm(T);
 a[9]=betam(T,ro1);
 a[10]=kappaS_Trom(T,ro1);
 a[11]=vis(T,ro1);
 a[12]=k(T,P);
 a[13]=0.0;
 a[14]=Pr_Trom(T,ro1);
 a[15]=ro1;
 a[16]=w_Trom(T,ro1);
 return a;
 }
public double[] propertym_Ps(double P,double s1)
{   double a[]=new double[17];
                double c[]=Trom_Psm(P,s1);
     double T=c[0];
     double ro1=c[1];
 a[0]=P;
 a[1]=T;
 a[2]=1.0/ro1;
 a[3]=h_Trom(T,ro1);
 a[4]=u_Trom(T,ro1);
 a[5]=s1;
 a[6]=2.0;
 a[7]=Cpm(T);
 a[8]=Cvm(T);
 a[9]=betam(T,ro1);
 a[10]=kappaS_Trom(T,ro1);
 a[11]=vis(T,ro1);
 a[12]=k(T,P);
 a[13]=0.0;
 a[14]=Pr_Trom(T,ro1);
 a[15]=ro1;
 a[16]=w_Trom(T,ro1);
 return a;
 }
//===========
public String[][] toString(String s, double v1, double v2)
{  String s1[][]=new String[17][4];
    if(option.equals("Turkish"))
   {
   if(mole)
   {s1[0][1]="P, basınç        ";
   s1[1][1]="T, sıcaklık     ";
   s1[2][1]="v, özgül hacim ";
   s1[3][1]="h, enthalpi        ";
   s1[4][1]="u, iç enerji ";
   s1[5][1]="s, entropi         ";
   s1[6][1]="x, doygunluk derecesi          ";
   s1[7][1]="Cp, sabit basınçta özgül ısı ";
   s1[8][1]="Cv, sabit hacimde özgül ısı";
```

```
s1[9][1]=""+'\u03B1'+" sabit basınçta ısıl genleşme katsayısı";
s1[10][1]=""+'\u03BA'+" izotermal sıkıştırılabilirlik" ;
s1[11][1]=""+'\u03B7'+" Dinamik vizkozite" ;
s1[12][1]="k ısıl iletim sabiti" ;
s1[13][1]=""+'\u03C3'+" yüzey gerilimi" ;
s1[14][1]="Prandtl sayısı" ;
s1[15][1]=""+'\u03C1'+" yoğunluk" ;
s1[16][1]=" ses hızı" ;
s1[0][3]=" kPA          ";
s1[1][3]=" deg K        ";
s1[2][3]=" m"+'\u00B3'+"/kmol       ";
s1[3][3]=" KJ/kmol        ";
s1[4][3]=" KJ/kmol        ";
s1[5][3]=" KJ/kmolK       ";
s1[6][3]=" kmol buhar/kmol karışım ";
s1[7][3]=" KJ/kmolK";
s1[8][3]=" KJ/kmolK";
s1[9][3]=" 1/K";
s1[10][3]=" kPa";
s1[11][3]=" Pa.s";
s1[12][3]=" W/mK";
s1[13][3]=" N/m";
s1[14][3]=" ";
s1[15][3]=" kmol/m^3";
s1[16][3]=" m/s";
}
else
{
s1[0][1]="P, basınç       ";
s1[1][1]="T, sıcaklık     ";
s1[2][1]="v, özgül hacim ";
s1[3][1]="h, enthalpi       ";
s1[4][1]="u, iç enerji ";
s1[5][1]="s, entropi        ";
s1[6][1]="x, doygunluk derecesi         ";
s1[7][1]="Cp, sabit basınçta özgül ısı ";
s1[8][1]="Cv, sabit hacimde özgül ısı";
s1[9][1]=""+'\u03B1'+" sabit basınçta ısıl genleşme katsayısı";
s1[10][1]=""+'\u03BA'+" izotermal sıkıştırılabilirlik" ;
s1[11][1]=""+'\u03B7'+" Dinamik vizkozite" ;
s1[12][1]="k ısıl iletim sabiti" ;
s1[13][1]=""+'\u03C3'+" yüzey gerilimi" ;
s1[14][1]="Prandtl sayısı" ;
s1[15][1]=""+'\u03C1'+" yoğunluk" ;
s1[16][1]=" ses hızı" ;
s1[0][3]=" kPA          ";
s1[1][3]=" deg K        ";
s1[2][3]=" m"+'\u00B3'+"/kg       ";
s1[3][3]=" KJ/kg        ";
s1[4][3]=" KJ/kg        ";
s1[5][3]=" KJ/kgK       ";
s1[6][3]=" kg buhar/kg karışım ";
s1[7][3]=" KJ/kgK";
s1[8][3]=" KJ/kgK";
s1[9][3]=" 1/K";
s1[10][3]=" kPa";
s1[11][3]=" Pa.s";
s1[12][3]=" W/mK";
s1[13][3]=" N/m";
s1[14][3]=" ";
s1[15][3]=" kg/m^3";
s1[16][3]=" m/s";
}

}  //end of "turkish"
else{
if(!mole)
{
s1[0][1]="P, pressure       ";
s1[1][1]="T, temperature     ";
s1[2][1]="v, specific volume ";
s1[3][1]="h, enthalpy       ";
s1[4][1]="u, internal energy ";
s1[5][1]="s, entropy        ";
s1[6][1]="x, quality        ";
s1[7][1]="Cp, specific heat at constant pressure ";
```

```
   s1[8][1]="Cv, specific heat at constant volume ";
   s1[9][1]=""+\u03B1'+" isobaric thermal expansion coefficient ";
   s1[10][1]=""+\u03BA'+" isothermal compressibility" ;
   s1[11][1]=""+\u03B7'+" Dynamic viscosity" ;
   s1[12][1]="k Thermal conductivity" ;
   s1[13][1]=""+\u03C3'+" Surface tension" ;
   s1[14][1]="Prandtl number" ;
   s1[15][1]=""+\u03C1'+" density" ;
   s1[16][1]=" speed of sound" ;
   s1[0][3]=" kPA          ";
   s1[1][3]=" deg K        ";
   s1[2][3]=" m"+\u00B3'+"/kg        ";
   s1[3][3]=" KJ/kg        ";
   s1[4][3]=" KJ/kg        ";
   s1[5][3]=" KJ/kgK        ";
   s1[6][3]=" kg vap/kg mix ";
   s1[7][3]=" KJ/kgK";
   s1[8][3]=" KJ/kgK";
   s1[9][3]=" 1/K";
   s1[10][3]=" kPa";
   s1[11][3]=" Pa.s";
   s1[12][3]=" W/mK";
   s1[13][3]=" N/m";
   s1[14][3]=" ";
   s1[15][3]=" kg/m^3";
   s1[16][3]=" m/s";
   }
   else
   {   s1[0][1]="P, pressure        ";
   s1[1][1]="T, temperature     ";
   s1[2][1]="v, specific volume ";
   s1[3][1]="h, enthalpy        ";
   s1[4][1]="u, internal energy ";
   s1[5][1]="s, entropy         ";
   s1[6][1]="x, quality         ";
   s1[7][1]="Cp, specific heat at constant pressure ";
   s1[8][1]="Cv, specific heat at constant volume ";
   s1[9][1]=""+\u03B1'+" isobaric thermal expansion coefficient ";
   s1[10][1]=""+\u03BA'+" isothermal compressibility" ;
   s1[11][1]=""+\u03B7'+" Dynamic viscosity" ;
   s1[12][1]="k Thermal conductivity" ;
   s1[13][1]=""+\u03C3'+" Surface tension" ;
   s1[14][1]="Prandtl number" ;
   s1[15][1]=""+\u03C1'+" density" ;
   s1[16][1]=" speed of sound" ;
   s1[0][3]=" kPA          ";
   s1[1][3]=" deg K        ";
   s1[2][3]=" m"+\u00B3'+"/kmol        ";
   s1[3][3]=" KJ/kmol         ";
   s1[4][3]=" KJ/kmol         ";
   s1[5][3]=" KJ/kmolK         ";
   s1[6][3]=" kmol vap/kmol mix ";
   s1[7][3]=" KJ/kmolK";
   s1[8][3]=" KJ/kmolK";
   s1[9][3]=" 1/K";
   s1[10][3]=" kPa";
   s1[11][3]=" Pa.s";
   s1[12][3]=" W/mK";
   s1[13][3]=" N/m";
   s1[14][3]=" ";
   s1[15][3]=" kmol/m^3";
   s1[16][3]=" m/s";
   }
  }
double pp[]=property(s,v1,v2);
for(int i=0;i<17;i++)
   {s1[i][2]=""+pp[i];}
    //s1[6][2]=phase(pp[6]);
for(int i=0;i<17;i++)
{s1[i][0]=""+i;}
return s1;
}

public String[][] toStringC(String s, double v1, double v2)
{  String s1[][]=new String[17][4];
   if(option.equals("Turkish"))
```

```
{
if(mole)
{s1[0][1]="P, basınç        ";
s1[1][1]="T, sıcaklık     ";
s1[2][1]="v, özgül hacim ";
s1[3][1]="h, enthalpi        ";
s1[4][1]="u, iç enerji ";
s1[5][1]="s, entropi         ";
s1[6][1]="x, doygunluk derecesi         ";
s1[7][1]="Cp, sabit basınçta özgül ısı ";
s1[8][1]="Cv, sabit hacimde özgül ısı";
s1[9][1]=""+'\u03B1'+" sabit basınçta ısıl genleşme katsayısı";
s1[10][1]=""+'\u03BA'+" izotermal sıkıştırılabilirlik" ;
s1[11][1]=""+'\u03B7'+" Dinamik vizkozite" ;
s1[12][1]="k ısıl iletim sabiti" ;
s1[13][1]=""+'\u03C3'+" yüzey gerilimi" ;
s1[14][1]="Prandtl sayısı" ;
s1[15][1]=""+'\u03C1'+" yoğunluk" ;
s1[16][1]=" ses hızı" ;
s1[0][3]=" kPA          ";
s1[1][3]=" deg K         ";
s1[2][3]=" m"+'\u00B3'+"/kmol        ";
s1[3][3]=" KJ/kmol         ";
s1[4][3]=" KJ/kmol         ";
s1[5][3]=" KJ/kmolK         ";
s1[6][3]=" kmol buhar/kmol karışım ";
s1[7][3]=" KJ/kmolK";
s1[8][3]=" KJ/kmolK";
s1[9][3]=" 1/K";
s1[10][3]=" kPa";
s1[11][3]=" Pa.s";
s1[12][3]=" W/mK";
s1[13][3]=" N/m";
s1[14][3]=" ";
s1[15][3]=" kmol/m^3";
s1[16][3]=" m/s";
}
else
{
s1[0][1]="P, basınç        ";
s1[1][1]="T, sıcaklık     ";
s1[2][1]="v, özgül hacim ";
s1[3][1]="h, enthalpi        ";
s1[4][1]="u, iç enerji ";
s1[5][1]="s, entropi         ";
s1[6][1]="x, doygunluk derecesi         ";
s1[7][1]="Cp, sabit basınçta özgül ısı ";
s1[8][1]="Cv, sabit hacimde özgül ısı";
s1[9][1]=""+'\u03B1'+" sabit basınçta ısıl genleşme katsayısı";
s1[10][1]=""+'\u03BA'+" izotermal sıkıştırılabilirlik" ;
s1[11][1]=""+'\u03B7'+" Dinamik vizkozite" ;
s1[12][1]="k ısıl iletim sabiti" ;
s1[13][1]=""+'\u03C3'+" yüzey gerilimi" ;
s1[14][1]="Prandtl sayısı" ;
s1[15][1]=""+'\u03C1'+" yoğunluk" ;
s1[16][1]=" ses hızı" ;
s1[0][3]=" kPA          ";
s1[1][3]=" deg K         ";
s1[2][3]=" m"+'\u00B3'+"/kg        ";
s1[3][3]=" KJ/kg         ";
s1[4][3]=" KJ/kg         ";
s1[5][3]=" KJ/kgK         ";
s1[6][3]=" kg buhar/kg karışım ";
s1[7][3]=" KJ/kgK";
s1[8][3]=" KJ/kgK";
s1[9][3]=" 1/K";
s1[10][3]=" kPa";
s1[11][3]=" Pa.s";
s1[12][3]=" W/mK";
s1[13][3]=" N/m";
s1[14][3]=" ";
s1[15][3]=" kg/m^3";
s1[16][3]=" m/s";
}

}  //end of "turkish"
```

```
else{
if(!mole)
{
s1[0][1]="P, pressure        ";
s1[1][1]="T, temperature     ";
s1[2][1]="v, specific volume ";
s1[3][1]="h, enthalpy        ";
s1[4][1]="u, internal energy ";
s1[5][1]="s, entropy         ";
s1[6][1]="x, quality         ";
s1[7][1]="Cp, specific heat at constant pressure ";
s1[8][1]="Cv, specific heat at constant volume ";
s1[9][1]=""+'\u03B1'+" isobaric thermal expansion coefficient ";
s1[10][1]=""+'\u03BA'+" isothermal compressibility" ;
s1[11][1]=""+'\u03B7'+" Dynamic viscosity" ;
s1[12][1]="k Thermal conductivity" ;
s1[13][1]=""+'\u03C3'+" Surface tension" ;
s1[14][1]="Prandtl number" ;
s1[15][1]=""+'\u03C1'+" density" ;
s1[16][1]=" speed of sound" ;
s1[0][3]=" kPA          ";
s1[1][3]=" deg K        ";
s1[2][3]=" m"+'\u00B3'+"/kg        ";
s1[3][3]=" KJ/kg        ";
s1[4][3]=" KJ/kg        ";
s1[5][3]=" KJ/kgK       ";
s1[6][3]=" kg vap/kg mix ";
s1[7][3]=" KJ/kgK";
s1[8][3]=" KJ/kgK";
s1[9][3]=" 1/K";
s1[10][3]=" kPa";
s1[11][3]=" Pa.s";
s1[12][3]=" W/mK";
s1[13][3]=" N/m";
s1[14][3]=" ";
s1[15][3]=" kg/m^3";
s1[16][3]=" m/s";
}
else
{   s1[0][1]="P, pressure        ";
s1[1][1]="T, temperature     ";
s1[2][1]="v, specific volume ";
s1[3][1]="h, enthalpy        ";
s1[4][1]="u, internal energy ";
s1[5][1]="s, entropy         ";
s1[6][1]="x, quality         ";
s1[7][1]="Cp, specific heat at constant pressure ";
s1[8][1]="Cv, specific heat at constant volume ";
s1[9][1]=""+'\u03B1'+" isobaric thermal expansion coefficient ";
s1[10][1]=""+'\u03BA'+" isothermal compressibility" ;
s1[11][1]=""+'\u03B7'+" Dynamic viscosity" ;
s1[12][1]="k Thermal conductivity" ;
s1[13][1]=""+'\u03C3'+" Surface tension" ;
s1[14][1]="Prandtl number" ;
s1[15][1]=""+'\u03C1'+" density" ;
s1[16][1]=" speed of sound" ;
s1[0][3]=" kPA          ";
s1[1][3]=" deg K        ";
s1[2][3]=" m"+'\u00B3'+"/kmol        ";
s1[3][3]=" KJ/kmol        ";
s1[4][3]=" KJ/kmol        ";
s1[5][3]=" KJ/kmolK       ";
s1[6][3]=" kmol vap/kmol mix ";
s1[7][3]=" KJ/kmolK";
s1[8][3]=" KJ/kmolK";
s1[9][3]=" 1/K";
s1[10][3]=" kPa";
s1[11][3]=" Pa.s";
s1[12][3]=" W/mK";
s1[13][3]=" N/m";
s1[14][3]=" ";
s1[15][3]=" kmol/m^3";
s1[16][3]=" m/s";
}
}
double pp[]=propertyC(s,v1,v2);
```

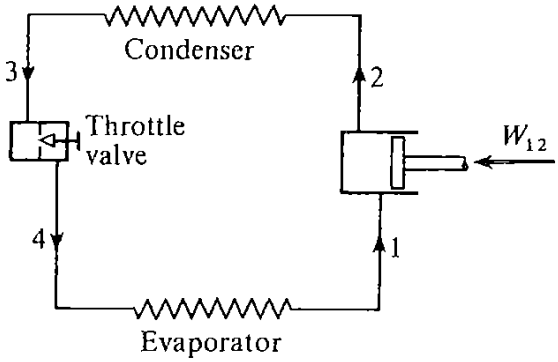
```
for(int i=0;i<17;i++)
   {s1[i][2]=""+pp[i];}
    //s1[6][2]=phase(pp[6]);
for(int i=0;i<17;i++)
{s1[i][0]=""+i;}
return s1;
}
//================
public static void main(String arg[])
{
 air_NIST air=new air_NIST();
    double T=300;
    double P=100.0;
    double ro=air.ro(T,P);
    double u=air.u_Tro1(T,ro);
    double h=air.h_Tro1(T,ro);
    double s=air.s_Tro1(T,ro);
    double Cp=air.Cp_Tro(T,ro);
    double Cv=air.Cv_Tro(T,ro);
    double M=28.965383561548776;
    double w=air.w_Tro(T,ro);
    System.out.println("T="+T+"P="+P+"ro="+ro+"u="+u+"h="+h+"s="+s+"Cv="+Cv+"Cp="+Cp+"w="+w);
    double ro1=ro*M;
    double u1=air.u_Tro(T,ro);
    double h1=air.h_Tro(T,ro);
    double s1=air.s_Tro(T,ro);
    double Cp1=air.Cp_Tro(T,ro)/M;
    double Cv1=air.Cv_Tro(T,ro)/M;
     System.out.println("T="+T+"P="+P+"ro="+ro1+"u1="+u1+"h1="+h1+"s1="+s1+"Cv="+Cv1+"Cp="+Cp1+"w="+w);
    double ro2=ro*air.M;
    double u2=air.u_Tro(T,ro)/air.M;
    double h2=air.h_Tro(T,ro)/air.M;
    double s2=air.s_Tro(T,ro)/air.M;
    double Cp2=air.Cp_Tro(T,ro)/air.M;
    double Cv2=air.Cv_Tro(T,ro)/air.M;
    System.out.println("T="+T+"P="+P+"ro="+ro2+"u2="+u2+"h2="+h2+"s2="+s2+"Cv2="+Cv2+"Cp2="+Cp2+"w="+w);
    //double a[]=air.property("tp",T,P);
    //String a1[][]=air.toString("tp",T,P);
    //Text.print(a1,"NIST EOS for air");
    double a1[]=air.Trom_Phm(P,h2);
    System.out.println("ro="+a1[0]+"T="+a1[1]);
    double a2[]=air.Trom_Psm(P,s2);
    System.out.println("T="+a1[0]+"ro="+a1[1]);
    System.out.println("T="+a2[0]+"ro="+a2[1]);
    Text.print(air.toString("tp",T,P),"mass");
    air.setMole(true);
     Text.print(air.toString("tp",T,P),"mole");
}
}
```

```
 ---------- Capture Output ----------
> "E:\co\java\bin\javaw.exe" brayton3_NISTtest
M=28.965383561548776
 Brayton cycle
Turbine power output W turbine = 10.061254307604202 kW
Compressor power input W kompresor = 15.959022333105706 kW
Net power output  Wnet = 5.897768025501504 kW
Heat exchanger heat removal Q  = 8.887159773972606 kW
refrigeration heat removal Qref = 2.9893917484711046 kW
COP refrigeration  = 1.5068683162079615
COP refrigeration  = 1.5068683162079615
Compressor inlet temperature = 26.850000000000023 degree C
Compressor inlet pressure  = 101.325 kPa
Compressor exit temperature = 154.41813556214487 degree C
Compressor exit  pressure   = 300.0 kPa
Turbine inlet temperature = 83.85000000000002 degree C
Turbine inlet pressure   = 300.0 kPA
Turbine exit temperature = 2.904547905522122 degree C
Turbine exit pressure   = 101.325 kPa

> Terminated with exit code 1.
```

4.2 STANDARD REFRIGERATION CYCLE

Carnot cycle is not a cycle to realise as a real engine. Using an turbine or expansion engine is expensive, instead of these, using of a simple throttle valve will be simplify the process. Of course COP value will be less due to this change, but the cost is also more affordable.



Now, if this change is adjusted to carnot cycle given in the previously

$W_{isentropiccompressor} = m(h_2 - h_1)$ $s_2 = s_1$ (4.2.1)

$Q_{condenser} = m(h_2 - h_3) = mT_2(s_2 - s_3) = mT_2(s_1 - s_4)$ (4.2.2)

Throttle valve: $h_3 = h_4$ (4.2.3)

$Q_{evaporator} = m(h_1 - h_4) = mT_1(s_1 - s_4)$ (4.2.4)

$W_{netin} = W_{isentropiccompressor} = Q_{condenser} - Q_{evaporator}$ (4.2.5)

$Coefficient of Performance for evaporator = \frac{Q_{evaporator}}{W_{compressor}}$ (4.2.6)

$Coefficient of Performance for condenser = \frac{Q_{condenser}}{W_{compressor}}$ (4.2.7)

Program 4.2.1 refrigeration cycle (turbine in carnot cycle is replaced with throttling valve)

```
public class ref_cycle1
{
// a R134 ideal carnot refrigeration cycle
// refrigerantEN b=new refrigerantEN("R134a");
// double a[]=b.property("tp",30.0,1.01325);
///---- String s values and meanings --------
//    tv=temperature-specific volume
//    tp=temperature-pressure
//    th=temperature-enthalpy
//    tu=temperature-internal energy
//    ts=temperature-entropy
//    tx=temperature quality
//    pv=pressure-specific volume
//    pt=pressure-temperature
//    ph=pressure-enthapy
//    pu=pressure-internal energy
//    ps=pressure-entropy
//    px=pressure-quality
//    vp=specific volume-pressure
//    vt=specific volume-temperature
//---- output values (method property) ---------
// r[0] P Pressure kPa
// r[1] t temperature degree C
// r[2] v specific volume m^3/kg
// r[3] h enthapy KJ/kg
// r[4] u internal energy KJ/kg
// r[5] s entropy KJ/kgK
// r[6] x quality kg vapour/kg total
// r[7] ro density kg/m^3
```

```
double T[];
double P[];
double h[];
double s[];
double x[];
double COP_condenser,COP_evaporator,COP_evap_carnot,COP_condenser_carnot;
double m;//kg/s
double Wp,W;//work
double Qevaporator,Qcondenser;
refrigerantEN b;
double c1[][];
double c3[][];
public ref_cycle1(double mi,double P1, double P2)
{
m=mi;
b=new refrigerantEN("R134a");
T=new double[9];
P=new double[9];
h=new double[9];
s=new double[9];
x=new double[9];

P[1]=P1;//compressor inlet
P[2]=P2;//compressor outlet
c1=new double[2][20];
c3=new double[2][20];
}
public double[][] line(double P1,double P2,double hi,int n)
{
double TT1[]=new double[n];
double ss1[]=new double[n];
double aa[];
double p;
for(int i=0;i<n;i++)
{p=P1+i*(P2-P1)/(n-1);
 aa=b.property("ph",p,hi);
 TT1[i]=aa[1];
 ss1[i]=aa[5];
}
double a[][]={TT1,ss1};
return a;
}

public void cycle()
{
// isentropic compressor output
double a1[]=b.property("px",P[2],1.0);
h[2]=a1[3];
s[2]=a1[5];
T[2]=a1[1];
// isentropik compressor input
s[1]=s[2];
a1=b.property("ps",P[1],s[1]);
T[1]=a1[1];
h[1]=a1[3];
x[1]=a1[6];
```

```
//expansion valve input-condenser output
P[3]=P[2];
a1=b.property("px",P[3],0.0);
h[3]=a1[3];
s[3]=a1[5];
T[3]=a1[1];
//expansion valve output
P[4]=P[1];
h[4]=h[3];
a1=b.property("ph",P[4],h[4]);
s[4]=a1[5];
T[4]=a1[1];
x[4]=a1[6];


// isentropik compressor
Wp=m*(h[2]-h[1]);
W=Wp;
Qevaporator=m*(h[1]-h[4]);
Qcondenser=m*(h[2]-h[3]);
COP_evaporator=Qevaporator/W;
COP_condenser_carnot=(T[2]+273.15)/(T[2]-T[1]);
COP_condenser=Qcondenser/W;
COP_evap_carnot=(T[1]+273.15)/(T[2]-T[1]);
}

public String toString()
{
cycle();
String ss="R134a ideal refrigeration cycle 1\n";
ss+=" compressor work = "+Wp+" kW\n";
ss+=" net work input= "+W+" kW\n";
ss+=" condenser heat output = "+Qcondenser+" kW\n";
ss+=" evaporator heat input = "+Qevaporator+" kW\n";
ss+=" COP evaporator    = " + COP_evaporator+" kW\n";
ss+=" COP condenser     = " + COP_condenser+" kW\n";
ss+=" COP evaporator  carnot = " + COP_evap_carnot+" kW\n";
ss+=" COP condenser   carnot = " + COP_condenser_carnot+" kW\n";
ss+=" h1 compressor input = "+h[1] +" kJ/kg\n";
ss+=" T1 compressor input = "+T[1] +" derece C\n";
ss+=" P1 compressor input = "+P[1] +" kPa \n";
ss+=" s1 compressor input = "+s[1] +" kJ/kgK\n";
ss+=" x1 compressor input = "+x[1] +" kgvapor/kgtotal"+"\n";
ss+=" T2 compressor output = "+T[3] +" derece C\n";
ss+=" P2 compressor output = "+P[3] +" kPa \n";
ss+=" s2 compressor output = "+s[2] +" kJ/kgK\n";
ss+=" x2 compressor output = "+x[2] +" kgvapor/kgtotal"+"\n";
ss+=" h3 turbine input = "+h[3] +" kJ/kg\n";
ss+=" T3 turbine input = "+T[3] +" derece C\n";
ss+=" P3 turbine input = "+P[3] +" kPa \n";
ss+=" s3 turbine input = "+s[3] +" kJ/kgK\n";
ss+=" x3 turbine input = "+x[3] +" kgvapor/kgtotal"+"\n";
ss+=" h4 turbine output = "+h[4] +" kJ/kg\n";
ss+=" T4 turbine output = "+T[4] +" derece C\n";
ss+=" s4 turbine output = "+s[4] +" kJ/kgK\n";
ss+=" x4 turbine output = "+x[4] +" kgvapor/kgtotal"+"\n";
return ss;
}
```

```
public double[][] TS()
{
double a[][]=new double[2][421];
double tc=b.r.Tc;
System.out.println("tc="+tc);
double dt=(tc-10.0)/100.0;
int i=0;
double t;
for(t=-90.0;t<=tc;t+=dt)
{double a1[]=b.property("tx",t,0.0);
 a[1][i]=t;
 a[0][i]=a1[5];
 i++;
}
double a3[]=b.property("tx",tc,1.0);
 a[1][i]=t;
 a[0][i]=a3[5];
 i++;
for(t=tc;t>=-90;t-=dt)
{double a2[]=b.property("tx",t,1.0);
 a[1][i]=t;
 a[0][i]=a2[5];
 i++;
}
return a;
}

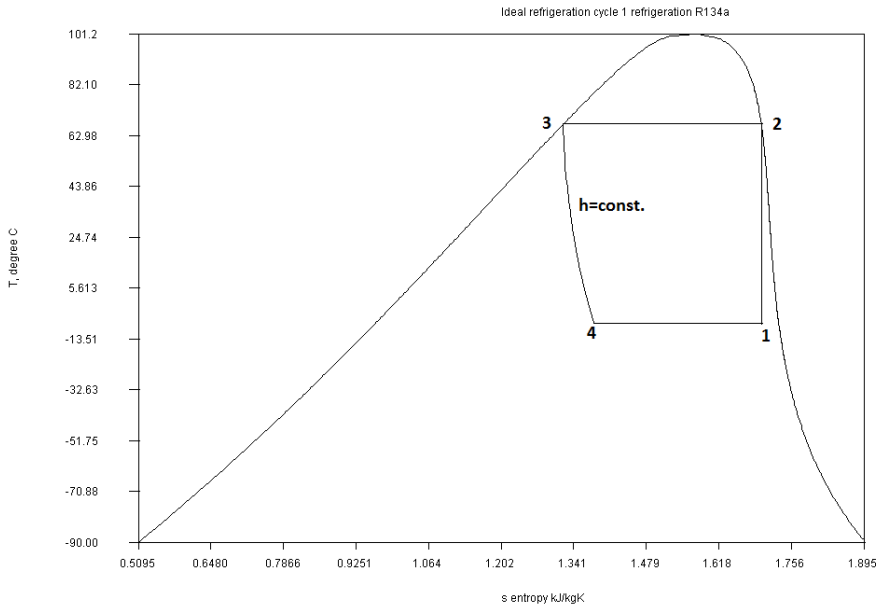
public void plot()
{
 Plot pp=new Plot(TS());
 double a[][]=line(P[3],P[4],h[3],15);
 double tt[]=a[0];
 double ss[]=a[1];
 System.out.println("n="+tt.length);
 double t1[]=new double[19];
 double s1[]=new double[19];
 t1[0]=T[1];t1[1]=T[2];t1[2]=T[3];
 s1[0]=s[1];s1[1]=s[2];s1[2]=s[3];
 for(int i=0;i<tt.length;i++)
 {t1[i+3]=tt[i];s1[i+3]=ss[i];}
 t1[18]=T[1];
 s1[18]=s[1];
 pp.setPlabel("Ideal refrigeration cycle 1 refrigeration R134a ");
 pp.setYlabel("T, degree C");
 pp.setXlabel("s entropy kJ/kgK");
 pp.addData(s1,t1);
 pp.plot();
}
}
```

Program 4.2.2 refrigeration cycle (turbine in carnot cycle is replaced with throttling valve) test program

```
public class ref_cycle1test
{
public static void main(String arg[])
{
double m=50/745.25928;  // kg/s çevrim R134a debisi
```

```
double P2=2000.0;         //  bar turbin giriş basıncı
double P1=220.0 ;         //  bar turbin çıkış basıncı
ref_cycle1 r=new ref_cycle1(m,P1,P2);
System.out.println(r.toString());
r.plot();
}
}
```

```
 ---------- Capture Output ----------
> "C:\java\bin\javaw.exe" ref_cycle1test
R134a ideal refrigeration cycle 1
 compressor work = 2.9194665890306633 kW
 net work input= 2.9194665890306633 kW
 condenser heat output = 8.611038292271756 kW
 evaporator heat input = 5.691571703241092 kW
 COP evaporator    = 1.9495245208923035 kW
 COP condenser     = 2.9495245208923038 kW
 COP evaporator  carnot = 3.5351212616738557 kW
 COP condenser   carnot = 4.535121261673855 kW
 h1 compressor input = 385.2301064454953 kJ/kg
 T1 compressor input = -7.63976989535311 derece C
 P1 compressor input = 220.0 kPa
 s1 compressor input = 1.6991352552692158 kJ/kgK
 x1 compressor input = 0.9566957691465375 kgvapor/kgtotal
 T2 compressor output = 67.46662970207672 derece C
 P2 compressor output = 2000.0 kPa
 s2 compressor output = 1.6991352552692158 kJ/kgK
 x2 compressor output = 0.0 kgvapor/kgtotal
 h3 turbine input = 300.39617385297873 kJ/kg
 T3 turbine input = 67.46662970207672 derece C
 P3 turbine input = 2000.0 kPa
 s3 turbine input = 1.3222852352261514 kJ/kgK
 x3 turbine input = 0.0 kgvapor/kgtotal
 h4 turbine output = 300.39617385297873 kJ/kg
 T4 turbine output = -7.63976989535311 derece C
 s4 turbine output = 1.3788234773837613 kJ/kgK
 x4 turbine output = 0.540763706713116 kgvapor/kgtotal
```



Compressor inlet point of the previous cycle is not very prefferable as well. It will be better if the compressor inlet is on saturation line or in superheated region.

Program 4.2.3 ideal refrigeration cycle

```
 public class ref_cycle2
{
// a R134 ideal refrigeration  cycle
// refrigerantEN b=new refrigerantEN("R134a");
// double a[]=b.property("tp",30.0,1.01325);
///---- String s values and meanings --------
//    tv=temperature-specific volume
//    tp=temperature-pressure
//    th=temperature-enthalpy
//    tu=temperature-internal energy
//    ts=temperature-entropy
//    tx=temperature quality
//    pv=pressure-specific volume
//    pt=pressure-temperature
//    ph=pressure-enthapy
//    pu=pressure-internal energy
//    ps=pressure-entropy
//    px=pressure-quality
//    vp=specific volume-pressure
//    vt=specific volume-temperature
//---- output values (method property) ---------
// r[0] P Pressure kPa
// r[1] t temperature degree C
// r[2] v specific volume m^3/kg
// r[3] h enthapy KJ/kg
// r[4] u internal energy KJ/kg
// r[5] s entropy KJ/kgK
// r[6] x quality kg vapour/kg total
// r[7] ro density kg/m^3

double T[];
double P[];
double h[];
double s[];
double x[];
double COP_condenser,COP_evaporator,COP_evap_carnot,COP_condenser_carnot;
double m;//kg/s
double Wp,W;//work
double Qevaporator,Qcondenser;
refrigerantEN b;
double c1[][];
double c3[][];
public ref_cycle2(double mi,double P1, double P2)
{
m=mi;
b=new refrigerantEN("R134a");
T=new double[9];
P=new double[9];
h=new double[9];
s=new double[9];
x=new double[9];

P[1]=P1;//compressor inlet
P[2]=P2;//compressor outlet
c1=new double[2][20];
```

```
c3=new double[2][20];
}
public double[][] line(double P1,double P2,double hi,int n)
{
double TT1[]=new double[n];
double ss1[]=new double[n];
double aa[];
double p;
for(int i=0;i<n;i++)
{p=P1+i*(P2-P1)/(n-1);
 aa=b.property("ph",p,hi);
 TT1[i]=aa[1];
 ss1[i]=aa[5];
}
double a[][]={TT1,ss1};
return a;
}
public double[][] line1(double T1,double T2,double Pi,int n)
{
double TT1[]=new double[n];
double ss1[]=new double[n];
double aa[];
for(int i=0;i<n;i++)
{TT1[i]=T1+i*(T2-T1)/(n-1);
 aa=b.property("tp",TT1[i],Pi);
 ss1[i]=aa[5];
}
double a[][]={TT1,ss1};
return a;
}

public void cycle()
{

// isentropik compressor input
double a1[]=b.property("px",P[1],1.0);
T[1]=a1[1];
h[1]=a1[3];
s[1]=a1[5];
x[1]=a1[6];
// isentropic compressor output
s[2]=s[1];
a1=b.property("ps",P[2],s[2]);
h[2]=a1[3];
T[2]=a1[1];
//saturation point
P[5]=P[2];
a1=b.property("px",P[5],1.0);
h[5]=a1[3];
T[5]=a1[1];
s[5]=a1[5];
//expansion valve input-condenser output
P[3]=P[2];
a1=b.property("px",P[3],0.0);
h[3]=a1[3];
s[3]=a1[5];
T[3]=a1[1];
//expansion valve output
```

```
P[4]=P[1];
h[4]=h[3];
a1=b.property("ph",P[4],h[4]);
s[4]=a1[5];
T[4]=a1[1];
x[4]=a1[6];


// isentropik compressor
Wp=m*(h[2]-h[1]);
W=Wp;
Qevaporator=m*(h[1]-h[4]);
Qcondenser=m*(h[2]-h[3]);
COP_evaporator=Qevaporator/W;
COP_condenser_carnot=(T[2]+273.15)/(T[2]-T[1]);
COP_condenser=Qcondenser/W;
COP_evap_carnot=(T[1]+273.15)/(T[2]-T[1]);
}

public String toString()
{
cycle();
String ss="R134a ideal refrigeration cycle 2\n";
ss+=" compressor work = "+Wp+" kW\n";
ss+=" net work input= "+W+" kW\n";
ss+=" condenser heat output = "+Qcondenser+" kW\n";
ss+=" evaporator heat input = "+Qevaporator+" kW\n";
ss+=" COP evaporator    = " + COP_evaporator+" kW\n";
ss+=" COP condenser     = " + COP_condenser+" kW\n";
ss+=" COP evaporator  carnot = " + COP_evap_carnot+" kW\n";
ss+=" COP condenser   carnot = " + COP_condenser_carnot+" kW\n";
ss+=" h1 compressor input = "+h[1] +" kJ/kg\n";
ss+=" T1 compressor input = "+T[1] +" derece C\n";
ss+=" P1 compressor input = "+P[1] +" kPa \n";
ss+=" s1 compressor input = "+s[1] +" kJ/kgK\n";
ss+=" x1 compressor input = "+x[1] +" kgvapor/kgtotal"+"\n";
ss+=" T2 compressor output = "+T[2] +" derece C\n";
ss+=" P2 compressor output = "+P[2] +" kPa \n";
ss+=" s2 compressor output = "+s[2] +" kJ/kgK\n";
ss+=" x2 compressor output = "+x[2] +" kgvapor/kgtotal"+"\n";
ss+=" T5 condenser saturation = "+T[5] +" derece C\n";
ss+=" P5 condenser saturation = "+P[5] +" kPa \n";
ss+=" s5 condenser saturation = "+s[5] +" kJ/kgK\n";
ss+=" x5 condenser saturation = "+x[5] +" kgvapor/kgtotal"+"\n";

ss+=" h3 turbine input = "+h[3] +" kJ/kg\n";
ss+=" T3 turbine input = "+T[3] +" derece C\n";
ss+=" P3 turbine input = "+P[3] +" kPa \n";
ss+=" s3 turbine input = "+s[3] +" kJ/kgK\n";
ss+=" x3 turbine input = "+x[3] +" kgvapor/kgtotal"+"\n";
ss+=" h4 turbine output = "+h[4] +" kJ/kg\n";
ss+=" T4 turbine output = "+T[4] +" derece C\n";
ss+=" s4 turbine output = "+s[4] +" kJ/kgK\n";
ss+=" x4 turbine output = "+x[4] +" kgvapor/kgtotal"+"\n";
return ss;
}

public double[][] TS()
```

```
{
double a[][]=new double[2][421];
double tc=b.r.Tc;
System.out.println("tc="+tc);
double dt=(tc-10.0)/100.0;
int i=0;
double t;
for(t=-90.0;t<=tc;t+=dt)
{double a1[]=b.property("tx",t,0.0);
 a[1][i]=t;
 a[0][i]=a1[5];
 i++;
}
double a3[]=b.property("tx",tc,1.0);
 a[1][i]=t;
 a[0][i]=a3[5];
 i++;
for(t=tc;t>=-90;t-=dt)
{double a2[]=b.property("tx",t,1.0);
 a[1][i]=t;
 a[0][i]=a2[5];
 i++;
}
return a;
}

public void plot()
{
 Plot pp=new Plot(TS());
 double a[][]=line(P[3],P[4],h[3],15);
 double tt[]=a[0];
 double ss[]=a[1];
 System.out.println("n="+tt.length);
 double t1[]=new double[20];
 double s1[]=new double[20];
 t1[0]=T[1];t1[1]=T[2];t1[2]=T[5];t1[3]=T[3];
 s1[0]=s[1];s1[1]=s[2];s1[2]=s[5];s1[3]=s[3];
 for(int i=0;i<tt.length;i++)
 {t1[i+4]=tt[i];s1[i+4]=ss[i];}
 t1[19]=T[1];
 s1[19]=s[1];
 pp.setPlabel("Ideal refrigeration cycle 1 refrigeration R134a ");
 pp.setYlabel("T, degree C");
 pp.setXlabel("s entropy kJ/kgK");
 pp.addData(s1,t1);
 pp.plot();
}
}
```

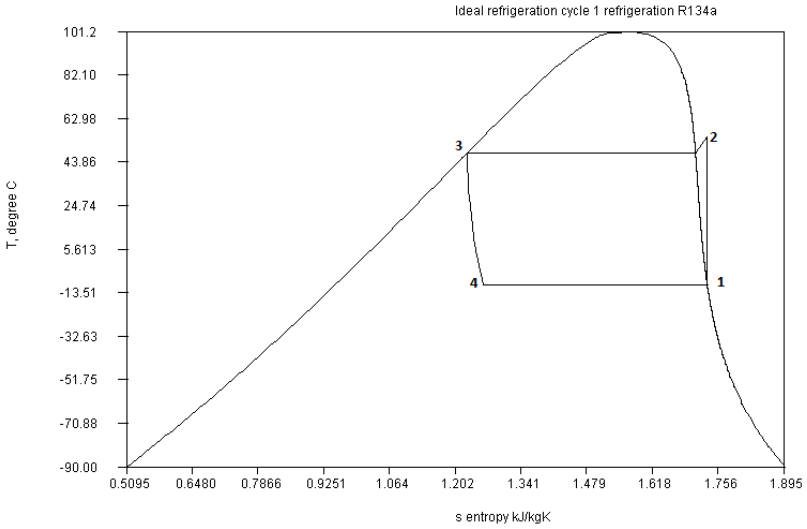
Program 4.2.4 ideal refrigeration cycle test program

```
import javax.swing.*;

public class ref_cycle2test
{
public static void main(String arg[])
{
double m=50/745.25928;  // kg/s çevrim R134a debisi
double P2=1250.0;         //  bar turbin giriş basıncı
```

```
double P1=200.0 ;         //  bar turbin çıkış basıncı
ref_cycle2 r=new ref_cycle2(m,P1,P2);
System.out.println(r.toString());
r.plot();
}}
```

```
---------- Capture Output ----------
> "C:\java\bin\javaw.exe" ref_cycle2test
R134a ideal refrigeration cycle 2
 compressor work = 2.5667447396341565 kW
 net work input= 2.5667447396341565 kW
 condenser heat output = 10.902434167277017 kW
 evaporator heat input = 8.33568942764286 kW
 COP evaporator    = 3.2475724207896746
 COP condenser     = 4.247572420789674
 COP evaporator  carnot = 4.040569717710718
 COP condenser   carnot = 5.040569717710718
 h1 compressor input = 392.85464093227904 kJ/kg
 T1 compressor input = -10.07576059284282 derece C
 P1 compressor input = 200.0 kPa
 s1 compressor input = 1.7341511989113898 kJ/kgK
 x1 compressor input = 1.0 kgvapor/kgtotal
 T2 compressor output = 55.032443889818154 derece C
 P2 compressor output = 1250.0 kPa
 s2 compressor output = 1.7341511989113898 kJ/kgK
 x2 compressor output = 0.0 kgvapor/kgtotal
 T5 condenser saturation = 47.87587075762405 derece C
 P5 condenser saturation = 1250.0 kPa
 s5 condenser saturation = 1.7093434574756143 kJ/kgK
 x5 condenser saturation = 0.0 kgvapor/kgtotal
 h3 turbine input = 268.60964290930445 kJ/kg
 T3 turbine input = 47.87587075762405 derece C
 P3 turbine input = 1250.0 kPa
 s3 turbine input = 1.2284066394805506 kJ/kgK
 x3 turbine input = 0.0 kgvapor/kgtotal
 h4 turbine output = 268.60964290930445 kJ/kg
 T4 turbine output = -10.07576059284282 derece C
 s4 turbine output = 1.2622418532709556 kJ/kgK
 x4 turbine output = 0.3976100345359028 kgvapor/kgtotal
```



Now a more realistic cycle can be investigated. In this cycle, compresor input will be superheated, and condenser output will be in liquid region. Pressure drop in condenser and evaporator will be taken into account. Throttling process in expansion valve will have a small entalphy change due to heat transfer to the surrounding.

Program 4.2.5 standard refrigeration cycle

```
 public class ref_cycle3
{
// a R134 ideal refrigeration  cycle
// refrigerantEN b=new refrigerantEN("R134a");
// double a[]=b.property("tp",30.0,1.01325);
///---- String s values and meanings --------
//    tv=temperature-specific volume
//    tp=temperature-pressure
//    th=temperature-enthalpy
//    tu=temperature-internal energy
//    ts=temperature-entropy
//    tx=temperature quality
//    pv=pressure-specific volume
//    pt=pressure-temperature
//    ph=pressure-enthapy
//    pu=pressure-internal energy
//    ps=pressure-entropy
//    px=pressure-quality
//    vp=specific volume-pressure
//    vt=specific volume-temperature
//---- output values (method property) ---------
// r[0] P Pressure kPa
// r[1] t temperature degree C
// r[2] v specific volume m^3/kg
// r[3] h enthapy KJ/kg
// r[4] u internal energy KJ/kg
// r[5] s entropy KJ/kgK
// r[6] x quality kg vapour/kg total
// r[7] ro density kg/m^3

double T[];
double P[];
double h[];
double s[];
double x[];
boolean supercritical=false;
String refname; //refrigerant name
double Tc,Pc;
double COP_condenser,COP_evaporator,COP_evap_carnot,COP_condenser_carnot;
double m;//kg/s mass flow rate of refrigerant
double Wp;//compressor work
double Wpi;//isentropic compressor work
double dPevap1;// evaporator pressure drop saturated region
double dPevap2;// evaporator pressure drop superheated region
double dPcond1;// condenser pressure drop superheated region
double dPcond2;// condenser pressure drop saturated region
double dPcond3;// condenser pressure drop liquid region
double dh;     // expansion valve enthalpy chenge (heat loss)
double dT1,dT6;
double eta_isent; // isentropic efficiency of compressor
double Qevaporator,Qcondenser; //condenser and evaporator heat transfer
refrigerantEN b; //refrigerantEN class object for properties
```

```
double c1[][];
double c3[][];
public ref_cycle3(String refnamei,double mi,double dT1i,double P1,double eta_isent1i, double
P2,double dT6i,double dPevap1i,double dPevap2i,double dPcond1i,double dPcond2i,double
dPcond3i,double dhi)
{
m=mi;
refname=refnamei;
dPevap1=dPevap1i;
dPevap2=dPevap2i;
dPcond1=dPcond1i;
dPcond2=dPcond2i;
dPcond3=dPcond3i;
dh=dhi;
dT1=dT1i;
dT6=dT6i;
eta_isent=eta_isent1i;
b=new refrigerantEN(refname);
T=new double[9];
P=new double[9];
h=new double[9];
s=new double[9];
x=new double[9];
Tc=b.r.Tc;//critical temperature degree C
Pc=b.r.Pc;//critical pressure kPa
P[1]=P1;//compressor inlet
P[2]=P2;//compressor outlet, evaporator superheated outlet
P[0]=P1-dPevap1; //evaporator saturation exit(x=1) pressure
P[4]=P[0]-dPevap2;//evaporator saturation inlet pressure
P[5]=P2-dPcond1;//condenser saturation inlet (x=1)
P[6]=P[5]-dPcond2;//condenser saturation exit (x=0)
P[3]=P[6]-dPcond3;//condenser  exit liquid state)
c1=new double[2][20];
c3=new double[2][20];
}

public void cycle()
{
// compressor saturation point 0
double a1[]=b.property("px",P[0],1.0);
T[0]=a1[1];
h[0]=a1[3];
s[0]=a1[5];
x[0]=a1[6];
// compressor inlet, evaporator exit point 1
T[1]=T[0]+dT1;
a1=b.property("tp",T[1],P[1]);
h[1]=a1[3];
s[1]=a1[5];
x[1]=a1[6];
// isentropic compressor exit point 7
s[7]=s[1];
P[7]=P[2];
a1=b.property("ps",P[7],s[7]);
h[7]=a1[3];
T[7]=a1[1];
//compressor exit point 2
h[2]=h[1]+(h[7]-h[1])/eta_isent;
```

```
a1=b.property("ph",P[2],h[2]);
T[2]=a1[1];
h[2]=a1[3];
s[2]=a1[5];
x[2]=a1[6];
// check if cycle is supercritical
if(P[3]>Pc)
{System.out.println("condenser is supercritical");
supercritical=true;
}
else supercritical=false;
//expansion valve input
if(!supercritical)
{
//saturation point
a1=b.property("px",P[5],1.0);
h[5]=a1[3];
T[5]=a1[1];
s[5]=a1[5];
a1=b.property("px",P[6],0.0);
h[6]=a1[3];
T[6]=a1[1];
s[6]=a1[5];
T[3]=T[6]-dT6;
a1=b.property("tx",T[3],0.0);
//a1=b.property("tp",T[3],P[3]);
}
else
{
T[3]=Tc-dT6;
a1=b.property("tp",T[3],P[3]);
}
h[3]=a1[3];
T[3]=a1[1];
s[3]=a1[5];
// point 4 evaporator inlet, expansion valve output
h[4]=h[3]+dh;
a1=b.property("ph",P[4],h[4]);
T[4]=a1[1];
s[4]=a1[5];
// isentropik compressor
Wpi=m*(h[7]-h[1]);
//compressor
Wp=m*(h[2]-h[1]);
Qevaporator=m*(h[1]-h[4]);
Qcondenser=m*(h[2]-h[3]);
COP_evaporator=Qevaporator/Wp;
COP_condenser_carnot=(T[2]+273.15)/(T[2]-T[1]);
COP_condenser=Qcondenser/Wp;
COP_evap_carnot=(T[1]+273.15)/(T[2]-T[1]);
}

public String toString()
{
cycle();
String ss="R134a ideal refrigeration cycle 2\n";
ss+=" compressor work = "+Wp+" kW\n";
ss+=" isentropic compressor work = "+Wpi+" kW\n";
```

```
ss+=" condenser heat output = "+Qcondenser+" kW\n";
ss+=" evaporator heat input = "+Qevaporator+" kW\n";
ss+=" COP evaporator    = " + COP_evaporator+" kW\n";
ss+=" COP condenser     = " + COP_condenser+" kW\n";
ss+=" COP evaporator  carnot = " + COP_evap_carnot+" kW\n";
ss+=" COP condenser   carnot = " + COP_condenser_carnot+" kW\n";
ss+=" h1 compressor input = "+h[1] +" kJ/kg\n";
ss+=" T1 compressor input = "+T[1] +" derece C\n";
ss+=" P1 compressor input = "+P[1] +" kPa \n";
ss+=" s1 compressor input = "+s[1] +" kJ/kgK\n";
ss+=" x1 compressor input = "+x[1] +" kgvapor/kgtotal"+"\n";
ss+=" h2 compressor input = "+h[2] +" kJ/kg\n";
ss+=" T2 compressor output = "+T[2] +" derece C\n";
ss+=" P2 compressor output = "+P[2] +" kPa \n";
ss+=" s2 compressor output = "+s[2] +" kJ/kgK\n";
ss+=" x2 compressor output = "+x[2] +" kgvapor/kgtotal"+"\n";
if(!supercritical)
{
ss+=" h5 compressor input = "+h[5] +" kJ/kg\n";
ss+=" T5 condenser saturation = "+T[5] +" derece C\n";
ss+=" P5 condenser saturation = "+P[5] +" kPa \n";
ss+=" s5 condenser saturation = "+s[5] +" kJ/kgK\n";
ss+=" x5 condenser saturation = "+x[5] +" kgvapor/kgtotal"+"\n";
ss+=" h6 compressor input = "+h[6] +" kJ/kg\n";
ss+=" T6 condenser saturation = "+T[6] +" derece C\n";
ss+=" P6 condenser saturation = "+P[6] +" kPa \n";
ss+=" s6 condenser saturation = "+s[6] +" kJ/kgK\n";
ss+=" x6 condenser saturation = "+x[6] +" kgvapor/kgtotal"+"\n";
}
ss+=" h3 turbine input = "+h[3] +" kJ/kg\n";
ss+=" T3 turbine input = "+T[3] +" derece C\n";
ss+=" P3 turbine input = "+P[3] +" kPa \n";
ss+=" s3 turbine input = "+s[3] +" kJ/kgK\n";
ss+=" x3 turbine input = "+x[3] +" kgvapor/kgtotal"+"\n";
ss+=" h4 turbine output = "+h[4] +" kJ/kg\n";
ss+=" T4 turbine output = "+T[4] +" derece C\n";
ss+=" s4 turbine output = "+s[4] +" kJ/kgK\n";
ss+=" x4 turbine output = "+x[4] +" kgvapor/kgtotal"+"\n";
return ss;
}

public double[][] TS(double ti)
{
double a[][]=new double[2][10000];
double tc=b.r.Tc;
double dt=(tc-1.0)/100.0;
int i=0;
double t;
for(t=ti;t<=tc;t+=dt)
{double a1[]=b.property("tx",t,0.0);
 a[1][i]=t;
 a[0][i]=a1[5];
 i++;
}
double a3[]=b.property("tx",tc,1.0);
 a[1][i]=t;
 a[0][i]=a3[5];
 i++;
```

```
for(t=tc;t>=ti;t-=dt)
{double a2[]=b.property("tx",t,1.0);
 a[1][i]=t;
 a[0][i]=a2[5];
 i++;
}
int n=i;
double b[][]=new double[2][n];
for(i=0;i<n;i++) {b[0][i]=a[0][i];b[1][i]=a[1][i];}
return b;
}
public double[][] line(double P1,double P2,double hi,double ho,int n)
{
double TT1[]=new double[n];
double ss1[]=new double[n];
double aa[];
double p,h;
for(int i=0;i<n;i++)
{p=P1+i*(P2-P1)/(n-1);
 h=hi+i*(ho-hi)/(n-1);
 aa=b.property("ph",p,h);
 TT1[i]=aa[1];
 ss1[i]=aa[5];
}
double a[][]={TT1,ss1};
return a;
}
public double[][] line1(double T1,double T2,double Pi,double Po,int n)
{
double TT1[]=new double[n];
double ss1[]=new double[n];
double aa[];
double P;
for(int i=0;i<n;i++)
{TT1[i]=T1+i*(T2-T1)/(n-1);
 P=Pi+i*(Po-Pi)/(n-1);
 aa=b.property("tp",TT1[i],P);
 ss1[i]=aa[5];
}
double a[][]={TT1,ss1};
return a;
}
public Plot plot1(Plot pp)
{double a[][]={{T[3],T[4]},{s[3],s[4]}};
 double tt[]=a[0];
 double ss[]=a[1];
 pp.addData(ss,tt);
 double b[][]=new double[2][15];
 double c[][];
 if(!supercritical)
 { c=line1(T[2],T[5],P[2],P[5],13);
   for(int i=0;i<c[0].length;i++)
   {b[0][i]=c[0][i];b[1][i]=c[1][i];}
    b[0][13]=T[6];b[1][13]=s[6];
    b[0][14]=T[3];b[1][14]=s[3];
 }
 else
 {c=line1(T[2],T[3],P[2],P[3],15);b=c;}
```

```
 tt=b[0];
 ss=b[1];
 pp.addData(ss,tt);
 double t1[]=new double[4];
 double s1[]=new double[4];
 t1[0]=T[4];t1[1]=T[0];t1[2]=T[1];t1[3]=T[2];
 s1[0]=s[4];s1[1]=s[0];s1[2]=s[1];s1[3]=s[2];
 pp.addData(s1,t1);
 double t2[]=new double[2];
 double s2[]=new double[2];
 t2[0]=T[1];t2[1]=T[7];
 s2[0]=s[1];s2[1]=s[7];
 pp.addData(s2,t2,1);

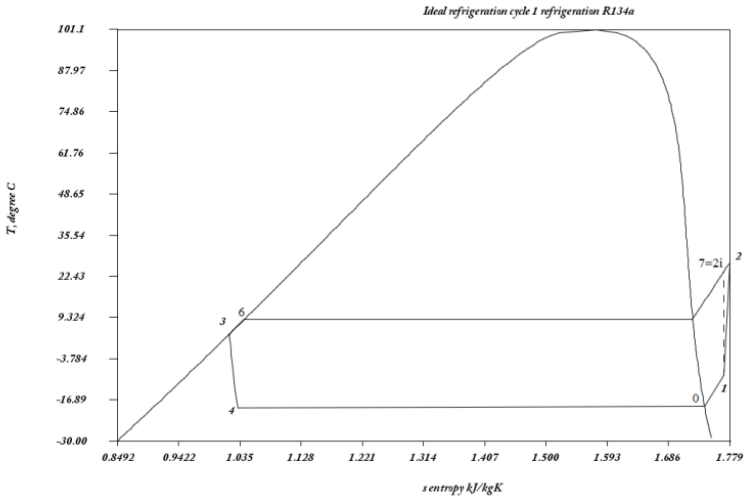
 return pp;
}
public void plot(double ti)
{   Plot pp=new Plot(TS(ti));
   pp= plot1(pp);
   pp.setPlabel("Ideal refrigeration cycle 1 refrigeration ");
   pp.setYlabel("T, degree C");
   pp.setXlabel("s entropy kJ/kgK");
            pp.plot();
}}
```

Program 4.2.5 standard refrigeration cycle test program

```
import javax.swing.*;

public class ref_cycle3test
{
public static void main(String arg[])
{
double m=0.05;  // kg/s çevrim R134a debisi
double P2=400.0;         //  kPa turbin giriş basıncı
double P1=140.0 ;         //  kPa turbin çıkış basıncı
double dT1i=10.0; //degree C
double eta_isent1i=0.9;
double dT6i=5.0; //degree C
double dPevap1i=2.0; //kPa
double dPevap2i=2.0; //kPa
double dPcond1i=2.0; //kPa
double dPcond2i=2.0; //kPa
double dPcond3i=2.0; //kPa
double dhi=2.0; //kJ/kgK
ref_cycle3 r=new
ref_cycle3("R22",m,dT1i,P1,eta_isent1i,P2,dT6i,dPevap1i,dPevap2i,dPcond1i,dPcond2i,dPc
ond3i,dhi);
System.out.println(r.toString());
r.plot(-50.0);
}
}
```

```
---------- Capture Output ----------
> "C:\java\bin\javaw.exe" ref_cycle3test
R134a ideal refrigeration cycle 2
 compressor work = 2.9946164677482345 kW
 isentropic compressor work = 2.695154820973411 kW
 condenser heat output = 12.412532218649785 kW
 evaporator heat input = 9.283734261742502 kW
 COP evaporator    = 3.100141324182089 kW
 COP condenser     = 4.144948894902471 kW
 COP evaporator  carnot = 3.990344898354031 kW
 COP condenser   carnot = 4.990344898354031 kW
 h1 compressor input = 400.58952866027346 kJ/kg
 T1 compressor input = -0.7139216789336409 derece C
 P1 compressor input = 200.0 kPa
 s1 compressor input = 1.7631762691950033 kJ/kgK
 x1 compressor input = 2.0 kgvapor/kgtotal
 h2 compressor input = 445.2248429128773 kJ/kg
 T2 compressor output = 67.55989556263937 derece C
 P2 compressor output = 1250.0 kPa
 s2 compressor output = 1.7763586400487221 kJ/kgK
 x2 compressor output = 1.1435914281628234 kgvapor/kgtotal
 h5 compressor input = 422.98465471180526 kJ/kg
 T5 condenser saturation = 47.71824296880384 derece C
 P5 condenser saturation = 1245.0 kPa
 s5 condenser saturation = 1.709402795937088 kJ/kgK
 x5 condenser saturation = 0.0 kgvapor/kgtotal
 h6 compressor input = 268.13803320986443 kJ/kg
 T6 condenser saturation = 47.56008650804041 derece C
 P6 condenser saturation = 1240.0 kPa
 s6 condenser saturation = 1.2268885030779424 kJ/kgK
 x6 condenser saturation = 0.0 kgvapor/kgtotal
 h3 turbine input = 260.2137464279225 kJ/kg
 T3 turbine input = 42.56008650804041 derece C
 P3 turbine input = 1235.0 kPa
 s3 turbine input = 1.201962034810739 kJ/kgK
 x3 turbine input = 0.0 kgvapor/kgtotal
 h4 turbine output = 262.2137464279225 kJ/kg
 T4 turbine output = -11.364829272039977 derece C
 s4 turbine output = 1.2393954627972106 kJ/kgK
 x4 turbine output = 0.0 kgvapor/kgtotal
```



Refrigeration cycles can be supercritical as well, this is specially the case for refrigerants such as R744(carbondioxide). Considering that as a natural refrigerant CO2 usage in refrigeration will be increased in the future.

```
public class refISO_cycle3
{
// a R134 ideal refrigeration  cycle
// refISO b=new refISO("R744");
// double a[]=b.property("tp",30.0,1.01325);
///---- String s values and meanings --------
//    tv=temperature-specific volume
//    tp=temperature-pressure
//    th=temperature-enthalpy
//    tu=temperature-internal energy
//    ts=temperature-entropy
//    tx=temperature quality
//    pv=pressure-specific volume
//    pt=pressure-temperature
//    ph=pressure-enthapy
//    pu=pressure-internal energy
//    ps=pressure-entropy
//    px=pressure-quality
//    vp=specific volume-pressure
//    vt=specific volume-temperature
//---- output values (method property) ---------
// r[0] P Pressure kPa
// r[1] t temperature degree C
// r[2] v specific volume m^3/kg
// r[3] h enthapy KJ/kg
// r[4] u internal energy KJ/kg
// r[5] s entropy KJ/kgK
// r[6] x quality kg vapour/kg total
// r[7] ro density kg/m^3

public double T[];
public double P[];
public double h[];
public double s[];
public double x[];
public boolean supercritical=false;
public String refname; //refrigerant name
public double Tc,Pc;
public double COP_condenser,COP_evaporator,COP_evap_carnot,COP_condenser_carnot;
public double m;//kg/s mass flow rate of refrigerant
public double N;//kmol/s molar flow rate;
public double Wp;//compressor work
public double Wpi;//isentropic compressor work
public double dPevap1;// evaporator pressure drop saturated region
public double dPevap2;// evaporator pressure drop superheated region
public double dPcond1;// condenser pressure drop superheated region
public double dPcond2;// condenser pressure drop saturated region
public double dPcond3;// condenser pressure drop liquid region
public double dh;     // expansion valve enthalpy chenge (heat loss)
public double dT1,dT6;
public double eta_isent; // isentropic efficiency of compressor
public double Qevaporator,Qcondenser; //condenser and evaporator heat transfer
refISO b; //refrigerantEN class object for properties
double c1[][];
```

```
double c3[][];
public refISO_cycle3(String refnamei,double mi,double dT1i,double P1,double eta_isent1i, double
P2,double dT6i,double dPevap1i,double dPevap2i,double dPcond1i,double dPcond2i,double
dPcond3i,double dhi)
{
m=mi;

refname=refnamei;
dPevap1=dPevap1i;
dPevap2=dPevap2i;
dPcond1=dPcond1i;
dPcond2=dPcond2i;
dPcond3=dPcond3i;
dh=dhi;
dT1=dT1i;
dT6=dT6i;
eta_isent=eta_isent1i;
b=new refISO(refname);
N=m/b.M;
T=new double[9];
P=new double[9];
h=new double[9];
s=new double[9];
x=new double[9];
Tc=b.Tc;//critical temperature degree C
Pc=b.Pc;//critical pressure kPa
P[1]=P1;//compressor inlet
P[2]=P2;//compressor outlet, evaporator superheated outlet
P[0]=P1+dPevap1; //evaporator saturation exit(x=1) pressure
P[4]=P[0]+dPevap2;//evaporator saturation inlet pressure
P[5]=P2-dPcond1;//condenser saturation inlet (x=1)
P[6]=P[5]-dPcond2;//condenser saturation exit (x=0)
P[3]=P[6]-dPcond3;//condenser  exit liquid state)
c1=new double[2][20];
c3=new double[2][20];
}

public void print(int i)
{System.out.println("T["+i+"] = "+T[i]+" P["+i+"] = "+P[i]+" h["+i+"] = "+h[i]+" s["+i+"] = "+s[i]+"
x["+i+"] = "+x[i]);}

public void cycle()
{
// compressor saturation point 0
double a1[]=b.property("px",P[0],1.0);
T[0]=a1[1];
h[0]=a1[3];
s[0]=a1[5];
x[0]=a1[6];
//print(0);
// compressor inlet, evaporator exit point 1
T[1]=T[0]+dT1;
a1=b.property("tp",T[1],P[1]);
h[1]=a1[3];
s[1]=a1[5];
x[1]=a1[6];
//print(1);
// isentropic compressor exit point 7
```

```
s[7]=s[1];
P[7]=P[2];
a1=b.property("ps",P[7],s[7]);
h[7]=a1[3];
T[7]=a1[1];
x[7]=a1[6];
//print(7);
//compressor exit point 2
h[2]=h[1]+(h[7]-h[1])/eta_isent;
a1=b.property("ph",P[2],h[2]);
T[2]=a1[1];
h[2]=a1[3];
s[2]=a1[5];
x[2]=a1[6];
//print(2);
// check if cycle is supercritical
if(P[3]>Pc)
{System.out.println("heat exchanger is supercritical");
supercritical=true;
}
else supercritical=false;
//expansion valve input

if(!supercritical)
{
//saturation point
a1=b.property("px",P[5],1.0);
h[5]=a1[3];
T[5]=a1[1];
s[5]=a1[5];
x[5]=a1[6];
//print(5);
a1=b.property("px",P[6],0.0);
h[6]=a1[3];
T[6]=a1[1];
s[6]=a1[5];
x[6]=a1[6];
//print(6);
T[3]=T[6]-dT6;
a1=b.property("tp",T[3],P[3]);
//a1=b.property("tx",T[3],0.0);
h[3]=a1[3];
s[3]=a1[5];
x[3]=a1[6];
//print(3);
}
else
{
T[3]=Tc-dT6;
a1=b.property("tp",T[3],P[3]);
h[3]=a1[3];
T[3]=a1[1];
s[3]=a1[5];
x[3]=a1[6];
//print(3);
}
// point 4 evaporator inlet, expansion valve output
h[4]=h[3]+dh;
```

```
double x4=b.x_Ph(P[4],h[4]);
a1=b.property("px",P[4],x4);
T[4]=a1[1];
s[4]=a1[5];
x[4]=a1[4];
x[4]=a1[6];
//print(4);
// isentropik compressor
Wpi=N*(h[7]-h[1]);
//compressor
Wp=N*(h[2]-h[1]);
Qevaporator=N*(h[1]-h[4]);
Qcondenser=N*(h[2]-h[3]);
COP_evaporator=Qevaporator/Wp;
COP_condenser_carnot=(T[2]+273.15)/(T[2]-T[1]);
COP_condenser=Qcondenser/Wp;
COP_evap_carnot=(T[1]+273.15)/(T[2]-T[1]);
}

public String toString()
{
cycle();
String ss=refname+" ideal refrigeration cycle \n";
ss+=" compressor work = "+Wp+" kW\n";
ss+=" isentropic compressor work = "+Wpi+" kW\n";
ss+=" condenser heat output = "+Qcondenser+" kW\n";
ss+=" evaporator heat input = "+Qevaporator+" kW\n";
ss+=" COP evaporator    = " + COP_evaporator+" kW\n";
if(!supercritical)
{ss+=" COP condenser     = " + COP_condenser+" kW\n";}
else
{ss+=" COP heat exchanger     = " + COP_condenser+" kW\n";}
ss+=" COP evaporator  carnot = " + COP_evap_carnot+" kW\n";
if(!supercritical)
ss+=" COP condenser   carnot = " + COP_condenser_carnot+" kW\n";
else
ss+=" COP heat exchanger   carnot = " + COP_condenser_carnot+" kW\n";
ss+=" h1 compressor input = "+h[1]/b.M +" kJ/kg\n";
ss+=" T1 compressor input = "+(T[1]-273.15) +" degree C\n";
ss+=" P1 compressor input = "+P[1] +" kPa \n";
ss+=" s1 compressor input = "+s[1]/b.M +" kJ/kgK\n";
ss+=" x1 compressor input = "+x[1] +" kgvapor/kgtotal"+"\n";
ss+=" h2 compressor input = "+h[2]/b.M +" kJ/kg\n";
ss+=" T2 compressor output = "+(T[2]-273.15) +" degree C\n";
ss+=" P2 compressor output = "+P[2] +" kPa \n";
ss+=" s2 compressor output = "+s[2]/b.M +" kJ/kgK\n";
ss+=" x2 compressor output = "+x[2] +" kgvapor/kgtotal"+"\n";
if(!supercritical)
{
ss+=" h5 condenser saturation = "+h[5]/b.M +" kJ/kg\n";
ss+=" T5 condenser saturation = "+(T[5]-273.15) +" degree C\n";
ss+=" P5 condenser saturation = "+P[5] +" kPa \n";
ss+=" s5 condenser saturation = "+s[5] +" kJ/kgK\n";
ss+=" x5 condenser saturation = "+x[5]/b.M +" kgvapor/kgtotal"+"\n";
ss+=" h6 condenser saturation = "+h[6]/b.M +" kJ/kg\n";
ss+=" T6 condenser saturation = "+(T[6]-273.15) +" degree C\n";
ss+=" P6 condenser saturation = "+P[6] +" kPa \n";
ss+=" s6 condenser saturation = "+s[6]/b.M +" kJ/kgK\n";
```

```
ss+=" x6 condenser saturation = "+x[6] +" kgvapor/kgtotal"+"\n";
}
ss+=" h3  expansion valve input = "+h[3]/b.M +" kJ/kg\n";
ss+=" T3  expansion valve input = "+(T[3]-273.15) +" degree C\n";
ss+=" P3  expansion valve input = "+P[3] +" kPa \n";
ss+=" s3  expansion valve input = "+s[3]/b.M +" kJ/kgK\n";
ss+=" x3  epansion valve input = "+x[3] +" kgvapor/kgtotal"+"\n";
ss+=" h4 expansion valve output = "+h[4]/b.M +" kJ/kg\n";
ss+=" T4 expansion valve output = "+(T[4]-273.15) +" degree C\n";
ss+=" s4 expansion valve output = "+s[4]/b.M +" kJ/kgK\n";
ss+=" x4 expansion valve output = "+x[4] +" kgvapor/kgtotal"+"\n";
return ss;
}

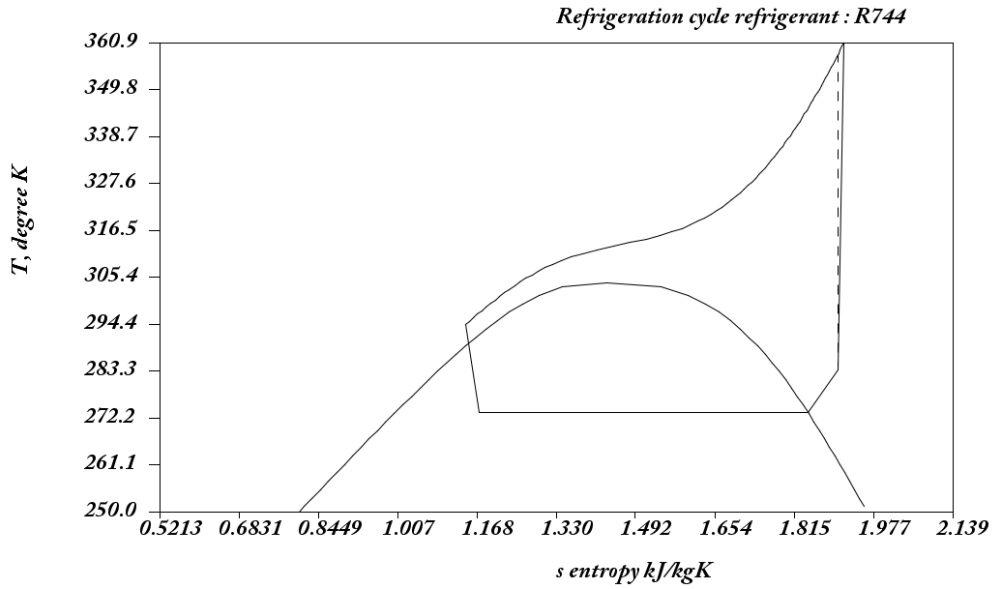

public double[][] line(double P1,double P2,double hi,double ho,int n)
{
double TT1[]=new double[n];
double ss1[]=new double[n];
double aa[];
double p,h;
for(int i=0;i<n;i++)
{p=P1+i*(P2-P1)/(n-1);
 h=hi+i*(ho-hi)/(n-1);
 aa=b.property("ph",p,h);
 TT1[i]=aa[1];
 ss1[i]=aa[5]/b.M;
}
double a[][]={TT1,ss1};
return a;
}
public double[][] line1(double T1,double T2,double Pi,double Po,int n)
{
double TT1[]=new double[n];
double ss1[]=new double[n];
double aa[];
double P;
for(int i=0;i<n;i++)
{TT1[i]=T1+i*(T2-T1)/(n-1);
 P=Pi+i*(Po-Pi)/(n-1);
 aa=b.property("tp",TT1[i],P);
 ss1[i]=aa[5]/b.M;
}
double a[][]={TT1,ss1};
return a;
}
public Plot plot1(Plot pp)
{double a[][]={{T[3],T[4]},{s[3]/b.M,s[4]/b.M}};
 double tt[]=a[0];
 double ss[]=a[1];
 pp.addData(ss,tt);
 double bb[][]=new double[2][15];
 double cc[][];
 if(supercritical)
 {cc=line1(T[2],T[3],P[2],P[3],80);bb=cc;}
 else //subcritical cycle
 { cc=line1(T[2],T[5],P[2],P[5],13);
   for(int i=0;i<cc[0].length;i++)
```

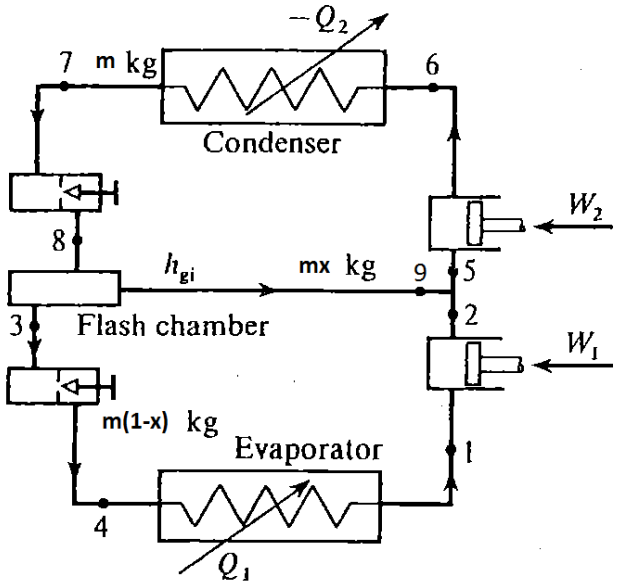
```
   {bb[0][i]=cc[0][i];bb[1][i]=cc[1][i];}
    bb[0][13]=T[6];bb[1][13]=s[6]/b.M;
    bb[0][14]=T[3];bb[1][14]=s[3]/b.M;
 }
 tt=bb[0];
 ss=bb[1];
 pp.addData(ss,tt);
 double t1[]=new double[4];
 double s1[]=new double[4];
 t1[0]=T[4];t1[1]=T[0];t1[2]=T[1];t1[3]=T[2];
 s1[0]=s[4]/b.M;s1[1]=s[0]/b.M;s1[2]=s[1]/b.M;s1[3]=s[2]/b.M;
 pp.addData(s1,t1);
 double t2[]=new double[2];
 double s2[]=new double[2];
 t2[0]=T[1];t2[1]=T[7];
 s2[0]=s[1]/b.M;s2[1]=s[7]/b.M;
 pp.addData(s2,t2,1);
 return pp;
}
public void plot(double ti)
{  double a[][]=b.TS_saturation();
   for(int i=0;i<a[0].length;i++) {a[1][i]=a[1][i]/b.M;}
   Plot pp=new Plot(a[1],a[0]);
   pp= plot1(pp);
   pp.setPlabel("Refrigeration cycle refrigerant : "+b.name);
   pp.setYlabel("T, degree K");
   pp.setXlabel("s entropy kJ/kgK");
              pp.plot();
}
}
```

```
import javax.swing.*;

public class refISO_cycle3test
{
public static void main(String arg[])
{
double m=0.05;  // kg/s çevrim R134a debisi
double P2=9000.0;         //  kPa turbin giriş basıncı
double P1=3500.0 ;         //  kPa turbin çıkış basıncı
double dT1i=10.0; //degree C
double eta_isent1i=0.9;
double dT6i=10.0; //degree C
double dPevap1i=10.0; //kPa
double dPevap2i=10.0; //kPa
double dPcond1i=10.0; //kPa
double dPcond2i=10.0; //kPa
double dPcond3i=10.0; //kPa
double dhi=10.0; //kJ/kgK
refISO_cycle3 r=new
refISO_cycle3("R744",m,dT1i,P1,eta_isent1i,P2,dT6i,dPevap1i,dPevap2i,dPcond1i,dPcond2i,
dPcond3i,dhi);
System.out.println(r.toString());
r.plot(50.0);
}
}
```

```
---------- Capture Output ----------
> "C:\java\bin\javaw.exe" refISO_cycle3test
heat exchanger is supercritical
R744 ideal refrigeration cycle
 compressor work = 2.3239323274386443 kW
 isentropic compressor work = 2.0915390946947783 kW
 condenser heat output = 12.328867048458902 kW
 evaporator heat input = 9.99357361508476 kW
 COP evaporator    = 4.300285983843308 kW
 COP heat exchanger     = 5.3051747259987305 kW
 COP evaporator  carnot = 7.185431145103652 kW
 COP heat exchanger   carnot = 8.185431145103653 kW
 h1 compressor input = 447.49431540176056 kJ/kg
 T1 compressor input = 10.26875975037035 degree C
 P1 compressor input = 3500.0 kPa
 s1 compressor input = 1.904419297963017 kJ/kgK
 h2 compressor input = 493.9729619505335 kJ/kg
 T2 compressor output = 87.7267088296864 degree C
 P2 compressor output = 9000.0 kPa
 s2 compressor output = 1.9173520752755946 kJ/kgK
h3  expansion valve input = 247.39562098135548 kJ/kg
 T3  expansion valve input = 20.980000000000018 degree C
 P3  expansion valve input = 8970.0 kPa
 s3  expansion valve input = 1.1451859384426248 kJ/kgK
 h4 expansion valve output = 247.6228431000654 kJ/kg
 T4 expansion valve output = 0.3764727762339817 degree C
 s4 expansion valve output = 1.1739723715818795 kJ/kgK
```





In order to improve COP of a refrigeration cycle, multistage refrigeration processes can be utilized. In the figure above, a two stage refrigeration cycle using flash chamber is shown. Flash chamber is a seperator seperating saturated liquid phase and saturated vapour phase of the the mixture. Energy balance in the flash chamber :

$mh_8 = mxh_9 + m(1-x)h_3$ (4.2.8)

Mixing process:

$mxh_9 + m(1-x)h_2 = mh_5$ (4.2.9)

$x = \frac{h_8 - h_3}{h_9 - h_3}$ (4.2.10)

Work output:

$W_1 = m(1-x)(h_2 - h_1)$ (4.2.11)

$W_2 = m(h_6 - h_5)$ (4.2.12)

$W = W_1 + W_2$ (4.2.13)

Evaporator heat transfer:

$Q_1 = m(1-x)(h_1 - h_4)$ (4.2.14)

Condenser heat transfer:

$Q_2 = m(h_6 - h_7)$ (4.2.15)

Program 4.2.6 standard refrigeration cycle program

```
public class ref_cycle4
{
// a R134 ideal refrigeration  cycle
// refrigerant b=new refrigerant("R134a");
// double a[]=b.property("tp",30.0,1.01325);
///---- String s values and meanings --------
//    tv=temperature-specific volume
//    tp=temperature-pressure
//    th=temperature-enthalpy
//    tu=temperature-internal energy
//    ts=temperature-entropy
//    tx=temperature quality
//    pv=pressure-specific volume
//    pt=pressure-temperature
//    ph=pressure-enthapy
//    pu=pressure-internal energy
//    ps=pressure-entropy
```

```
//    px=pressure-quality
//    vp=specific volume-pressure
//    vt=specific volume-temperature
//---- output values (method property) ---------
// r[0] P Pressure kPa
// r[1] t temperature degree C
// r[2] v specific volume m^3/kg
// r[3] h enthapy KJ/kg
// r[4] u internal energy KJ/kg
// r[5] s entropy KJ/kgK
// r[6] x quality kg vapour/kg total
// r[7] ro density kg/m^3

double T[];
double P[];
double h[];
double s[];
double x[];
boolean supercritical=false;
String refname; //refrigerant name
double Tc,Pc;
double COP_condenser,COP_evaporator,COP_evap_carnot,COP_condenser_carnot;
double m;//kg/s mass flow rate of refrigerant
double Wp1,Wp2;//compressor work
double Wp1i,Wp2i;//isentropic compressor work
double Wp;//net compressor work
double dPevap1;// evaporator pressure drop saturated region
double dPevap2;// evaporator pressure drop superheated region
double dPcomp1;// evaporator pressure drop saturated region
double dPcond1;// condenser pressure drop superheated region
double dPcond2;// condenser pressure drop saturated region
double dPcond3;// condenser pressure drop liquid region
double etaisent2,etaisent6; //isentropic efficiencies of turbines
double dh1,dh2;     // expansion valve enthalpy chenge (heat loss)
double dT1,dT7;
double eta_isent; // isentropic efficiency of compressor
double Qevaporator,Qcondenser; //condenser and evaporator heat transfer
public refrigerant b; //refrigerant class object for properties
double c1[][];
double c3[][];
public ref_cycle4(String refnamei,double mi,double dT1i,double P1, double P2,double etaisent2i,double
P6,double etaisent6i,double dT7i,
double dPevap1i,double dPevap2i,double dPcomp1i,double dPcond1i,double dPcond2i,double
dPcond3i,double dh1i,double dh2i)
{
m=mi;
refname=refnamei;    // refrigerant name
dPevap1=dPevap1i;    // evaporator pressure drop saturated region
dPevap2=dPevap2i;    // evaporator pressure drop superheated region and compressor 1 inlet valve
dPcomp1=dPcomp1;     // compressor 1 exit valve
dPcond1=dPcond1i;    // condenser pressure drop superheated region compressor2 exit valve
dPcond2=dPcond2i;    // condenser pressure drop saturated region
dPcond3=dPcond3i;    // condenser pressure drop liquid region
dh1=dh1i; dh2=dh2i;  // enthalpy drop(heat gain) in expansion valves
dT1=dT1i;
dT7=dT7i;
etaisent2=etaisent2i;
etaisent6=etaisent6i;
```

```
b=new refrigerant(refname);
T=new double[14];
P=new double[14];
h=new double[14];
s=new double[14];
x=new double[14];
Tc=b.r.Tc;//critical temperature degree C
Pc=b.r.Pc;//critical pressure kPa
P[1]=P1;//compressor inlet
P[2]=P2;//compressor outlet, evaporator superheated outlet
P[0]=P1-dPevap1; //evaporator saturation exit(x=1) pressure
P[4]=P[0]-dPevap2;//evaporator saturation inlet pressure
P[5]=P[2]-dPcomp1;// mixing point pressure
P[9]=P[5];//flash tank saturated vapour exit
P[8]=P[5];//flash tank inlet
P[3]=P[5];//flash tank saturated liquid exit
P[6]=P6;//condenser inlet
P[12]=P[6];// 12=6i isentropic compressor exit point
P[10]=P[6]-dPcond1; // condenser  exit liquid state)
P[13]=P[10]-dPcond2;// condenser saturated liquid point
P[7]=P[13]-dPcond3; // condenser exit (liquid)
c1=new double[2][20];
c3=new double[2][20];
}

public void cycle()
{
// compressor saturation point 0
double a1[]=b.property("px",P[0],1.0);
T[0]=a1[1];
h[0]=a1[3];
s[0]=a1[5];
x[0]=a1[6];
// compressor inlet, evaporator exit point 1
T[1]=T[0]+dT1;
a1=b.property("tp",T[1],P[1]);
h[1]=a1[3];
s[1]=a1[5];
x[1]=a1[6];
s[11]=s[1];
P[11]=P[2];
// isentropic compressor exit point 11 (2i)
a1=b.property("ps",P[11],s[11]);
h[11]=a1[3];
T[11]=a1[1];
h[2]=h[1]+(h[11]-h[1])/etaisent2;
// compressor 1 exit point 2
a1=b.property("ph",P[2],h[2]);
s[2]=a1[5];
T[2]=a1[1];
// point 10 condenser saturated vapour point
x[10]=1.0;
a1=b.property("px",P[10],x[10]);
T[10]=a1[1];
h[10]=a1[3];
s[10]=a1[5];
// point 13 condenser saturated liquid point
x[13]=0.0;
```

```
a1=b.property("px",P[13],x[13]);
T[13]=a1[1];
h[13]=a1[3];
s[13]=a1[5];
T[7]=T[13]-dT7;
// point 7 condenser exit
if(T[7]==T[13] && P[7]==P[13])
{h[7]=h[13];T[7]=T[13];s[7]=s[13];x[7]=x[13];
}
else
{
 a1=b.property("tp",T[7],P[7]);
 h[7]=a1[3];
 s[7]=a1[5];
 x[7]=a1[6];
}
//point 3 flash chamber saturated liquid
x[3]=0.0;
a1=b.property("px",P[3],x[3]);
T[3]=a1[1];
h[3]=a1[3];
s[3]=a1[5];
x[3]=a1[6];
//point 4 flash chamber saturated liquid
h[4]=h[3]-dh2;
a1=b.property("ph",P[4],h[4]);
T[4]=a1[1];
h[4]=a1[3];
s[4]=a1[5];
x[4]=a1[6];
//point 9 flash chamber saturated vapour
x[9]=1.0;
a1=b.property("px",P[9],x[9]);
T[9]=a1[1];
h[9]=a1[3];
s[9]=a1[5];
// point 8 flash chamber input
h[8]=h[7]+dh2;
a1=b.property("ph",P[8],h[8]);
T[8]=a1[1];
s[8]=a1[5];
x[8]=a1[6]; //indicates seperation parameter as well
// point 5 compressor 2 inlet point
h[5]=x[8]*h[9]+(1.0-x[8])*h[2];
a1=b.property("ph",P[5],h[5]);
T[5]=a1[1];
s[5]=a1[5];
//point 12 isentropic compressor2 output
s[12]=s[5];
P[12]=P[6];
a1=b.property("ps",P[12],s[12]);
h[12]=a1[3];
T[12]=a1[1];
h[6]=h[5]+(h[12]-h[5])/etaisent6;
a1=b.property("ph",P[6],h[6]);
T[6]=a1[1];
s[6]=a1[5];
x[6]=a1[6];
```

```
 // isentropik compressor 1
Wp1i=m*(1.0-x[8])*(h[11]-h[1]);
//compressor 1
Wp1=m*(1.0-x[8])*(h[2]-h[1]);
Wp=Wp1+Wp2;
// isentropik compressor 2
Wp2i=m*(h[12]-h[5]);
//compressor 2
Wp2=m*(h[6]-h[5]);
Qevaporator=m*(1.0-x[8])*(h[1]-h[4]);
Qcondenser=m*(h[6]-h[7]);
COP_evaporator=Qevaporator/Wp;
COP_condenser_carnot=(T[6]+273.15)/(T[10]-T[0]);
COP_condenser=Qcondenser/Wp;
COP_evap_carnot=(T[0]+273.15)/(T[10]-T[0]);
}

public String toString()
{
cycle();
String ss="two stage flash point refrigeration cycle\n";
ss+="Flash ratio : "+x[8]+"\n";
ss+=" compressor1 work = "+Wp1+" kW\n";
ss+=" compressor2 work = "+Wp2+" kW\n";
ss+=" total compressor work = "+Wp+" kW\n";
ss+=" isentropic compressor 1 work = "+Wp1i+" kW\n";
ss+=" isentropic compressor 2 work = "+Wp2i+" kW\n";
ss+=" total isentropic compressor work = "+(Wp1i+Wp2i)+" kW\n";
ss+=" condenser heat output = "+Qcondenser+" kW\n";
ss+=" evaporator heat input = "+Qevaporator+" kW\n";
ss+=" COP evaporator    = " + COP_evaporator+" kW\n";
ss+=" COP condenser     = " + COP_condenser+" kW\n";
ss+=" COP evaporator  carnot = " + COP_evap_carnot+" kW\n";
ss+=" COP condenser   carnot = " + COP_condenser_carnot+" kW\n";

ss+=" h1 compressor 1 inlet = "+h[1] +" kJ/kg\n";
ss+=" T1 compressor 1 inlet = "+T[1] +" derece C\n";
ss+=" P1 compressor 1 inlet = "+P[1] +" kPa \n";
ss+=" s1 compressor 1 inlet = "+s[1] +" kJ/kgK\n";
ss+=" x1 compressor 1 inlet = "+x[1] +" kgvapor/kgtotal"+"\n";

ss+=" h2 compressor 1 exit = "+h[2] +" kJ/kg\n";
ss+=" T2 compressor 1 exit = "+T[2] +" derece C\n";
ss+=" P2 compressor 1 exit = "+P[2] +" kPa \n";
ss+=" s2 compressor 1 exit = "+s[2] +" kJ/kgK\n";
ss+=" x2 compressor 1 exit = "+x[2] +" kgvapor/kgtotal"+"\n";

ss+=" h5 compressor 2 inlet = "+h[5] +" kJ/kg\n";
ss+=" T5 compressor 2 inlet = "+T[5] +" derece C\n";
ss+=" P5 compressor 2 inlet = "+P[5] +" kPa \n";
ss+=" s5 compressor 2 inlet = "+s[5] +" kJ/kgK\n";
ss+=" x5 compressor 2 inlet = "+x[5] +" kgvapor/kgtotal"+"\n";

ss+=" h6 compressor 2 exit = "+h[6] +" kJ/kg\n";
ss+=" T6 compressor 2 exit = "+T[6] +" derece C\n";
ss+=" P6 compressor 2 exit = "+P[6] +" kPa \n";
ss+=" s6 compressor 2 exit = "+s[6] +" kJ/kgK\n";
ss+=" x6 compressor 2 exit = "+x[6] +" kgvapor/kgtotal"+"\n";
```

```
ss+=" h8 flash chamber input = "+h[3] +" kJ/kg\n";
ss+=" T8 flash chamber input = "+T[3] +" derece C\n";
ss+=" P8 flash chamber input = "+P[3] +" kPa \n";
ss+=" s8 flash chamber input = "+s[3] +" kJ/kgK\n";
ss+=" x8 flash chamber input = "+x[3] +" kgvapor/kgtotal"+"\n";

ss+=" h4 expansion valve exit-evaporator inlet = "+h[4] +" kJ/kg\n";
ss+=" T4 expansion valve exit-evaporator inlet = "+T[4] +" derece C\n";
ss+=" P4 expansion valve exit-evaporator inlet = "+P[4] +" kPa \n";
ss+=" s4 expansion valve exit-evaporator inlet = "+s[4] +" kJ/kgK\n";
ss+=" x4 expansion valve exit-evaporator inlet = "+x[4] +" kgvapor/kgtotal"+"\n";
return ss;
}

public double[][] TS(double ti)
{
double a[][]=new double[2][10000];
double tc=b.r.Tc;
double dt=(tc-1.0)/100.0;
int i=0;
double t;
for(t=ti;t<=tc;t+=dt)
{double a1[]=b.property("tx",t,0.0);
 a[1][i]=t;
 a[0][i]=a1[5];
 i++;
}
double a3[]=b.property("tx",tc,1.0);
 a[1][i]=t;
 a[0][i]=a3[5];
 i++;
for(t=tc;t>=ti;t-=dt)
{double a2[]=b.property("tx",t,1.0);
 a[1][i]=t;
 a[0][i]=a2[5];
 i++;
}
int n=i;
double b[][]=new double[2][n];
for(i=0;i<n;i++) {b[0][i]=a[0][i];b[1][i]=a[1][i];}
return b;
}
public double[][] line(double P1,double P2,double hi,double ho,int n)
{
double TT1[]=new double[n];
double ss1[]=new double[n];
double aa[];
double p,h;
for(int i=0;i<n;i++)
{p=P1+i*(P2-P1)/(n-1);
 h=hi+i*(ho-hi)/(n-1);
 aa=b.property("ph",p,h);
 TT1[i]=aa[1];
 ss1[i]=aa[5];
}
double a[][]={TT1,ss1};
return a;
```

```
}
public double[][] line1(double T1,double T2,double Pi,double Po,int n)
{
double TT1[]=new double[n];
double ss1[]=new double[n];
double aa[];
double P;
double x=1.0;
aa=b.property("tx",T1,x);
TT1[0]=T1;
ss1[0]=aa[5];
for(int i=1;i<n;i++)
{TT1[i]=T1+i*(T2-T1)/(n-1);
 P=Pi+i*(Po-Pi)/(n-1);
 aa=b.property("tp",TT1[i],P);
 ss1[i]=aa[5];
}
double a[][]={TT1,ss1};
Text.printT(a);
return a;
}

public void plot(double ti)
{
 Plot pp=new Plot(TS(ti));
 //enthalpy line
 double a34[][]=line(P[3],P[4],h[3],h[4],10);
 double a78[][]=line(P[7],P[8],h[7],h[7],10);
 double tt[]=a34[0];
 double ss[]=a34[1];
 pp.addData(ss,tt);

 tt=a78[0];
 ss=a78[1];
 //System.out.println("tt="+Matrix.toString(tt));
 //System.out.println("tt="+Matrix.toString(ss));
 pp.addData(ss,tt);

 double a10[][]=line1(T[0],T[1],P[0],P[1],15);
 tt=a10[0];
 ss=a10[1];
 pp.addData(ss,tt);

 double a92[][]=line1(T[9],T[2],P[9],P[2],16);
  tt=a92[0];
  ss=a92[1];
  pp.addData(ss,tt);
  double a10_6[][]=line1(T[10],T[6],P[10],P[6],16);
  tt=a10_6[0];
  ss=a10_6[1];
  pp.addData(ss,tt);
 double tt1[]={T[10],T[13],T[7]};
 double ss1[]={s[10],s[13],s[7]};
 pp.addData(ss1,tt1);
 double tt2[]={T[4],T[0]};
 double ss2[]={s[4],s[0]};
 pp.addData(ss2,tt2);
 double tt3[]={T[9],T[3]};
```

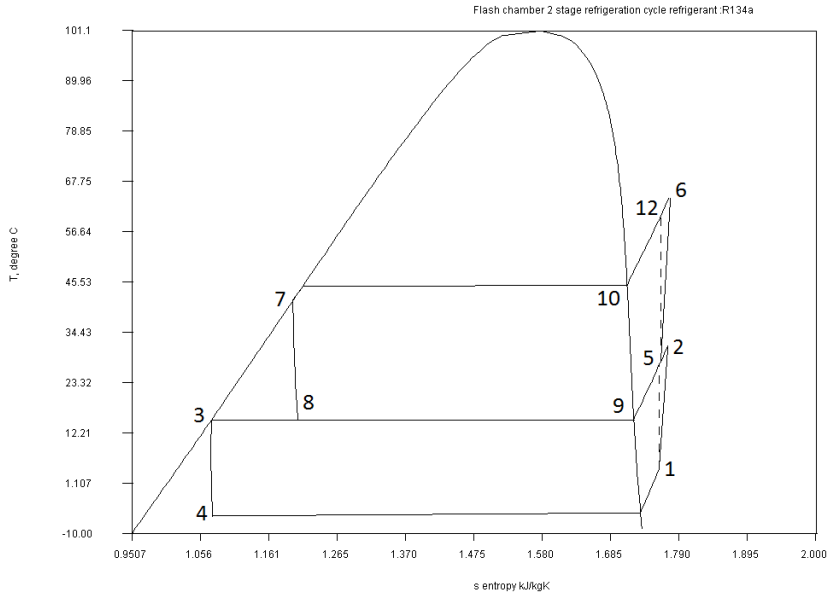
```
 double ss3[]={s[9],s[3]};
 pp.addData(ss3,tt3);
 double tt4[]={T[1],T[2]};
 double ss4[]={s[1],s[2]};
 pp.addData(ss4,tt4);
  double tt5[]={T[5],T[6]};
 double ss5[]={s[5],s[6]};
 pp.addData(ss5,tt5);
 pp.addData(ss4,tt4);
  double tt6[]={T[1],T[11]};
 double ss6[]={s[1],s[11]};
 pp.addData(ss6,tt6,1);
  double tt7[]={T[5],T[12]};
 double ss7[]={s[5],s[12]};
 pp.addData(ss7,tt7,1);
 pp.setPlabel("Flash chamber 2 stage refrigeration cycle refrigerant :"+refname);
 pp.setYlabel("T, degree C");
 pp.setXlabel("s entropy kJ/kgK");
 pp.plot();
}}
```

Program 4.2.7 standard refrigeration cycle test program

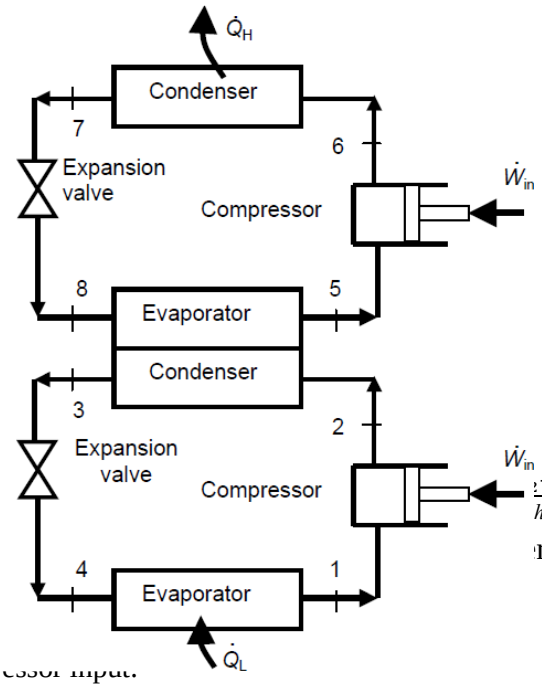
```
 import javax.swing.*;
public class ref_cycle4test
{
public static double Pres(String refrigerant,double T)
{refrigerant r=new refrigerant(refrigerant);
double a[]=r.property("tx",T,0.0);
double P1=a[0];
return P1;
}
public static void main(String arg[])
{
double m=1.0;  // kg/s çevrim R134a debisi
String refrigerant="R134a";
refrigerant r=new refrigerant(refrigerant);
double P1=Pres(refrigerant,-5.0);
double P2=Pres(refrigerant,15.0);
double P6=Pres(refrigerant,45.0);
double dT1i=10.0; //degree C
double etaisent2i=0.8;
double etaisent6i=0.8;
double dT7i=3.0; //degree C
double dPevap1i=5.0;
double dPevap2i=5.0;
double dPcomp1i=5.0;
double dPcond1i=5.0;
double dPcond2i=5.0;
double dPcond3i=5.0;
double dh1i=1.0;
double dh2i=1.0;
ref_cycle4 re=new ref_cycle4("R134a",m,dT1i,P1,P2,etaisent2i,P6,etaisent6i,dT7i,
dPevap1i,dPevap2i,dPcomp1i,dPcond1i,dPcond2i,dPcond3i,dh1i,dh2i);
re.cycle();
System.out.println(re.toString());
re.plot(-10.0);
}
}
```

```
---------- Capture Output ----------
> "D:\co\java\bin\javaw.exe" ref_cycle4test
two stage flash point refrigeration cycle
Flash ratio : 0.21120156195879478
 compressor1 work = 14.805863773886465 kW
 compressor2 work = 24.050989363605083 kW
 total compressor work = 38.85685313749155 kW
 isentropic compressor 1 work = 11.844691019109163 kW
 isentropic compressor 2 work = 19.240791490884078 kW
 total isentropic compressor work = 31.08548250999324 kW
 condenser heat output = 184.53255326182955 kW
 evaporator heat input = 145.46449856237925 kW
 COP evaporator    = 3.743599566533758 kW
 COP condenser     = 4.749034941375139 kW
 COP evaporator  carnot = 5.31152153784105 kW
 COP condenser   carnot = 6.700686171842741 kW
 h1 compressor 1 inlet = 403.89090915702633 kJ/kg
 T1 compressor 1 inlet = 4.452127613408595 derece C
 P1 compressor 1 inlet = 243.34658575187083 kPa
 s1 compressor 1 inlet = 1.760202180243217 kJ/kgK
 x1 compressor 1 inlet = 2.0 kgvapor/kgtotal
 h2 compressor 1 exit = 422.66105760033884 kJ/kg
 T2 compressor 1 exit = 31.330509752005508 derece C
 P2 compressor 1 exit = 488.3740471839527 kPa
 s2 compressor 1 exit = 1.7726125368700008 kJ/kgK
 x2 compressor 1 exit = 0.0 kgvapor/kgtotal
 h5 compressor 2 inlet = 419.3684277030723 kJ/kg
 T5 compressor 2 inlet = 27.850293806875605 derece C
 P5 compressor 2 inlet = 488.3740471839527 kPa
 s5 compressor 2 inlet = 1.761736309839008 kJ/kgK
 x5 compressor 2 inlet = 0.0 kgvapor/kgtotal
 h6 compressor 2 exit = 443.4194170666774 kJ/kg
 T6 compressor 2 exit = 64.4402485342654 derece C
 P6 compressor 2 exit = 1159.9066190539932 kPa
 s6 compressor 2 exit = 1.776078845629143 kJ/kgK
 x6 compressor 2 exit = 1.1389855564604832 kgvapor/kgtotal
 h8 flash chamber input = 220.478147230664 kJ/kg
 T8 flash chamber input = 14.999989757599444 derece C
 P8 flash chamber input = 488.3740471839527 kPa
 s8 flash chamber input = 1.0724586959517781 kJ/kgK
 x8 flash chamber input = 0.0 kgvapor/kgtotal
 h4 expansion valve exit-evaporator inlet = 219.478147230664 kJ/kg
 T4 expansion valve exit-evaporator inlet = -6.104624271642147 derece C
 P4 expansion valve exit-evaporator inlet = 233.34658575187083 kPa
 s4 expansion valve exit-evaporator inlet = 1.0734651632666865 kJ/kgK
 x4 expansion valve exit-evaporator inlet = 0.13599536311661237 kgvapor/kgtotal

> Terminated with exit code 0.
```



Two single stage refrigeration cycles can be combined to make a cascade refrigeration cycle. In this cycle, upper cycle evaporator and lower cycle condenser is merged to form a single heat exchanger.



Comp

$\eta_{C_{uppe}}$ $\frac{-h_5)}{h_5)}$

Mass r cycle:

$\dot{m}_A(h_$

Compressor input:

$$W_1 = \dot{m}_A(h_6 - h_5)$$

$W_2 = \dot{m}_b(h_2 - h_1)$ (4.2.17-18)

$W_T = W_1 + W_2$ (4.2.19)

Evaporator heat transfer (Heat removal from refrigerated space):

$Q_{evap} = \dot{m}_B(h_1 - h_4)$ (4.2.20)

Condenser heat transfer:

$Q_{cond} = \dot{m}_A(h_6 - h_7)$ (4.2.21)

Coefficient of performance of the system (COPsys):

$$COP_{sys} = \frac{Q_{evap}}{W_T}$$

Program 4.2.8 cascade refrigeration cycle program

```
 public class ref_cycle7
{ // Cascade cycle
//   two stage refrigeration  cycle
//   refrigerantEN b=new refrigerantEN("R134a");
//   double a[]=b.property("tp",30.0,1.01325);
///---- String s values and meanings --------
//    tv=temperature-specific volume
//    tp=temperature-pressure
//    th=temperature-enthalpy
//    tu=temperature-internal energy
//    ts=temperature-entropy
//    tx=temperature quality
//    pv=pressure-specific volume
//    pt=pressure-temperature
//    ph=pressure-enthapy
//    pu=pressure-internal energy
//    ps=pressure-entropy
//    px=pressure-quality
//    vp=specific volume-pressure
//    vt=specific volume-temperature
//---- output values (method property) ---------
// r[0] P Pressure kPa
// r[1] t temperature degree C
// r[2] v specific volume m^3/kg
// r[3] h enthapy KJ/kg
// r[4] u internal energy KJ/kg
// r[5] s entropy KJ/kgK
// r[6] x quality kg vapour/kg total
// r[7] ro density kg/m^3
ref_cycle3 rA,rB;
double mA,mB;
double Qheatexchanger;
double Wp;
double Qevaporator;
double Qcondenser;
double COP_evaporator;
double COP_condenser;
Plot pp;
public ref_cycle7(String refnameAi,double mAi,double dT1Ai,double P1A,double eta_isent1Ai, double
P2A,double dT6Ai,double dPevap1Ai,double dPevap2Ai,double dPcond1Ai,double dPcond2Ai,double
dPcond3Ai,double dhAi,
String refnameBi,double dT1Bi,double P1B,double eta_isent1Bi, double P2B,double dT6Bi,double
dPevap1Bi,double dPevap2Bi,double dPcond1Bi,double dPcond2Bi,double dPcond3Bi,double dhBi)
{
// top cycle
mA=mAi;
rA=new
ref_cycle3(refnameAi,mAi,dT1Ai,P1A,eta_isent1Ai,P2A,dT6Ai,dPevap1Ai,dPevap2Ai,dPcond1Ai,dP
cond2Ai,dPcond3Ai,dhAi);
rA.cycle();
double m=1.0;
// bottom cycle
rB=new
ref_cycle3(refnameBi,m,dT1Bi,P1B,eta_isent1Bi,P2B,dT6Bi,dPevap1Bi,dPevap2Bi,dPcond1Bi,dPcon
d2Bi,dPcond3Bi,dhBi);
rB.cycle();
mB=rA.Qevaporator/(rB.h[2]-rB.h[3]);
```

```
//rB=new
ref_cycle3(refnameBi,mB,dT1Bi,P1A,eta_isent1Bi,P2A,dT6Bi,dPevap1Bi,dPevap2Bi,dPcond1Bi,dPcond2Bi,dPcond3Bi,dhBi);
}

public void cycle()
{
// top cycle
rA=new
ref_cycle3(rA.refname,mA,rA.dT1,rA.P[1],rA.eta_isent,rA.P[2],rA.dT6,rA.dPevap1,rA.dPevap2,rA.dPcond1,rA.dPcond2,rA.dPcond3,rA.dh);
rA.cycle();
rB=new
ref_cycle3(rB.refname,mB,rB.dT1,rB.P[1],rB.eta_isent,rB.P[2],rB.dT6,rB.dPevap1,rB.dPevap2,rB.dPcond1,rB.dPcond2,rB.dPcond3,rB.dh);
rB.cycle();
Wp=rA.Wp+rB.Wp;
Qevaporator=rB.Qevaporator;
Qcondenser=rA.Qcondenser;
Qheatexchanger=rB.Qcondenser;
COP_evaporator=Qevaporator/Wp;
COP_condenser=Qcondenser/Wp;
}

public String toString()
{
cycle();
String ss=" cascade refrigeration cycle \n";
ss+=" compressor work = "+Wp+" kW\n";
ss+=" condenser heat output = "+Qcondenser+" kW\n";
ss+=" evaporator heat input = "+Qevaporator+" kW\n";
ss+=" heat exchanger input/output = "+Qheatexchanger+" kW\n";
ss+=" COP evaporator    = " + COP_evaporator+" kW\n";
ss+=" COP condenser     = " + COP_condenser+" kW\n";
ss+=" Top cycle  \n";
ss+=" refrigerant = "+rA.refname+"\n";
ss+=" m = "+mA+"kg/s\n";
ss+=" h1 compressor input = "+rA.h[1] +" kJ/kg\n";
ss+=" T1 compressor input = "+rA.T[1] +" derece C\n";
ss+=" P1 compressor input = "+rA.P[1] +" kPa \n";
ss+=" s1 compressor input = "+rA.s[1] +" kJ/kgK\n";
ss+=" x1 compressor input = "+rA.x[1] +" kgvapor/kgtotal"+"\n";
ss+=" h2 compressor input = "+rA.h[2] +" kJ/kg\n";
ss+=" T2 compressor output = "+rA.T[2] +" derece C\n";
ss+=" P2 compressor output = "+rA.P[2] +" kPa \n";
ss+=" s2 compressor output = "+rA.s[2] +" kJ/kgK\n";
ss+=" x2 compressor output = "+rA.x[2] +" kgvapor/kgtotal"+"\n";
if(!rA.supercritical)
{
ss+=" h5 compressor input = "+rA.h[5] +" kJ/kg\n";
ss+=" T5 condenser saturation = "+rA.T[5] +" derece C\n";
ss+=" P5 condenser saturation = "+rA.P[5] +" kPa \n";
ss+=" s5 condenser saturation = "+rA.s[5] +" kJ/kgK\n";
ss+=" x5 condenser saturation = "+rA.x[5] +" kgvapor/kgtotal"+"\n";
ss+=" h6 compressor input = "+rA.h[6] +" kJ/kg\n";
ss+=" T6 condenser saturation = "+rA.T[6] +" derece C\n";
ss+=" P6 condenser saturation = "+rA.P[6] +" kPa \n";
ss+=" s6 condenser saturation = "+rA.s[6] +" kJ/kgK\n";
```

```
ss+=" x6 condenser saturation = "+rA.x[6] +" kgvapor/kgtotal"+"\n";
}
ss+=" h3 turbine input = "+rA.h[3] +" kJ/kg\n";
ss+=" T3 turbine input = "+rA.T[3] +" derece C\n";
ss+=" P3 turbine input = "+rA.P[3] +" kPa \n";
ss+=" s3 turbine input = "+rA.s[3] +" kJ/kgK\n";
ss+=" x3 turbine input = "+rA.x[3] +" kgvapor/kgtotal"+"\n";
ss+=" h4 turbine output = "+rA.h[4] +" kJ/kg\n";
ss+=" T4 turbine output = "+rA.T[4] +" derece C\n";
ss+=" s4 turbine output = "+rA.s[4] +" kJ/kgK\n";
ss+=" x4 turbine output = "+rA.x[4] +" kgvapor/kgtotal"+"\n";
ss+=" Bottom cycle  \n";
ss+=" refrigerant = "+rB.refname+"\n";
ss+=" m = "+mB+"kg/s\n";
ss+=" h1 compressor input = "+rB.h[1] +" kJ/kg\n";
ss+=" T1 compressor input = "+rB.T[1] +" derece C\n";
ss+=" P1 compressor input = "+rB.P[1] +" kPa \n";
ss+=" s1 compressor input = "+rB.s[1] +" kJ/kgK\n";
ss+=" x1 compressor input = "+rB.x[1] +" kgvapor/kgtotal"+"\n";
ss+=" h2 compressor input = "+rB.h[2] +" kJ/kg\n";
ss+=" T2 compressor output = "+rB.T[2] +" derece C\n";
ss+=" P2 compressor output = "+rB.P[2] +" kPa \n";
ss+=" s2 compressor output = "+rB.s[2] +" kJ/kgK\n";
ss+=" x2 compressor output = "+rB.x[2] +" kgvapor/kgtotal"+"\n";
if(!rA.supercritical)
{
ss+=" h5 compressor input = "+rB.h[5] +" kJ/kg\n";
ss+=" T5 condenser saturation = "+rB.T[5] +" derece C\n";
ss+=" P5 condenser saturation = "+rB.P[5] +" kPa \n";
ss+=" s5 condenser saturation = "+rB.s[5] +" kJ/kgK\n";
ss+=" x5 condenser saturation = "+rB.x[5] +" kgvapor/kgtotal"+"\n";
ss+=" h6 compressor input = "+rB.h[6] +" kJ/kg\n";
ss+=" T6 condenser saturation = "+rB.T[6] +" derece C\n";
ss+=" P6 condenser saturation = "+rB.P[6] +" kPa \n";
ss+=" s6 condenser saturation = "+rB.s[6] +" kJ/kgK\n";
ss+=" x6 condenser saturation = "+rB.x[6] +" kgvapor/kgtotal"+"\n";
}
ss+=" h3 turbine input = "+rB.h[3] +" kJ/kg\n";
ss+=" T3 turbine input = "+rB.T[3] +" derece C\n";
ss+=" P3 turbine input = "+rB.P[3] +" kPa \n";
ss+=" s3 turbine input = "+rB.s[3] +" kJ/kgK\n";
ss+=" x3 turbine input = "+rB.x[3] +" kgvapor/kgtotal"+"\n";
ss+=" h4 turbine output = "+rB.h[4] +" kJ/kg\n";
ss+=" T4 turbine output = "+rB.T[4] +" derece C\n";
ss+=" s4 turbine output = "+rB.s[4] +" kJ/kgK\n";
ss+=" x4 turbine output = "+rB.x[4] +" kgvapor/kgtotal"+"\n";
return ss;
}

public void plot(double ti)
{  double a[][]=rA.TS(ti);
            pp=new Plot(a[0],a[1]);
   pp=rA.plot1(pp);
   a=rB.TS(ti);
   pp.addData(a[0],a[1]);
   pp=rB.plot1(pp);
   pp.setPlabel("Cascade refrigeration  refrigerant 1="+rA.refname+" refrigerant 2="+rB.refname );
   pp.setYlabel("T, degree C");
```

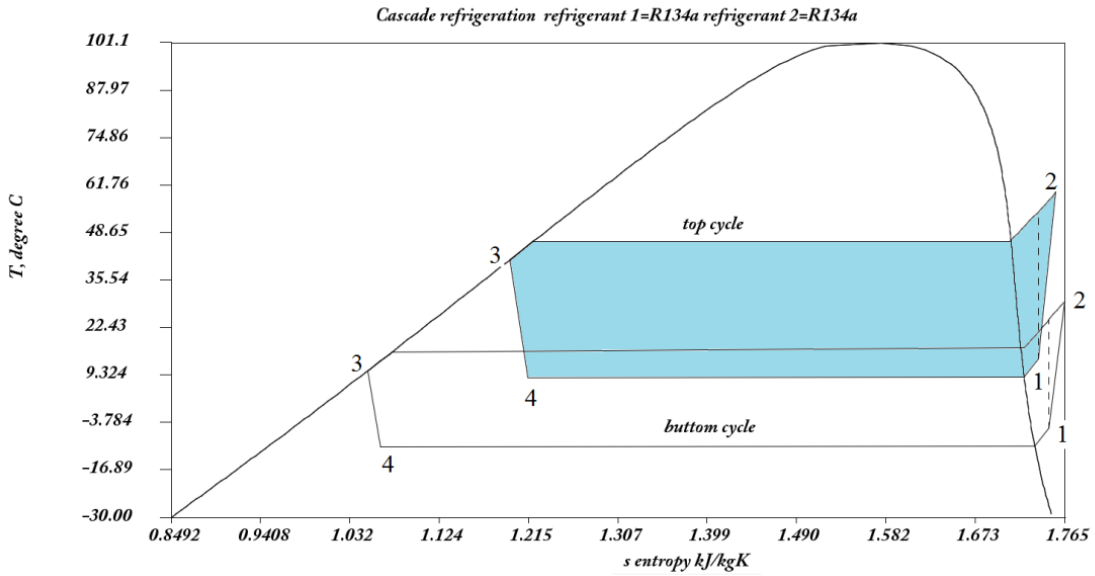
```
    pp.setXlabel("s entropy kJ/kgK");
    pp.plot();
}
}
```

Since upper and lower cycles are completely independent, they can run with same refrigerant or two different refrigerants

Program 4.2.9 cascade refrigeration cycle test program

```
import javax.swing.*;

public class ref_cycle7testA
{
public static void main(String arg[])
{
double mA=1;  // kg/s çevrim R134a debisi
String refA="R134a";
double P2A=1200.0;         //  kPa turbin giriş basıncı
double P1A=400.0 ;         //  kPa turbin çıkış basıncı
double dT1Ai=5.0; //degree C
double eta_isent1Ai=0.8;
double dT6Ai=5.0; //degree C
double dPevap1Ai=2.0; //kPa
double dPevap2Ai=2.0; //kPa
double dPcond1Ai=2.0; //kPa
double dPcond2Ai=2.0; //kPa
double dPcond3Ai=2.0; //kPa
double dhAi=2.0; //kJ/kgK
String refB="R134a";
double P2B=500.0;         //  kPB turbin giriş bBsıncı
double P1B=200.0 ;         //  kPB turbin çıkış bBsıncı
double dT1Bi=5.0; //degree C
double etB_isent1Bi=0.8;
double dT6Bi=5.0; //degree C
double dPevap1Bi=2.0; //kPB
double dPevap2Bi=2.0; //kPB
double dPcond1Bi=2.0; //kPB
double dPcond2Bi=2.0; //kPB
double dPcond3Bi=2.0; //kPB
double dhBi=2.0; //kJ/kgK
ref_cycle7 r=new
ref_cycle7(refA,mA,dT1Ai,P1A,eta_isent1Ai,P2A,dT6Ai,dPevap1Ai,dPevap2Ai,dPcond1Ai,dPcond2Ai,
dPcond3Ai,dhAi,
refB,dT1Bi,P1B,etB_isent1Bi,P2B,dT6Bi,dPevap1Bi,dPevap2Bi,dPcond1Bi,dPcond2Bi,dPcond3Bi,dhBi);
System.out.println(r.toString());
r.cycle();
r.plot(-30.0);}}
```



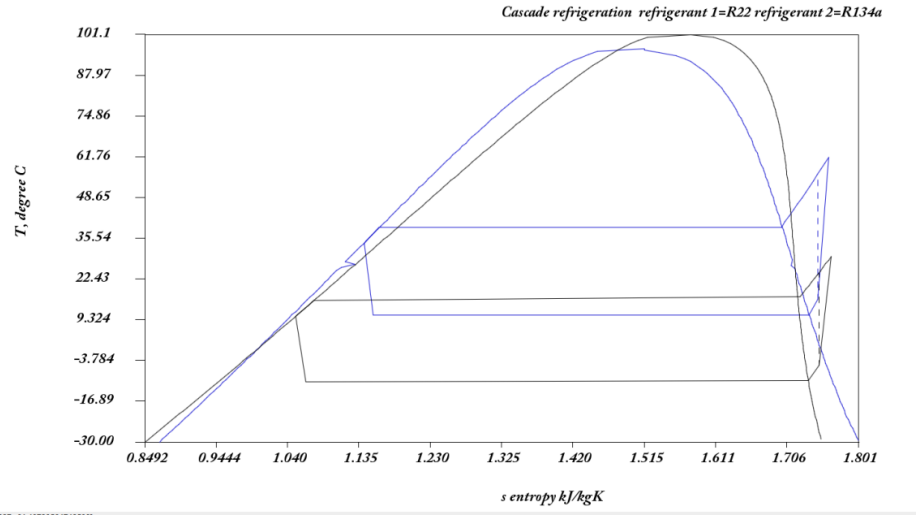
Program 4.2.10 cascade refrigeration cycle test program

```
---------- Capture Output ----------
> "C:\java\bin\java.exe" ref_cycle7testA
cascade refrigeration cycle
compressor work = 46.557689380580285 kW
condenser heat output = 179.1910789630847 kW
evaporator heat input = 129.2012201024127 kW
heat exchanger input/output = 147.90723649915645 kW
COP evaporator    = 2.7750780122757535 kW
COP condenser     = 3.8487966509314617 kW
Top cycle
refrigerant = R134a
m = 1.0kg/s
h1 compressor input = 408.2319432599678 kJ/kg
T1 compressor input = 13.760391726590042 derece C
P1 compressor input = 400.0 kPa
s1 compressor input = 1.7384553609550266 kJ/kgK
x1 compressor input = 2.0 kgvapor/kgtotal
h2 compressor input = 437.51578572389604 kJ/kg
T2 compressor output = 59.75600041799221 derece C
P2 compressor output = 1200.0 kPa
s2 compressor output = 1.7561889728460591 kJ/kgK
x2 compressor output = 1.0966552920873422 kgvapor/kgtotal
h5 compressor input = 422.3858140744719 kJ/kg
T5 condenser saturation = 46.21416207180537 derece C
P5 condenser saturation = 1198.0 kPa
s5 condenser saturation = 1.7100129797825283 kJ/kgK
x5 condenser saturation = 0.0 kgvapor/kgtotal
h6 compressor input = 265.93693684559605 kJ/kg
T6 condenser saturation = 46.14926677186208 derece C
P6 condenser saturation = 1196.0 kPa
s6 condenser saturation = 1.2201990973101455 kJ/kgK
x6 condenser saturation = 0.0 kgvapor/kgtotal
h3 turbine input = 258.32470676081135 kJ/kg
T3 turbine input = 41.14926677186208 derece C
P3 turbine input = 1194.0 kPa
s3 turbine input = 1.1966004528854404 kJ/kgK
x3 turbine input = 0.0 kgvapor/kgtotal
h4 turbine output = 260.32470676081135 kJ/kg
```

T4 turbine output = 8.611307337942751 derece C
s4 turbine output = 1.214757586220795 kJ/kgK
x4 turbine output = 0.0 kgvapor/kgtotal
Bottom cycle
refrigerant = R134a
m = 0.7160847400458633kg/s
h1 compressor input = 396.66384167379755 kJ/kg
T1 compressor input = -5.329546896665409 derece C
P1 compressor input = 200.0 kPa
s1 compressor input = 1.748643091085814 kJ/kgK
x1 compressor input = 2.0 kgvapor/kgtotal
h2 compressor input = 420.7864712323407 kJ/kg
T2 compressor output = 29.609873382254033 derece C
P2 compressor output = 500.0 kPa
s2 compressor output = 1.7647141124425834 kJ/kgK
x2 compressor output = 1.0706127884130356 kgvapor/kgtotal
h5 compressor input = 407.5782465453429 kJ/kg
T5 condenser saturBtion = 15.582823738854056 derece C
P5 condenser saturBtion = 498.0 kPa
s5 condenser saturBtion = 1.720567043376862 kJ/kgK
x5 condenser saturBtion = 0.0 kgvapor/kgtotal
h6 compressor input = 221.13917141892864 kJ/kg
T6 condenser saturBtion = 15.457077674540868 derece C
P6 condenser saturBtion = 496.0 kPa
s6 condenser saturBtion = 1.0747486638109192 kJ/kgK
x6 condenser saturBtion = 0.0 kgvapor/kgtotal
h3 turbine input = 214.23656417848645 kJ/kg
T3 turbine input = 10.457077674540868 derece C
P3 turbine input = 494.0 kPa
s3 turbine input = 1.050685283471824 kJ/kgK
x3 turbine input = 0.0 kgvapor/kgtotal
h4 turbine output = 216.23656417848645 kJ/kg
T4 turbine output = -10.585295785211429 derece C
s4 turbine output = 1.0638402933582916 kJ/kgK
x4 turbine output = 0.0 kgvapor/kgtotal

```
import javax.swing.*;
public class ref_cycle7test
{
public static void main(String arg[])
{
double mA=1;  // kg/s çevrim R134a debisi
String refA="R22";
double P2A=1500.0;         //  kPa turbin giriş basıncı
double P1A=700.0 ;         //  kPa turbin çıkış basıncı
double dT1Ai=5.0; //degree C
double eta_isent1Ai=0.8;
double dT6Ai=5.0; //degree C
double dPevap1Ai=2.0; //kPa
double dPevap2Ai=2.0; //kPa
double dPcond1Ai=2.0; //kPa
double dPcond2Ai=2.0; //kPa
double dPcond3Ai=2.0; //kPa
double dhAi=2.0; //kJ/kgK
String refB="R134a";
double P2B=500.0;         //  kPB turbin giriş bBsıncı
double P1B=200.0 ;         //  kPB turbin çıkış bBsıncı
double dT1Bi=5.0; //degree C
```

```
double etB_isent1Bi=0.8;
double dT6Bi=5.0; //degree C
double dPevap1Bi=2.0; //kPB
double dPevap2Bi=2.0; //kPB
double dPcond1Bi=2.0; //kPB
double dPcond2Bi=2.0; //kPB
double dPcond3Bi=2.0; //kPB
double dhBi=2.0; //kJ/kgK
ref_cycle7 r=new
ref_cycle7(refA,mA,dT1Ai,P1A,eta_isent1Ai,P2A,dT6Ai,dPevap1Ai,dPevap2Ai,dPcond1Ai,dPcond2Ai,dPcond3
Ai,dhAi,refB,dT1Bi,P1B,etB_isent1Bi,P2B,dT6Bi,dPevap1Bi,dPevap2Bi,dPcond1Bi,dPcond2Bi,dPcond3Bi,dhBi
);
System.out.println(r.toString());
r.cycle();
r.plot(-30.0);}}
```



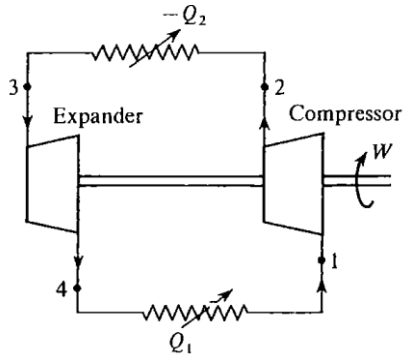
Program 4.2.11 cascade refrigeration cycle test program

```
---------- Capture Output ----------
> "C:\java\bin\java.exe" ref_cycle7test
 cascade refrigeration cycle
 compressor work = 43.63978004024978 kW
 condenser heat output = 194.35609347721999 kW
 evaporator heat input = 147.0858959305664 kW
 heat exchanger input/output = 168.38129220334156 kW
 COP evaporator    = 3.3704545667944785 kW
 COP condenser     = 4.453645121445657 kW
 Top cycle
 refrigerant = R22
 m = 1.0kg/s
 h1 compressor input = 412.1929202446289 kJ/kg
 T1 compressor input = 15.82384668927146 derece C
 P1 compressor input = 700.0 kPa
 s1 compressor input = 1.7471638120291129 kJ/kgK
 x1 compressor input = 2.0 kgvapor/kgtotal
 h2 compressor input = 436.16772151850734 kJ/kg
 T2 compressor output = 61.50679237571848 derece C
 P2 compressor output = 1500.0 kPa
 s2 compressor output = 1.7616085057473234 kJ/kgK
 x2 compressor output = 1.1195970417376142 kgvapor/kgtotal
 h5 compressor input = 416.1036313700978 kJ/kg
 T5 condenser saturation = 39.026896625615855 derece C
 P5 condenser saturation = 1498.0 kPa
```

s5 condenser saturation = 1.6999540073912813 kJ/kgK
x5 condenser saturation = 0.0 kgvapor/kgtotal
h6 compressor input = 248.2654432673927 kJ/kg
T6 condenser saturation = 38.97310626089193 derece C
P6 condenser saturation = 1496.0 kPa
s6 condenser saturation = 1.1618967780114848 kJ/kgK
x6 condenser saturation = 0.0 kgvapor/kgtotal
h3 turbine input = 241.81162804128735 kJ/kg
T3 turbine input = 33.97310626089193 derece C
P3 turbine input = 1494.0 kPa
s3 turbine input = 1.1418973251732016 kJ/kgK
x3 turbine input = 0.0 kgvapor/kgtotal
h4 turbine output = 243.81162804128735 kJ/kg
T4 turbine output = 10.728386486701817 derece C
s4 turbine output = 1.1541583870802548 kJ/kgK
x4 turbine output = 0.0 kgvapor/kgtotal
Bottom cycle
refrigerant = R134a
m = 0.8152087532019033kg/s
h1 compressor input = 396.66384167379755 kJ/kg
T1 compressor input = -5.329546896665409 derece C
P1 compressor input = 200.0 kPa
s1 compressor input = 1.748643091085814 kJ/kgK
x1 compressor input = 2.0 kgvapor/kgtotal
h2 compressor input = 420.7864712323407 kJ/kg
T2 compressor output = 29.609873382254033 derece C
P2 compressor output = 500.0 kPa
s2 compressor output = 1.7647141124425834 kJ/kgK
x2 compressor output = 1.0706127884130356 kgvapor/kgtotal
h5 compressor input = 407.5782465453429 kJ/kg
T5 condenser saturation = 15.582823738854056 derece C
P5 condenser saturation = 498.0 kPa
s5 condenser saturation = 1.720567043376862 kJ/kgK
x5 condenser saturation = 0.0 kgvapor/kgtotal
h6 compressor input = 221.13917141892864 kJ/kg
T6 condenser saturation = 15.457077674540868 derece C
P6 condenser saturation = 496.0 kPa
s6 condenser saturation = 1.0747486638109192 kJ/kgK
x6 condenser saturation = 0.0 kgvapor/kgtotal
h3 turbine input = 214.23656417848645 kJ/kg
T3 turbine input = 10.457077674540868 derece C
P3 turbine input = 494.0 kPa
s3 turbine input = 1.050685283471824 kJ/kgK
x3 turbine input = 0.0 kgvapor/kgtotal
h4 turbine output = 216.23656417848645 kJ/kg
T4 turbine output = -10.585295785211429 derece C
s4 turbine output = 1.0638402933582916 kJ/kgK
x4 turbine output = 0.0 kgvapor/kgtotal

4.3 BRAYTON-ERICSSON CYCLE

Brayton Cycle was investigated as a power cycle in the previous chapter. If cycle operates in inverse direction a gas cycle for refrigeration is defined. Brayton cycle used in airplanes as air conditining cycle.



$W_{compressor} = m(h_2 - h_1)$ (4.3.1)

$W_{isentropiccompressor} = m(h_{2s} - h_1)$ $s_{2s} = s_1$ (4.3.2)

$\eta_{isentropiccompressor} = \frac{W_{compressor}}{W_{isentropiccompressor}}$ (4.3.3)

$Q_{heatexchanger} = m(h_2 - h_3)$ $P_2 = P_3$ (4.3.4)

$W_{turbine} = m(h_3 - h_4)$ (4.3.5)

$W_{isentropicturbine} = m(h_3 - h_{4s})$ $s_3 = s_{4s}$ $P_4 = P_1$ (4.3.6)

$Q_{refrigeration} = m(h_1 - h_4)$ (4.3.7)

$\eta_{isentropicturbine} = \frac{W_{turbine}}{W_{isentropicturbine}}$ (4.3.8)

$COP_{heatexchanger} = \frac{Q_{heatexchanger}}{W_{compressor} - W_{turbine}}$ (4.3.9)

$COP_{refrigeration} = \frac{Q_{refrigeration}}{W_{compressor} - W_{turbine}}$ (4.3.10)

Program 4.3.1 cascade refrigeration cycle test program

```
//Brayton-Ericsson refrigeration cycle
public class brayton3
{
Gas g;
double T[];
double P[];
double s[];
double h[];
double Cp[];
double etakompressor;
double etaturbin;
double Wturbin,Wkompresor,Wnet,Q,Qref,COP,COPref,T1,P1,T2,P2,T3,P3,T4,P4;
public brayton3(double m,double T1,double P1, double P2, double T3,double etakompressori,double
etaturbini)
{ set(m,T1,P1,P2,T3,etakompressori,etaturbini);}

public double set(double m,double T1,double P1, double P2, double T3,double etakompressori,double
etaturbini)
{
 T=new double[6];
 P=new double[6];
 s=new double[6];
 h=new double[6];
 Cp=new double[6];
 g=new Gas("air");
 g.mole=false;
 P[1]=P1;
 P[2]=P2;
```

```
  P[4]=P1;
  P[3]=P2;
  T[1]=T1;
  T[3]=T3;
  etakompressor=etakompressori;
  etaturbin=etaturbini;
// 1
double a1[]=g.property(T[1],P[1]);
h[1]=a1[3];
s[1]=a1[5];
double pr1=a1[17];
// 2i = 0
double pr2=P[2]/P[1]*pr1;
T[0]=g.Ti(pr2,P[2]);
a1=g.property(T[0],P[2]);
h[0]=a1[3];
s[0]=a1[5];
Cp[0]=a1[9];
// 2
h[2]=(h[0]-h[1])/etakompressor+h[1];
T[2]=(h[2]-h[0])/Cp[0]+T[0];
a1=g.property(T[2],P[2]);
s[2]=a1[5];
//3
a1=g.property(T[3],P[3]);
h[3]=a1[3];
s[3]=a1[5];
double pr3=a1[17];
double pr4=P1/P2*pr3;
//4i = 5
T[5]=g.Ti(pr4,P[1]);
a1=g.property(T[5],P[1]);
h[5]=a1[3];
s[5]=a1[5];
Cp[5]=a1[9];
//4
h[4]=h[3]-etaturbin*(h[3]-h[5]);
T[4]=(h[4]-h[5])/Cp[5]+T[5];
a1=g.property(T[4],P[4]);
s[4]=a1[5];
Wturbin=m*(h[3]-h[4]);
Wkompresor=m*(h[2]-h[1]);
Q=m*(h[2]-h[3]);
Qref=m*(h[1]-h[4]);
Wnet=Wkompresor-Wturbin;
COP=Q/Wnet;
COPref=Qref/Wnet;
return T[4];
}
public double[][] line1(double T1,double T2,double P,int n)
{
double TT1[]=new double[n];
double ss1[]=new double[n];
double aa[];

for(int i=0;i<n;i++)
{TT1[i]=T1+i*(T2-T1)/(n-1);
 aa=g.property(TT1[i],P);
```

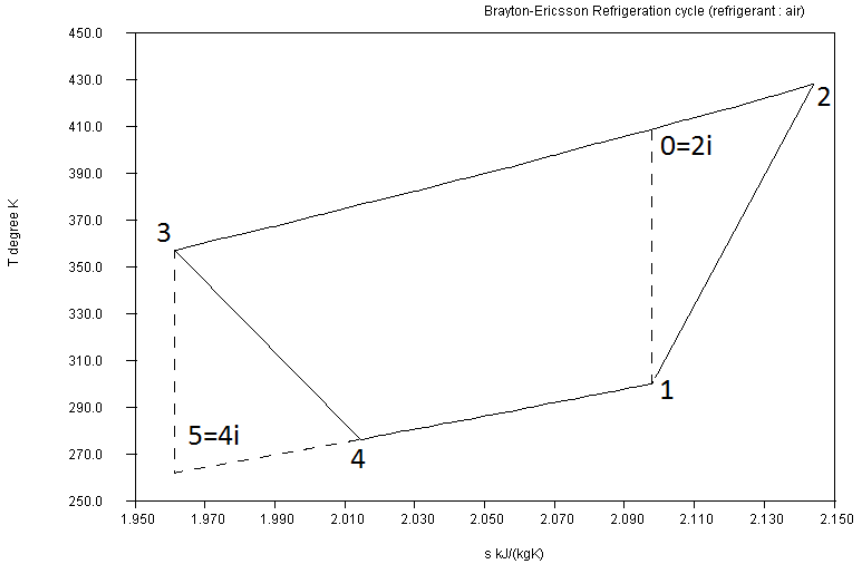
```
 ss1[i]=aa[5];
}
double a[][]={TT1,ss1};
return a;
}
public String toString()
{
String s=" Brayton cycle \n";
s+="Turbine power output W turbine = "+Wturbin+" kW \n";
s+="Compressor power input W kompresor = "+Wkompresor+" kW\n";
s+="Net power output  Wnet = "+Wnet+" kW\n";
s+="Heat exchanger heat removal Q  = "+Q+" kW\n";
s+="refrigeration heat removal Qref = "+Qref+" kW \n";
s+="COP refrigeration  = "+COP+"\n";
s+="COP refrigeration  = "+COP+"\n";
s+="Compressor inlet temperature = "+(T[1]-273.15)+" degree C\n";
s+="Compressor inlet pressure  = "+P[1]+" kPa\n";
s+="Compressor exit temperature = "+(T[2]-273.15)+" degree C\n";
s+="Compressor exit  pressure   = "+P[2]+" kPa\n";
s+="Turbine inlet temperature = "+(T[3]-273.15)+" degree C\n";
s+="Turbine inlet pressure   = "+P[3]+" kPA\n";
s+="Turbine exit temperature = "+(T[4]-273.15)+" degree C\n";
s+="Turbine exit pressure   = "+P[4]+" kPa\n";
return s;
}
public void plot()
{
 double a23[][]=line1(T[2],T[3],P[2],15);

 double tt[]=a23[0];
 double ss[]=a23[1];
 Plot pp=new Plot(ss,tt);
 double a41[][]=line1(T[4],T[1],P[1],15);
 tt=a41[0];
 ss=a41[1];
 pp.addData(ss,tt);
 double t1[]={T[1],T[2]};
 double s1[]={s[1],s[2]};
 pp.addData(s1,t1);
 double t2[]={T[1],T[0]};
 double s2[]={s[1],s[0]};
 pp.addData(s2,t2,1);
 double t3[]={T[3],T[4]};
 double s3[]={s[3],s[4]};
 pp.addData(s3,t3);
 double t4[]={T[3],T[5],T[4]};
 double s4[]={s[3],s[5],s[4]};
 pp.addData(s4,t4,1);
 pp.plot();
}}
```

```
 public class brayton3test
{
  public static void main(String arg[])
  {
                double mhava=0.124; //kg/s air inlet mass flow rate
                double T1=300.0; //derece K air inlet temperature
                double P1=1.01325;// bar air inlet pressure
```

```
            double P2=3;// bar compressor exit pressure
            double T3=357.0; //derece K turbine inlet temperature
            double etakompresor=0.85; //compressor isentropic efficiency
            double etaturbin=0.85; ///turbine isentropic efficiency
    brayton3 bry=new brayton3(mhava,T1,P1,P2,T3,etakompresor,etaturbin);
    System.out.println(bry.toString());
    bry.plot();
  }
}
```

```
---------- Capture Output ----------
> "C:\java\bin\javaw.exe" brayton3test
 Brayton cycle
Turbine power output W turbine = 10.082410266562684 kW
Compressor power input W kompresor = 15.982194445826448 kW
Net power output  Wnet = 5.899784179263763 kW
Heat exchanger heat removal Q  = 8.8694836816727 kW
refrigeration heat removal Qref = 2.969699502408937 kW
COP refrigeration  = 1.5033573114160128
COP refrigeration  = 1.5033573114160128
Compressor inlet temperature = 26.850000000000023 degree C
Compressor inlet pressure  = 1.01325 kPa
Compressor exit temperature = 154.517442534128 degree C
Compressor exit  pressure   = 3.0 kPa
Turbine inlet temperature = 83.85000000000002 degree C
Turbine inlet pressure   = 3.0 kPA
Turbine exit temperature = 2.9618911756290345 degree C
Turbine exit pressure   = 1.01325 kPa
```



If a gas is to be liquified, its temperature must be reduced to a value below its critical temperature (for example for nitrogen the critical temperature is -147 C). Gases to be liquified is used as the refrigerant of the cycle, liquified part removed from cycle and new fresh gas is added in equal amount. Due to required very low temperature to liquified gases such as Nitrohen, Oxygen, helium etc, very high pressures are required. This consumes coniderable amount of energy. For that reason multisage compressors with intercoolers can be used to reduced total energy requirement for compression. Linde liquifaction cycle is given below as an example.

If a gas is to be liquified, its temperature must be reduced to a value below its critical temperature (for example for nitrogen the critical temperature is -147 C)

$mxh_5 + m(1-x)h_6 = mh_7$ (4.4.1) mixing process (amount added is equal to amount liquid extracted)

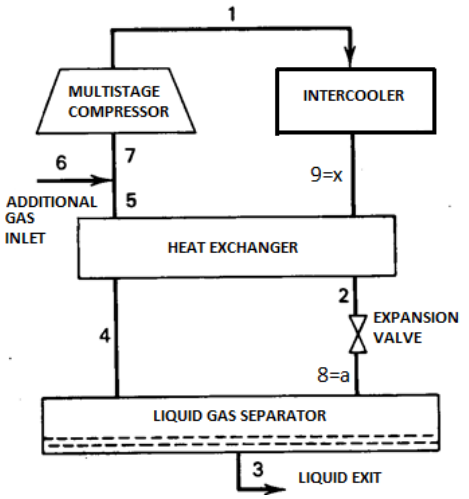
$h_8 = (1-x)h_3 + xh_4$ (4.4.2) saturaded state separation process

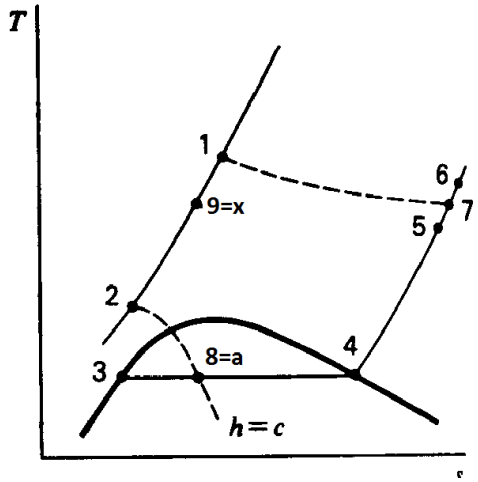
$h_8 = h_2$ (4.4.3) Expansion process

$mx(h_5 - h_4) = m(h_9 - h_2)$ (4.4.4) Heat Exchanger

$W_{compressor} = m\frac{n}{n-1}(P_1v_1 - P_7v_7)$ (4.4.5) Polytropic multistage compressor work output

$q_{compressor} = m(h_1 - h_7) - W_{compressor}$ (4.4.6) Polytropic multistage compressor heat output





Program 4.4.1 liquifying cycle program

```
public class ref_cycle5
{
// gas liquifyning cylcle
// refrigerantEN b=new refrigerantEN("R728");
// double a[]=b.property("tp",30.0,1.01325);
///---- String s values and meanings --------
//    tv=temperature-specific volume
//    tp=temperature-pressure
```

```
//    th=temperature-enthalpy
//    tu=temperature-internal energy
//    ts=temperature-entropy
//    tx=temperature quality
//    pv=pressure-specific volume
//    pt=pressure-temperature
//    ph=pressure-enthapy
//    pu=pressure-internal energy
//    ps=pressure-entropy
//    px=pressure-quality
//    vp=specific volume-pressure
//    vt=specific volume-temperature
//---- output values (method property) ---------
// r[0] P Pressure kPa
// r[1] t temperature degree C
// r[2] v specific volume m^3/kg
// r[3] h enthapy KJ/kg
// r[4] u internal energy KJ/kg
// r[5] s entropy KJ/kgK
// r[6] x quality kg vapour/kg total
// r[7] ro density kg/m^3

double T[];
double P[];
double h[];
double s[];
double x[];
double v[];
double xx;//quality at liquid separator inlet
boolean supercritical=false;
String refname; //refrigerant name
double Tc,Pc;
double n;
double COP_condenser,COP_evaporator;
double m;//kg/s mass flow rate of refrigerant
double mliquid;//liquid extraction rate
double Wc;//net compressor work
double Qcompressor;
double Qintercooler;
double Qheatexchanger;
double Qevaporator,Qcondenser; //condenser and evaporator heat transfer
refrigerantEN b; //refrigerantEN class object for properties
public ref_cycle5(String refnamei,double mi,double P4,double P1,double T5,double T6,double
xi,double ni)
{
m=mi;
n=ni;
xx=xi;
refname=refnamei;    // refrigerant name
b=new refrigerantEN(refname);
T=new double[14];
P=new double[14];
h=new double[14];
s=new double[14];
x=new double[14];
v=new double[14];
Tc=b.r.Tc;//critical temperature degree C
Pc=b.r.Pc;//critical pressure kPa
```

```
P[3]=P4;
P[4]=P4;
P[8]=P4;
P[6]=P4;
P[5]=P4;
P[7]=P4;
P[1]=P1;
P[2]=P1;
P[9]=P1;//point x
T[5]=T5;
T[6]=T6;
}

public void cycle()
{
// separator saturation point 3 x=0
double a1[]=b.property("px",P[3],0.0);
T[3]=a1[1];
h[3]=a1[3];
s[3]=a1[5];
x[3]=a1[6];
v[3]=a1[2];
// separator saturation point 4 x=0
a1=b.property("px",P[4],1.0);
T[4]=a1[1];
h[4]=a1[3];
s[4]=a1[5];
x[4]=a1[6];
v[4]=a1[2];
// 8=a separator inlet point
a1=b.property("px",P[4],xx);
T[8]=a1[1];
h[8]=a1[3];
s[8]=a1[5];
x[8]=a1[6];
v[8]=a1[2];
// 2 heat exchanger exit-expansion inlet
h[2]=h[8];
a1=b.property("ph",P[2],h[2]);
T[2]=a1[1];
s[2]=a1[5];
x[2]=a1[6];
v[2]=a1[2];
// separator saturation point 5 x=0
a1=b.property("tp",T[5],P[5]);
h[5]=a1[3];
s[5]=a1[5];
x[5]=a1[6];
v[5]=a1[2];
// additional gas inlet point 6
a1=b.property("tp",P[6],T[6]);
h[6]=a1[3];
s[6]=a1[5];
x[6]=a1[6];
v[6]=a1[2];
// 9=x intercooler exit
h[9]=xx*(h[5]-h[4])+h[2];
a1=b.property("ph",P[9],h[9]);
```

```
T[9]=a1[1];
s[9]=a1[5];
x[9]=a1[6];
v[9]=a1[2];

// multistage compressor inlet point 7
h[7]=(1-xx)*h[6]+xx*h[5];
a1=b.property("ph",P[7],h[7]);
T[7]=a1[1];
h[7]=a1[3];
s[7]=a1[5];
x[7]=a1[6];
v[7]=a1[2];

// multistage compressor exit point 1
v[1]=Math.pow((P[7]/P[1]),(1.0/n))*v[7];
double TT1=P[1]*1e3*v[1]/(8314.5)-273.15;
a1=b.property("pv",P[1],v[1]);
T[1]=a1[1];
h[1]=a1[3];
s[1]=a1[5];
x[1]=a1[6];
v[1]=a1[2];
xx=(h[7]-h[6])/(h[5]-h[6]);
//point 8=a liquid separator inlet
a1=b.property("px",P[8],xx);
T[8]=a1[1];
h[8]=a1[3];
s[8]=a1[5];
x[8]=a1[6];
v[8]=a1[2];
//point 2 Heat exchanger exit
h[2]=h[8];
a1=b.property("ph",P[2],h[2]);
T[2]=a1[1];
h[2]=a1[3];
s[2]=a1[5];
x[2]=a1[6];
v[2]=a1[2];
 // multistage polytropic compressor
Wc=m*n/(n-1)*(P[1]*v[1]-P[7]*v[7]);
Qcompressor=m*(h[1]-h[7]);
Qintercooler=m*(h[1]-h[9]);
Qheatexchanger=m*(h[9]-h[2]);
mliquid=(1.0-xx)*m;//kg/s liquid extraction rate
}

public String toString()
{
cycle();
String ss="gas liquifying plant\n";
ss+="polytropic constant n="+n+"\n";
ss+="liquid extraction rate : "+mliquid+"\n";
ss+=" multistage compressor work = "+Wc+" kW\n";
ss+=" multistage compressor heat transfer = "+Qcompressor+" kW\n";
ss+=" intercooler heat transfer = "+Qintercooler+" kW\n";
ss+=" heat exchanger heat transfer = "+Qheatexchanger+" kW\n";
```

```
ss+=" h7 compressor inlet = "+h[7] +" kJ/kg\n";
ss+=" T7 compressor inlet = "+T[7] +" degree C\n";
ss+=" P7 compressor inlet = "+P[7] +" kPa \n";
ss+=" s7 compressor inlet = "+s[7] +" kJ/kgK\n";
ss+=" x7 compressor inlet = "+x[7] +" kgvapor/kgtotal"+"\n";
ss+=" v7 compressor inlet = "+v[7] +" m^3/kg"+"\n";
ss+=" h1 compressor exit = "+h[1] +" kJ/kg\n";
ss+=" T1 compressor exit = "+T[1] +" degree C\n";
ss+=" P1 compressor exit = "+P[1] +" kPa \n";
ss+=" s1 compressor exit = "+s[1] +" kJ/kgK\n";
ss+=" x1 compressor exit = "+x[1] +" kgvapor/kgtotal"+"\n";
ss+=" v1 compressor inlet = "+v[1] +" m^3/kg"+"\n";
ss+=" h6 gas feed = "+h[6] +" kJ/kg\n";
ss+=" T6 gas feed = "+T[6] +" degree C\n";
ss+=" P6 gas feed = "+P[6] +" kPa \n";
ss+=" s6 gas feed = "+s[6] +" kJ/kgK\n";
ss+=" x6 gas feed = "+x[6] +" kgvapor/kgtotal"+"\n";
ss+=" v6 compressor inlet = "+v[6] +" m^3/kg"+"\n";

ss+=" h5 heat exchager exit = "+h[5] +" kJ/kg\n";
ss+=" T5 heat exchager exit = "+T[5] +" degree C\n";
ss+=" P5 heat exchager exit = "+P[5] +" kPa \n";
ss+=" s5 heat exchager exit = "+s[5] +" kJ/kgK\n";
ss+=" x5 heat exchager exit = "+x[5] +" kgvapor/kgtotal"+"\n";
ss+=" v5 compressor inlet = "+v[5] +" m^3/kg"+"\n";

ss+=" h3 liquid separator liquid discharge = "+h[3] +" kJ/kg\n";
ss+=" T3 liquid separator liquid discharge = "+T[3] +" degree C\n";
ss+=" P3 liquid separator liquid discharge = "+P[3] +" kPa \n";
ss+=" s3 liquid separator liquid discharge = "+s[3] +" kJ/kgK\n";
ss+=" x3 liquid separator liquid discharge = "+x[3] +" kgvapor/kgtotal"+"\n";
ss+=" v3 compressor inlet = "+v[3] +" m^3/kg"+"\n";

ss+=" h4 liquid separator gas discharge = "+h[4] +" kJ/kg\n";
ss+=" T4 liquid separator gas discharge = "+T[4] +" degree C\n";
ss+=" P4 liquid separator gas discharge = "+P[4] +" kPa \n";
ss+=" s4 liquid separator gas discharge = "+s[4] +" kJ/kgK\n";
ss+=" x4 liquid separator gas discharge = "+x[4] +" kgvapor/kgtotal"+"\n";
ss+=" v4 compressor inlet = "+v[4] +" m^3/kg"+"\n";

ss+=" h2 expansion valve inlet = "+h[2] +" kJ/kg\n";
ss+=" T2 expansion valve inlet = "+T[2] +" degree C\n";
ss+=" P2 expansion valve inlet = "+P[2] +" kPa \n";
ss+=" s2 expansion valve inlet = "+s[2] +" kJ/kgK\n";
ss+=" x2 expansion valve inlet = "+x[2] +" kgvapor/kgtotal"+"\n";
ss+=" v2 compressor inlet = "+v[2] +" m^3/kg"+"\n";
return ss;
}

}
```

Program 4.4.1 liquifying cycle test program

```
import javax.swing.*;
public class ref_cycle5test
{
public static double Pres(String refrigerant,double T)
{refrigerantEN r=new refrigerantEN(refrigerant);
double a[]=r.property("tx",T,0.0);
```

```
double P1=a[0];
return P1;
}
public static void main(String arg[])
{
double m=1.0;  // kg/s çevrim R134a debisi
String refrigerant="R728";
refrigerantEN r=new refrigerantEN(refrigerant);
double P4=Pres(refrigerant,-195.0);//kPA
double P1=4000.0;//kPA
double T5=30.0;
double T6=40.0;
double T7=5.0;
double T9=120.0;
double n=1.2; //compressor polytropic constant
double x=0.99;
ref_cycle5 re=new ref_cycle5(refrigerant,m,P4,P1,T5,T6,x,n);
re.cycle();
System.out.println(re.toString());
}
}
```

```
---------- Capture Output ----------
> "C:\java\bin\javaw.exe" ref_cycle5test
gas liquifying plant
polytropic constant n=1.2
liquid extraction rate : 0.009999999999999898
 multistage compressor work = 442.2351408982641 kW
 multistage compressor heat transfer = 4155.660889416219 kW
 intercooler heat transfer = 4160.854857184115 kW
 heat exchanger heat transfer = 234.54009562200986 kW
 h7 compressor inlet = 315.50298302351456 kJ/kg
 T7 compressor inlet = 30.811638650885733 degree C
 P7 compressor inlet = 111.16014521822753 kPa
 s7 compressor inlet = 7.21501566705719 kJ/kgK
 x7 compressor inlet = 2.1998956109055205 kgvapor/kgtotal
 v7 compressor inlet = 0.811592005587896 m^3/kg
 h1 compressor exit = 4471.1638724397335 kJ/kg
 T1 compressor exit = 279.13446740548966 degree C
 P1 compressor exit = 4000.0 kPa
 s1 compressor exit = 11.238933130696067 kJ/kgK
 x1 compressor exit = 3.0 kgvapor/kgtotal
 v1 compressor inlet = 0.0409806355038701 m^3/kg
 h6 gas feed = 399.00272119066653 kJ/kg
 T6 gas feed = 40.0 degree C
 P6 gas feed = 111.16014521822753 kPa
 s6 gas feed = 7.086179046532035 kJ/kgK
 x6 gas feed = 3.0 kgvapor/kgtotal
 v6 compressor inlet = 2.851609538861844 m^3/kg
 h5 heat exchager exit = 314.65955132485647 kJ/kg
 T5 heat exchager exit = 30.0 degree C
 P5 heat exchager exit = 111.16014521822753 kPa
 s5 heat exchager exit = 7.213029510233074 kJ/kgK
 x5 heat exchager exit = 3.0 kgvapor/kgtotal
 v5 compressor inlet = 0.8094248860775289 m^3/kg
 h3 liquid separator liquid discharge = -120.39405559892191 kJ/kg
 T3 liquid separator liquid discharge = -194.99981112354098 degree C
 P3 liquid separator liquid discharge = 111.16014521822753 kPa
```

```
s3 liquid separator liquid discharge = 2.854994391306452 kJ/kgK
x3 liquid separator liquid discharge = 0.0 kgvapor/kgtotal
v3 compressor inlet = 0.0012462512632194695 m^3/kg
h4 liquid separator gas discharge = 77.75036382787654 kJ/kg
T4 liquid separator gas discharge = -194.99981112354098 degree C
P4 liquid separator gas discharge = 111.16014521822753 kPa
s4 liquid separator gas discharge = 5.390444248770784 kJ/kgK
x4 liquid separator gas discharge = 1.0 kgvapor/kgtotal
v4 compressor inlet = 0.19906500474458172 m^3/kg
h2 expansion valve inlet = 75.76891963360858 kJ/kg
T2 expansion valve inlet = -155.6449303020083 degree C
P2 expansion valve inlet = 4000.0 kPa
s2 expansion valve inlet = 4.6553785094688775 kJ/kgK
x2 expansion valve inlet = 3.0 kgvapor/kgtotal
v2 compressor inlet = 0.008713563166911232 m^3/kg

> Terminated with exit code 0.
```

4.5 ABSORPTION REFRIGERATION

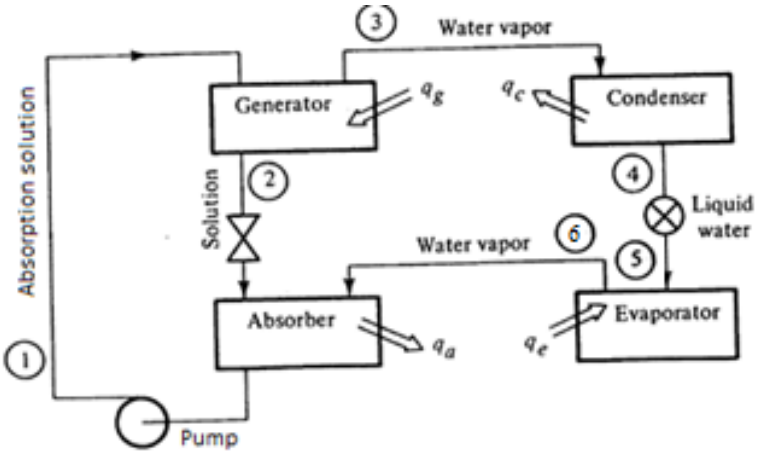
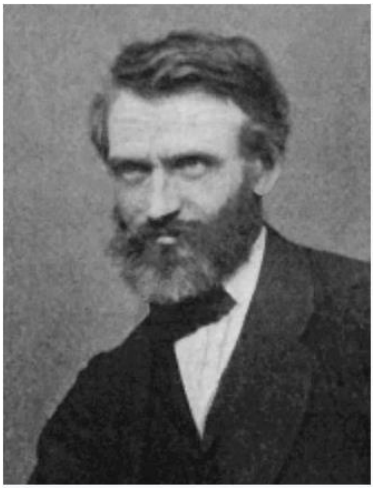


Figure 4.5.1 A simple absorption refrigeration cycle



Ferdinand Carre

Ferdinand Carre invented the absorbtion system and took a United States patent in 1860. The absorbtion cycle is similar in certain respects to the vapor compression cycle. A simple absorption refrigeration cycle is shown in the figure above. In order to pressurize low pressure vapor from evaporator to high pressure of condenser, simple refrigeration cycle

utilise a compressor. In absorbtion cycle, the same duty will be done by a generator, an absorber and a liquid pump. The absorbtion system first absorbs the low pressure vapor in an apropriate absorption liquid. Heat is generated by the absorption process. This heat be removed from the absorber by a heat echange system. Absorber fluid is then pressurised by a liquid pump and send to a generator. Liquid heated up in generator and released vapour is sent the condenser. Most common absorption pairs are Lithium Bromide-Water, Ammonia-Water and Lithium Chloride-Water. Cycle can be better explained as an example process. Let us assume LiBr-Water pair is used for the cycle given in above figure. If pump absorption solution mass flow rate m_1=0.6 kg/s and Absorber temperature $T_{absorber}$=100 C and Generator temperature $T_{generator}$=100 C, calculate the cycle. Condenser (saturation) temperature is 40 C and evaporator temperature is 10 C.

Total mass flow balance:

$m_2 + m_3 = m_1$ (4.5.1)

LiBr balance:

$m_1 w_1 = m_2 w_2$ (4.5.2)

$Q_{generator} = m_3 h_3 + m_2 h_2 - m_1 h_1$ (4.5.3)

$Q_{condenser} = m_3 (h_3 - h_4)$ (4.5.4)

$Q_{evaporator} = m_3 (h_5 - h_6)$ (4.5.5)

$Q_{absorberr} = m_2 h_2 + m_3 h_6 - m_1 h_1$ (4.5.6)

Program 4.5.1 absorbtion cycle

```
 public class LiBrproptest
{
public static void main(String arg[])
{LiBr_H2O lip=new LiBr_H2O();
double t0=30.0;//degree C Absorber
double t2=100.0;//degree C Generator
double t3=40.0;//degree C Condenser
double t4=t3;
double t5=10.0;//degreee C Evaporator
double m1=0.6; //kg/s LiBr-Water solution
 double a[]=lip.st.property("tx",t3,0.0);
 double P3=a[0]*100;//water vapor (saturated)
 double P2=lip.Psw(t3);//generator
 double w2=lip.W(t2,P2);//generator
 double P5=lip.Psw(t5);//saturated water
 double P6=P5;// water vapor
 double P0=P5;// water vapor
 double P1=P2;
 double w1=lip.W(t0,P0);//absorber-pump
 double m2=m1*w1/w2;//generator exit
 System.out.println("P3="+P3+" kPa condenser=");
 System.out.println("P2="+P2+" kPa generator w2="+w2+"kg LiBr/kg mix");
 System.out.println("P5=P6="+P5+" kPA evaporator ");
 System.out.println("P1="+P1+" kPa generator w1="+w1+"kg LiBr/kg mix");
 System.out.println("m1="+m1+" kg/s solution absorber exit");
 System.out.println("m2="+m2+" kg/s solution generator exit");
 double m3=m1-m2;//water vapor going to condenser
 System.out.println("m3="+m3+" kg/s water vapor generator exit");
 double h0=lip.h_w(t0,w1);//absorber
 System.out.println("h0="+h0+" kJ/kg solution absorber");
 double ro1=lip.ro_w(t0,w1);
 double v1=1.0/ro1;
```

```
System.out.println("v1="+v1+"m^3/kg"+"ro1="+ro1+"kg/m^3");
double dh=v1*(P1-P0);//enthalpy change in pump
double h1=h0+dh;
System.out.println("h0="+h0+"h1="+h1+" kJ/kg pump output");
double h2=lip.h_w(t2,w2);//Generator
System.out.println("h2="+h2+" kJ/kg solution generator");
a=lip.st.property("tp",t2,P3*1e-2);
double h3=a[3];// kJ/kg superheated water at t2,P3
System.out.println("h3="+h3+" kJ/kg generator exit superheated water");
a=lip.st.property("tx",t4,0.0);
double h4=a[3]; // kJ/kg saturated liquid water at t3
System.out.println("h4="+h4+" kJ/kg condenser  exit saturated water");
double h5=h4;
a=lip.st.property("ph",P5*1e-2,h5);
double x5=a[5];
a=lip.st.property("tx",t5,1.0);
double h6=a[3];
System.out.println("h5="+h5+" kJ/kg evaporator inlet saturated water");
System.out.println("x5="+x5+" kg vapor/kg evaporator inlet saturated water");
System.out.println("h6="+h6+" kJ/kg  saturated water vapor absorption inlet");
double Qgenerator=m3*h3+m2*h5-m1*h1;
double Qabsorber=m2*h2+m3*h6-m1*h1;
double Qcondenser=m3*(h3-h4);
double Qevaporator=m3*(h6-h5);
double Wpump=m1*(h1-h0);
System.out.println("Qgenerator="+Qgenerator+" kw  ");
System.out.println("Qabsorber="+Qabsorber+" kw  ");
System.out.println("Qcondenser="+Qcondenser+" kw  ");
System.out.println("Qevaporator="+Qevaporator+" kw  ");
System.out.println("Wpump="+Wpump+" kw  ");
double COP=Qevaporator/(Wpump+Qgenerator);
 System.out.println("COP="+COP+" ");
}}
```

```
---------- Capture Output ----------
> "C:\java\bin\javaw.exe" LiBrproptest
P3=7.383294297190836 kPa condenser=
P2=7.383294297190836 kPa generator w2=0.6652288543991745kg LiBr/kg mix
P5=P6=1.227529277542396 kPA evaporator
P1=7.383294297190836 kPa generator w1=0.4913386036641896kg LiBr/kg mix
m1=0.6 kg/s solution absorber exit
m2=0.4431605157367623 kg/s solution generator exit
m3=0.15683948426323768 kg/s water vapor generator exit
tt=30.0v=0.0010043347674437307hsl=125.78774901764382
h0=60.434860678227906 kJ/kg solution absorber
roo=51400.89250314854row=1516.593156380927
v1=6.593726180238856E-4m^3/kgro1=1516.593156380927kg/m^3
h0=60.434860678227906h1=60.43891962112485 kJ/kg pump output
tt=100.0v=0.0010435090224857138hsl=419.0369217541938
h2=268.6248377904894 kJ/kg solution generator
h3=2687.789139508786 kJ/kg generator exit superheated water
h4=167.55922426315783 kJ/kg condenser  exit saturated water
h5=167.55922426315783 kJ/kg evaporator inlet saturated water
x5=0.5943952146183389 kg vapor/kg evaporator inlet saturated water
h6=2519.754780813521 kJ/kg  saturated water vapor absorption inlet
Qgenerator=459.54374291712725 kw
Qabsorber=477.97761017488267 kw
Qcondenser=395.2715601319075 kw
```

```
Qevaporator=368.9171379756383 kw
Wpump=0.0024353657381681157 kw
COP=0.8027857817339044

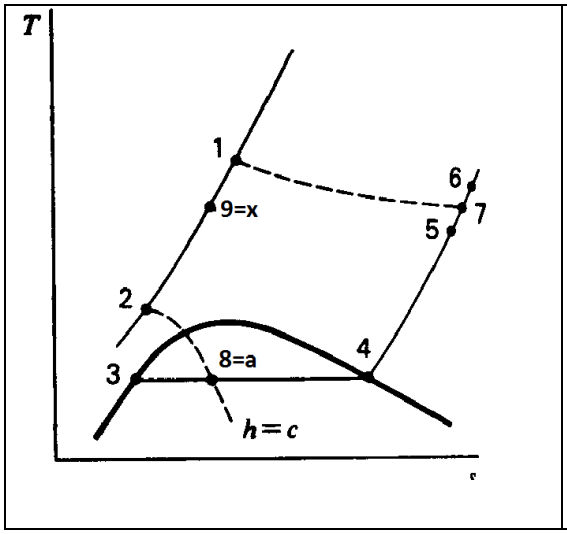
> Terminated with exit code 0.
```

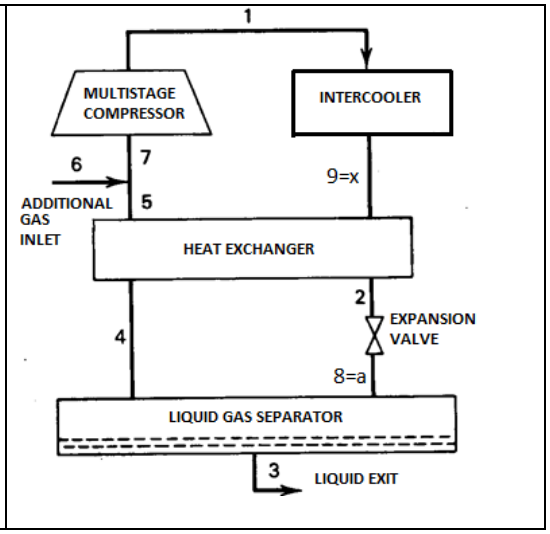
PROBLEMS:

PROBLEM 1: In a tank air is existed at P=1000 kPa pressure and T=300 K temperature. This air will be used for cooling purposes. In order to achive this, air will be expanded to P=100 kPa by using throttling process (constant enthalpy process). Calculate the exit temperature of air. Use air_NIST as a real EOS for your calculation.

PROBLEM 2: Flash chamber refrigeration cycle uses R134a as the refrigerant. Evaporator saturation temperature is -5 ℃, Condenser saturation temperature is 45 ℃, Flash chamber saturation temperature is 20 ℃ . Isentropic efficiencies of bothh compressors are 85%. Assume 5 ℃ superheat at evaporator exirt and 5 ℃ of condenser subcooling at condenser exit. Assume that there are no pressure drop at condenser and evaporator. Calculate cycle and COP values of the cycle.

PROBLEM 3: Oxygen (R732) to be liquified by using Linde cycle. Compressor compressed politropically (Pv^n=const) with a polyropical constant n=1.1. Saturation temperature is T=-200 ℃. Compresor output pressure is 4500 kPa. Evaporator inlet dryness fraction x_8=0.97. Oxygen inlet temperature is T_6=40 ℃ and heat exchanger exit temperature T_5= 30 ℃, Compressur inlet temperature T_7= 35 ℃, Calculate process parameters.





5.0 AIR CONDITIONING

5.1 PROPERTIES OF WET AIR

Equation of states for dry air and water was investigated previously.The two fluid can be mixed to give properties of wet air. Details of wet air properties are investigated in every thermodynamics book. Mixing rulle usually assumes ideal gas for each gas, but water properties can be obtained by using actual eqaution of state. Wet air properties used in air conditioning and refrigeration applications. The basic mixing rule is based on adding up partial pressures of each component to obtain total pressure of the system.

$P = P_{dryair} + P_{watervapor}$ (5.1.1)

Total mass of the gas is found by summing the mass of the component gases

$m = m_{dryair} + m_{watervapor}$ (5.1.2), The ratio of water vapor to dry air gives us specific moisture

The specific moisture can be calculated by using ideal gas law

$P_{dryair}V = m_{dryair}(R/M_{dryair})T$ (5.1.3),

$P_{watervapor}V = m_{watervapor}(R/M_{watervapor})T$ (5.1.4),

In these terms, P is pressure, m is mass, T is temperature in degree Kelvin. From these equations:

$w = \left(\frac{M_{watervapor}}{M_{dryair}}\right)\frac{P_{watervapor}}{P-P_{watervapor}}$ (5.1.5),

where ratio of molar masses are equal to:

$$\left(\frac{M_{watervapor}}{M_{dryair}}\right) = \frac{18.016}{28.964197} = 0.6220093$$

water vapor molar mass to ratio of saturated water vapor molar mass at same temperature.

$\varphi = \frac{x_{watervopor}}{x_{saturatedwatervopor}} = \frac{P_{watervopor}}{P_{saturatedwatervopor}}$ (5.1.6),

Degree of saturation $DOS = \frac{m_{watervapor}}{m_{saturatedwatervapor}} = \frac{w_{watervapor}}{w_{saturatedwatervapor}}$ (5.1.7),

The enthalpy of wet air can be calculated summing partial enthalpies

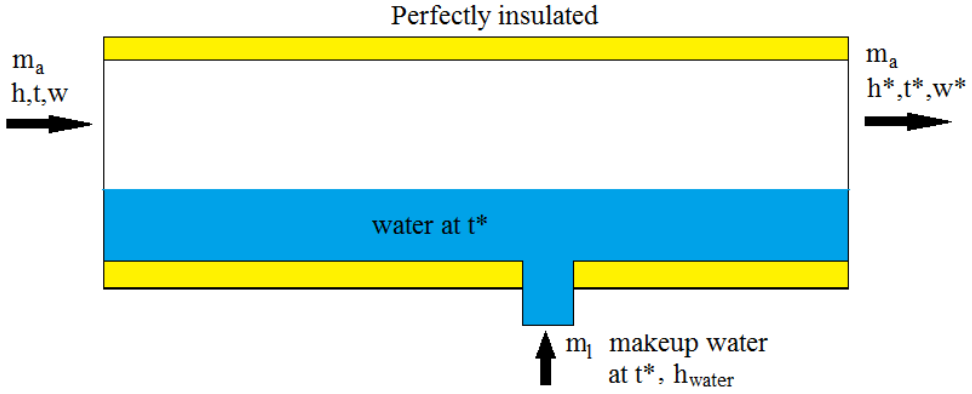
$h = \frac{H}{m_{dryair}} = h_{dryair} + \frac{m_{watervapor}}{m_{dryair}}h_{watervapor} = h_{dryair} + wh_{watervapor}$ (5.1.8),

Entropy is also calculated the same way

$s = \frac{s}{m_{dryair}} = s_{dryair} + \frac{m_{watervopor}}{m_{dryair}}s_{watervapor} = s_{dryair} + ws_{watervapor}$ (5.1.9),

Another concept used for wet air is adyabatic saturation temperature. If air flow through an infinite length channel filled with water at the bottom and all walss are insulated, it will absorb water and will be reached to adyabatic saturation point. The temparature of adyabatic saturation point is also called wet air temperature, it is an idealised thermodynamic concept and can be calculated from the energy balance of the infinitely long channel. Basic energy equation:

Energy of the air entering the channel = enegy of the air leaving the channel + energy of evaporated water,



So $m_a h + m_l h^*_{water} = m_a h^*$

$$m_l = m_a(w * -w)$$

$h + (w^* - w)h^*_{water} = h^*$ (5.1.10),

Wet air temperature is close to the wet bulb temperature which defines similar process as isothermal process rather than adiabatic process. In an isothermal process temperature is depends on the heat transfer coefficient. The last property should be defined is dew point temperature. If air is cooled without changing the pressure, in some temperature condensatiıon point is reached. It is called dew point temperature. In order to calculate properties of wet air a program, wetair.java is developed and codes are given in Appendix 7.

By using this program, thermodynamic properties of humidified air can be calculated by giving a pair (plus air pressure) of thermodynamic properties. Following properties can be defined in the program:
“tdb_twb" dry air - wet air temperature
"tdb_rh" dry air temperature - relative humidity
"tdb_w" dry air temperature - specifice humidity
"tdb_tdew" dry air temperature - çiğ noktası temperature
"tdb_pv" dry air temperature - su buharı basıncı
"tdb_dos" dry air temperature – degree of saturation
"tdb_h" dry air temperature - enthalpy
"tdb_s" dry air temperature - entropy
"w_rh" specific humidity - relative humidity
"w_h" specific humidity - enthalpy
After specifying known thermodynamic couple that properties are given to the program and results are obtained. For example to obtain properties 25 C dry bulb temperature and 20 degree C wet bulb temperature and 1.01325 bar pressure, the following code should be called

```
wetair y=new wetair();
double a[]=y.property("tdb_twb",25.0,20.0,1.0132);
```

A is a vector that carries out all thermodynamic properties. The parameters of a is as follows:

a[0]= P, pressure
a[1]= T, dry bulb temperature
a[2]= v, specific volume
a[3]= Pv partial pressure of water vapor
a[4]= Pa partial pressure of dry air
a[5]= w, specific humidity
a[6]= rh, relative humidity
a[7]= dos, degree of saturation
a[8]= h enthalpy
a[9]= s, entropy
a[10]= dew point temperature
a[11]= ha, enthalpy of dry air
a[12]= hv, enthalpy of water vapor
a[13]= hv*w, latent enthalpy
a[14]= T wet air temperature

A simple program is given to show this call

Program 5.1.1 Properties of wetair

```
public class wetairtest
{
public static String output(double b[])
{
String s="";
for(int i=0;i<b.length;i++)
{s+="a"+i+": "+b[i]+"\n";}
return s;
}
public static void main(String arg[])
{
wetair wa=new wetair();
double a[]=wa.property("tdb_twb",25.0,20.0,1.01325);
System.out.println(output(a));
}
}
```

```
---------- Capture Output ----------
> "C:\java\bin\javaw.exe" wetairtest
a0: 1.01325
a1: 25.0
```

```
a2: 0.844681235079535
a3: 0.020117894070933378
a4: 0.9931321059290666
a5: 0.012599267065824954
a6: 0.6348526592806563
a7: 0.6274558633488576
a8: 57.1893891270598
a9: 0.1983765100615907
a10: 17.59459840312558
a11: 25.092773476196555
a12: 2547.498634894733
a13: 32.09661565086324
a14: 20.0
> Terminated with exit code 0.
```

If only result set is required, and the values will not be used, a simpler call will be:

```
wetair y=new wetair();
System.out.println(y.toString("tdb_twb",25.0,20.0,1.0132));
```

This will give direct output with unit definitions.
Program 5.1.2 calling the properties of wetair

```
public class wetairtest1
{
public static void main(String arg[])
{
wetair wa=new wetair();
String s=wa.toString("tdb_twb",25.0,20.0,1.01325);
System.out.println(s);
}}
```

```
---------- Capture Output ----------
> "C:\java\bin\javaw.exe" wetairtest1
0 P,_pressure_____ 1.01325000000000_bar_____
1 T,_dry_bulb_temperature_____ 25.00000000000000_deg_C_____
2 v,_dry_air_specific_vol_____ 0.84468123507953_m^3/kg_____
3 Pv_water_vapor_pressure_____ 0.02011789407093_bar_____
4 Pa_air_partial_pressure_____ 0.99313210592907_bar_____
5 w,_humidity_ratio_____ 0.01259926706582_kg_vapor/kg_dry_air_____
6 rh,relative_humidity_____ 0.63485265928066_____
7 dos,degree_of_saturation_____ 0.62745586334886_____
8 h_enthalpy_____ 57.18938912705980_KJ/kg_dry_air_____
9 s,_entropy_____ 0.19837651006159_KJ/kg_dry_air_K_____
10 tdew,dew_point_temperature___ 17.59459840312558_deg_C_____
11 ha,_partial_air_enthalpy_____ 25.09277347619656_KJ/kg_dry_air_____
12 hv,partial_water_vap_enthalpy 2547.49863489473300_KJ/kg_dry_air_____
13 hv*w,_latent_enthalpy_____ 32.09661565086324_KJ/kg_dry_air_____
14 T_wet_bulb_temperature_____ 20.00000000000000_deg_C_____
> Terminated with exit code 0.
```

If only results are required from a Graphic user interface program, psTEN.java is prepared for you. Program will open with a pressure input window. When the desired pressure is given a psychrometric diagram will be drawn for us to data entry. Data can be inputted from here by using Mouse, or from the third page manual entry page for more accurate specifications. In both cases all the selected points will be entered to the psychrometric chart and all thermodynamic properties will be listed as a separate

list. Values in this list can directly be copied by using ctr-C and past into any programs (such as MS Word, Open Office write or an editör program)

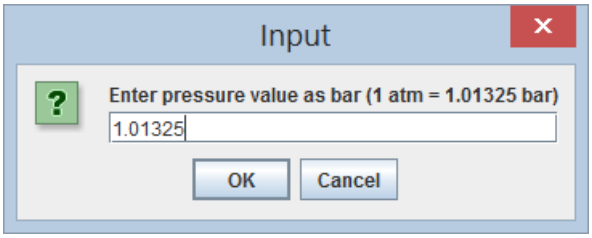


Figure 5.1.1 psTEN.java program entry point

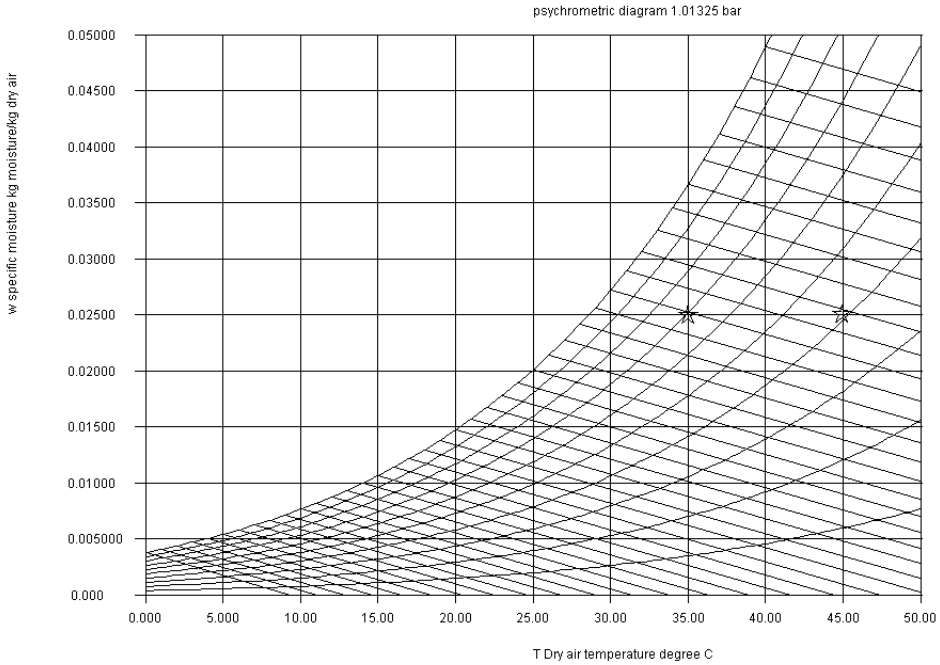


Figure 5.1.2 psTEN.java program dynamic graphic data entry psychrometric diagram

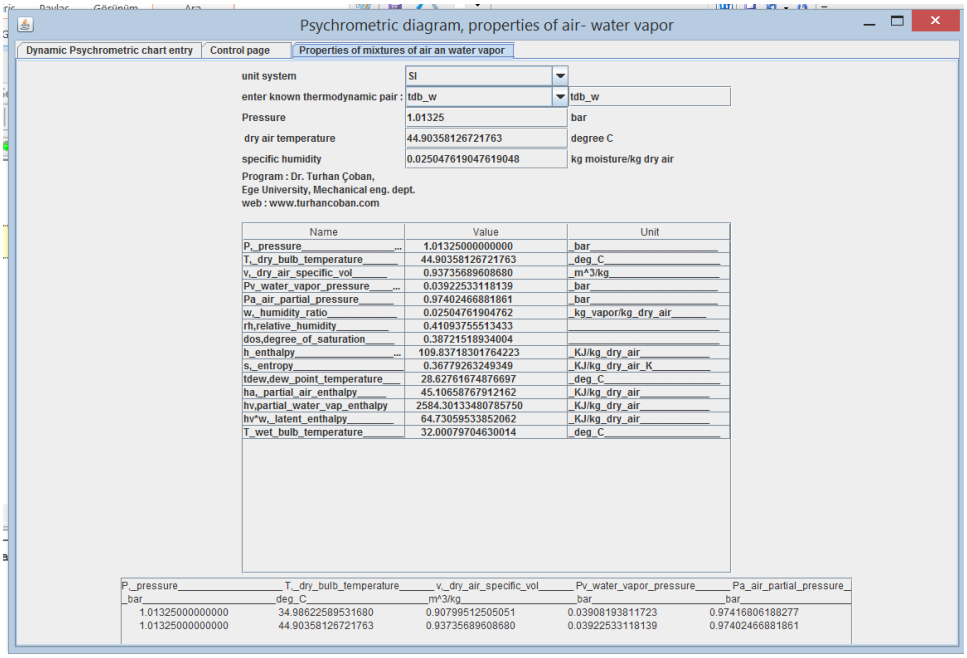


Figure 5.1.3 psTEN.java program data entry and output

Of course, if a certain psychrometric processes to be defined, a basic form of the equation should be utilized rather than a A GUI program.

5.2 AIR CONDITIONING (PSYCHROMETRIC) PROCESSES

In order to keep living area in comfortable conditions, we neded processes called air conditioning processes. It will involve adjusting of moisture content and temperature of the space to comfort conditions. In this chapter, first simple air conditioning processes will be investigated, and then the real process will be looked at.

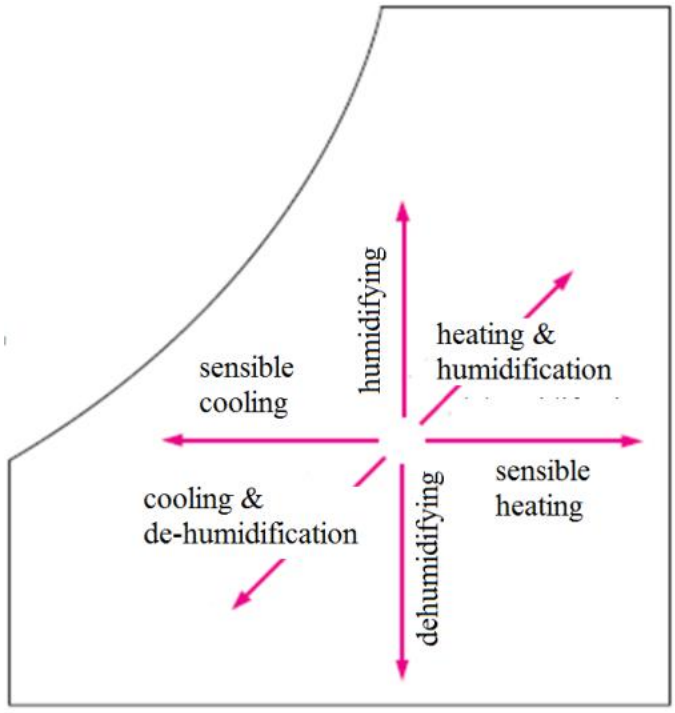


Figure 5.2.1 Simple air conditioning processes in psychrometric diagram

In figure 5.2.1 some of the simple air conditioning processes shown in psychrometric diagram. As it is seen from the diagrams, horizontal lines are sensible heating and cooling while vertical lines are humidification and dehumidification processes.

5.2.1.sensible heating & cooling:

An air stream can be heated by using hot water coils or closed heat exchangers with hot fluid recurculating inside. Or heat source can be an oven or radiator. In this case specific moisture in the room is not changing but temperature of the steam is increasing. Cooling is the inverse process. During the heating process, relative humidiy of the space is decreasing. And during the cooling process relative humidity is increasing. About 50% relative humidiy is usualy desired point for human comfort, therefore if it changes too much from the ideal point, we might feel uncomfortable, even though the room temperature in comfort zone.

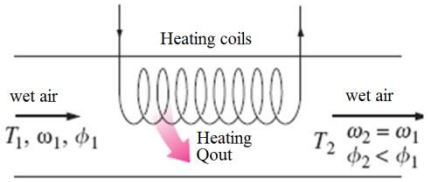


Figure 5.2.1-1 heating of wet air

The basic energy equations for the process:

$m_{a,1} = m_{a,2} = m_a$ for air and

$w_1 = w_2$ for water

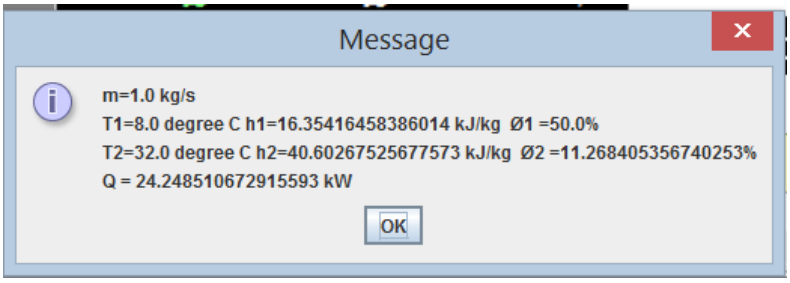
and heat transfer:

$Q = m_a (h_2 - h_1)$ (Kw) veya $q = h_2 - h_1$ (kj/kg) (5.2.1)

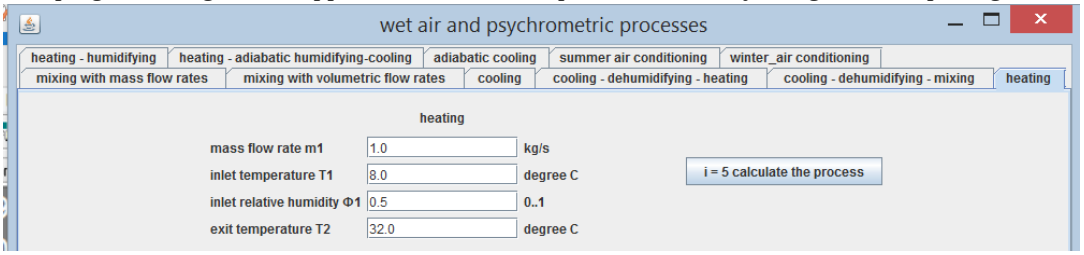
In this equation m_a dry air mass flow rates and h_1 and h_2 is enthalpy of wet air, but the difference of dry air enthapies will result the same value due to constant latent enthalpy values. This simple process can be investigated a small program. As an example, if m_a= 1 kg/s dry air is flowing with an inlet temperature of 8 C and relative humidity %50 and heated to 32 C. A small java program will be developed to calculate the process.

Program 5.2.1.1 heating of wet air

```
import javax.swing.*;
public class wetairtest2
{
public static void main(String arg[])
{
wetair wa=new wetair();
double m=1.0;//  kg/s
double p=1.01325;//  bar
double t1=8.0;//  degree C
double rh1=0.5;
double a[]=wa.property("tdb_rh",t1,rh1,p);
double h1=a[8];
double w1=a[5];
double w2=w1;
double t2=32.0;//   degree C
a=wa.property("tdb_w",t2,w2,p);
double h2=a[8];
double rh2=a[6];
double Q=m*(h2-h1);
String s="m="+m+" kg/s\n";
s+="T1="+t1+" degree C h1="+h1+" kJ/kg  "+'\u00D8'+"1 ="+rh1*100+"%\n";
s+="T2="+t2+" degree C h2="+h2+" kJ/kg  "+'\u00D8'+"2 ="+rh2*100+"%\n";
s+="Q = "+Q+" kW";
JOptionPane.showMessageDialog(null,s);
}}
```



A graphic user interface program wetairprocesses and wetairprocessTable is developed. The codes of these programs are given in Appendix 7. If the same problem solved by using this GUI package:



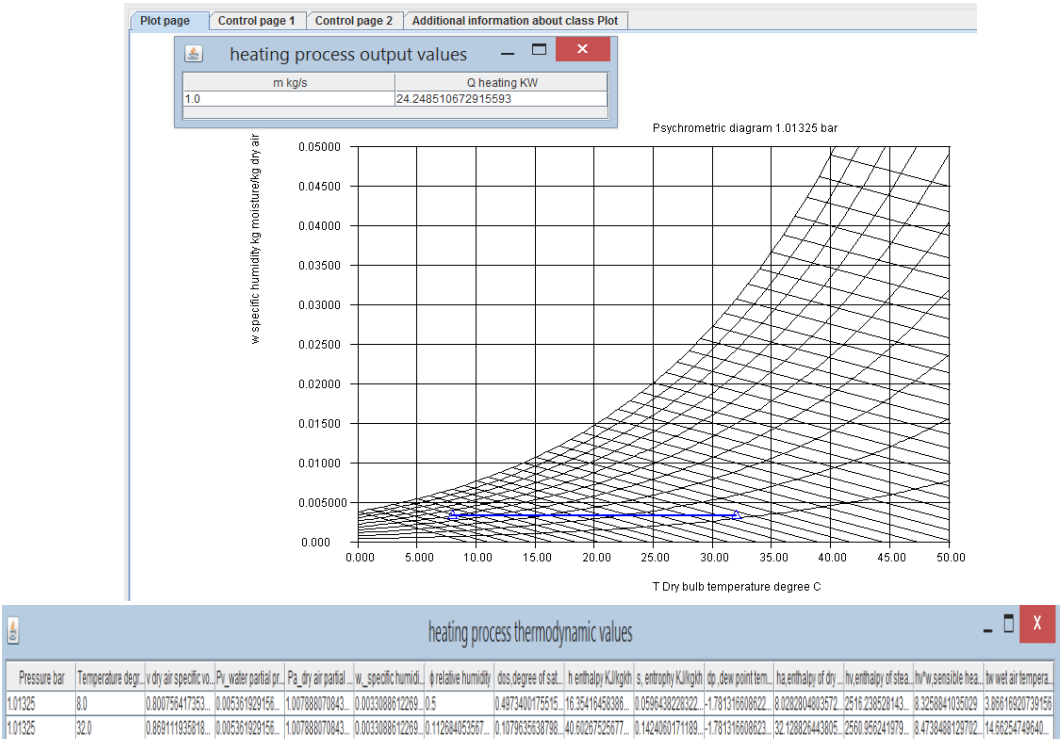


Figure 5.2.1.1 GUI output showing the heating process

A cooling process can be calculated similarly. As an example if m_a= 1 kg/s of dry air is flowing with an inlet temperature of 35 C and relative humidity %50 and cooled to 25 C.

Program 5.2.1.2 cooling of wet air

```
import javax.swing.*;
public class wetairtest3
{
public static void main(String arg[])
{
wetair wa=new wetair();
double m=1.0;//  kg/s
double p=1.01325;//  bar
double t1=35.0;//  degree C
double rh1=0.5;
double a[]=wa.property("tdb_rh",t1,rh1,p);
double h1=a[8];
double w1=a[5];
double w2=w1;
double t2=25.0;//   degree C
a=wa.property("tdb_w",t2,w2,p);
double h2=a[8];
double rh2=a[6];
double Q=m*(h2-h1);
String s="m="+m+" kg/s\n";
s+="T1="+t1+" degree C h1="+h1+" kJ/kg  "+'\u00D8'+"1 ="+rh1*100+"%\n";
s+="T2="+t2+" degree C h2="+h2+" kJ/kg  "+'\u00D8'+"2 ="+rh2*100+"%\n";
s+="Q = "+Q+" kW";
JOptionPane.showMessageDialog(null,s);
}}
```



If the same problem solved by using this GUI package:

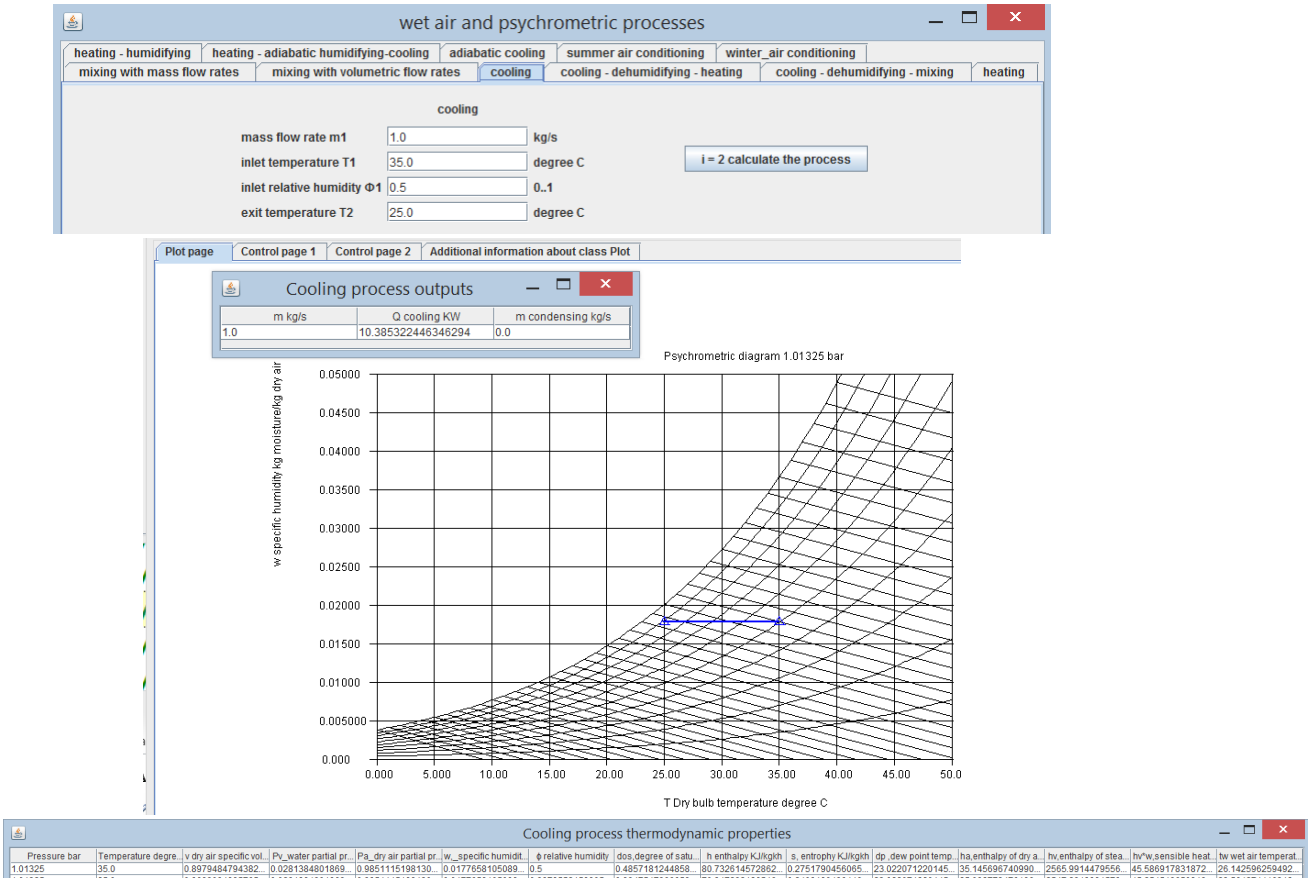


Figure 5.2.1.2 GUI output showing the cooling process

5.2.2. Heating and humidifying:

Uncomfort of dry air in heating process can be solved by moisturizing the air, after heating. Air is first heated and then humidified by passing it through a humidifier. Humidification can be done adiabatically or by using steam, it can be done together with some heating.

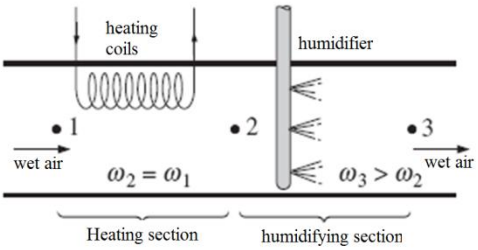


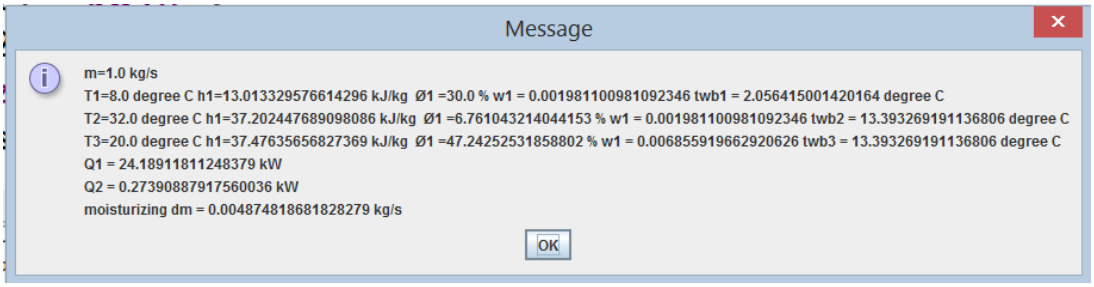
Figure 5.2.2-1 heating and humidifying of wet air

As a first example if an adiabatik moisturising is assumed with m_a=1 kg/s, T_1=8^{0}C, ϕ_1(relative humidity)=%30 ,T_2=32^{0}C, T_3=20^{0}C, $T_{3wetbulb}$= $T_{2wetbulb}$ (adiabatic moisturizing process)

Program 5.2.2.1 heating and adiabatic humidifying

```
 import javax.swing.*;
public class wetairtest4
{
public static void main(String arg[])
{
wetair wa=new wetair();
double m=1.0;//  kg/s
double p=1.01325;//  bar
double t1=8.0;//  degree C
double rh1=0.3;
double a[]=wa.property("tdb_rh",t1,rh1,p);
double h1=a[8];
double w1=a[5];
double twb1=a[14];
double w2=w1;
double t2=32.0;//   degree C
a=wa.property("tdb_w",t2,w2,p);
double h2=a[8];
double rh2=a[6];
double twb2=a[14];
double t3=20.0;
double twb3=twb2;
a=wa.property("tdb_twb",t3,twb3,p);
double h3=a[8];
double rh3=a[6];
double w3=a[5];
double Q1=m*(h2-h1);
double Q2=m*(h3-h2);
double dm=m*(w3-w2);

String s="m="+m+" kg/s\n";
s+="T1="+t1+" degree C h1="+h1+" kJ/kg  "+'\u00D8'+"1 ="+rh1*100+" % w1 = "+w1+" twb1 =
"+twb1+" degree C\n";
s+="T2="+t2+" degree C h1="+h2+" kJ/kg  "+'\u00D8'+"1 ="+rh2*100+" % w1 = "+w2+" twb2 =
"+twb2+" degree C\n";
s+="T3="+t3+" degree C h1="+h3+" kJ/kg  "+'\u00D8'+"1 ="+rh3*100+" % w1 = "+w3+" twb3 =
"+twb3+" degree C\n";
s+="Q1 = "+Q1+" kW\n";
s+="Q2 = "+Q2+" kW\n";
s+="moisturizing dm = "+dm+" kg/s";
JOptionPane.showMessageDialog(null,s);
}
}
```



We can investigate this process by using GUI program as well.

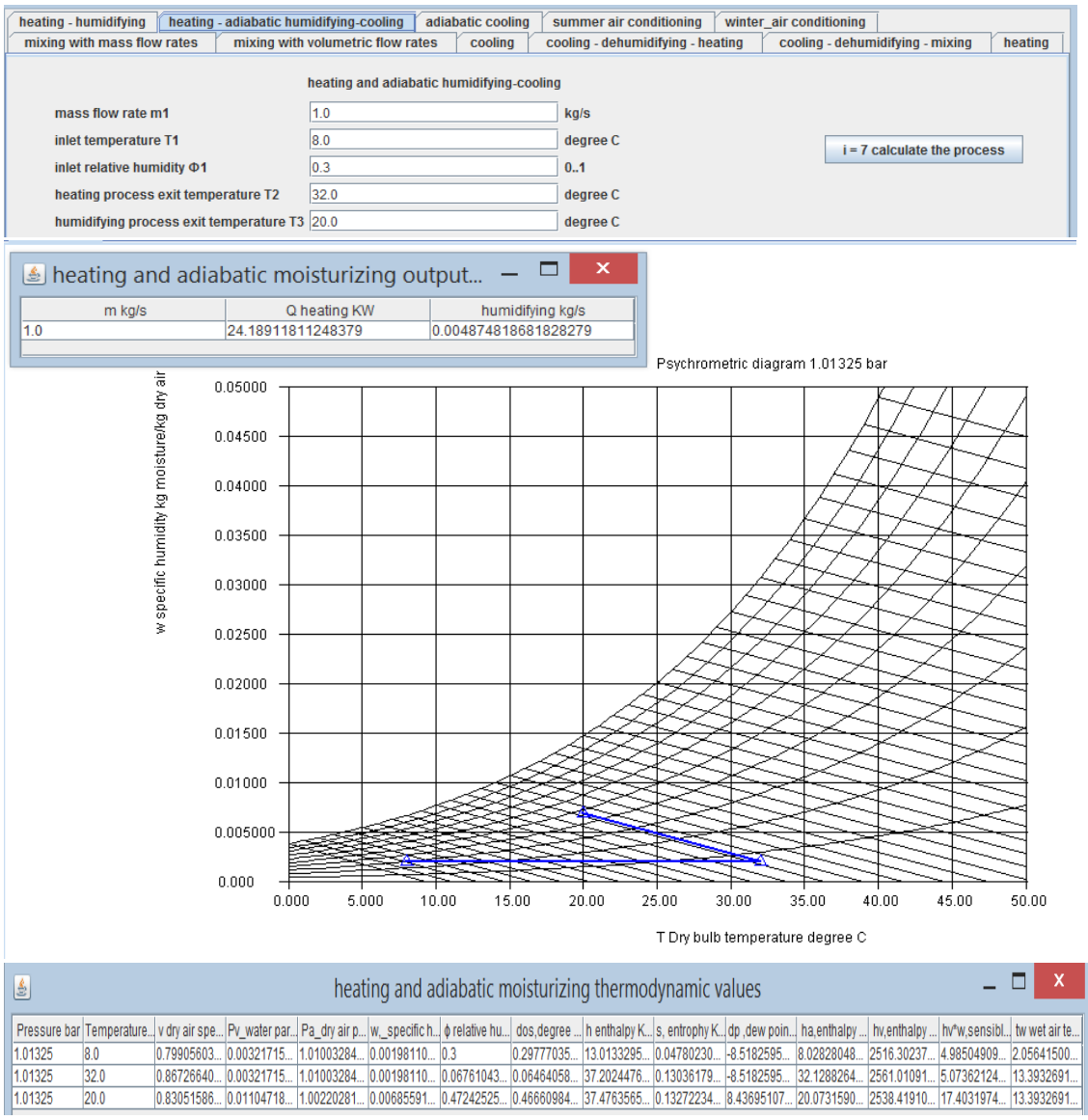
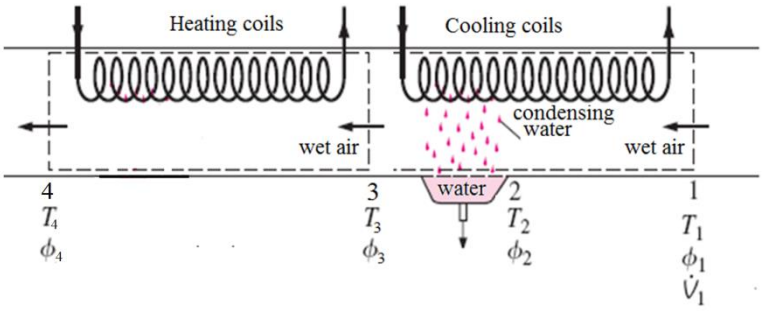


Figure 5.2.2.1 GUI output showing the heating and adiabatic moisturizing process

5.2.3 .Cooling and dehumidification and reheating:

Moisture content of air remains constant during sensible cooling process. If relative humidity of exit air is too high, removal of moisture is a requirement for obtaining a better condition. But after removal of moisture cols air has 100% relative humidiy, to take the value of relative humidiy to comfortable levels reheat of the air can be applied.



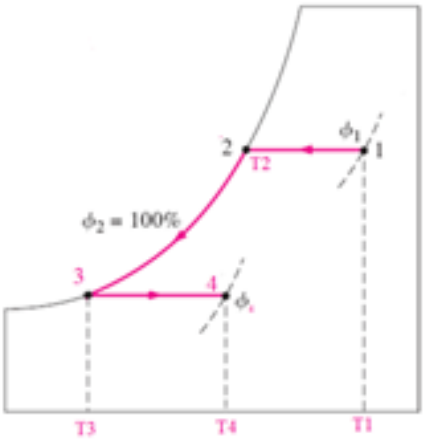


Figure 5.2.3-1 cooling and dehumidifying and heating process

Basic equations:

$w_1 = w_2$ (5.2.2)

$\varphi_2 = \varphi_3$ (5.2.3)

$w_3 = w_4$ (5.2.4)

$Q_{cooling} = m_{a1}(h_1 - h_3)$ (5.2.5)

$Q_{heating} = m_{a1}(h_4 - h_3)$ (5.2.6)

$\Delta m_{condensation} = m_{a1}(w_2 - w_1)$ (5.2.7)

As an example problem following values are considered: with m_a=1 kg/s, T_1=32 C, ϕ_1(relative humidity)=%70 ,T_4=22^{0}C, ϕ_1(relative humidity)=%45

Program 5.2.3.1 cooling-dehumidifying-heating process

```
import javax.swing.*;
public class wetairtest5
{
public static void main(String arg[])
{
wetair wa=new wetair();
double m=1.0;//  kg/s
double p=1.01325;//  bar
double t1=32.0;//  degree C
double rh1=0.7;
double a[]=wa.property("tdb_rh",t1,rh1,p);
double h1=a[8];
double w1=a[5];
double twb1=a[14];

double w2=w1;
double rh2=1.0;
a=wa.property("w_rh",w2,rh2,p);
double t2=a[1];//   degree C
double twb2=a[14];
double h2=a[8];

double t4=22.0;//  degree C
double rh4=0.45;
a=wa.property("tdb_rh",t4,rh4,p);
double h4=a[8];
double w4=a[5];
double twb4=a[14];
```

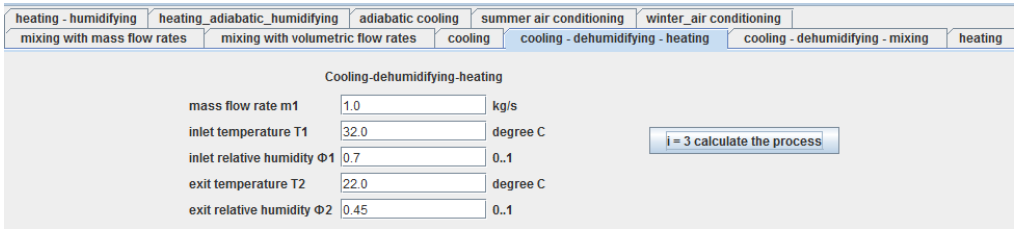
```
double w3=w4;
double rh3=1.0;
a=wa.property("w_rh",w3,rh3,p);
double t3=a[1];
double h3=a[8];
double twb3=a[14];
double Qcooling=m*(h1-h3);
double Qheating=m*(h4-h3);
double dm=m*(w2-w3);

String s="m="+m+" kg/s\n";
s+="T1="+t1+" degree C h1="+h1+" kJ/kg  "+'\u00D8'+"1 ="+rh1*100+" % w1 = "+w1+" twb1 =
"+twb1+" degree C\n";
s+="T2="+t2+" degree C h1="+h2+" kJ/kg  "+'\u00D8'+"2 ="+rh2*100+" % w1 = "+w2+" twb2 =
"+twb2+" degree C\n";
s+="T3="+t3+" degree C h1="+h3+" kJ/kg  "+'\u00D8'+"3 ="+rh3*100+" % w1 = "+w3+" twb3 =
"+twb3+" degree C\n";
s+="T4="+t3+" degree C h4="+h4+" kJ/kg  "+'\u00D8'+"4 ="+rh4*100+" % w1 = "+w3+" twb3 =
"+twb4+" degree C\n";

s+="Qcooling = "+Qcooling+" kW\n";
s+="Qheating = "+Qheating+" kW\n";
s+="de-moisturizing dm = "+dm+" kg/s";
JOptionPane.showMessageDialog(null,s);
}
}
```



The same process can be calculated from GUI



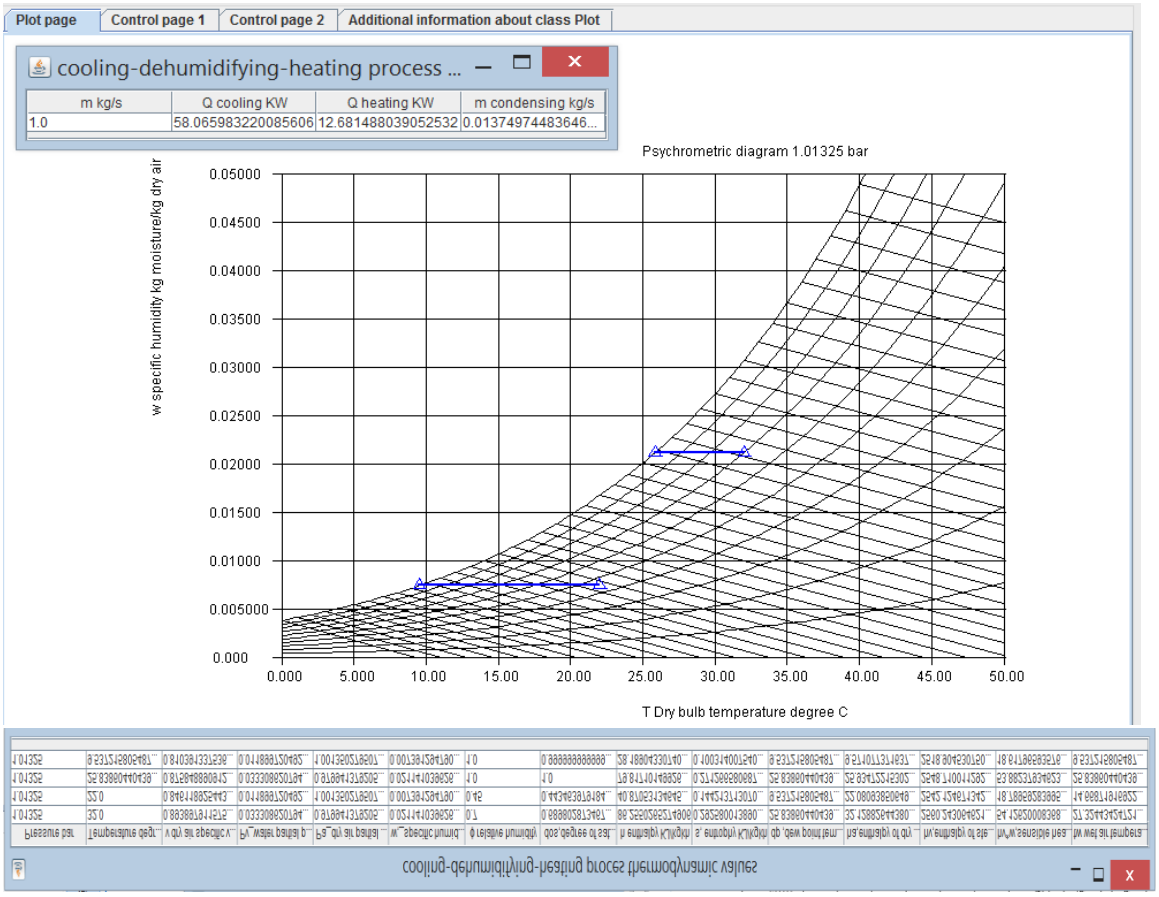


Figure 5.2.3.2 GUI output showing the cooling-demoisturising-heating process

5.2.4 Adiabatic mixing of air streams

Mixing process is used quite commonly as a part of air conditioning processes. Mixing process is shown in the figure below. In the mixing process heat escaping to the environment are usually samll and it can be ignored

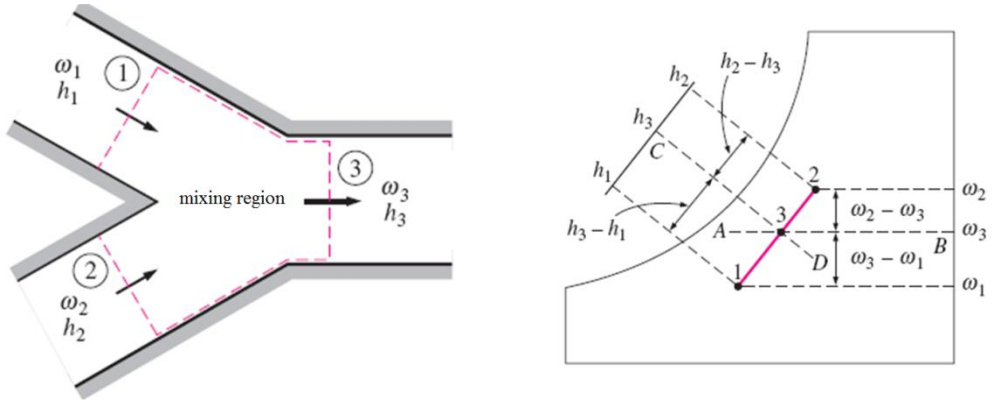


Figure 5.2.4.1 Adiabatic mixing process

Basic equations:

$m_{a1} + m_{a2} = m_{a3}$ Air mass balance (5.2.8)

$m_{a1}w_1 + m_{a2}w_2 = m_{a3}w_3$ water vapor mass balance (5.2.9)

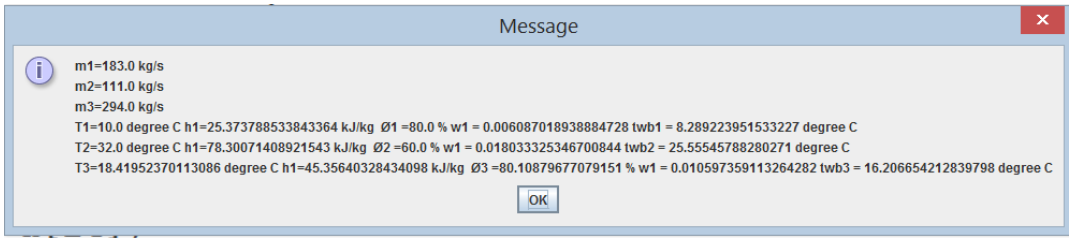
$m_{a1}h_1 + m_{a2}h_2 = m_{a3}h_3$ energy balance (5.2.10)

Adiabatic mixing process example data is; m_{a1}= 183 kg/s, T_1=10^{0}C, ϕ_1=%80, m_{a2}=111kg/s,T_2=32^{0}C,ϕ_2=%60

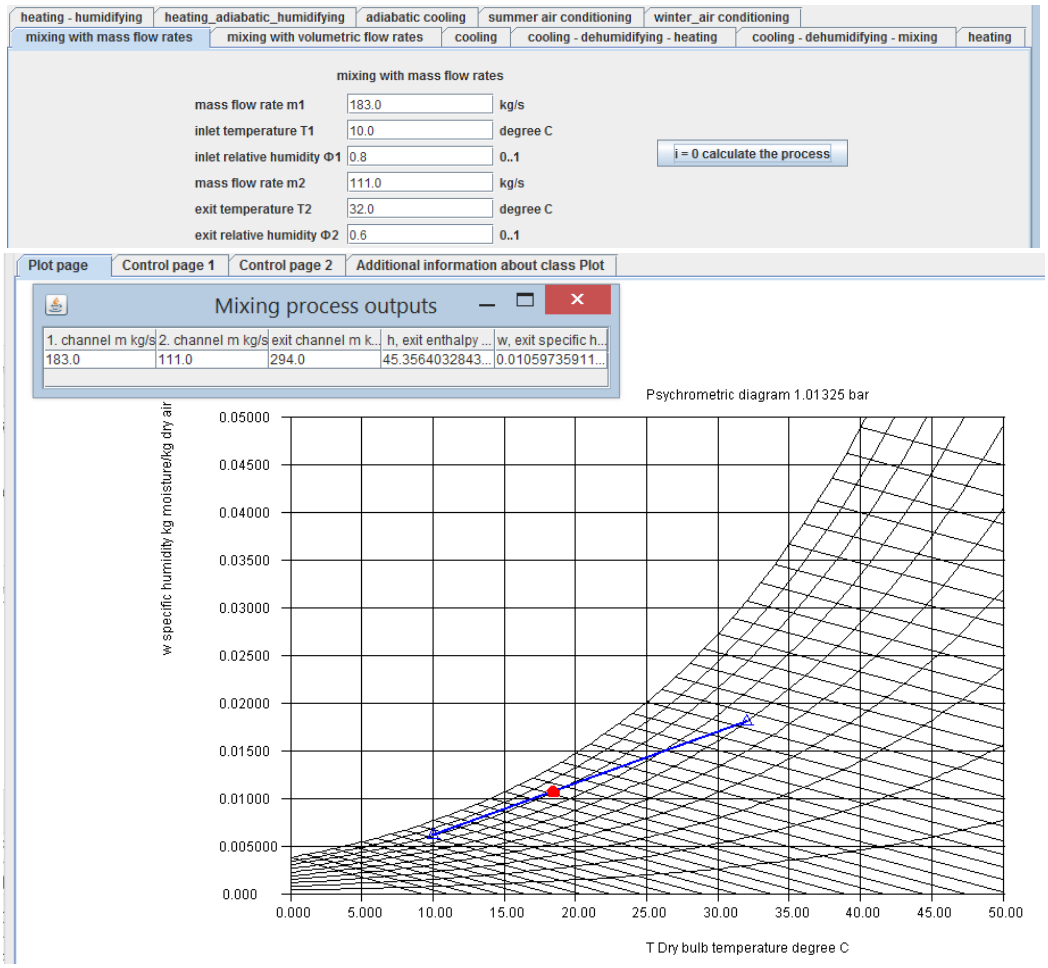
Program 5.2.4.1 adiabatic mixing process

```
import javax.swing.*;
public class wetairtest6
{ // adiabatic mixing process
public static void main(String arg[])
{
wetair wa=new wetair();
double m1=183.0;//  kg/s
double m2=111.0;//  kg/s
double m3=m1+m2; //kg/s
double p=1.01325;//  bar
double t1=10.0;//  degree C
double rh1=0.8;
double t2=32.0;//  degree C
double rh2=0.6;
double a[]=wa.property("tdb_rh",t1,rh1,p);
double h1=a[8];
double w1=a[5];
double twb1=a[14];
//point 2
a=wa.property("tdb_rh",t2,rh2,p);
double h2=a[8];
double w2=a[5];
double twb2=a[14];
double w3=(m1*w1+m2*w2)/m3;
double h3=(m1*h1+m2*h2)/m3;
a=wa.property("w_h",w3,h3,p);
double t3=a[1];
double rh3=a[6];
double twb3=a[14];

String s="m1="+m1+" kg/s\n";
s+="m2="+m2+" kg/s\n";
s+="m3="+m3+" kg/s\n";
s+="T1="+t1+" degree C h1="+h1+" kJ/kg  "+'\u00D8'+"1 ="+rh1*100+" % w1 = "+w1+" twb1 = "+twb1+"
degree C\n";
s+="T2="+t2+" degree C h1="+h2+" kJ/kg  "+'\u00D8'+"2 ="+rh2*100+" % w1 = "+w2+" twb2 = "+twb2+"
degree C\n";
s+="T3="+t3+" degree C h1="+h3+" kJ/kg  "+'\u00D8'+"3 ="+rh3*100+" % w1 = "+w3+" twb3 = "+twb3+"
degree C\n";
JOptionPane.showMessageDialog(null,s);
}
}
```



The same solution with GUI



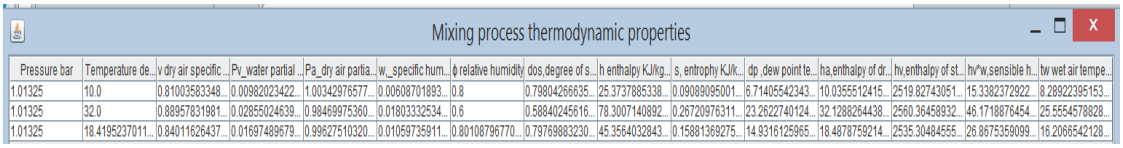


Figure 5.2.4.2 GUI output showing the adyabatic mixing process

5.2.5 cooling, dehumification and mixing process

In order to heat up air to take it to more comfortable air conditions after moisture removal is an expansive process, a similar but not as effective effect can be obtained by mixing cold saturated air with hot air of the room.

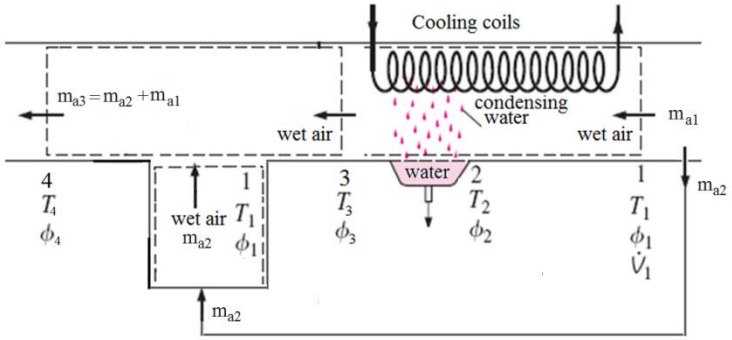


Figure 5.2.5-1 cooling and dehumidifying and mixing process

Basic equations:

$w_1 = w_2$ (5.2.11)

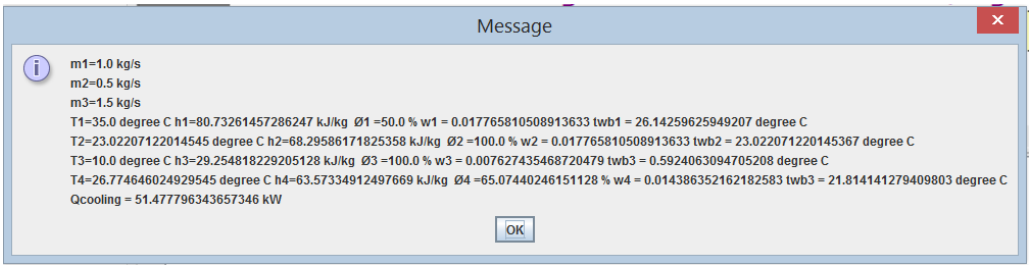
$\varphi_2 = \varphi_3$ (5.2.12)
$w_3 = w_4$ (5.2.13)
$Q_{cooling} = m_{a1}(h_1 - h_3)$ (5.2.14)
$m_{a1} + m_{a2} = m_{a3}$ Air mass balance (5.2.15)
$m_{a1}w_3 + m_{a2}w_1 = m_{a3}w_4$water vapor mass balance (5.2.16)
$m_{a1}h_3 + m_{a2}h_1 = m_{a3}h_4$ energy balance (5.2.17)
$\Delta m_{condensation} = m_{a1}(w_2 - w_1)$ (5.2.18)
As a sample problem, following values will be considered m_1= 1 kg/s, T_1=35^{0}C, ϕ_1(relative humidity)=%50, T_4=10^{0}C, m_2=0.5kg/s

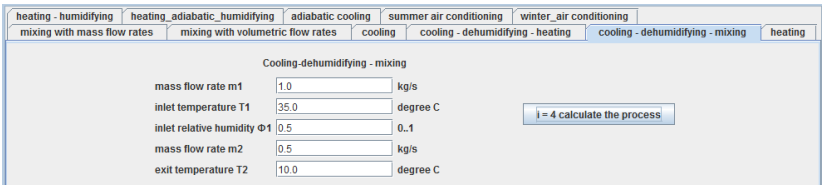
Program 5.2.5.1 cooling-dehumidifying and adiabatic mixing process

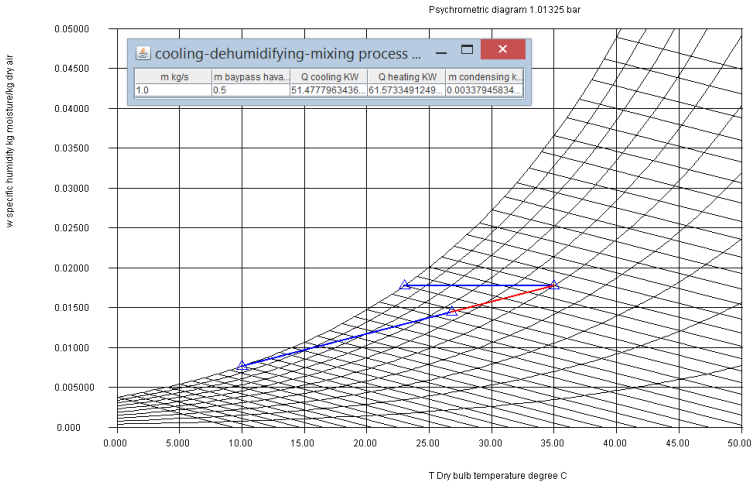
```
import javax.swing.*;
public class wetairtest6
{ // cooling-demoisturizing and adiabatic mixing process
public static void main(String arg[])
{
wetair wa=new wetair();
double m1=1.0;//  kg/s
double m2=0.5;//  kg/s
double m3=m1+m2; //kg/s
double p=1.01325;//  bar
double t1=35.0;//  degree C
double rh1=0.5;
double t3=10.0;//  degree C
double rh3=1.0;
double a[]=wa.property("tdb_rh",t1,rh1,p);
double h1=a[8];
double w1=a[5];
double twb1=a[14];
//point 2
double w2=w1;
double rh2=1.0;
a=wa.property("w_rh",w2,rh2,p);
double t2=a[1];
double h2=a[8];
double twb2=a[14];
//point 4
a=wa.property("tdb_rh",t3,rh3,p);
double h3=a[8];
double w3=a[5];
double twb3=a[14];

double w4=(m1*w1+m2*w3)/m3;
double h4=(m1*h1+m2*h3)/m3;
a=wa.property("w_h",w4,h4,p);
double t4=a[1];
double rh4=a[6];
double twb4=a[14];
double Qcooling=m1*(h1-h3);
String s="m1="+m1+" kg/s\n";
s+="m2="+m2+" kg/s\n";
s+="m3="+m3+" kg/s\n";
s+="T1="+t1+" degree C h1="+h1+" kJ/kg  "+'\u00D8'+"1 ="+rh1*100+" % w1 = "+w1+" twb1 =
"+twb1+" degree C\n";
s+="T2="+t2+" degree C h2="+h2+" kJ/kg  "+'\u00D8'+"2 ="+rh2*100+" % w2 = "+w2+" twb2 =
"+twb2+" degree C\n";
s+="T3="+t3+" degree C h3="+h3+" kJ/kg  "+'\u00D8'+"3 ="+rh3*100+" % w3 = "+w3+" twb3 =
"+twb3+" degree C\n";
```

```
s+="T4="+t4+" degree C h4="+h4+" kJ/kg  "+'\u00D8'+"4 ="+rh4*100+" % w4 = "+w4+" twb3 =
"+twb4+" degree C\n";
s+="Qcooling = "+Qcooling+"\n";
JOptionPane.showMessageDialog(null,s);
}
}
```







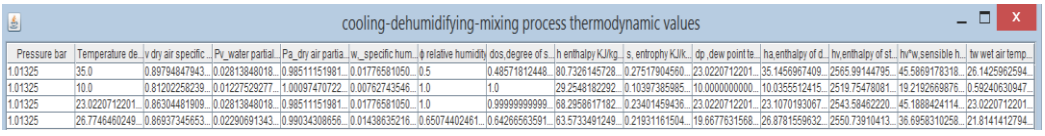


Figure 5.2.5.2 GUI output showing the cooling-dehumidification- mixing process

5.2.6 evaporative cooling process

Cooling processes are basically based on refrigeration cycle. This processes can be utilized any kind of outside conditions, but their cost is high. In very hot and dry conditions, evaporative cooling can be an alternative or first satage before refrigerative cooling to reduce the cost. Evaporative cooling based on evaporation of water. While water is evaporating it latent heat of evaporation is supplied from water and environmental air. As a result water and air cooled down. This methods are used for centruies to cool down the drinking water as well as watermelons in summer times. You might also noticed that if you water your garden n a hot day, it will get cooler. The reason for this is the evaporative cooling effect.

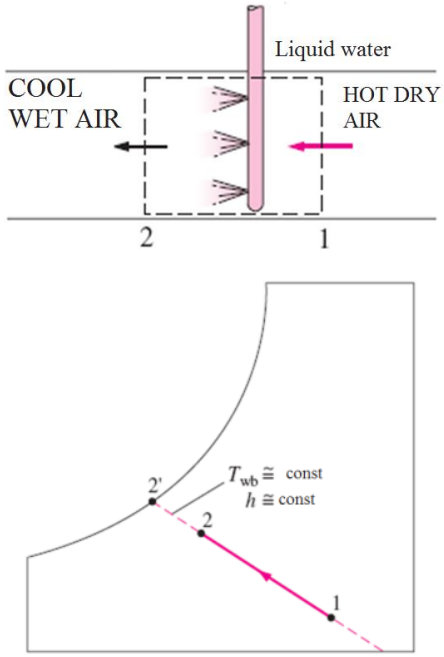


Figure 5.2.6.1 Evaporative cooling process

Basic equations:

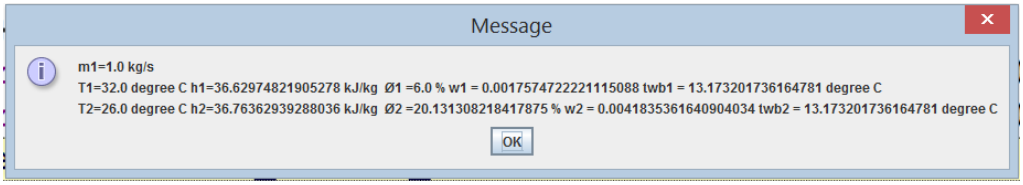
$T_{wetbulb1} = T_{wetbulb2}$ (5.2.19)

$$\varphi_1 < \varphi_2 < 100\%$$

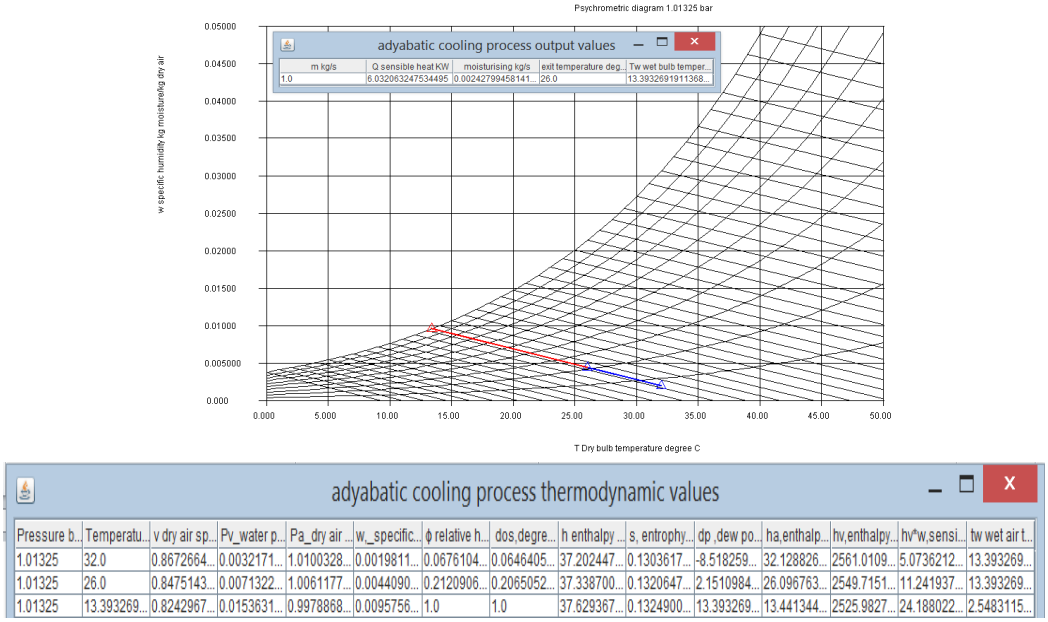
Example data for the evaporative cooling: m_1= 1 kg/s, T_1=32^{0}C, ϕ_1(relative humidity)=%6, T_2=26^{0}C

Program 5.2.6.1 Evaporative cooling process

```
import javax.swing.*;
public class wetairtest7
{ // adiabatic cooling process
public static void main(String arg[])
{
wetair wa=new wetair();
double m1=1.0;//  kg/s
double p=1.01325;//  bar
double t1=32.0;//  degree C
double rh1=0.06;
double t2=26.0;//  degree C
double a[]=wa.property("tdb_rh",t1,rh1,p);
double h1=a[8];
double w1=a[5];
double twb1=a[14];
double twb2=twb1;
//point 2
a=wa.property("tdb_twb",t2,twb2,p);
double h2=a[8];
double rh2=a[6];
double w2=a[5];
String s="m1="+m1+" kg/s\n";
s+="T1="+t1+" degree C h1="+h1+" kJ/kg  "+'\u00D8'+"1 ="+rh1*100+" % w1 = "+w1+" twb1 =
"+twb1+" degree C\n";
s+="T2="+t2+" degree C h2="+h2+" kJ/kg  "+'\u00D8'+"2 ="+rh2*100+" % w2 = "+w2+" twb2 =
"+twb2+" degree C\n";
JOptionPane.showMessageDialog(null,s);
}
}
```



Again the data can be run through the GUI program



Pressure b...	Temperatu...	v dry air sp...	Pv_water p...	Pa_dry air ...	w_specific...	ϕ relative h...	dos,degre...	h enthalpy ...	s, entrophy...	dp ,dew po...	ha,enthalp...	hv,enthalpy...	hv*w,sensi...	tw wet air t...
1.01325	32.0	0.8672664...	0.0032171...	1.0100328...	0.0019811...	0.0676104...	0.0646405...	37.202447...	0.1303617...	-8.518259...	32.128826...	2561.0109...	5.0736212...	13.393269...
1.01325	26.0	0.8475143...	0.0071322...	1.0061177...	0.0044090...	0.2120906...	0.2065052...	37.338700...	0.1320647...	2.1510984...	26.096763...	2549.7151...	11.241937...	13.393269...
1.01325	13.393269...	0.8242967...	0.0153631...	0.9978868...	0.0095756...	1.0	1.0	37.629367...	0.1324900...	13.393269...	13.441344...	2525.9827...	24.188022...	2.5483115...

Figure 5.2.6.1 GUI output showing the adiabatic humidification process

5.2.7 Single zone winter air conditioning

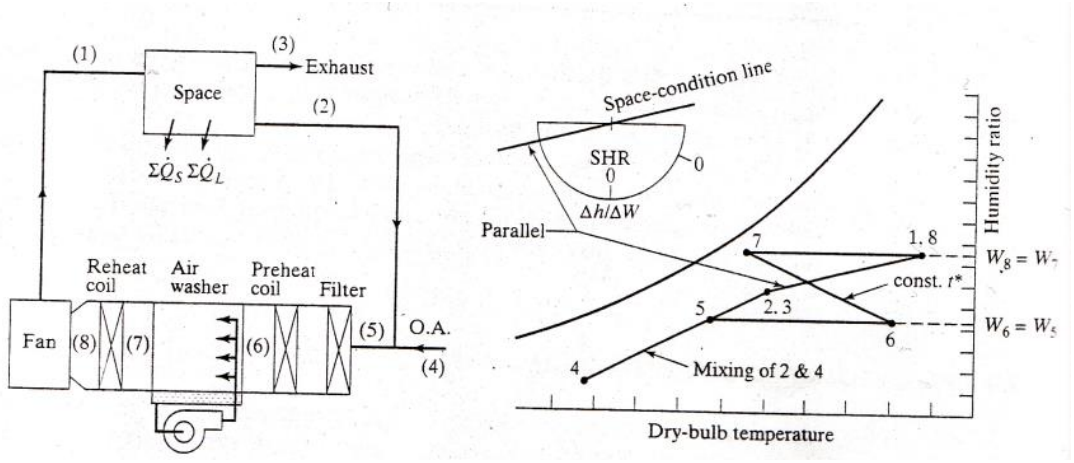


Figure 5.2.7.1 a single space winter air conditioning process

Figure 5.2.7.1 shows winter air conditioning process of a single zone. While some of the air is exhausted to replace fresh air, the remaining air circulates through a series of air conditioning processes, first, it is preheated, and then moisturised in an air washer. After a last heating process ir is send to the room. In the room, the warm air inlet is mixed up with the air in the room to create the comfort state required. The details of the individual processes are defined earlier as single processes so, we are only given here process diagrams and program codes to create winter air conditioning. It should be note that process can be mad efor a multizone system with a much complicated details.

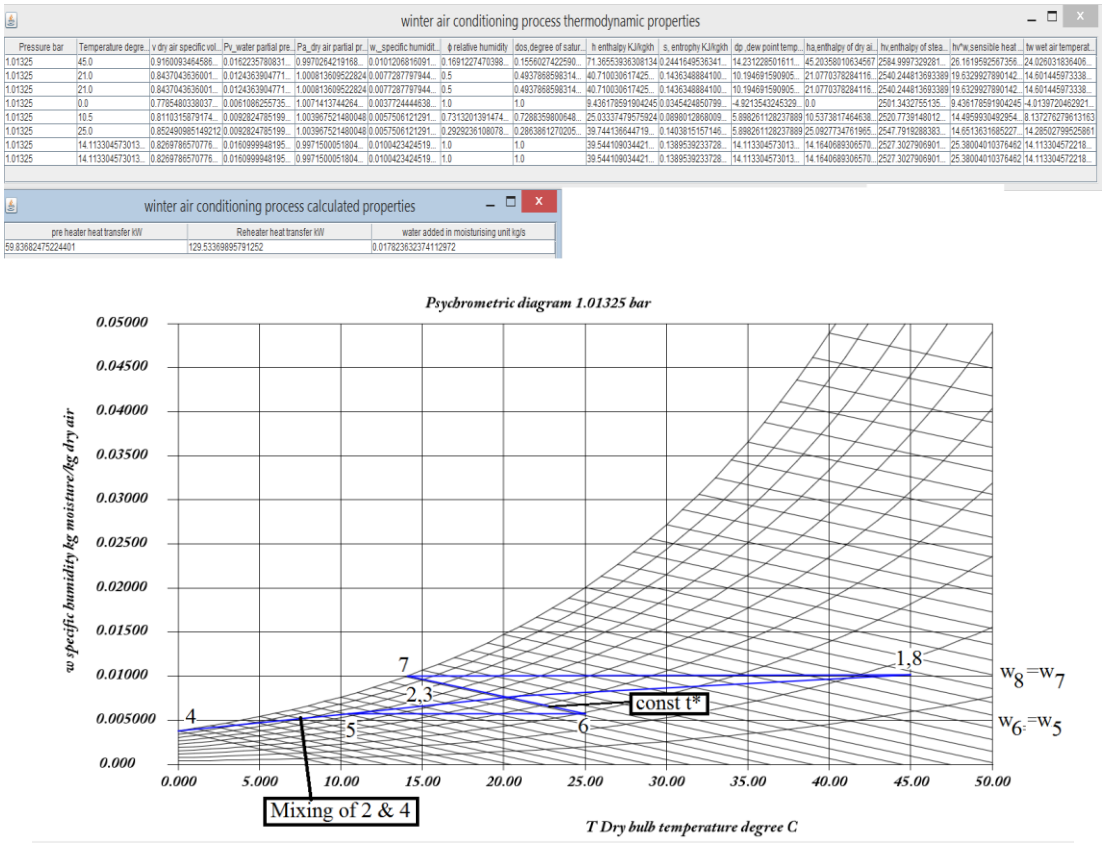


Figure 5.2.7.2 a single space winter air conditioning process

Program 5.2.7.1 single zone winter air conditioning process

```
import javax.swing.*;
public class wetairtest10
{ // cooling-demoisturizing and adiabatic winter air conditioning process
Plot pi;
double P;
wetair yh;
double tmin=0;
double tmax=50.0;
String heading[]={"Pressure bar",
"Temperature degree C",
"v dry air specific volume m^3/kgkh",
"Pv_water partial pressure bar",
"Pa_dry air partial pressure bar",
"w,_specific humidity kg moist./kg dry air",
'\u03D5'+" relative humidity",
"dos,degree of saturation",
"h enthalpy KJ/kgkh",
"s, entrophy KJ/kgkh",
"dp ,dew point temperature degree C",
"ha,enthalpy of dry air KJ/kgkh",
"hv,enthalpy of steam KJ/kgkh",
"hv*w,sensible heat KJ/kgkh",
"tw wet air temperature C"};
```

```
public double[][] winter_air_conditioning(double Pi,double T1,double T2,double
f2,double T4,double f4,double T6,double Qsensible,double Qlatent)
{   P=Pi;
   yh=new wetair();
   ps_plot_hazirla(tmin,tmax,P);
          // room air conditions
          double a2[]=yh.property("tdb_rh",T2,f2,P);
          double h2=a2[8];
          double w2=a2[5];
          double hv2=a2[12];
          // outside air inlet
          double a4[]=yh.property("tdb_rh",T4,f4,P);
          double h4=a4[8];
          double w4=a4[5];
          // pre heating inlet
          double h5=h2*0.5+h4*0.5;
          double w5=w2*0.5+w4*0.5;
          double t5=0.5*T2+0.5*T4;
          // pre heating exit
          double w6=w5;
          double a6[]=yh.property("tdb_w",T6,w6,P);
          double h6=a6[8];
          // reheating entry

          double  m_supply_air=Qsensible/(1.0216*(T1-T2));
          //first guess w1
          double w1;
          double t7;
          double a1new[];
          double a7[];
          double a7new[];
          double h7new;
          double a6guess[];
          double t6guess;
          double titer=0.001;
          double  w1new=w2+titer;
           a1new=yh.property("tdb_w",T1,w1new,P);
          double hv1=a1new[12];
          double hort=(hv1+hv2)*0.5;
          w1=w2+(Qlatent/(m_supply_air*hort));
          a7new=yh.property("w_h",w1,h6,P);
          t7=a7new[1];
          double w7=w1;
          double mw=m_supply_air*(w7-w6);
          double a1[]=yh.property("tdb_w",T1,w1,P);
          double h1=a1[8];
          double Q56=m_supply_air*(h6-h5); // pre heater heating kW
```

```
            double Q78=(m_supply_air+mw)*(h1-h6);// re_heater heating kW
            // last evaluation
            double a1yeni[]=yh.property("tdb_w",T1,w1,P);
            double a2yeni[]=yh.property("tdb_rh",T2,f2,P);
            double a3yeni[]=a2yeni;
            double a4yeni[]=yh.property("tdb_rh",T4,f4,P);
            double a5yeni[]=yh.property("tdb_w",t5,w5,P);
            double a6yeni[]=yh.property("tdb_w",T6,w6,P);
            double a7yeni[]=yh.property("tdb_w",t7,w7,P);
            double a[][]=new double[8][a1yeni.length];
    a[0]=a1yeni;
    a[1]=a2yeni;
    a[2]=a[1];
    a[3]=a4yeni;
    a[4]=a5yeni;
    a[5]=a6yeni;
    a[6]=a7yeni;
    a[7]=a[6];
            addData(a4yeni,a5yeni,5,0,0,255);
    addData(a5yeni,a2yeni,5,0,0,255);
            addData(a2yeni,a1yeni,5,0,0,255);
            addData(a1yeni,a7yeni,5,0,0,255);
            addData(a7yeni,a6yeni,5,0,0,255);
            addData(a6yeni,a5yeni,5,0,0,255);
    ps_plot(0,50);
            print(a,heading,"winter air conditioning process thermodynamic properties");
            String baslik2[]={"pre heater heat transfer kW","Reheater heat transfer
kW","water added in moisturising unit kg/s"};
            double b[][]={{Q56,Q78,mw}};
            print(b,baslik2,"winter air conditioning process calculated properties");

            return a;
}
public void print(double a[][],String s)
{//String heading[]={"winter air conditioning"};
 Text.print(a,heading,"winter air conditioning data");}

public void print(double a[][],String heading2[],String s)
{String heading[];Text.print(a,heading2,s);}
public void ps_plot_hazirla(double tmin,double tmax,double P1)
     {
              double x1[][]=new double[62][51];
              double y1[][]=new double[62][51];
              double A[][][]=new double[2][62][51];
              A=ps_verisi(tmin,tmax,P1);
              for(int i=0;i<A[0].length;i++)
               for(int j=0;j<A[0][0].length;j++)
```

```
            {x1[i][j]=A[0][i][j];y1[i][j]=A[1][i][j];}
 pi=new Plot(x1,y1);
 String ss2="Psychrometric diagram "+P+" bar";
 pi.setPlabel(ss2);
 pi.setXlabel("T Dry bulb temperature degree C");
 pi.setYlabel("w specific humidity kg moisture/kg dry air");
 pi.setMinMax(tmin,tmax,0.0,0.05);
 pi.setXgrid(1);
 pi.setYgrid(1);
 }
public static double[][][] ps_verisi(double tmin,double tmax,double P)
{
      //prepare pyschrometric chart data for pressure P
wetair w=new wetair();
      double aa[]=new double[14];
double bb[]=new double[14];
      //f2 ff=new f2();
      //double x1[][]=new double[62][51];
      //double y1[][]=new double[62][51];
      double A[][][]=new double[2][62][51];
//initilise as a function plot
int i,j;
i=0;
double h;
for(double rh=0.1;rh<=1.0;rh+=0.1)
{   j=0;
   for(double t=tmin;t<=tmax;t+=1.0)
   {
         aa=w.property1("tdb_rh",t,rh,P);
         h=aa[8];
           if(rh>0.99)
     {
           h=aa[8];
     A[0][11+j][0]=t;A[1][11+j][0]=aa[5];
     bb=w.property1("w_h",0.0,h,P);
     A[0][11+j][1]=bb[10];A[1][11+j][1]=0.0;
     }
           A[0][i][j]=t;
     A[1][i][j]=aa[5];
           j++;
     }
   i++;
}
return A;
}
 public void addData(double a1[][],int plottype,int red,int green,int blue)
 { double x1[]=new double[2];
```

```
        double y1[]=new double[2];
        x1[0]=a1[0][1];
        y1[0]=a1[0][5];
        x1[1]=a1[1][1];
        y1[1]=a1[1][5];
        pi.addData(x1,y1,plottype,red,green,blue);
      }

     public void addData(double a1[],double a2[],int plottype,int red,int green,int
blue)
     { double x1[]=new double[2];
       double y1[]=new double[2];
       x1[0]=a1[1];
       y1[0]=a1[5];
       x1[1]=a2[1];
       y1[1]=a2[5];
       pi.addData(x1,y1,plottype,red,green,blue);
     }

     public void addData(double a1[],int plottype,int red,int green,int blue)
     { double x1[]=new double[1];
       double y1[]=new double[1];
       x1[0]=a1[1];
       y1[0]=a1[5];
       pi.addData(x1,y1,plottype,red,green,blue);
     }

     public void ps_plot(double tmin,double tmax)
     {   pi.setMinMax(tmin,tmax,0,0.05);
                pi.plot();
            }

public static void main(String arg[])
{ wetairtest10 wat=new wetairtest10();
double p=1.01325; //  bar
double T1=45.0;   //  room inlet temperature degree C
double T2=21.0;   //  room exit temperature
double f2=0.5;    //   room exit relative humidity
double T4=0.0;    //  outside inlet temperature
double f4=1.0;    //   outside  relative humidity
double T6=25.0;   //   preheater exit temperature
double Qsensible=100.0; //  sensible heat load
double Qlatent=25.0;   //  latent heat load
double a[][]=wat.winter_air_conditioning(p,T1,T2,f2,T4,f4,T6,Qsensible,Qlatent);
}
}
```

5.2.8 Single zone summer air conditioning process

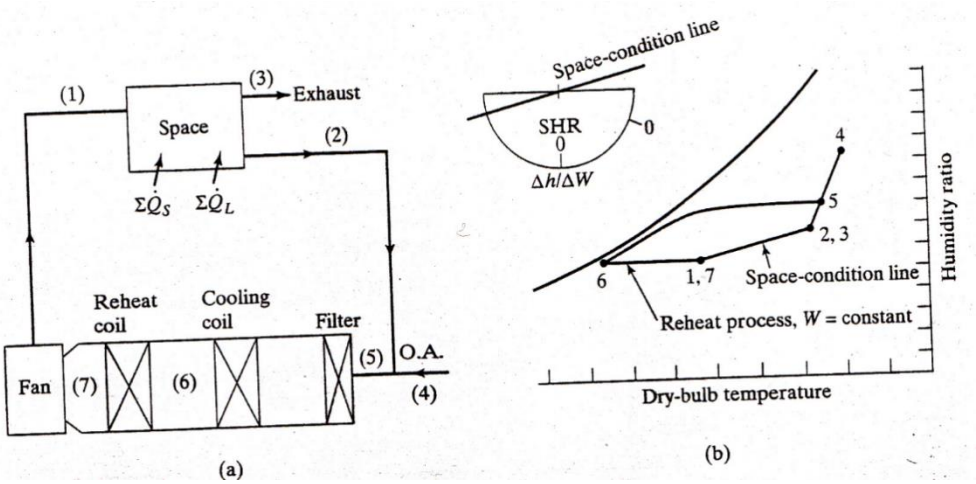
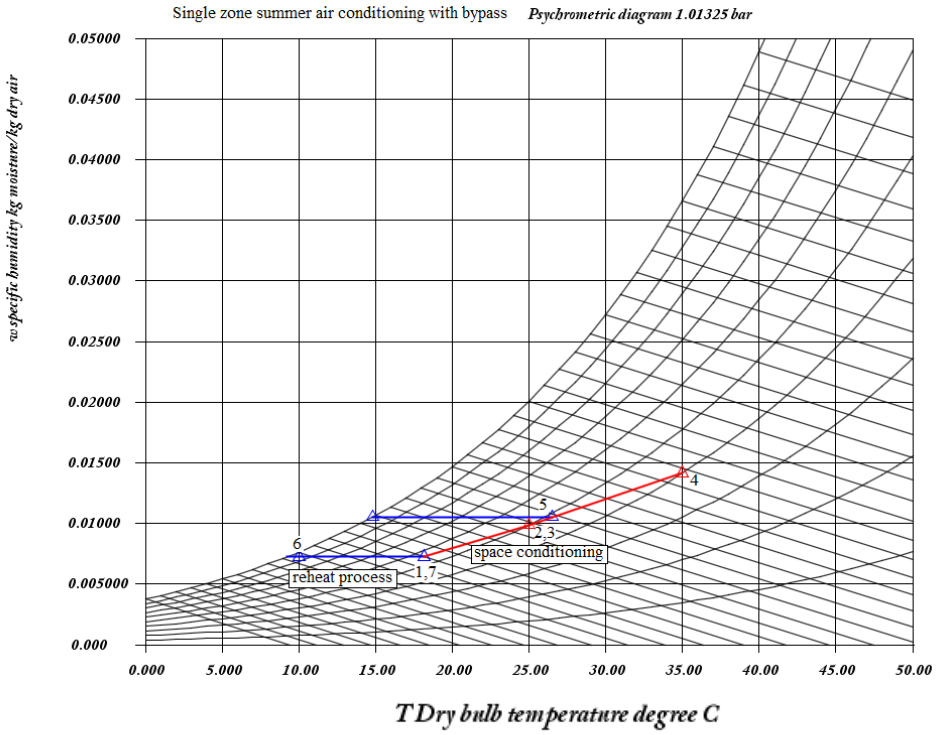


Figure 5.2.81 a single space summer air conditioning process with a bypass

Summer comfort air conditioning process include cooling and dehumidification processes. Dehumidification usually applied by cooling, but since dehumidified air has very high relative humidity, it is dropped either by mixing this air with the air bypassed from the room. Air can be mixed in the room directly, but cleaning of the air required also some filtering process. Or it can be reheat by using a heating coil. In the given process reheating is used.



summer air conditioning data

Pressure bar	Temperature ...	v dry air specif...	Pv_water parti...	Pa_dry air par...	w_specific hu...	ϕ relative hum...	dos,degree of...	h enthalpy KJ/...	s, entrophy KJ...	dp ,dew point ...	ha,enthalpy of...	hv,enthalpy of ...	hv*w,sensible...	tw wet air tem...
1.01325	18.18907891...	0.834995376...	0.011661528...	1.001588471...	0.007241623...	0.558356798...	0.553214731...	36.61297328...	0.129713841...	9.236995232...	18.25526938...	2535.026050...	18.35770389...	13.04789836...
1.01325	25.0	0.858099633...	0.015844537...	0.997405462...	0.009880470...	0.5	0.492057122...	50.26440154...	0.175628665...	13.86700289...	25.09277347...	2547.614331...	25.17162806...	17.88662108...
1.01325	25.0	0.858099633...	0.015844537...	0.997405462...	0.009880470...	0.5	0.492057122...	50.26440154...	0.175628665...	13.86700289...	25.09277347...	2547.614331...	25.17162806...	17.88662108...
1.01325	35.0	0.892847862...	0.022510784...	0.990739215...	0.014131916...	0.4	0.386367279...	71.41004813...	0.245116228...	19.38725809...	35.14569674...	2566.131204...	36.26435138...	23.93488029...
1.01325	26.50924955...	0.863330438...	0.016850188...	0.996399811...	0.010518187...	0.486223230...	0.477534715...	53.43624852...	0.186179408...	14.81726776...	26.61384280...	2550.405168...	26.82563928...	18.89184789...
1.01325	10.0	0.811524982...	0.011661528...	1.001588471...	0.007241623...	0.95	0.949417848...	28.28279788...	0.100711597...	9.236995232...	10.03555124...	2519.772943...	18.24724664...	9.581246535...
1.01325	9.236995232...	0.809338164...	0.011661528...	1.001588471...	0.007241623...	1.0	1.0	27.50672583...	0.097967047...	9.236995232...	9.269763010...	2518.352845...	18.23696282...	9.236995231...
1.01325	14.81726776...	0.829629399...	0.016850188...	0.996399811...	0.010518187...	1.0	1.0	41.46688420...	0.145425452...	14.81726776...	14.87067124...	2528.592824...	26.59621295...	14.81726776...

single zone summer air conditioni...

evaporator heat transfer k...	heater heat transfer kW	condansation mass flow ...
754.6035194079752	249.90526195869305	0.09829691816952062

Program 5.28.1 single zone summer air conditioning process

```
import javax.swing.*;
public class wetairtest11
{ // cooling-demoisturizing and adiabatic summer air conditioning process
Plot pi;
double P;
wetair yh;
double tmin=0;
double tmax=50.0;
String heading[]={"Pressure bar",
"Temperature degree C",
"v dry air specific volume m^3/kgkh",
"Pv_water partial pressure bar",
"Pa_dry air partial pressure bar",
"w,_specific humidity kg moist./kg dry air",
'\u03D5'+" relative humidity",
"dos,degree of saturation",
"h enthalpy KJ/kgkh",
"s, entrophy KJ/kgkh",
"dp ,dew point temperature degree C",
"ha,enthalpy of dry air KJ/kgkh",
"hv,enthalpy of steam KJ/kgkh",
"hv*w,sensible heat KJ/kgkh",
"tw wet air temperature C"};
public double[][] summer_air_conditioning(double Pi,double m1, double m3,double T2, double f2,
                    double T4, double f4, double T6,double f6,double dQsensible,double dQlatent)
{ // m1 mass flow rate,    room girişi kg/s
  // m3 mass flow rate,    eksoz havası kg/s
  // T2 room temperature,    degree C
  // f2 room relative humidity,   0-1
  // T4 outside air temperature, degree C
  // f4 outside air relative humidity 0-1
  // T6 evaporator exit temperature
  // f6 evaporator exit relative humidity
  // f2 giriş relative humidity, ikinci kanal 0-1
  // dQsensible sensible heat input kW
  // dQlatent   latent heat input kW

  //2 nd point room exit
  P=Pi;
  ps_plot_hazirla(tmin,tmax,P);
  yh=new wetair();
  double a2[]=yh.property("tdb_rh",T2,f2,P);
  double h2=a2[8];
  double w2=a2[5];
  double ha2=a2[11];
  //6 th point evaporator exit
  double a6[]=yh.property("tdb_rh",T6,f6,P);
  double h6=a6[8];
  double w6=a6[5];
  double w1=w6;
  double ha1=ha2-dQsensible/m1;
  //15 degree room temperature evaluation
  // 1A noktası
  double a1A[]=yh.property("tdb_w",15.0,w1,P);
  double ha1A=a1A[11];
  double Cp1=(ha2-ha1A)/(T2-15.0);
  double T1=ha1/Cp1;
  double a1[]=yh.property("tdb_w",T1,w1,P);
```

```
 double h1=a1[8];
 double m2=m1-m3;
 double m4=m3;
 double m5=m1;
 // outside air entry
 double a4[]=yh.property("tdb_rh",T4,f4,P);
 double h4=a4[8];
 double w4=a4[5];

 double h5=(m4*h4+m2*h2)/m5;
 double w5=(m4*w4+m2*w2)/m5;
 double a5[]=yh.property("w_h",w5,h5,P);
 double T5=a5[1];
 double Q56=m5*(h5-h6);//KW
 double Q67=m5*(h1-h6);//KW
 double dm=m5*(w5-w6);
 double a8[]=yh.property("w_rh",w6,1.0,P);
 double a9[]=yh.property("w_rh",w5,1.0,P);
 double a[][]=new double[8][a1.length];
 a[0]=a1;
 a[1]=a2;
 a[2]=a2;
 a[3]=a4;
 a[4]=a5;
 a[5]=a6;
 a[6]=a8;
 a[7]=a9;
 // Plot
 // thick blue line
 addData(a9,a5,5,0,0,255);
 //triangle
 addData(a9,a5,24,0,0,255);
 addData(a8,a1,5,0,0,255);
 addData(a6,a1,24,0,0,255);
 addData(a1,a2,5,255,0,0);
 addData(a2,a4,5,255,0,0);
 addData(a2,a4,24,255,0,0);
 ps_plot();
 print(a,"single zone summer air conditioning process thermodynamic properties");
 String baslik2[]={"evaporator heat transfer kW","heater heat transfer kW","condansation mass flow
rate kg/s"};
 double b[][]={{Q56,Q67,dm}};
 print(b,baslik2,"single zone summer air conditioning process exit properties");
 return a;
}
public void print(double a[][],String s)
{//String heading[]={"summer air conditioning"};
 Text.print(a,heading,"summer air conditioning data");}

public void print(double a[][],String heading2[],String s)
{String heading[];Text.print(a,heading2,s);}
public void ps_plot_hazirla(double tmin,double tmax,double P1)
    {
            double x1[][]=new double[62][51];
            double y1[][]=new double[62][51];
            double A[][][]=new double[2][62][51];
            A=ps_verisi(tmin,tmax,P1);
            for(int i=0;i<A[0].length;i++)
```

```
            for(int j=0;j<A[0][0].length;j++)
                {x1[i][j]=A[0][i][j];y1[i][j]=A[1][i][j];}
 pi=new Plot(x1,y1);
 String ss2="Psychrometric diagram "+P+" bar";
 pi.setPlabel(ss2);
 pi.setXlabel("T Dry bulb temperature degree C");
 pi.setYlabel("w specific humidity kg moisture/kg dry air");
 pi.setMinMax(tmin,tmax,0.0,0.05);
 pi.setXgrid(1);
 pi.setYgrid(1);
 }
public static double[][][] ps_verisi(double tmin,double tmax,double P)
{
        //prepare pyschrometric chart data for pressure P
wetair w=new wetair();
        double aa[]=new double[14];
double bb[]=new double[14];
        //f2 ff=new f2();
        //double x1[][]=new double[62][51];
        //double y1[][]=new double[62][51];
        double A[][][]=new double[2][62][51];
//initilise as a function plot
int i,j;
i=0;
double h;
for(double rh=0.1;rh<=1.0;rh+=0.1)
{   j=0;
   for(double t=tmin;t<=tmax;t+=1.0)
   {
            aa=w.property1("tdb_rh",t,rh,P);
            h=aa[8];
              if(rh>0.99)
     {
              h=aa[8];
     A[0][11+j][0]=t;A[1][11+j][0]=aa[5];
     bb=w.property1("w_h",0.0,h,P);
     A[0][11+j][1]=bb[10];A[1][11+j][1]=0.0;
     }
              A[0][i][j]=t;
     A[1][i][j]=aa[5];
              j++;
     }
   i++;
}
return A;
}
 public void addData(double a1[][],int plottype,int red,int green,int blue)
 { double x1[]=new double[2];
   double y1[]=new double[2];
   x1[0]=a1[0][1];
   y1[0]=a1[0][5];
   x1[1]=a1[1][1];
   y1[1]=a1[1][5];
   pi.addData(x1,y1,plottype,red,green,blue);
 }

 public void addData(double a1[],double a2[],int plottype,int red,int green,int blue)
 { double x1[]=new double[2];
```

```
      double y1[]=new double[2];
      x1[0]=a1[1];
      y1[0]=a1[5];
      x1[1]=a2[1];
      y1[1]=a2[5];
      pi.addData(x1,y1,plottype,red,green,blue);
    }

    public void addData(double a1[],int plottype,int red,int green,int blue)
    { double x1[]=new double[1];
      double y1[]=new double[1];
      x1[0]=a1[1];
      y1[0]=a1[5];
      pi.addData(x1,y1,plottype,red,green,blue);
    }

    public void ps_plot()
    {   pi.setMinMax(tmin,tmax,0,0.05);
                pi.plot();
              }

public static void main(String arg[])
{ wetairtest11 wat=new wetairtest11();
double p=1.01325; //  bar
double T1=30.0;   //  room inlet temperature degree C
double m1=30.0;//kg/s
double m3=4.5; //kg/s
double T2=25.0;   //  room exit temperature
double f2=0.5;    //   room exit relative humidity
double T4=35.0;    //  outside inlet temperature
double f4=0.4;    //   outside  relative humidity
double T6=10.0;   //   preheater exit temperature
double f6=0.95;
double Qsensible=205.0; //  sensible heat load
double Qlatent=88.0;   //  latent heat load
double a[][]=wat.summer_air_conditioning(p,m1,m3,T2,f2,T4,f4,T6,f6,Qsensible,Qlatent);
}
}
```

5.3 cooling towers

Cooling towers cooled the water by using evaporative cooling principle. Hot water enters cooling tower. At the other side environmental air enters. Some of the air evaporates and this evaporation effect cools the remaining water. Of course discharged air has a high humidification ratio. In the following figure the basic shematic diagram of a cooling tower is shown in the figure below. The basic mass and energy balance for this system will be:

$m_{a3} = m_{a4} = m_a$ (dry air) (5.3.1)

$m_1 + m_5 + m_{v3} = m_2 + m_{v4}$ (water) or (5.3.2)

$m_1 + m_5 + m_a w_3 = m_2 + m_a w_4$ (water) (5.3.3)

Assuming that feed water flow rate is equivalent to water loss through evaporation process

$m_5 = m_a(w_4 - w_3)$ (5.3.4)

We can also assume that mass flow rate of hot water going into the cooling tower is the same as the cold water exit

$m_1 = m_2$ (5.3.5)

Energy balance:

$m_a h_3 + m_5 h_5 + m_1 h_1 = m_a h_4 + m_2 h_2$ (5.3.6)

$m_a h_3 + m_a(w_4 - w_3) h_5 + m_1 h_1 = m_a h_4 + m_1 h_2.$ (5.3.7)

In real cooling tower analysis, exit condition of moist air can be calculated through mass and heat transfer, but as now, it will be assumed that exit conditions of wet air is known.

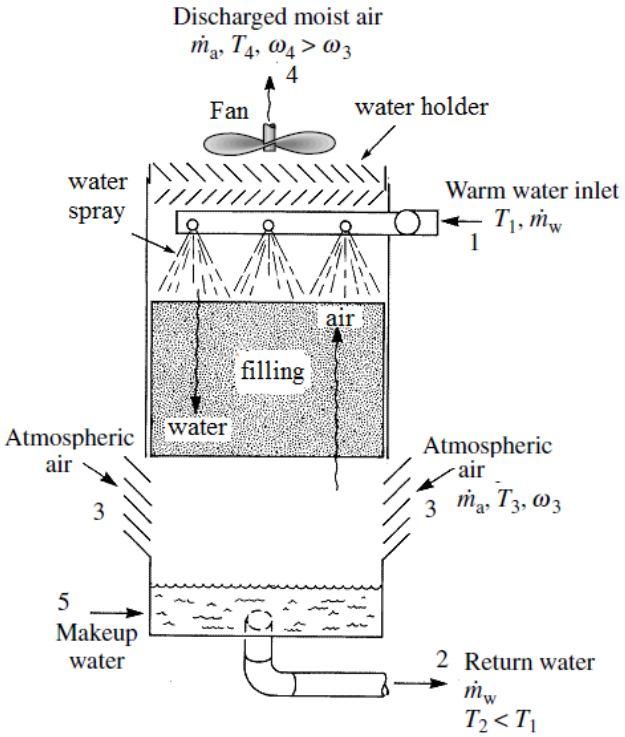


Figure 5.3.1 schematic view of a cooling tower

As a case study let us look at an example problem:

A liquid water stream enters a cooling tower from a condenser at 38 C with a known mass flow rate. A stream of cooled water is returned to the condenser at 30 C and the same flow rate. Makeup water is added at 20 C. Atmospheric air enters the tower at 24 C and ϕ= 40%. Moist air exits the tower at 35 C and ϕ=90%. Determine the mass flow rates of the dry air and the makeup water, in kg/h.

Program 5.3.1 Cooling tower example program

```
import javax.swing.*;
public class wetairtest8
{ // adiabatic cooling process
public static void main(String arg[])
{
wetair wa=new wetair();
steam st=new steam();
//1 hot water inlet
double m1=1.0;//  kg/s
double p=1.01325;//  bar
double T1=38.0;// degree C
double a1[]=st.property("tp",T1,p);
double h1=a1[3];
//2 cold water outlet
double T2=30.0;// degree C
double m2=m1;
a1=st.property("tp",T2,p);
double h2=a1[3];
// atmospheric air inlet
double T3=24.0;//  degree C
```

```
double rh3=0.4;
double a[]=wa.property("tdb_rh",T3,rh3,p);
double h3=a[8];
double w3=a[5];
// cooling tower air exit
double T4=35.0;//  degree C
double rh4=0.9;
a=wa.property("tdb_rh",T4,rh4,p);
double h4=a[8];
double w4=a[5];
// feed water
double T5=20.0;// degree C
a1=st.property("tp",T5,p);
double h5=a1[3];
//air mass flow rate
double ma=(m1*h1-m2*h2)/(h4-h3-(w4-w3)*h5);
double m5=ma*(w4-w3);
double Qwater=m1*h1-m2*h2; //kW
double Qair=ma*(h4-h3); //kW
String s="Wet Cooling Tower\n";
s+="Warm water inlet\n";
s+="m1="+m1+" kg/s\n";
s+="T1="+T1+" degree C\n";
s+="h1="+h1+" kJ/kg\n";
s+="Cold water outlet\n";
s+="m2="+m2+" kg/s\n";
s+="T2="+T2+" degree C\n";
s+="h2="+h2+" kJ/kg\n";
s+="Feed water inlet\n";
s+="m5="+m5+" kg/s\n";
s+="T5="+T5+" degree C\n";
s+="h5="+h5+" kJ/kg\n";
s+="Atmospheric air inlet\n";
s+="ma="+ma+" kg/s\n";
s+="T3="+T3+" degree C\n";
s+="h3="+h3+" kJ/kg\n";
s+="w3="+w3+" kJ/kg\n";
s+=""+'\u00D8'+"3="+rh3+" \n";
s+=" air outlet\n";
s+="ma="+ma+" kg/s\n";
s+="T4="+T4+" degree C\n";
s+="h4="+h4+" kJ/kg\n";
s+="w4="+w4+" kJ/kg\n";
s+=""+'\u00D8'+"4="+rh4+" \n";
s+="Qwater = "+Qwater+" kW\n";
s+="Qair = "+Qair+" kW\n";
System.out.println(s);
}
}
```

```
---------- Capture Output ----------
> "C:\java\bin\javaw.exe" wetairtest8
Wet Cooling Tower
Warm water inlet
m1=1.0 kg/s
T1=38.0 degree C
h1=159.2910629405701 kJ/kg
```

Cold water outlet
m2=1.0 kg/s
T2=30.0 degree C
h2=125.87625939254804 kJ/kg
Feed water inlet
m5=0.0114278760818863 kg/s
T5=20.0 degree C
h5=84.05043818331433 kJ/kg
Atmospheric air inlet
ma=0.45151290670932415 kg/s
T3=24.0 degree C
h3=42.969116948158174 kJ/kg
w3=0.007416100378795528 kJ/kg
Ø3=0.4
air outlet
ma=0.45151290670932415 kg/s
T4=35.0 degree C
h4=119.10275793472566 kJ/kg
w4=0.03272628733485382 kJ/kg
Ø4=0.9
Qwater33.414803548022064 kW
Qair34.37532154020922 kW

> Terminated with exit code 0.

6.0 COMBUSTION AND THERMOCHEMISTRY

6.1 Basics of combustion process

Most of the energy we use today are based on a chemical reaction called combustion. Combustion process requires a fuel, basically a hydrocarbon and oxidant (air or oxygen). Fuel and oxidants together called reactants. When a chemical reaction occurs, the bonds within molecules of the reactants are broken, and atoms and electrons rearrange to form products. In combustion reactions, rapid oxidation of combustible elements of the fuel results in energy release as combustion products are formed. Fuel is mostly made of Hydrogen and Carbon atoms, but some Sulfur and other trace elements can also existed. In solid fuels also non-combustible elements can exist. They can form solids after combustion which is called ash. Combustion is complete when carbon atoms burned to carbondioxide, hydrogen atoms burned to water and sulfur atoms burned to sulfurdioxide, and all other combustible elements are fully oxidized. When this condition is not fulfilled combustion is incomplete. Combustion reaction can be summarized as

Reactants → Products or fuel+oxidizer→Products

As an oxidizer usuakky air is used, except special cases like requirement of high temperatures. Air is a complex mixture of several gases. Air composition is given in table 6.1.1

Table 6.1.1 Composition of atmospheric air

Name	Formula	Mol %
Nitrogen	N_2	78.084
Oxygen	O_2	20.946
Argon	Ar	0.9340
Carbondioxide	CO_2	0.0397
Neon	Ne	0.001818
Helium	He	0.000524

Methane	CH_4	0.000179
Water vapor	H_2O	0.00001-0.05

As it is seen from the table major elements in composition is nitrogen, oxygen and argon. As another simplification, we usually ignore argon as well and assume air composition as 79 % N_2 and 21% O_2 or per kmole of O_2 79/21=3.761905 kmol of N_2 per kmole of oxygen. Nitrogen is noncombustible in normal conditions, but it can be converted to nitrous oxides in relatively small quantities. These are not significant in energy balance, but has big enviromental effects.

It is customary to write the reaction formula with one kmol of fuel input in reactions such as: CH4 or ($0.9CH_4$+$0.2C_2H_6$). When combustion is complete and there are no excess oxygen in products this type of reactions are called stoichiometric reactions. As an example reaction, stoichiometric combustion reaction of methane(CH_4) can be given as follows:

$$\mathbf{CH_4+2(O_2+3.761905\ N_2)\rightarrow CO_2+2H_2O+7.52381N_2}$$

As it is seen from the equation, amount of nitrogen entered into reaction is emerged exactly the same amount in products, oxygen is not existed in products and carbon burned to carbondioxide and hydrogen burned to water. Amounts of chemicals appeared in products are depends on the mass balances of the atoms. Number of atomsa re conserved in reaction, so product mol numbers of molecules are depends on the number of atoms in reactants. In above reaction one carbon atom went into the reactions, therefore one carbondioxide appears in products, and 4 hydrogen atom went into reaction so two water molecules are emerged as a result. From emerging CO_2 and H_2O, required O_2 in the oxydant can be calculated to be 2 from O atomic balance

O balance ; 2x=2+2 x=2

Amount of air going into the reaction is important concept in combustion. For the above reaction

Amount of air per kmol of fuel=2*(1+ 3.761905)=9.52381 kmol air/kmol fuel. Enthalpy definition for combustion process usually taken 298.16 degree K(25 degree C) as reference point. Furthermore a state of enthalpy existed due to chemical potentials of reactions. During a chemical reacton, some chemical bonds are broken, and a new ones are formed. The chemical energy associated with these bonds are in general is different. The energy level due to chemical reactions at the reference temperature of 298.16 K is called enthalpy of formation. Since the temperature in this defination is constatnt, it can be given as a single reference value for each molecules in reaction. So enthalpy of a molecule going into the reaction calculated as:

$h_T(T) = h(T) - h(298.16) + \Delta h_f(298.16)$ where $\Delta h_f(298.16)$ is formation enthalpy and $h_T(T)$ is the total enthalpy used in chemical reaction processes. The first law of thermodynamic for a combustion states than

$$Q + W = \sum_{products} N_i \left(h_{Ti}(T_{pruduct}) + \frac{V_i^2}{2} + gz_i\right) - \sum_{reac\,tan\,ts} N_i \left(h_{Ti}(T_{reac\,tan\,t}) \frac{V_i^2}{2} + gz_i\right) \quad (6.1.1)$$

In this equation potential and kinetic energy terms are ignored, equation simplified as:

$$Q + W = \sum_{products} N_i h_{Ti}(T_{pruduct}) - \sum_{reac\,tan\,ts} N_i h_{Ti}(T_{reac\,tan\,t}) \quad (6.1.2)$$

The work term is only apply to fuel cell type applications. If only a combustion chamber is considered, work term can be considered zero and heat transfer of a combustion reaction becomes:

$$Q = \sum_{products} N_i h_{Ti}(T_{pruduct}) - \sum_{reac\,tan\,ts} N_i h_{Ti}(T_{reac\,tan\,t}) \quad (6.1.3)$$

when N_i is the mol numbers of the molecules going into the reactions. Second hand side of the equation is the enthalpy of the reaction

$$H = \sum_{products} N_i h_{Ti}(T_{pruduct}) - \sum_{reac\,tan\,ts} N_i h_{Ti}(T_{reac\,tan\,t}), \quad (6.1.4)$$

similarly entropy of the reaction can be written as

$$S^0 = \sum_{products} N_i s^0_{Ti}(T_{pruduct}) - \sum_{reac\,tan\,ts} N_i s^0_{Ti}(T_{reac\,tan\,t}) \quad (6.1.5)$$

$$S = \sum_{products} N_i \left[s^0_{Ti}(T_{pruduct}) - R\,ln(P_i)\right] - \sum_{reac\,tan\,ts} N_i \left[s^0_{Ti}(T_{reac\,tan\,t}) - R\,ln(P_i)\right] \quad (6.1.6)$$

$$S = S^0 + \sum_{reac\,tan\,ts} N_i [R\,ln(P_i)] - \sum_{products} N_i [R\,ln(P_i)] \quad (6.1.7)$$

Gibbs free energy of reaction:

$$G^0 = \sum_{products} N_i \left[h_{Ti}(T_{pruduct}) - T_{product} s^0_{Ti}(T_{pruduct})\right] - \sum_{reac\,tan\,ts} N_i [h_{Ti}(T_{reac\,tan\,t}) - T_{reac\,tan\,t} s_{Ti}(T_{reac\,tan\,t})] \quad (6.1.8)$$

$$G = G^0 + \sum_{reac\,tan\,ts} N_i [R\,ln(P_i)] - \sum_{products} N_i [R\,ln(P_i)] = G^0 - R\,ln\left(\frac{\prod_{products} P_i^{N_i}}{\prod_{reac\,tan\,ts} P_i^{N_i}}\right) \quad (6.1.9)$$

As a first example heat transfer for a reaction is calculated by using polynomial eqautions as a base for Cp as:

Program 6.1.1 CH_4 fuel combustion calculated in excel by using a 4th degree polynomial functions

Combustion

	a	b	c	d	e	f	hf
CH4	1.9399E+01	6.5653E-02	-1.4662E-05	-6.3296E-10	5.8194E-13	-5.1165E-17	-7.4850E+04
O2	2.6756E+01	1.1507E-02	-4.8733E-06	1.2462E-09	-1.6495E-13	8.8224E-18	0.0000E+00
N2	2.7946E+01	3.6088E-03	2.1541E-06	-1.4913E-09	2.9670E-13	-1.9491E-17	0.0000E+00
CO2	2.3852E+01	5.4070E-02	-3.1694E-05	9.2267E-09	-1.2969E-12	7.0209E-17	-3.9352E+05
H2O	3.1748E+01	4.9577E-03	7.9500E-06	-4.0642E-09	7.1971E-13	-4.4041E-17	-2.4182E+05

	T	T*T	T^3	T^4	T^5	T^6
Tref	2.9816E+02	8.8899E+04	2.6506E+07	7.9031E+09	2.3564E+12	7.0258E+14
Tr	2.9816E+02	8.8899E+04	2.6506E+07	7.9031E+09	2.3564E+12	7.0258E+14
Tp	1.0000E+03	1.0000E+06	1.0000E+09	1.0000E+12	1.0000E+15	1.0000E+18

		Ni	hi	hi+hf	Hi	
R	**CH4**	1.0000E+00	0.0000E+00	-7.4850E+04	-7.4850E+04	
	O2	2.0000E+00	0.0000E+00	0.0000E+00	0.0000E+00	
	N2	7.5238E+00	0.0000E+00	0.0000E+00	0.0000E+00	**-7.4850E+04**
P	**CO2**	1.0000E+00	3.3170E+04	-3.6035E+05	-3.6035E+05	
	H2O	2.0000E+00	2.6226E+04	-2.1559E+05	-4.3119E+05	
	O2	0.0000E+00	2.2722E+04	2.2722E+04	0.0000E+00	
	N2	7.5238E+00	2.1633E+04	2.1633E+04	1.6277E+05	**-6.2877E+05**
					Q	**-5.539225E+05**

To check the accuracy the same calculation can be repeated in java environment

Program 6.1.2 CH_4 fuel combustion calculated in java by using a 4th degree polynomial functions

```
public class reactiontest3
{

static double ht(int i,double T)
{ double Tr=298.16;
double R=8.3145;
double C[][]=
{{1.939869186926E+01,6.565333278907E-02,-1.466182490573E-05,-6.329616609155E-
10,5.819409563279E-13,-5.116465644319E-17},
 {2.675596850695E+01,1.150718911476E-02,-4.873333185805E-06,1.246226000157E-09,-
1.649499826041E-13,8.822371434893E-18},
 {2.794626725797E+01,3.608818970753E-03,2.154061888770E-06,-1.491279827848E-
09,2.967027913986E-13,-1.949118588911E-17},
 {2.385184692766E+01,5.407033093625E-02,-3.169416460166E-05,9.226660264546E-09,-
1.296877816241E-12,7.020934713906E-17},
 {3.174807138649E+01,4.957708828333E-03,7.949985639501E-06,-4.064180760029E-
09,7.197141669681E-13,-4.404120668938E-17}};
 double hf[]={-74850.0,0.0,0.0,-393520.0,-241820.0};
 double hh=(C[i][0]*(T-Tr)+C[i][1]/2.0*(T*T-Tr*Tr)+C[i][2]/3.0*(T*T*T-
Tr*Tr*Tr)+C[i][3]/4.0*(T*T*T*T-Tr*Tr*Tr*Tr)+C[i][4]/5.0*(T*T*T*T*T-
Tr*Tr*Tr*Tr*Tr)+C[i][5]*(T*T*T*T*T*T*T-Tr*Tr*Tr*Tr*Tr*Tr*Tr))+hf[i];
 return hh;
}
public static void main(String arg[])
{
String s[]={"ch4","o2","n2","co2","h2o"};

double N[][]={{1.0,0.0},{2.0,0.0},{7.523809524,7.523809524},{0.0,1.0},{0.0,2.0}};
double Tproduct=1000.0;//degree K
double Treactant=298.16;//degree K
int k=s.length;
double Qproduct=0;
double Qreactant=0;
for(int i=0;i<k;i++)
{  Qproduct+=N[i][1]*ht(i,Tproduct);}
for(int i=0;i<k;i++)
{ Qreactant+=N[i][0]*ht(i,Treactant);}
double Q=Qproduct-Qreactant;
System.out.println("Q="+Q+" kW");
}}
```

```
 ---------- Capture Output ----------
> "C:\java\bin\javaw.exe" reactiontest3
Q=-553984.3967321854 kW

> Terminated with exit code 0.
```

Now we can calculate the same result with partially continious curve fitting functions defined in Gas.java class

Program 6.1.3 CH_4 fuel combustion calculated in java by using partial continous polynomial curve fitting with Gas class

```
 public class reactiontest1
{
public static void main(String arg[])
{
```

```
String s[]={"ch4","o2","n2","co2","h2o"};
double N[][]={{1.0,0.0},{2.0,0.0},{7.523809524,7.523809524},{0.0,1.0},{0.0,2.0}};
double Tproduct=1000.0;//degree K
double Treactant=298.16;//degree K
int k=s.length;
Gas g[]=new Gas[k];
for(int i=0;i<k;i++)
{g[i]=new Gas(s[i]);
}
double Qproduct=0;
double Qreactant=0;
for(int i=0;i<k;i++)
{Qproduct+=N[i][1]*g[i].ht(Tproduct);}
for(int i=0;i<k;i++)
{Qreactant+=N[i][0]*g[i].ht(Treactant);}
double Q=Qproduct-Qreactant;
System.out.println("Q="+Q);
}}
```

```
---------- Capture Output ----------
> "C:\java\bin\javaw.exe" reactiontest1
Q=-555419.69706795

> Terminated with exit code 0.
```

As it is seen from the results there are the difference in results. To check accuracy the same program can also be run by using cubic spline curve fitting version of the ideal gas equation of state.

Program 6.1.4 CH_4 fuel combustion calculated in java by using cubic spline curve fitting with GasCS class

```
public class reactiontest1CS
{
public static void main(String arg[])
{
String s[]={"ch4","o2","n2","co2","h2o"};
double N[][]={{1.0,0.0},{2.0,0.0},{7.523809524,7.523809524},{0.0,1.0},{0.0,2.0}};
double Tproduct=1000.0;//degree K
double Treactant=298.16;//degree K
int k=s.length;
GasCS g[]=new GasCS[k];
for(int i=0;i<k;i++)
{g[i]=new GasCS(s[i]);
}
double Qproduct=0;
double Qreactant=0;
for(int i=0;i<k;i++)
{Qproduct+=N[i][1]*g[i].ht(Tproduct);}
for(int i=0;i<k;i++)
{Qreactant+=N[i][0]*g[i].ht(Treactant);}
double Q=Qproduct-Qreactant;
System.out.println("Q="+Q);
}
}
```

```
---------- Capture Output ----------
> "C:\java\bin\javaw.exe" reactiontest1CS
Q=-555425.1381705669

> Terminated with exit code 0.
```

The results indicated that partially continious curve fitting and cubic spline curve fitting function results are close enough so both can be utilized for practical purposes, but single polynomial curve fitting error levels indicates to be bigger.
What would happened if combustion product teperature is 298.16 K as well? The value obtained for this case is called **heating value of the fuel**. Result by using program 6.1.4 will be

```
 ---------- Capture Output ----------
> "C:\java\bin\javaw.exe" reactiontest1CS
Q=-802300.9008195192

> Terminated with exit code 0.
```

There is only one problem here, at 298.16 K the water in product will condense to liquid form, therefore the enthalpy of evaporation should also be considered (of course at 298.16 K value) in the equation

Program 6.1.5 CH_4 fuel heating value calculated in java by using cubic spline curve fitting with GasCS class with liquid water exit

```
 public class reactiontest2CS
{
public static void main(String arg[])
{
String s[]={"ch4","o2","n2","co2","h2ol"};
double N[][]={{1.0,0.0},{2.0,0.0},{7.523809524,7.523809524},{0.0,1.0},{0.0,2.0}};
double Tproduct=298.16;//degree K
double Treactant=298.16;//degree K
int k=s.length;
GasCS g[]=new GasCS[k];
for(int i=0;i<k;i++)
{g[i]=new GasCS(s[i]);
}
double Qproduct=0;
double Qreactant=0;
for(int i=0;i<k;i++)
{Qproduct+=N[i][1]*g[i].ht(Tproduct);}
for(int i=0;i<k;i++)
{Qreactant+=N[i][0]*g[i].ht(Treactant);}
double Q=Qproduct-Qreactant;
System.out.println("Q="+Q);
}
}
```

```
 ---------- Capture Output ----------
> "C:\java\bin\javaw.exe" reactiontest2CS
Q=-890308.9008195192

> Terminated with exit code 0.
```

As it is seen from the result the value is higher now. Logically at this temperature range water should condense in the product, therefore the second result is the actual value of maximum possible heat transfer from a combustion reaction. It is called higher heating value. For commercial purposes, people also use the first assumption (vapor phase of the water output at 298.16 K) and it is called lower heating value. By taking reference point of lower heating value and condensing type boilers some manufacturers claims that their boiler efficiency is more than 100%., which is of course a false claim.

Another limit in combustion is adiabatic flame temperature. If combustion gases is heated by the reaction and no heat is ejected from the reaction, products will reach to a maximum temperature with zero heat transfer . This limit is called adiabatic flame temperature. Curve fitting should applied to find this limit value.

Program 6.1.6 CH_4 adiabatic flame temperature calculated in java by using bisection curve fitting with Gas class

```
 import javax.swing.*;
public class reactiontest4
{
public static double Q(String s[],double N[][],double Treactant,double Tproduct)
{int k=s.length;
Gas g[]=new Gas[k];
for(int i=0;i<k;i++)
{g[i]=new Gas(s[i]);
}
double Qproduct=0;
double Qreactant=0;
for(int i=0;i<k;i++)
{Qproduct+=N[i][1]*g[i].ht(Tproduct);}
for(int i=0;i<k;i++)
{Qreactant+=N[i][0]*g[i].ht(Treactant);}
double Q=Qproduct-Qreactant;
return Q;
}

public static double Tadiabatic(String s[],double N[][],double Treactant,double Tproduct_low,double
Tproduct_high)
{
//bisection root finding method
double xl=Tproduct_low;
double xu=Tproduct_high;
double test;
double xr=0;
double es,ea;
double fxl,fxr;
int maxit=100,iter=0;
es=0.0000001;
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= Q(s,N,Treactant,xl);
   fxr= Q(s,N,Treactant,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 arg[])
{
String s[]={"ch4","o2","n2","co2","h2o"};
double N[][]={{1.0,0.0},{2.0,0.0},{7.523809524,7.523809524},{0.0,1.0},{0.0,2.0}};
double Tproduct=1000.0;//degree K
double Treactant=298.16;//degree K
double Tadiabatic=Tadiabatic(s,N,Treactant,298.16,3000.0);
double Q=Q(s,N,Treactant,Tadiabatic);
System.out.println("Tadiabatic="+Tadiabatic);
System.out.println("Q="+Q);
}
}
```

```
---------- Capture Output ----------
> "C:\java\bin\javaw.exe" reactiontest4
Tadiabatic=2326.6911578521513
Q=-2.3041048552840948E-4

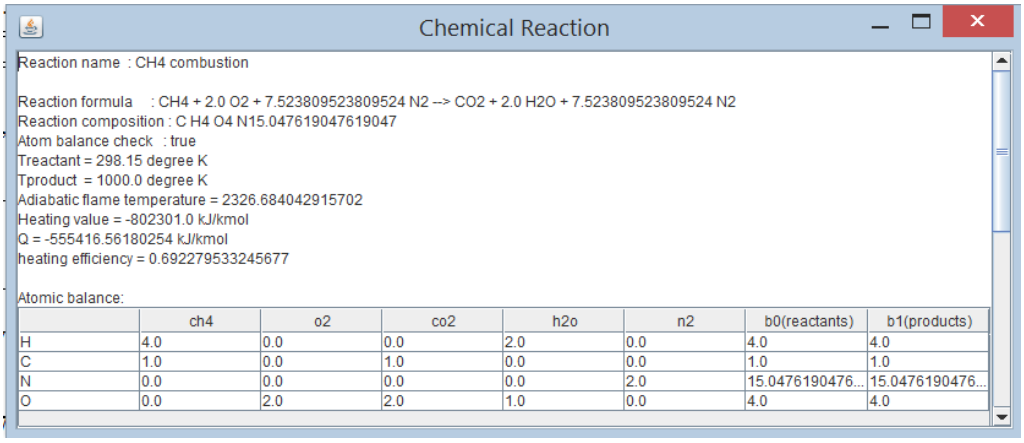
> Terminated with exit code 0.
```

In order to make this calculations more formal, a chemical reaction class is created and listed in appendix 8.1 By using this class we can give the gas composition directly or it can be read from a file called Reaction.txt through a given reaction name.

Program 6.1.7 CH_4 combustion calculated in java by using class chemicalReaction in A8.1

```
import javax.swing.*;
import java.io.*;

public class reactiontest6
{
public static void main(String arg[]) throws IOException
{
String s[]={"ch4","o2","co2","h2o","n2"};
double N[][]={{1.0,0.0},{2.0,0.0},{0.0,1.0},{0.0,2.0},{2.0*79/21,2.0*79/21}};
double Tproduct=1000.0;//degree K
double Treactant=298.15;//degree K
chemicalReaction cr=new chemicalReaction("CH4 combustion",s,N);
cr.print(Treactant,Tproduct);
}
}
```



	ch4	o2	co2	h2o	n2	b0(reactants)	b1(products)
H	4.0	0.0	0.0	2.0	0.0	4.0	4.0
C	1.0	0.0	1.0	0.0	0.0	1.0	1.0
N	0.0	0.0	0.0	0.0	2.0	15.0476190476...	15.0476190476...
O	0.0	2.0	2.0	1.0	0.0	4.0	4.0

A common thing we have done so far, is used the general reaction concepts. In doing so we have to give all coefficients of the reaction. Combustion is a special chemical raction and it happens between a fuel and oxydant(normally air), therefore, instead of giving all constants, fuel combination and amount of air can be given to maket he basic calculations. A class called combustion is developed for this purpose and program list is given in appendix 8.2. Let us try program in the same example program.

Program 6.1.8 CH_4 adiabatic flame temperature and reaction calculated in java by using class combustion in A8.2

```
import javax.swing.*;
import java.io.*;

public class combustiontest
{
public static void main(String arg[])
{
String s[]={"ch4"};
double N[]={1.0};
double a_f=1.0;
double Tproduct=1000.0;//degree K
double Tfuel=298.16;//degree K
double Tair=298.16;//degree K
combustion cc=new combustion("CH4 combustion",s,N,a_f);
System.out.println(cc.toString());
double Tadiabatic=cc.Taf(Tfuel,Tair,298.16,3000.0);
double Q=cc.H(Tfuel,Tair,Tproduct);
System.out.println("Tadiabatic="+Tadiabatic);
System.out.println("Q="+Q);
}}
```

```
---------- Capture Output ----------
> "C:\java\bin\javaw.exe" combustiontest
[CH4] + 2.0000*[ O2 + (79/21) *N2] --> CO2 + 2.0000 H2O + 7.5238 N2
Tadiabatic=2326.691141496301
Q=-555419.6970720381

> Terminated with exit code 0.
```

More complex reactions can also be given. For example 10% excess air for 90% ch4 and 10% c2h6 fuel will give:

```
---------- Capture Output ----------
> "C:\java\bin\javaw.exe" combustiontest
[0.9000 CH4 + 0.1000 C2H6] + 2.3650*[ O2 + (79/21) *N2] --> 1.1000 CO2 + 2.1000 H2O + 8.8969
N2 + 0.2150 O2
Tadiabatic=2196.5441922754053
Q=-577682.314331347

> Terminated with exit code 0.
```

10% depleted air for 90% ch4 and 10% c2h6 fuel will give:

```
---------- Capture Output ----------
> "C:\java\bin\javaw.exe" combustiontest
[0.9000 CH4 + 0.1000 C2H6] + 1.9350*[ O2 + (79/21) *N2] --> 0.6700 CO2 + 2.1000 H2O + 7.2793
N2 + 0.4300 CO
Tadiabatic=2224.182699812055
```

```
Q=-500631.88144321425
> Terminated with exit code 0.
```

CO apeared in the last reaction, because there are not enough oxygen existed for completed reaction.

6.2 Chemical equilibrium , Stoichiometric algorithm

Actual products formed in a reaction is not a clear cut concept as we defined in the previous chapter. The actual products formed in a chemical reactions can only be determined experimentally and as a function of the environmental values such as temperature, pressure and time. When these parameters changed actual products can change as well. One way of estimating what products will be formed is to use equilibrium concept. All physical systems are tend to minimize their energy levels when there are no any external force or energy to the system. For example a bubble takes spherical shape under the influence of surface tension which has the minimal area for the given volume. Chemical reactions when temperature and pressure is constant and long enough time is given, tends to minimize it energy level, specifically giibs free energy level (or maximize its entropy level). This state is called chemical equilibrium. For example consider a closed system consisting initially of a gaseous mixture of carbondioxide , oxygen and carbonic acid. A reaction might take place is
$CO_2 + H_2O \rightleftharpoons H_2CO_3$
At equilibrium the system will consist basically of three components, CO_2, H_2CO_3 and H_2O, for not all the components gases. This is what is called soda reaction. When soda bottle opened equailibrium condition (pressured in the system) is changed and rection slides to a new equilibrium states, so that some omount of CO_2 and H_2O is formed and amount of carbonic acid is reduced . Changes in the amounts of these components during the opening of soda bottle:
$dn_{CO_2} = -dn_{H_2CO_3}\ dn_{H_2O} = -dn_{H_2CO_3}$ (6.2.1) where dn denotes the differential change in the representative component.
$-\frac{dn_{CO_2}}{1} = -\frac{dn_{H_2O}}{1} = \frac{dn_{H_2CO_3}}{1}$ (6.2.2)
Equilibrium is a condition of balance. When equilibrium is established amount of CO_2 and H_2O and H_2CO_3 tends to be balanced with each other. But when equilibrium is broken and a new conditions arise, like opening of a soda bottle, a new equilibrium will be established. And during these process total gibbs free energy of the system will be minimised.
$dG(T,P) = 0$ (6.2.3) This amount can be determined from chemical potential changes (see the first chapter for details)

$$dG(T,P) = 1 * \mu_{CO_2} dn_{CO_2} + 1 * \mu_{H_2O} dn_{H_2O} + 1 * \mu_{H_2CO_3} dn_{H_2CO_3} = 0 \quad (6.2.4)$$

$$dG(T,P) = \left(-1\mu_{CO_2} - 1 * \mu_{H_2O} + 1 * \mu_{H_2CO_3}\right) dn_{H_2O} = 0 \quad (6.2.5)$$

This is called the equilibrium reaction.For a more general equilibrium equation such as:
$\nu_A A + \nu_B B \rightleftharpoons \nu_c C + \nu_D D$ (6.2.6) where ν's are stoichiometric coefficients. The following equilibrium case is existed:
$-\frac{dn_A}{\nu_A} = -\frac{dn_B}{\nu_B} = \frac{dn_C}{\nu_C} = \frac{dn_D}{\nu_D} = d\varepsilon$ (6.2.7) where $d\varepsilon$is the proportionality factor. From which the following expressions are obtained.
$dn_A = -\nu_A d\varepsilon\ \ dn_B = -\nu_B d\varepsilon\ dn_c = \nu_c d\varepsilon\ \ dn_D = \nu_D d\varepsilon$ (6.2.8)
For this case the Gibbs free energy equation takes form:

$$dG(T,P) = -\nu_A \mu_A - \nu_B \mu_B + \nu_C \mu_C + \nu_D \mu_D = 0 \quad (6.2.9)$$

For ideal gas mixtures chemical potential can be expressed as specific gibbs free energy as
$\mu_i = h_i + Ts_i{}^0 + RT\ ln\left[\frac{P_i}{P_{ref}}\right] = g_i^0 + RT\ ln\left[\frac{P_i}{P_{ref}}\right]$ (6.2.10) where P_i is the partial pressure of the gas. For and ideal gas, partial pressure can be expressed as a function of components in the total components

$y_i = \frac{n_i}{n_{total}}$ $P_i = y_i P = \frac{n_i}{n_{total}} P$. (6.2.11) Substituting these into the equation gives:

$\mu_i = g_i^0 + RT\, ln\left[\frac{n_i}{n_{total}} \frac{P}{P_{ref}}\right]$ (6.2.12) Then the basic Gibbs free energy equation becomes

$dG(T,P) = -\nu_A \left[g_A^0 + RT\, ln\left(\frac{n_A}{n_{total}} \frac{P}{P_{ref}}\right)\right] - \nu_B \left[g_B^0 + RT\, ln\left(\frac{n_B}{n_{total}} \frac{P}{P_{ref}}\right)\right] +$

$\nu_C \left[g_C^0 + RT\, ln\left(\frac{n_C}{n_{total}} \frac{P}{P_{ref}}\right)\right] + \nu_D \left[g_D^0 + RT\, ln\left(\frac{n_D}{n_{total}} \frac{P}{P_{ref}}\right)\right] = 0$ (6.2.13)

Arranging equation:

$dG^0(T,P) = -\nu_A g_A^0 - \nu_B g_B^0 + \nu_C g_C^0 + \nu_D g_D^0$ (6.2.14)

$dG(T,P) = dG^0(T,P) - RT\left[\nu_A\, ln\left(\frac{n_A}{n_{total}} \frac{P}{P_{ref}}\right) + \nu_B\, ln\left(\frac{n_B}{n_{total}} \frac{P}{P_{ref}}\right) - \nu_C\, ln\left(\frac{n_C}{n_{total}} \frac{P}{P_{ref}}\right) -\right.$

$\left.\nu_D\, ln\left(\frac{n_D}{n_{total}} \frac{P}{P_{ref}}\right)\right] = 0$ (6.2.15)

This equation can be written as:

$-\frac{dG^0(T,P)}{RT} = \left[\nu_A\, ln\left(\frac{n_A}{n_{total}} \frac{P}{P_{ref}}\right) + \nu_B\, ln\left(\frac{n_B}{n_{total}} \frac{P}{P_{ref}}\right) - \nu_C\, ln\left(\frac{n_C}{n_{total}} \frac{P}{P_{ref}}\right) -\right.$

$\left.\nu_D\, ln\left(\frac{n_D}{n_{total}} \frac{P}{P_{ref}}\right)\right]$ (6.2.16) or

$exp\left(-\frac{dG^0(T,P)}{RT}\right) = \left[\frac{\left(\frac{n_A}{n_{total}} \frac{P}{P_{ref}}\right)^{\nu_A} * \left(\frac{n_B}{n_{total}} \frac{P}{P_{ref}}\right)^{\nu_B}}{\left(\frac{n_C}{n_{total}} \frac{P}{P_{ref}}\right)^{\nu_C} \left(\frac{n_D}{n_{total}} \frac{P}{P_{ref}}\right)^{\nu_D}}\right] = \left[\left(\frac{n_A^{\nu_A} n_B^{\nu_B}}{n_C^{\nu_C} n_D^{\nu_D}}\right)\right] \left(\frac{P}{n_{total} P_{ref}}\right)^{\nu_A + \nu_B - \nu_C - \nu_D}$

(6.2.17)

The left hand side of the equation is called equilibrium constant, which is only function of temperature for ideal gases.

$K(T) = exp\left(-\frac{dG^0(T,P)}{RT}\right)$ (6.2.18)

Equilibrium constant evaluation is already defined in chemical Reaction class. As an example let us evaluate equilibrium constant of the equilibrium reaction

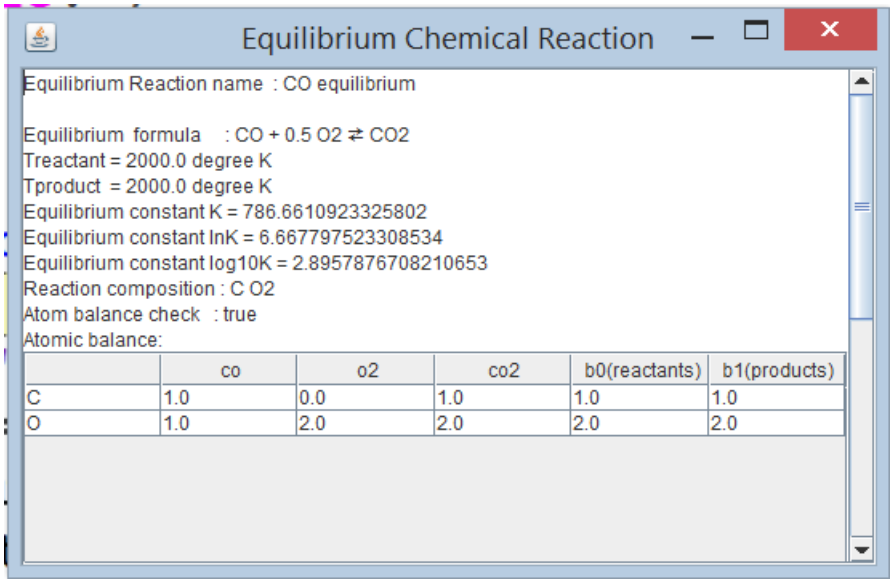
$CO + \frac{1}{2} O_2 \rightleftharpoons CO_2$

At T=2000 K

Program 6.2.1 calculation of equailibrium constant by using ChemicalReaction class

```
import javax.swing.*;
import java.io.*;

public class reactiontest7
{
public static void main(String arg[]) throws IOException
{
String s[]={"co","o2","co2"};
double N[][]={{1.0,0.0},{0.5,0.0},{0.0,1.0}};
double Tproduct=2000.0;  // degree K
double Treactant=2000.0; // degree K
chemicalReaction cr=new chemicalReaction("CO equilibrium",s,N);
cr.equilibrium_print(Treactant,Tproduct);
}
}
```



	co	o2	co2	b0(reactants)	b1(products)
C	1.0	0.0	1.0	1.0	1.0
O	1.0	2.0	2.0	2.0	2.0

Now we can use this knowledge to calculate an equilibrium composition. For a better understanding of the process, we would like to solve an example problem by hand, before establishing a computer solution. One kmol of CO an done kmol of O_2 established an equilibrium at 3000 K. The equilibrium reaction for this is as follows:

$CO + \frac{1}{2}O_2 \rightleftharpoons CO_2$

The reaction will be

$CO+O_2 \rightarrow n_0CO+n_1O_2+n_2CO_2$

Find the equilibrium composition. System pressure is P=1.01325 bar. (Pref=1.01325 bar)

The reaction will be

$CO+O_2 \rightarrow n_0CO+n_1O_2+n_2CO_2$

Find the equilibrium composition.

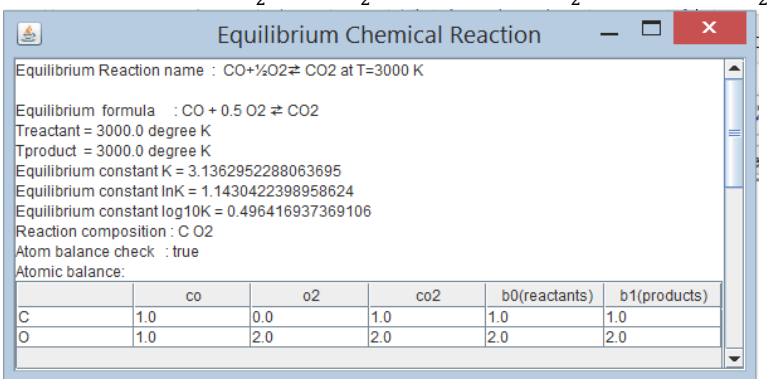
Atomic balance for C: $1=n_0+n_2$

Atomic balance for O: $3=n_0+2n_{1+}2n_2$

Equilibrium constant at T=3000 K :

K=3.1362952

$n_2 = 1 - n_0 \;\; n_1 = \frac{3-n_0-2n_2}{2} = \frac{1+n_0}{2} \; n_{total} = n_0 + \frac{1+n_0}{2} + 1 - n_0 = \frac{3+n_0}{2}$



	co	o2	co2	b0(reactants)	b1(products)
C	1.0	0.0	1.0	1.0	1.0
O	1.0	2.0	2.0	2.0	2.0

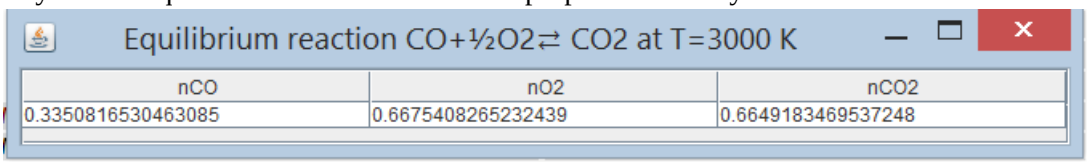
$$ln(3.1362952) = \left[ln\left(\frac{2n_0}{3+n_0}\right) + 0.5\, ln\left(\frac{1+n_0}{3+n_0}\right) - ln\left(\frac{2(1-n_0)}{3+n_0}\right)\right]$$

If this equation is solved by excel solver

n0=nCO	0.335081662
n1=nO2	0.667540831
n2=NCO	0.664918338

ntotal	1.667540831
lnK	1.143042231
	-1.143042319
equation	-8.789E-08

Now the same solution can be carried out by computer program. In order to solve the system a system of equation solver is used. For this purpose continuity method is used.



Program 6.2.1 fi_xj abstract class to define chemical equilibrium class

```
public abstract  class fi_xj extends Mathd
{
  // 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
  abstract public double[] func(double x[]);

public double[] xp(double x[],double j,double m,double h)
{
//h = dx
//j=-4..4 multiplication factor
//m derivative taken value
int k=x.length;
double xx[]=new double[k];
for(int i=0;i<k;i++)
{             if(i==m) xx[i]=x[i]+j*h;
   else     xx[i]=x[i];
}
return xx;
}
public double[][] dfunc(double x[])
{
double h=1.0e-4;
return dfunc(x,1,h);
}
public double[][] dfunc(double x[],int n)
{
double h=1.0e-4;
return dfunc(x,n,h);
}

public double[][] dfunc(double x[],int n,double h)
{
int k=x.length; //number of x values (dimension)
double df[][]=new double[k][k];
for(int i=0;i<k;i++)
{ for(int j=0;j<k;j++)
  {df[i][j]=dfunc(x,n,i,j,h);}
}
return df;
```

```
}

public double dfunc(double x[],int n,int j,int m)
{
double h=1.0e-6;
return dfunc(x,n,j,m,h);
}

public double dfunc(double x[],int n,int j,int m,double h)
{ // derivative of x[m]
  //m derivative taken value
  //jth equation
  //nth derivative of the equation (n=0) function value
  //n=1 first derivative n=2 second derivative..
int k=x.length; //number of x values (dimension)
double hh=1/h;
double ff[][]=new double[9][3];
double xp[][]=new double[9][3];
double df;
int j1;
double x1=0;
x1=x[m];
    for(int jj=-4;jj<=4;jj++)
    {  j1=jj+4;
                xp[j1]=xp(x,jj,m,h);
       ff[j1]=func(xp[j1]);
    }
if(n==0)
{df=ff[4][j];}
else if(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;}
else if(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;}
else if(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;}
else if(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;}
else if(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;}
else if(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;}
else if(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;}
else if(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 6.2.2 Chemical equilibrium function

```
 import java.io.*;
import java.util.*;
import javax.swing.*;
import java.awt.*;
import java.awt.event.*;
public class f_equilibrium extends fi_xj
{ public double n0[];
  public double n[];
  int nn,n1,n2;
  String name[];
  double TR,TP;
  public chemicalReaction cr;
  public double A[][];
  public double b[];
  public Gas g[];
  double Pref=1.01325;//bar
  double P;
  chemicalReaction chr[];
  public f_equilibrium(double n0i[],String  namei[], chemicalReaction chri[],double T1i,double
T2i,double Pi)
  { nn=n0i.length;
    int nn1=namei.length;
    if(nn!=nn1) System.out.println("array size error nn="+nn+"nn1="+nn1);
    P=Pi;
    n2=chri.length;
    n1=nn-n2;
   n0=new double[nn];
   n=new double[nn];
   g=new Gas[nn];
   name=new String[nn];
   TR=T1i;
   TP=T2i;
   chr=new chemicalReaction[n2];
   for(int i=0;i<n2;i++) {chr[i]=chri[i];}
   for(int i=0;i<nn;i++)
   {
              n0[i]=n0i[i];
    name[i]=namei[i];
    g[i]=new Gas(name[i]);
   }
   double N[][]=new double[nn][2];
   for(int i=0;i<nn;i++)
   {  N[i][0]=n0[i];
      N[i][1]=n[i];
   }
   cr=new chemicalReaction("base reaction",name,N);
   cr.conservation();
   A=cr.A;
   b=cr.b0;
  }
  public double[] func(double n[])
  { double b1[]=new double[nn];
    double nt=0;
    for(int i=0;i<nn;i++)
    {nt+=n[i];}
              for(int ii=0;ii<n1;ii++)
              { for(int ij=0;ij<nn;ij++)
```

```
    { double carpim=A[ii][ij]*n[ij];
                        b1[ii]+=carpim;
    }
    b1[ii]=b1[ii]-b[ii];
  }
  double lnK;
  double total;
  double mu[][]=new double[nn][2];
  double nx[];
  for(int ii=n1;ii<nn;ii++)
            {
              int ij=ii-n1;
              int nn2=chr[ij].ngas;
              nx=new double[nn2];
              for(int i=0;i<nn;i++)
      {
                    for(int j=0;j<nn2;j++)
         {  if(name[i].equals(chr[ij].name[j]))
            {   mu[i][0]=chr[ij].N[j][0];mu[i][1]=chr[ij].N[j][1];
                nx[j]=n[i];

                        break;
                      }
        }
       }
              total=0;
              for(int i=0;i<chr[ij].ngas;i++)
    { if(nx[i]!=0)
      {total-=mu[i][0]*Math.log(nx[i]/nt*P/Pref);
       total+=mu[i][1]*Math.log(nx[i]/nt*P/Pref);
      }
              }
    lnK=chr[ij].lnK(TR,TP);
    b1[ii]=lnK-total;
            }
 return b1;
}
   public  void equilibrium_print(double ni[])
   {
              double x[]=new double[nn];
              double x0[]=new double[nn];
              double nnt=0;
              double nn0t=0;
              for(int i=0;i<nn;i++){nnt+=ni[i];nn0t+=n0[i];}
              for(int i=0;i<nn;i++){x[i]=ni[i]/nnt;x0[i]=n0[i]/nn0t;}
   JTextArea jta;
   JPanel jpan=new JPanel();
   jpan.setLayout(new BorderLayout());
   String s="";
   s+="Treactant = "+TR+" degree K\n";
   s+="Tproduct  = "+TP+ " degree K\n";
   s+="P         = "+P+" bar\n";
   s+="=========================================\n";
   for(int i=0;i<n2;i++)
   {
   s+="Equilibrium Reaction   : "+chr[i].toStringE()+"\n";
   s+="Equilibrium constant K = "+chr[i].K(TR,TP)+"\n";
   s+="Equilibrium constant lnK = "+chr[i].lnK(TR,TP)+"\n";
```

```
    s+="=========================================\n";
    }

    //for(int j=0;j<nn;j++)
    //{s+="n_"+name[j]+" = ["+n0[j]+" ,  "+ni[j]+" ]\n";}
    jta=new JTextArea(s);

    String heading[]={"name","n0 mole in","x0 mole ratio in","n mole out","x mole ratio out"};
    String s1[][]=new String[nn+1][5];
    for(int i=0;i<nn;i++)
    {s1[i][0]=name[i];
     s1[i][1]=""+n0[i];
     s1[i][2]=""+x0[i];
     s1[i][3]=""+ni[i];
     s1[i][4]=""+x[i];
    }
    s1[nn][0]="total";
    s1[nn][1]=""+nn0t;
    s1[nn][2]=""+1;
    s1[nn][3]=""+nnt;
    s1[nn][4]=""+1;
            JTable jt;
            genelModel gm=new genelModel(s1,heading);
            jt=new JTable(gm);
             jpan.add(jta,BorderLayout.NORTH);
             jpan.add(new JScrollPane(jt),BorderLayout.SOUTH);
            String bb="Equilibrium Chemical Reaction";
            JFrame cerceve=new JFrame(bb);
    cerceve.addWindowListener(new BasicWindowMonitor());
    cerceve.getContentPane().add(new JScrollPane(jpan));
    cerceve.pack();
    cerceve.setVisible(true);
            }
}
```

Below listed continuity method (it is also called homotophy method) to solve non-linear system of equation. This method is relatively less dependent to initial estimation of the system of equation solution, therefore a good selection for solving the system of non-linear equation. The method details is as follows:When a problem of system of nonlinear equations of the form F(x)=0 desired to be solved, 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))]

Where λ=0 coresponds to initial guess of the solution , x(0) ,and where λ=1 value corresponds the actual solution set x(1)= x*

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

0=G(1,x) = F(x)

Therefore at x(1)=x* solution set will be obtained. If a function G(λ,x) satisfies the above equation can be found, it will also find us the solution. Function G is called a homotopy between the function G(0,x) and G(1,x)=F(x). 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

$$0 = \frac{\partial G(\lambda, x)}{\partial \lambda} + \frac{\partial G(\lambda, x)}{\partial x} x'(\lambda)$$

if x'(λ) is isolated form this equation, it becomes:

$$x'(\lambda) = -\left[\frac{\partial G(\lambda, x(\lambda)}{\partial x}\right]^{-1}\left[\frac{\partial G(\lambda, x(\lambda))}{\partial \lambda}\right]x'(\lambda)$$

If G(λ,x) = λF(x) + (1-λ)[F(x) - F(x(0))] equation is substituted into the differential equation

$$\left[\frac{\partial G(\lambda, x(\lambda)}{\partial x}\right] = \begin{bmatrix} \frac{\partial f_1(x(\lambda))}{\partial x_1} & \frac{\partial f_1(x(\lambda))}{\partial x_2} & \frac{\partial f_1(x(\lambda))}{\partial x_3} \\ \frac{\partial f_2(x(\lambda))}{\partial x_1} & \frac{\partial f_2(x(\lambda))}{\partial x_2} & \frac{\partial f_2(x(\lambda))}{\partial x_3} \\ \frac{\partial f_3(x(\lambda))}{\partial x_1} & \frac{\partial f_3(x(\lambda))}{\partial x_2} & \frac{\partial f_3(x(\lambda))}{\partial x_3} \end{bmatrix} = J(x(\lambda)$$

Forms a Jacobian matrix. and

$$\left[\frac{\partial G(\lambda, x(\lambda)}{\partial x}\right] = F(x(0))$$

Differential equation becomes

$$x'(\lambda) = \frac{dx(\lambda)}{d\lambda} = -[J(x(\lambda))]^{-1}F(x(0))0 \leq \lambda \leq 1$$

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 sixth order Runge-Kutta differential equation solution will be defined here to solve our homotopy problem. If equation $\frac{dx(\lambda)}{d\lambda} = f(\lambda, x(\lambda))$ is given the 6th order Runge-Kutta method to numerically solve this differential equation is defined as:

$y_{i+1} = y_i + (1/90)*(7k_1 + 32k_3 +12k_4+32k_5+7k_6)h$

$k_1=f(x_i,y_i)$

$k_2=f(x_i+0.25h , y_i+0.25k_1h)$

$k_3=f(x_i+0.25h , y_i+0.125k_1h+0.125k_2h)$

$k_4=f(x_i+0.5h , y_i – 0.5k_2h+k_3h)$

$k_5=f(x_i+0.75h , y_i + (3/16)k_1h+(9/16)k_4h)$

$k_6=f(x_i+h , y_i - (3/7)k_1h+(2/7)k_2h+(12/7)k_3h - (12/7)k_4h+(8/7)k_5h)$

This equation can be given as Buthcher tableu as:

$$\begin{array}{c|ccccc} 0 & 0 & 0 & 0 & 0 & 0 \\ 1/4 & 1/4 & 0 & 0 & 0 & 0 \\ 1/4 & 1/8 & 1/8 & 0 & 0 & 0 \\ 2/4 & 1/4 & -1/4 & 1 & 0 & 0 \\ 3/4 & 3/16 & 0 & 0 & 9/16 & 0 \\ 1 & -3/7 & 2/7 & 12/7 & -12/7 & 8/7 \\ \hline & 1/90 & 7/90 & 32/90 & 12/90 & 7/90 \end{array}$$

In these equations h is finite difference step size. Solution starts by using the initial value λ=0 , x0(0) and adds h into λ in each iteration step. The code given here uses 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 approach solution with a relatively rough estimate with homotophy following with a Newton-Raphson type of method, which is quite efficient when the estimation approaches the correct roots. In the following code methods of Homotophy with 6th order Runge-Kutta methods and combination of Homotophy with 4th order Runge-Kutta and Newton-Raphson method is given.

Program 6.2.3 continuity method to solve non-linear system of equations

```
 import java.util.*;
import java.awt.*;
import java.applet.Applet;
```

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

public class continuity
{
// (Continuity-homotopy) method to solve a system of nonlinear eqations
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[] continuationRK6(fi_xj 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_continuationRK6(fi_xj 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,20);
double ti=1.0;
int i;
```

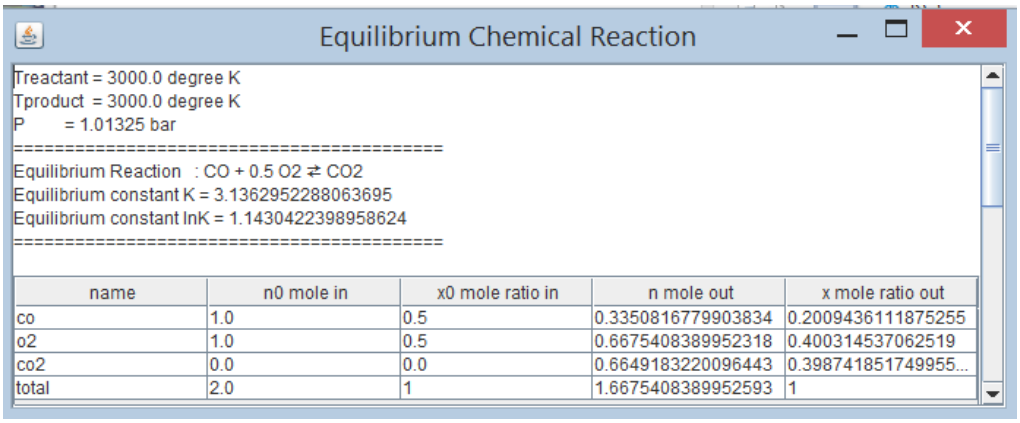
```
int nmax=600;
double tolerance=1.0e-10;
int n=x.length;
double b[];
b=new double[n];
for(i=0;i<n;i++)
{
b[i]=1.0;
}
i=0;
while( i++ < nmax && Matrix.abs(b) > tolerance )
{ b=Matrix.multiply(Matrix.divide(f.func(x),f.dfunc(x)),-ti);
  x=Matrix.add(x,b);
}
if(i >= nmax) JOptionPane.showMessageDialog(null,"Warning : 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[] newton(fi_xj 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=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;
 }
 }
```

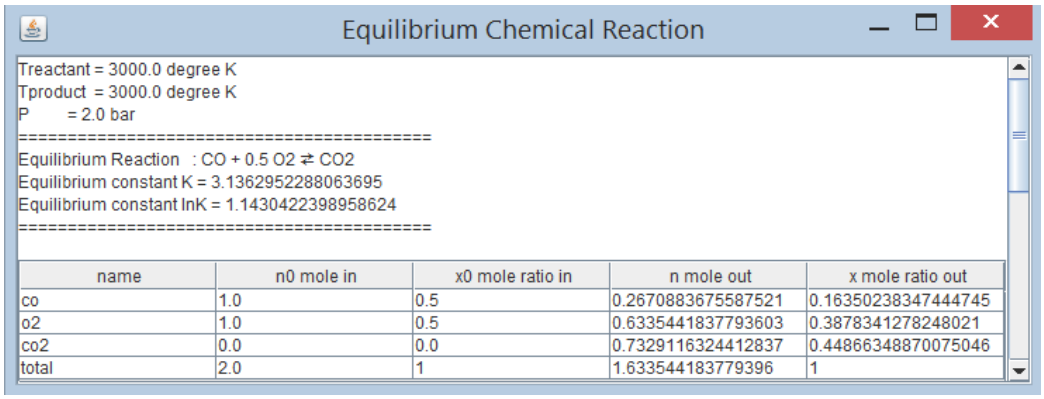
Program 6.2.4 example equilibrium problem

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

public class equilibrium2
{
  public static void main(String arg[])
  {   String s[]={"co","o2","co2"};
      double N[][]={{1.0,0.0},{0.5,0.0},{0.0,1.0}};
      chemicalReaction r[]=new chemicalReaction[1];
      r[0]=new chemicalReaction("r1",s,N);
      double Tproduct=3000.0;//degree K
      double Treactant=3000.0;//degree K
      double n0[]={1,1,0};
      double P=1.01325;//bar
              f_equilibrium fe=new f_equilibrium(n0,s,r,Treactant,Tproduct,P);
              double n[]={0.3,0.3,0.3};
              double [] r1= continuity.continuationRK6(fe,n,10);
      fe.equilibrium_print(r1);
  }
}
```

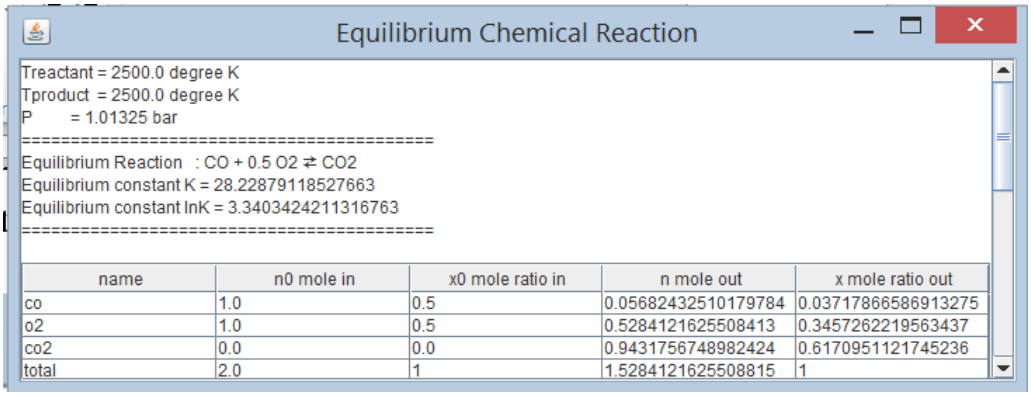


The same result is established in here as well. Now if conditions change, a new equilibrium condition will result. If the pressure applied in this problem becomes P=2 bar, the new equilibrium condition will be:



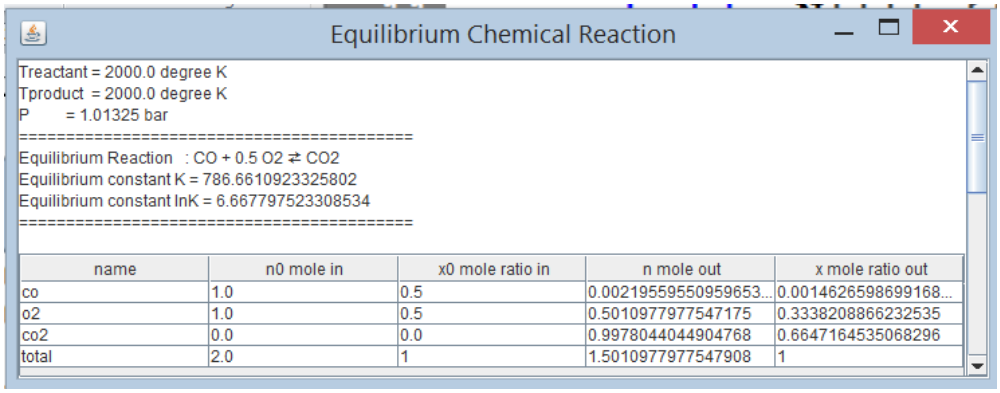
name	n0 mole in	x0 mole ratio in	n mole out	x mole ratio out
co	1.0	0.5	0.2670883675587521	0.16350238347444745
o2	1.0	0.5	0.6335441837793603	0.3878341278248021
co2	0.0	0.0	0.7329116324412837	0.44866348870075046
total	2.0	1	1.633544183779396	1

If the temperature drops to 2500 K for a pressure P=1.01325 bar:



name	n0 mole in	x0 mole ratio in	n mole out	x mole ratio out
co	1.0	0.5	0.05682432510179784	0.03717866586913275
o2	1.0	0.5	0.5284121625508413	0.3457262219563437
co2	0.0	0.0	0.9431756748982424	0.6170951121745236
total	2.0	1	1.5284121625508815	1

As it is seen from the results, CO vanishes when the temperature drops. If the temperature drops to 2000 K for a pressure P=1.01325 bar:



name	n0 mole in	x0 mole ratio in	n mole out	x mole ratio out
co	1.0	0.5	0.00219559550959653...	0.0014626598699168...
o2	1.0	0.5	0.5010977977547175	0.3338208866232535
co2	0.0	0.0	0.9978044044904768	0.6647164535068296
total	2.0	1	1.5010977977547908	1

For T=3000 K and P=1.01325 bar, if N2 is added to the reaction. Equilibrium reaction is still the same:

$CO + \frac{1}{2}O_2 \rightleftharpoons CO_2$

The reaction will be

$CO+O_2+1.88N_2 \rightarrow n_0CO+n_1O_2+n_2CO_2+1.88N_2$

Find the equilibrium composition.

Atomic balance for C: $1=n_0+n_2$

Atomic balance for O: $3=n_0+2n_1+2n_2$

Equilibrium constant at T=3000 K :

K=3.1362952

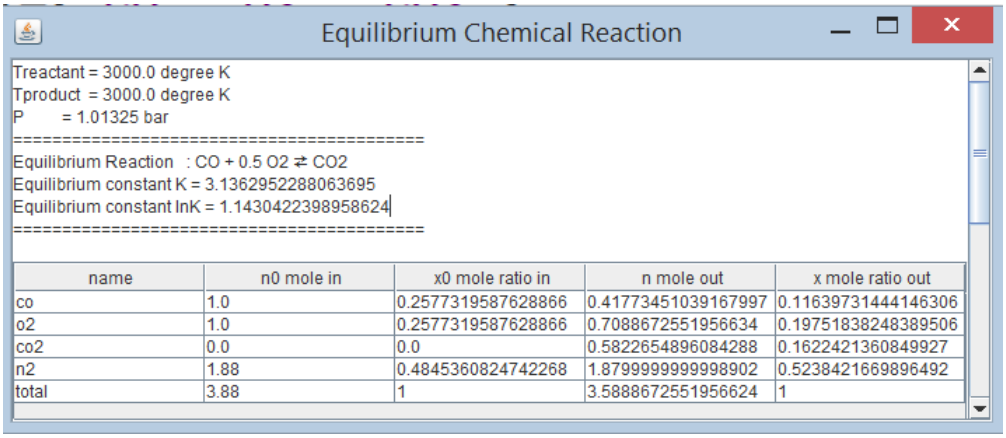
$n_2 = 1 - n_0 \;\; n_1 = \frac{3-n_0-2n_2}{2} = \frac{1+n_0}{2} \; n_{total} = n_0 + \frac{1+n_0}{2} + 1 - n_0 + 1.88 = \frac{6.76+n_0}{2}$

$ln(3.1362952) = \left[ln\left(\frac{2n_0}{6.76+n_0}\right) + 0.5\, ln\left(\frac{1+n_0}{6.76+n_0}\right) - ln\left(\frac{2(1-n_0}{6.76+n_0}\right)\right]$

If this equation is solved by excel solver

n0=nCO	0.417734523

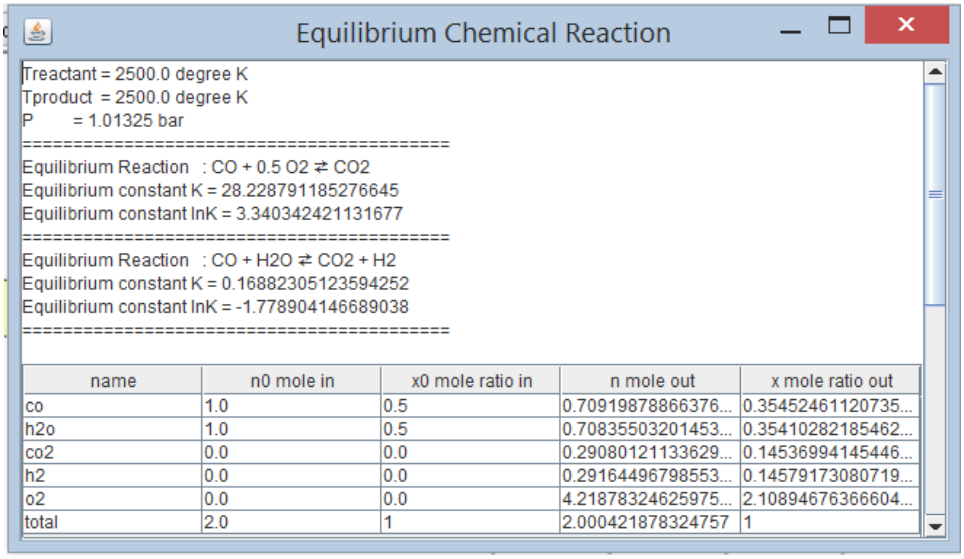
n1=nO2	0.708867261
n2=nCO	0.582265477
n3=nN2	1.88
ntotal	3.588867261
lnK	1.143042231
	-1.143042181
equation	5.00389E-08



As it is seen from the example, inert gas that does not goes into the reaction, still changes the equilibrium composition.Now let us look at more than one simultaneous reactions going on together:One mole of CO and one mole of H_2O are heated to 2500 K and 1.01325 bar. Determine the equilibrium composition if it is assumed that only CO, CO_2, H_2, H_2O and O_2 are presented.

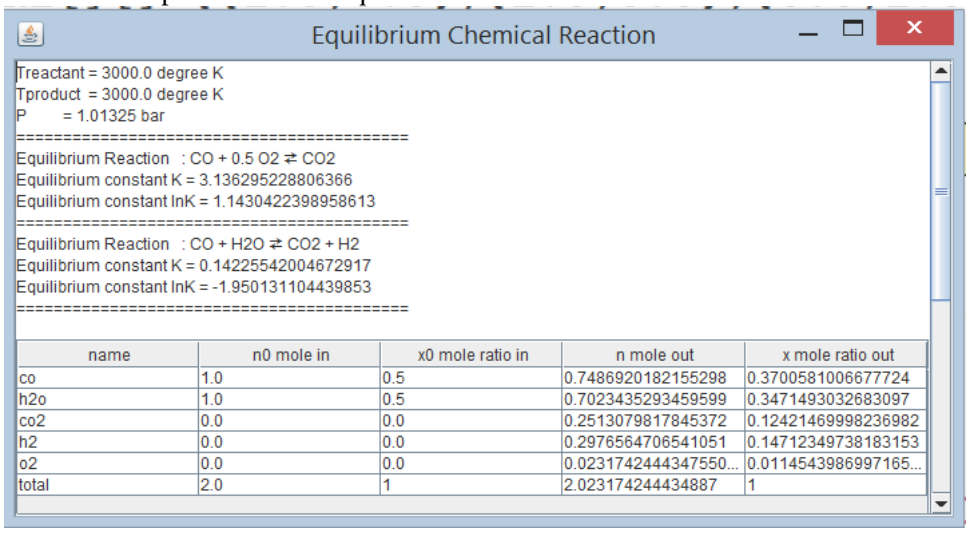
Program 6.2.5 example equilibrium problem

```
 import java.util.*;
import java.awt.*;
import java.applet.Applet;
import java.awt.event.*;
import javax.swing.*;
public class equilibrium1
{public static void main(String arg[])
 {   chemicalReaction r[]=new chemicalReaction[2];
     String s[]={"co","h2o","co2","h2","o2"};
     double N1[][]={{1.0,0.0},{0,0},{0,1},{0.0,0.0},{0.5,0.0}};
     r[0]=new chemicalReaction("r0",s,N1);
     double N2[][]={{1.0,0.0},{1.0,0.0},{0.0,1.0},{0.0,1.0},{0,0}};
     r[1]=new chemicalReaction("r1",s,N2);
     double Tproduct=2500.0;//degree K
     double Treactant=2500.0;//degree K
     double n0[]={1.0,1.0,0.0,0.0,0.0};
     double P=1.01325;//bar
              f_equilibrium fe=new f_equilibrium(n0,s,r,Treactant,Tproduct,P);
              double n[]={0.5,0.5,0.5,0.5,0.5};
              //double r1[]=fe.func(n);
              //double [] r1= continuity.continuationRK6(fe,n,4);
              double [] r1= continuity.newton_continuationRK6(fe,n);
              String ss[]={"nCO","nH2O","nCO2","nH2","nO2"};
              String ss1="Equilibrium reaction CO+"+'\u00BD'+"O2"+'\u21C4'+" CO2 \n";
              ss1+="CO+"+"at T="+Tproduct+" K ";
              fe.equilibrium_print(r1);
 }}
```



name	n0 mole in	x0 mole ratio in	n mole out	x mole ratio out
co	1.0	0.5	0.70919878866376...	0.35452461120735...
h2o	1.0	0.5	0.70835503201453...	0.35410282185462...
co2	0.0	0.0	0.29080121133629...	0.14536994145446...
h2	0.0	0.0	0.29164496798553...	0.14579173080719...
o2	0.0	0.0	4.21878324625975...	2.10894676366604...
total	2.0	1	2.000421878324757	1

As a result of heating, a system consisting initially of 1 kmol of CO_2, kmol of O_2, and kmol of N_2 forms an equilibrium mixture of CO_2, CO, O_2, N_2, and NO at 3000 K, 1.01325 bar. Determine the composition of the equilibrium mixture.



name	n0 mole in	x0 mole ratio in	n mole out	x mole ratio out
co	1.0	0.5	0.7486920182155298	0.3700581006677724
h2o	1.0	0.5	0.7023435293459599	0.3471493032683097
co2	0.0	0.0	0.2513079817845372	0.12421469998236982
h2	0.0	0.0	0.2976564706541051	0.14712349738183153
o2	0.0	0.0	0.0231742444347550...	0.0114543986997165...
total	2.0	1	2.023174244434887	1

If high enough temperatures are attained, nitrogen can combine with oxygen to form components such as nitric oxide. Even trace amounts of oxides of nitrogen in products of combustion can be a source of air pollution.

6.3 Chemical equilibrium with Gibbs free energy minimisation

Defining equilibrium equations each time is a time consuming process. Instead, gibss free energy for the full reaction can be minimised to obtain an equilibrium state. Total Gibbs free energy of the reaction can be defined as:

$$g = \sum_{j=1}^{NS} \mu_j n_j \quad (6.3.1)$$

In this equation g is gibbs free energy and NS is the number of species in the reaction, μ is chemical potential, and n_j is mole number for jth species. Chemical potential per kilomol j is defined to be

$$\mu_j = \left(\frac{\partial g}{\partial n_j}\right)_{T,P,n_{i\neq j}} \quad (6.3.2)$$

The condition for chemical equilibrium is the minimization of free energy. The minimisation is usually subject to certain constraints, such as the following mass-balance constraints:

$\sum_{j=1}^{NS} A_{ij}\, n_j - b_i^0 = 0$ (i=1..na) or $b_i = \sum_{j=1}^{NS} A_{ij}\, n_j$ so: $b_i - b_i^0 = 0$ (i=1..na) (6.3.3) where na is the number of chemical elements. A_{ij} is number of kilogram atoms per kmole of species j. And b_i^0 is the assigned number of kilogram atoms element i per kmol of total reactants. Assuming ideal gas:

$$\mu_j = \mu_j^0 + RT\, ln\left(\frac{n_j}{n_t}\frac{P}{P_{ref}}\right) \quad (6.3.4)$$

$$\mu_j = g_j^0(T) + RT\, ln\left(\frac{n_j}{n_t}\right) + RT\, ln\left(\frac{P}{P_{ref}}\right) \quad (6.3.5)$$

So gibbs free energy equation that we will minimise will be:

$$G = \sum_{j=1}^{NG} \mu_j n_j + \sum_{i=1}^{na} \lambda_i (b_i - b_i^0) \quad (6.3.6)$$

Where $\lambda_i$are lagrangian multipliers. The minimum value of the equation will give solution**,** but this equation is quite non-linear and relatively difficult to solve. We would like to introduce NASA method to solve the equation in here. The condition for equilibrium becomes:

$$\delta G = \sum_{j=1}^{NG}\left(\mu_j + \sum_{i=1}^{na} \lambda_i A_{ij}\right)\delta n_j + \sum_{i=1}^{na}(b_i - b_i^0)\,\delta\lambda_i = 0 \quad (6.3.7)$$

Treating the variations δn_j and $\delta\lambda_i$ as independent variables gives

$$\mu_j + \sum_{i=1}^{na} \lambda_i A_{ij} = 0 \quad (6.3.8) \quad \text{and}$$

$$b_i - b_i^0 = 0 \quad (6.3.9)$$

By opening these equations to Taylor series and simplification the following set of equations are obtained:

$$\sum_{i=1}^{na}\sum_{j=1}^{NG} A_{kj}A_{ij}n_j\pi_i + \left(\sum_{j=1}^{NG} A_{kj}n_j\right)\Delta\, ln\, n + \left(\sum_{j=1}^{NG}\frac{A_{kj}n_j H_j^0}{RT}\right)\Delta\, ln\, T = b_k^0 - b_k + \sum_{j=1}^{NG}\frac{A_{kj}n_j\mu_j}{RT} \quad \text{k=1..na} \quad (6.3.10)$$

$$\sum_{i=1}^{na}\sum_{j=1}^{NG} A_{kij}n_j\pi_i + \left(\sum_{j=1}^{NG} n_j - n\right)\Delta\, ln\, n + \left(\sum_{j=1}^{NG}\frac{n_j H_j^0}{RT}\right)\Delta\, ln\, T = n - \sum_{j=1}^{NG} n_j + \sum_{j=1}^{NG}\frac{n_j\mu_j}{RT} \quad (6.3.11)$$

Where $\pi_i = -\frac{\lambda_i}{RT}$, H_j^0 (6.3.12) is the satandart molar enthalpy for species j at temperature T.

A program called gibbs.java is developed to solve this set of eqautions and the code list is given at Appendix 8.3. We can check some of the previously investigated problems by using Gibbs free energy minimisation:

One kmol of CO an done kmol of O_2 established an equilibrium at 3000 K.

Program 6.3.1 example gibbs free energy inimisation by using gibbs class

```
public class gibbstest3 {

        public static void main(String[] arg){
                String iLK[]={ "co","o2","co2"};
         double[] n0={1,1,0};
                double Te[]={3000,3000,3000};
                double T1=3000;
                double P1=1.01325;
                int nmax1=500;//iterasyon sayısı
                System.out.println("---------------------------------");
                gibbs g1=new gibbs(iLK,n0,Te,T1,P1,nmax1);
                g1.getEq();
                for(int i=0;i<g1.n.length;i++) System.out.println(g1.gs[i].Formula+"   "+g1.n[i][0]);
                System.out.println("Q= "+g1.delQ());
                System.out.println("A= \n"+Matrix.toString(g1.A));
                System.out.println("b= \n"+Matrix.toString(g1.b0));
                System.out.println("---------------------------------");

        }

}
```

```
---------- Capture Output ----------
> "C:\java\bin\javaw.exe" gibbstest3
```

```
---------------------------------
CO   0.3336866135142944
O2   0.6668433067571505
CO2   0.6663133864857074
Q = -181692.07676724726
A =  1.000000000000000       0.000000000000000       1.000000000000000
       1.000000000000000       2.000000000000000       2.000000000000000
b=   1.000000000000000
       3.000000000000000
---------------------------------
> Terminated with exit code 0.
```

As addition to previous programs, gibbs can be able to calculate adiabatic flame temperatures of an equilibrium state as well.

Program 6.3.2 example gibbs free energy inimisation by using gibbs class

```
public class gibbstest2 {
   public static void main(String[] arg)
   {
       String s0[]={ "co","h2o","co2","h2","o2" };
             double[] n0={1,1,0,0,0};
                       double Te[]={2500,2500,2500,2500,2500};
                       double T1=2500;//degree K
                       double P1=1.01325;//bar
                       int nmax1=500;//iterasyon sayısı
                       System.out.println("---------------------------------");
                       gibbs g1=new gibbs(s0,n0,Te,T1,P1);
                       g1.getEq();
                       for(int i=0;i<g1.n.length;i++) System.out.println(g1.gs[i].Formula+"   "+g1.n[i][0]);
       System.out.println("---------------------------------");
                       System.out.println("Q= "+g1.delQ());
                       T1=g1.set_Q(s0,n0,Te,0.0,P1);
                       System.out.println("T1="+T1);
                       g1.getEq();
                       for(int i=0;i<g1.n.length;i++) System.out.println(g1.gs[i].Formula+"   "+g1.n[i][0]);
                       System.out.println("---------------------------------");
             }
}
```

```
---------- Capture Output ----------
> "C:\java\bin\javaw.exe" gibbstest2
---------------------------------
CO   0.7091934909063149
H2O   0.7083603316367073
CO2   0.2908065090936887
H2   0.29163966836329364
O2   4.1657963479952635E-4
---------------------------------
Q= -6442.059798113942

//adiabatic flame temperature
T1=2566.515550936884
CO   0.7124848532408645
H2O   0.7108823034004179
CO2   0.28751514675913026
H2   0.2891176965995817
O2   8.012749202241687E-4
---------------------------------
```

```
Q= -8.02592279427472E-5

> Terminated with exit code 0.
```

A graphic user interface (GUI) is prepared for gibbs class utilisation

Program 6.3.3 example gibbs free energy inimisation Graphic user interface gibbsTableEN using gibbs class

```
//=======================================================
// Termodinamics-Numerical Analysis package
// Gibbs free energy minimisation
// Dr. Turhan Coban
// Ege University, School of Engineering, Department of Mechanical Eng.
// web : www.turhancoban.com
// =======================================================

import java.lang.Integer;
import java.awt.*;
import java.awt.event.*;
import java.awt.font.*;
import java.awt.geom.*;
import java.awt.image.*;
import javax.swing.*;
import java.util.Locale;
import java.text.*;
import java.io.*;
import java.awt.*;
import java.awt.event.*;
import java.util.*;
import javax.swing.table.*;

public class gibbsTableEN extends JApplet implements  ActionListener,ItemListener
{    private static final long serialVersionUID = 37259887589L;
    String state[]={" product temperature is given         "," heat transfer is given          "};
    String g1[]={"co","o2","co2"};
    double gn1[]={1.0,1.0,0.0};
    double gT[]={3000.0,3000.0,3000.0};
    double Te=3000.0;
    double Pe=1.01325;
    double Qe=0;
    JPanel panel[];
    JTextArea output;
    gibbs gib;
    JButton b1;
    girdiP3EN gp3;
    boolean caseT=true;
    JCheckBox hhv;
    JTextField hhv_lhv;
  public String setArea()
  {
   String s="";
   s+="    Dr. Turhan Çoban, \n";
   s+="    Ege University, School of Engineering, Department of Mechanical Eng.\n";
   s+="    web : www.turhancoban.com\n";
   s+="    =======================================================\n";
   for(int i=0;i<gib.n.length;i++) s+=gib.gs[i].Formula+"   "+gib.n[i][0]+"\n";
   Qe=gib.delQ();
```

```
s+="Q   = "+Qe+" KJ/kmol\n";
s+="Te  = "+Te+" degree K\n";
s+="Pe  = "+Pe+" bar\n";
s+="---------------------------------";
return s;
}

public void init()
{
  //ilk değerler
  gp3=new girdiP3EN(g1,gn1,gT,Te,Pe,"Gibbs free energy minimisation");
          gib=new gibbs(g1,gn1,gT,Te,Pe);
  gib.getEq();
  Qe=gib.delQ();
  hhv=new JCheckBox("exit temperature/reaction energy ? ",true);
  hhv.addItemListener(this);
  hhv_lhv=new JTextField(state[0]);
  hhv_lhv.setFont(new Font("TimesRoman",Font.BOLD,14));
  panel=new JPanel[2];
  panel[1]= new JPanel(new FlowLayout());
  panel[1].add(hhv);
  panel[1].add(hhv_lhv);
          JTabbedPane jtp=new JTabbedPane();
          b1=new JButton("Gibbs free energy minimisation");
          b1.addActionListener(this);
  panel[0]= new JPanel(new BorderLayout());

  output=new JTextArea(setArea());
  output.setFont(new Font("TimesRoman",Font.BOLD,14));
  panel[0].add(gp3,BorderLayout.NORTH);
  panel[0].add(panel[1],BorderLayout.CENTER);
  panel[0].add(b1,BorderLayout.SOUTH);
  //b1.addActionListener(this);
  JScrollPane skrolPane=new JScrollPane(panel[0]);
  jtp.addTab("Input Panel",skrolPane);
  jtp.addTab("Output Panel",new JScrollPane(output));
  add(jtp);
}

public void actionPerformed(ActionEvent e)
{
if (e.getSource()==b1)
{
  g1=gp3.gas;
  gn1=gp3.ngas;
  gT=gp3.Tgas;
  Pe=gp3.Pegas;
  if(caseT)
  { Te=gp3.Tegas;
    gib.setgibbs(g1,gn1,gT,Te,Pe);
    gib.getEq();
    Qe=gib.delQ();

  }
  else
  { Qe=Double.parseDouble(gp3.t2[gp3.n].getText());
    Te=gib.set_Q(g1,gn1,gT,Qe,Pe);
    gib.getEq();
```

```
      }
    }
    else if (e.getSource()==gp3.t2[gp3.n])
    {
     g1=gp3.gas;
     gn1=gp3.ngas;
     gT=gp3.Tgas;
     Pe=gp3.Pegas;
     if(caseT)
     {Te=Double.parseDouble(gp3.t2[gp3.n].getText());
      gib.setgibbs(g1,gn1,gT,Te,Pe);
      gib.getEq();
      Qe=gib.delQ();
     }
     else
     { Qe=Double.parseDouble(gp3.t2[gp3.n].getText());
       Te=gib.set_Q(g1,gn1,gT,Qe,Pe);
       gib.getEq();
     }
    }
    output.setText(setArea());
    repaint();
  }

public void itemStateChanged(ItemEvent e)
{   if(e.getSource()==hhv)
    { if(e.getStateChange()==ItemEvent.SELECTED)
       {caseT=true;
       hhv_lhv.setText(state[0]);
       gp3.jl[3].setText("Exit temperature of the gas : ");
       gp3.jl[4].setText("degree K ");
       g1=gp3.gas;
       gn1=gp3.ngas;
       gT=gp3.Tgas;
       Te=gp3.Tegas;
       Pe=gp3.Pegas;
       gib.setgibbs(g1,gn1,gT,Te,Pe);
       gib.getEq();
       output.setText(setArea());
       }
      else
       {caseT=false;
       hhv_lhv.setText(state[1]);
       gp3.jl[3].setText("energy output of the reaction : ");
       gp3.jl[4].setText("KJ/kmol ");
       g1=gp3.gas;
       gn1=gp3.ngas;
       gT=gp3.Tgas;
       Qe=gp3.Tegas;
       Pe=gp3.Pegas;
       Te=gib.set_Q(g1,gn1,gT,Qe,Pe);
       gib.getEq();
       output.setText(setArea());
       }
    }
    repaint();
}
    public static void main(String s[])
```

```
    {
      //main program
            JFrame f = new JFrame("Gibbs free energy minimisation");
       f.setDefaultCloseOperation( JFrame.EXIT_ON_CLOSE );
            JApplet applet = new gibbsTableEN();
       f.getContentPane().add("Center", applet);
       applet.init();
       f.pack();
       f.setSize(new Dimension(1000,600));
       f.setVisible(true);
    }
  }
```

Program 6.3.4 example gibbs free energy minization Graphic user interface gibbsTableEN using gibbs class, input-output screen program girdiP3EN

```
import java.lang.Integer;
import java.awt.*;
import java.awt.event.*;
import java.awt.font.*;
import java.awt.geom.*;
import java.awt.image.*;
import javax.swing.*;
import java.util.Locale;
import java.text.*;
import java.io.*;
import java.awt.*;
import java.awt.event.*;
import java.util.*;
import javax.swing.table.*;

public class girdiP3EN extends JPanel implements ActionListener,ItemListener,Serializable
{ private static final long serialVersionUID = 80948739856L;
 String gas[];
 double ngas[];
 double Tgas[];
 double Tegas;
 double Pegas;
 int n1;
 JPanel p1,p2,p3,p4,p;
 JLabel l1[];        // Label prompt unit
 JLabel l2[];        // Label prompt unit
 JTextField t1[];
 JComboBox<String> c2[];
 JTextField t2[];
 JTextField t3[];
 JLabel jl[];

 double a[];

 JLabel heading;
 //JButton jb;
 int n;

 public girdiP3EN(String g1[],double gn1[],double gT1[],double Tegasi,double Pegasi,String hh)
 {
        Gas1 g=new Gas1("ch4");
        gas=new String[g1.length];
        ngas=new double[gn1.length];
```

```
            Tgas=new double[gn1.length];
            Tegas=Tegasi;
            Pegas=Pegasi;
    n1=g1.length;
    String baslik[]={"Gas name","reactant moles","reactant temperature degree K"};
            jl=new JLabel[7];
            jl[0]=new JLabel(baslik[0]);
            jl[1]=new JLabel(baslik[1]);
            jl[2]=new JLabel(baslik[2]);
            jl[3]=new JLabel("exit temperature of the gas : ");
            jl[4]=new JLabel("degree K ");
            jl[5]=new JLabel("exitpressure of the gas : ");
            jl[6]=new JLabel("bar ");

    StringTokenizer token=new StringTokenizer(g.readGasNames());
    String st[]=new String[token.countTokens()];
    int i=0;
    while(token.hasMoreTokens())
    {
    st[i++]=new String(token.nextToken());
    }
    n=g1.length;
    l1=new JLabel[n];
    l2=new JLabel[n];
    t1=new JTextField[n+2];
    a=new double[n];
    heading=new JLabel(hh);
    p1=new JPanel();
    p2=new JPanel();
    p3=new JPanel();
    p4=new JPanel();
    p4.setLayout(new GridLayout(n+3,3,5,5));
    c2=new JComboBox[n];
    t2=new JTextField[n+2];
    t3=new JTextField[n];
    p4.add(jl[0]);
    p4.add(jl[1]);
    p4.add(jl[2]);
    for(int iz=0;iz<n1;iz++)
    { gas[iz]=g1[iz];
      ngas[iz]=gn1[iz];
      Tgas[iz]=gT1[iz];
                c2[iz]=new JComboBox<String>(st);
                t2[iz]=new JTextField(ngas[iz]+"              ");
      t3[iz]=new JTextField(Tgas[iz]+"              ");
                t2[iz].setFont(new Font("TimesRoman",Font.BOLD,14));
                t3[iz].setFont(new Font("TimesRoman",Font.BOLD,14));
      c2[iz].setSelectedItem(gas[iz]);
      t2[iz].addActionListener(this);
                p4.add(c2[iz]);
                p4.add(t2[iz]);
                p4.add(t3[iz]);
      c2[iz].addItemListener(this);
      t2[iz].addActionListener(this);
      t3[iz].addActionListener(this);
    }
    t2[n]=new JTextField(Tegas+"              ");
    t2[n+1]=new JTextField(Pegas+"              ");
```

```
  p4.add(jl[3]);
  p4.add(t2[n]);
  p4.add(jl[4]);

  p4.add(jl[5]);
  p4.add(t2[n+1]);
  p4.add(jl[6]);

  t2[n].addActionListener(this);
  t2[n+1].addActionListener(this);
     p=new JPanel()
     {
      public Dimension getPrefferedSize()
      {
         Dimension size=super.getPreferredSize();
         size.width=200;
         size.height=100;
         return size;
      }
      };
      p.add(p4,BorderLayout.NORTH);
            add(p);
}

public void actionPerformed( ActionEvent e)
{
int i=0;
for(i=0;i<n;i++)
{           if(e.getSource()==t2[i])
  { ngas[i]=Double.parseDouble(t2[i].getText());
    break;
  }
  else if(e.getSource()==t3[i])
  { Tgas[i]=Double.parseDouble(t3[i].getText());
    break;
  }
}
if(e.getSource()==t2[n])
  { Tegas=Double.parseDouble(t2[n].getText());
  }
else if(e.getSource()==t2[n+1])
  { Pegas=Double.parseDouble(t2[n+1].getText());}
repaint();
}

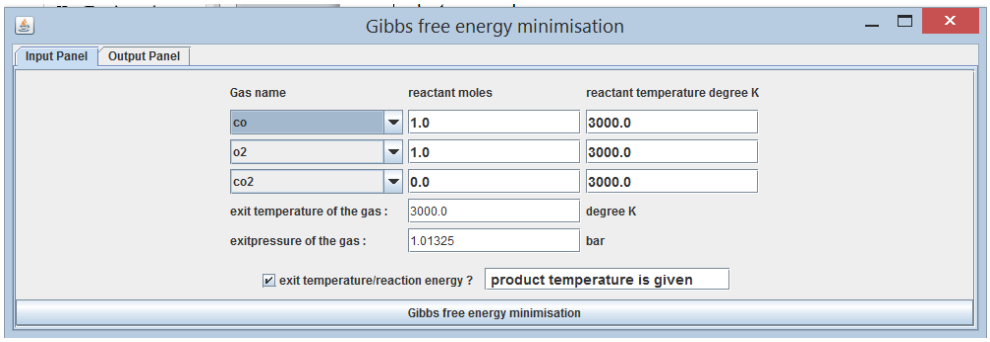
public void itemStateChanged(ItemEvent ev)
{
for(int i=0;i<n1;i++)
{           if(ev.getSource()==c2[i])
  { gas[i]=(String)c2[i].getSelectedItem();break;}
}
}

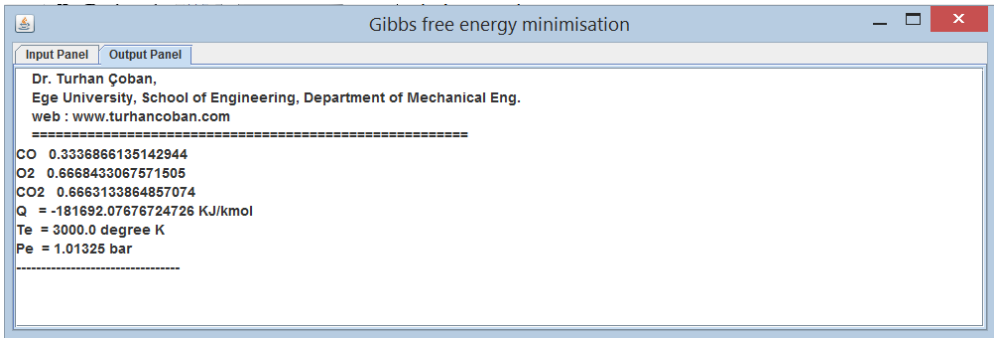
public double[] readNgas()
{return ngas;}

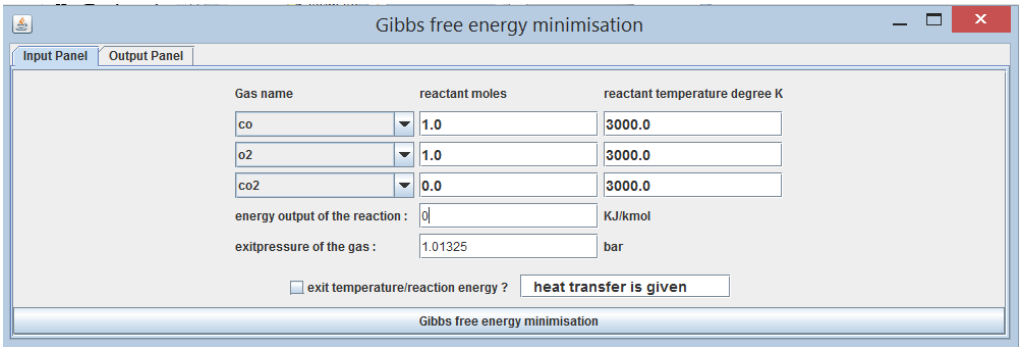
public double[] readTgas()
{return ngas;}
```

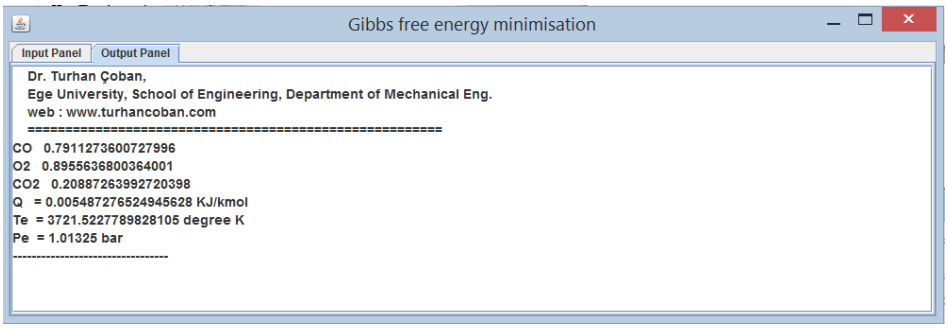
```
public String[] readgas()
{return gas;}

}
```









As it is seen from the output at T=3721 K, the reaction reached to adiabatic flame temperature (Total heat transfer is zero)

As a second method, system of equation in the form of

$\mu_j + \sum_{i=1}^{na} \lambda_i A_{ij} = 0$ (6.3.8) and

$b_i - b_i^0$=0 (6.3.9)

$\lambda_i$terms are very big compare to $\mu_j$terms. This makes solution of the nonlinear system difficult. In order to make the sets easier to solve a normalisation factor can be applied to the set as:

$$C = 10RT$$

$\pi_i = \frac{\lambda_i}{C}$ so eqautions become:

$\mu_j + \sum_{i=1}^{na} C\pi_i A_{ij} = 0$ (6.3.8a) and $b_i - b_i^0 = 0$ (6.3.9a)

can now be directly solved by using nonlinear system of equations methods. Equations are formed as a function in f_gibbs class and solved by using continuity(homotophy) methods combined with Newton-Raphson method. Metod detailes were described previously.

Program 6.3.5 example gibbs free energy minization , by directly solving non-linear system of equations, f_gibbs general input function

```
 import java.io.*;
import java.util.*;
import javax.swing.*;
import java.awt.*;
import java.awt.event.*;
class f_gibbs extends fi_xj
{       public double[][] A;
        public gibbsSpecies[] gs;
        public double[] n0;
        public String name[];
        double[] n;
        double[] del_lnnm;
        double[] del_nm;
        double[] lnnm;
        double nt;
        double lnnt;
        double del_lnnt;
        public double R;
        public int NG;//Gas species number
        public int l;//element number
        double[][] d_coeff;
        double[] d_const;
        double[] b0;
        double[] b;
        double[] mu;
        double[] mutemp;
        double P;
        double T;
        double T0;
        double[] Te;
        double etemp=0;
        double Tref=298.15;
        double G=0;
        double Gnew=0;
        double diff=0;
        double SIZE=25.328436;;
        double ntahminilk;
        int it_max=100;
        int it_max0=100;
        int o2=-1;
        int n2=-1;
```

```
        double x1[];
        double x2[];
        public double C;
                        public f_gibbs(String igs[],double[] in0,double iTe[],double iT,double iP)
                        {    R=8.3145;
                            C=50*R*iT;
                            Te=new double[iTe.length];
                            name=new String[iTe.length];
                            for(int i=0;i<iTe.length;i++)
                            {Te[i]=iTe[i];name[i]=igs[i];}
                            setgibbs(igs,in0,iT,iP);
                      }
                    public void setgibbs(String igs[],double[] in0,double iT,double iP){
                            gs=new gibbsSpecies[igs.length];
                            for(int i=0;i<igs.length;i++){
                                    gs[i]=new gibbsSpecies(igs[i],"gas");
                            }
                    int nn=igs.length;
                            int nat=110;
                            double B[][]=new double[nat][nn];
                            int nk=0;
                            for(int i=0;i<nn;i++){
                                    for(nk=0;nk<gs[i].atomList.length;nk++){
                                            B[gs[i].atomList[nk].number-
1][i]=gs[i].atomList[nk].N;
                                            }
                                    }
                            int natcount=0;
                            boolean zeroflag[]=new boolean[110];
                            for(int j=0;j<nat;j++)
                            {zeroflag[j]=false;for(int
i=0;i<nn;i++){if(B[j][i]!=0){zeroflag[j]=true;}};if(zeroflag[j]) natcount++;}
                            A=new double[natcount][nn];
                            int k=0;
                            for(int j=0;j<nat;j++)
                            {if(zeroflag[j]){A[k]=B[j];k++;}}
                            n0=in0;
                            P=iP;
                            T=iT;
                            T0=iT;
                            mu=new double[gs.length];
                            l=A.length;
                            NG=A[0].length;
                            b=new double[l];
                            nt=getnt(n0);
                            ntahminilk=nt/n0.length;
                            n=new double[n0.length];
                            for(int i=0;i<n0.length;i++){
                                    n[i]=ntahminilk;
                            }
                            nt=getnt(n);
                            b0=new double[l];
                            for(int ii=0;ii<b0.length;ii++)
                            {for(int ij=0;ij<NG;ij++)
                                    { double carpim=A[ii][ij]*n0[ij];
                                      b0[ii]+=carpim;
                                    }
                            }
```

```
                calMu(n0);
        }
        public double getnt(double[] jn)
        {
                double nt_inner=0;
                for(int i=0;i<jn.length;i++) {nt_inner+=jn[i];}
                return nt_inner;
        }
        public void setN0(double[] in0){
           for(int i=0;i<n0.length;i++) n0[i]=in0[i];
        }

        public void setT(double iTeq){
                T=iTeq;
                calMu(n0);
        }
        public void setTDefault(){
                T=T0;
        }

        public void calMu(double[] in){
                for(int i=0;i<mu.length;i++){
                        double inT=getnt(in);
                        mu[i]=gs[i].mu(T,P,in[i],inT);
                }
        }
        public double[] getb(double[] in){
                double inner_b[]=new double[A.length];
                for(int ii=0;ii<inner_b.length;ii++){
                        inner_b[ii]=0;
                        for(int ij=0;ij<NG;ij++){
                                inner_b[ii]+=A[ii][ij]*in[ij];
                        }
                }
                return inner_b;
        }
        public double A_lambda(double[] in,int j)
        {
                double inner_b=0;
                for(int i=0;i<in.length;i++)
                {inner_b+=A[i][j]*in[i];}
                return inner_b;
        }
        public double getG(double[] in)
        {
                calMu(in);
                double munt=0;
                for(int j=0;j<mu.length;j++) mu[j]*=in[j];
                for(int i=0;i<mu.length;i++) munt+=mu[i];
                return munt;
        }
public double[] func(double x[])
{
//çözumu istenen fonksiyon
int n1=x.length;
double ff[]=new double[n1];
double G;
double x1[]=new double[NG];
```

```
double x2[]=new double[l];
for(int i=0;i<NG;i++) {x1[i]=x[i];//System.out.println("x1["+i+"] = "+x1[i]);
}     //n_i
for(int i=NG;i<(NG+l);i++) {x2[i-NG]=x[i];} //lambda_k
calMu(x1);
double f1[]=mu;
double xx=0;

for(int i=0;i<NG;i++) {ff[i]=f1[i];}
for(int j=0;j<NG;j++) {xx=A_lambda(x2,j);ff[j]+=C*xx;}
b=getb(x1);
//System.out.println("b="+Matrix.toString(b)+"b0="+Matrix.toString(b0));
for(int i=NG;i<(NG+l);i++) {ff[i]=b[i-NG]-b0[i-NG];}
return ff;
}
    public  void equilibrium_print(double ni[])
    { int nn=ni.length;
            double x[]=new double[nn];
            double x0[]=new double[nn];
            double nnt=0;
            double nn0t=0;
            for(int i=0;i<nn;i++){nnt+=ni[i];nn0t+=n0[i];}
            for(int i=0;i<nn;i++){x[i]=ni[i]/nnt;x0[i]=n0[i]/nn0t;}
    JTextArea jta;
    JPanel jpan=new JPanel();
    jpan.setLayout(new BorderLayout());
    String s="";
    //s+="Treactant = "+TR+" degree K\n";
    //s+="Tproduct  = "+TP+ " degree K\n";
    s+="P         = "+P+" bar\n";
    s+="=========================================\n";
    jta=new JTextArea(s);
    String heading[]={"name","n0 mole in","x0 mole ratio in","n mole out","x mole ratio out"};
    String s1[][]=new String[nn+1][5];
    for(int i=0;i<nn;i++)
    {s1[i][0]=name[i];
     s1[i][1]=""+n0[i];
     s1[i][2]=""+x0[i];
     s1[i][3]=""+ni[i];
     s1[i][4]=""+x[i];
    }
    s1[nn][0]="total";
    s1[nn][1]=""+nn0t;
    s1[nn][2]=""+1;
    s1[nn][3]=""+nnt;
    s1[nn][4]=""+1;
            JTable jt;
            genelModel gm=new genelModel(s1,heading);
            jt=new JTable(gm);
             jpan.add(jta,BorderLayout.NORTH);
             jpan.add(new JScrollPane(jt),BorderLayout.SOUTH);
            String bb="Equilibrium Chemical Reaction";
            JFrame cerceve=new JFrame(bb);
    cerceve.addWindowListener(new BasicWindowMonitor());
    cerceve.getContentPane().add(new JScrollPane(jpan));
    cerceve.pack();
    cerceve.setVisible(true);
            }
```

```
}
```

Program 6.3.6 example gibbs free energy minization , by directly solving non-linear system of equations, gibbs1

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

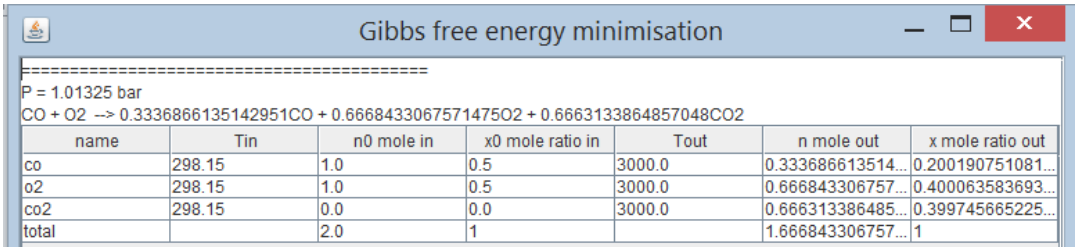
public class gibbs1
{           public static double[] gibbs(String igs[],double n0[],double Ti[],double To,double Pi)
   {f_gibbs f=new f_gibbs(igs,n0,Ti,To,Pi);
   double R=8.3145;
    int NG=f.NG;
    int l=f.l;
    int n1=NG+l;
    double n[]=new double[n1];

    for(int i=0;i<NG;i++)
    {n[i]=0.5;}
    for(int i=NG;i<n1;i++)
    {n[i]=10000.0/(50.0*R*To);}
    double [] r1= continuity.newton_continuationRK6(f,n);
    double r2[]=new double[NG];
    for(int i=0;i<NG;i++) {r2[i]=r1[i];}
    return r2;
   }
    public  static void equilibrium_print(String name[],double n0[],double ni[],double Ti[],double
To,double Pi)
    { int nn=ni.length;
            double x[]=new double[nn];
            double x0[]=new double[nn];
            double nnt=0;
            double nn0t=0;
            for(int i=0;i<nn;i++){nnt+=ni[i];nn0t+=n0[i];}
            for(int i=0;i<nn;i++){x[i]=ni[i]/nnt;x0[i]=n0[i]/nn0t;}
    JTextArea jta;
    JPanel jpan=new JPanel();
    jpan.setLayout(new BorderLayout());
    String s="";
    s+="=========================================\n";
    s+="P = "+Pi+" bar \n";
    s+=toString(name,n0,ni);
    jta=new JTextArea(s);
    String heading[]={"name","Tin","n0 mole in","x0 mole ratio in","Tout","n mole out","x mole ratio
out"};
    String s1[][]=new String[nn+1][7];
    for(int i=0;i<nn;i++)
    {s1[i][0]=name[i];
     s1[i][1]=""+Ti[i];
     s1[i][2]=""+n0[i];
     s1[i][3]=""+x0[i];
     s1[i][4]=""+To;
     s1[i][5]=""+ni[i];
     s1[i][6]=""+x[i];
    }
    s1[nn][0]="total";
```

```
    s1[nn][1]=" ";
    s1[nn][2]=""+nn0t;
    s1[nn][3]=""+1;
    s1[nn][4]=" ";
    s1[nn][5]=""+nnt;
    s1[nn][6]=""+1;
              JTable jt;
              genelModel gm=new genelModel(s1,heading);
              jt=new JTable(gm);
               jpan.add(jta,BorderLayout.NORTH);
               jpan.add(new JScrollPane(jt),BorderLayout.SOUTH);
              String bb="Gibbs free energy minimisation";
              JFrame cerceve=new JFrame(bb);
    cerceve.addWindowListener(new BasicWindowMonitor());
    cerceve.getContentPane().add(new JScrollPane(jpan));
    cerceve.pack();
    cerceve.setVisible(true);
              }
    public static String toString(String name[],double n0[],double n[])
    {
     // writes chemical reaction in chemistry norm
     int nn=n0.length;
     String s="";
     String gasname;
     for(int i=0;i<nn;i++)
     {
               Gas g=new Gas(name[i]);
     if(n0[i]!=0)
        { if(n0[i]!=1.0) s+=""+n0[i];
          s+=g.toString();
          if(i!=(nn-1)) s+=" + ";
          else
          { if(s.endsWith(" + ")) s=s.substring(0,s.length()-2);
           s+=" --> ";}
        }
     else
        { if(s.endsWith(" + ")) s=s.substring(0,s.length()-2);
                    if(i==(nn-1)) s+=" --> ";
                  }
     }
     for(int i=0;i<nn;i++)
     {
               Gas g=new Gas(name[i]);
     if(n[i]!=0)
        { if(n[i]!=1.0) s+=""+n[i];
          s+=g.toString();
          if(i!=(nn-1)) s+=" + ";
          else s+=" ";
        }
     }
    return s;
    }
public static void main(String arg[])
{      String iLK[]={ "co","o2","co2" };
              double[] n0={1,1,0};
                        double Te[]={298.15,298.15,298.15};
                        double T1=3000;
                        double P1=1.01325;
```

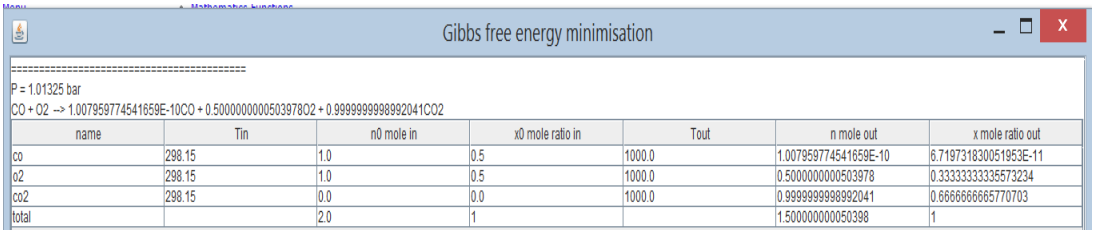
```
                    double [] r1= gibbs(iLK,n0,Te,T1,P1);
                    equilibrium_print(iLK,n0,r1,Te,T1,P1);
  }
}
```

Non linear continuity method listed before as Program 6.2.3 which is using a combination of continuity and Newton-Raphson non-linear system of equation solution methods.



name	Tin	n0 mole in	x0 mole ratio in	Tout	n mole out	x mole ratio out
co	298.15	1.0	0.5	3000.0	0.333686613514...	0.200190751081...
o2	298.15	1.0	0.5	3000.0	0.666843306757...	0.400063583693...
co2	298.15	0.0	0.0	3000.0	0.666313386485...	0.399745665225...
total		2.0	1		1.666843306757...	1

If the temperature drops CO will vanish and reaction wil be converted to a complete combustion

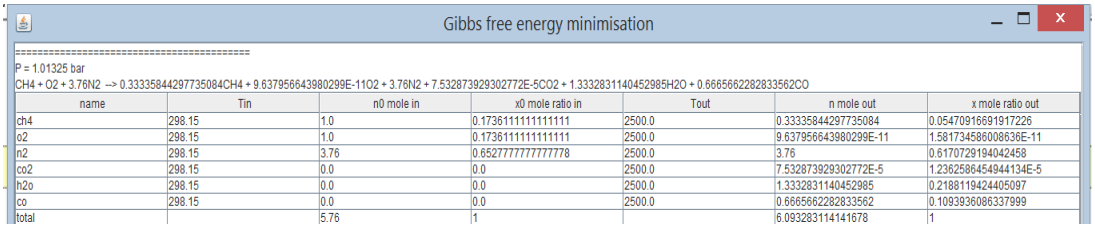


name	Tin	n0 mole in	x0 mole ratio in	Tout	n mole out	x mole ratio out
co	298.15	1.0	0.5	1000.0	1.007959774541659E-10	6.719731830051953E-11
o2	298.15	1.0	0.5	1000.0	0.5000000000503978	0.33333333335573234
co2	298.15	0.0	0.0	1000.0	0.9999999998992041	0.6666666665770703
total		2.0	1		1.500000000050398	1

If no additional gases is defined, equilibrium state will find the chemical balance

If additional gases is given in equilibrium, gas combination will drastically change

```
  public static void main(String arg[])
   {      String iLK[]={ "ch4","o2","n2","co2","h2o","co" };
             double[] n0={1,1,3.76,0,0,0};
                       double Te[]={298.15,298.15,298.15,298.15,298.15,298.15};
                       double T1=2500;
                       double P1=1.01325;
                       double [] r1= gibbs(iLK,n0,Te,T1,P1);
                       equilibrium_print(iLK,n0,r1,Te,T1,P1);
   }
```



name	Tin	n0 mole in	x0 mole ratio in	Tout	n mole out	x mole ratio out
ch4	298.15	1.0	0.17361111111111111	2500.0	0.33335844297735084	0.05470916691917226
o2	298.15	1.0	0.17361111111111111	2500.0	9.637956643980299E-11	1.581734586008636E-11
n2	298.15	3.76	0.6527777777777778	2500.0	3.76	0.6170729194042458
co2	298.15	0.0	0.0	2500.0	7.532873929302772E-5	1.2362586454944134E-5
h2o	298.15	0.0	0.0	2500.0	1.3332831140452985	0.2188119424405097
co	298.15	0.0	0.0	2500.0	0.6665662282833562	0.1093936086337999
total		5.76	1		6.093283114141678	1

Gibbs free energy minimisation can also be applied by directly minimizing gibbs function, but since equation is very unstable results may not be obtained by using any optimisation method.

Program 6.3.7 example gibbs free energy minization , by gibbs equation function f1_gibbs

```
class f1_gibbs extends f_xj
{           public double[][] A;
```

```
public gibbsSpecies[] gs;
public String name[];
public double[] n0;
public double R;
double[] n;
double[] del_lnnm;
double[] del_nm;
double[] lnnm;
double nt;
double lnnt;
double del_lnnt;
public int NG;//Gas species number
public int l;//element number
double[][] d_coeff;
double[] d_const;
double[] b0;
double[] b;
double[] mu;
double[] mutemp;
double P;
double T;
double T0;
double[] Te;
double etemp=0;
double Tref=298.15;
double G=0;
double Gnew=0;
double diff=0;
double SIZE=25.328436;;
double ntahminilk;
int it_max=100;
int it_max0=100;
int o2=-1;
int n2=-1;
double x1[];
double x2[];
double C;
        public f1_gibbs(String igs[],double[] in0,double iTe[],double iT,double iP)
        {    R=8.3145;
        C=50*R*iT;
                Te=new double[iTe.length];
            name=new String[iTe.length];
            for(int i=0;i<iTe.length;i++)
            {Te[i]=iTe[i];name[i]=igs[i];}
            setgibbs(igs,in0,iT,iP);
    }
public void setgibbs(String igs[],double[] in0,double iT,double iP){
        gs=new gibbsSpecies[igs.length];
        for(int i=0;i<igs.length;i++){
                gs[i]=new gibbsSpecies(igs[i],"gas");
        }
int nn=igs.length;
        int nat=110;
        double B[][]=new double[nat][nn];
        int nk=0;
        for(int i=0;i<nn;i++){
                for(nk=0;nk<gs[i].atomList.length;nk++){
```

```
                                                B[gs[i].atomList[nk].number-
1][i]=gs[i].atomList[nk].N;
                                                }
                                        }
                                int natcount=0;
                                boolean zeroflag[]=new boolean[110];
                                for(int j=0;j<nat;j++)
                                {zeroflag[j]=false;for(int
i=0;i<nn;i++){if(B[j][i]!=0){zeroflag[j]=true;}};if(zeroflag[j]) natcount++;}
                                A=new double[natcount][nn];
                                int k=0;
                                for(int j=0;j<nat;j++)
                                {if(zeroflag[j]){A[k]=B[j];k++;}}
                                n0=in0;
                                P=iP;
                                T=iT;
                                T0=iT;
                                mu=new double[gs.length];
                                l=A.length;
                                NG=A[0].length;
                                b=new double[l];
                                nt=getnt(n0);
                                ntahminilk=nt/n0.length;
                                n=new double[n0.length];
                                for(int i=0;i<n0.length;i++){
                                        n[i]=ntahminilk;
                                }
                                nt=getnt(n);
                                b0=new double[l];
                                for(int ii=0;ii<b0.length;ii++)
                                {for(int ij=0;ij<NG;ij++)
                                        { double carpim=A[ii][ij]*n0[ij];
                                          b0[ii]+=carpim;
                                        }
                                }
                                calMu(n0);
                        }
                        public double getnt(double[] jn)
                        {
                                double nt_inner=0;
                                for(int i=0;i<jn.length;i++) {nt_inner+=jn[i];}
                                return nt_inner;
                        }
                        public void setN0(double[] in0){
                           for(int i=0;i<n0.length;i++) n0[i]=in0[i];
                        }

                        public void setT(double iTeq){
                                T=iTeq;
                                calMu(n0);
                        }
                        public void setTDefault(){
                                T=T0;
                        }

                        public void calMu(double[] in){
                                for(int i=0;i<mu.length;i++){
                                        double inT=getnt(in);
```

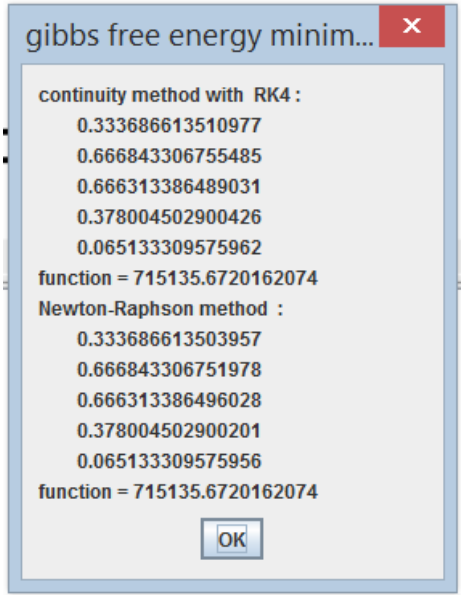
```
                                        mu[i]=gs[i].mu(T,P,in[i],inT);
                                }
                        }
                        public double[] getb(double[] in){
                                double inner_b[]=new double[A.length];
                                for(int ii=0;ii<inner_b.length;ii++){
                                        inner_b[ii]=0;
                                        for(int ij=0;ij<NG;ij++){
                                                inner_b[ii]+=A[ii][ij]*in[ij];
                                        }
                                }
                                return inner_b;
                        }
                        public double A_lambda(double[] in,int j)
                        {
                                double inner_b=0;
                                for(int i=0;i<in.length;i++)
                                {inner_b+=A[i][j]*in[i];}
                                return inner_b;
                        }
                        public double getG(double[] in)
                        {
                                calMu(in);
                                double munt=0;
                                for(int j=0;j<mu.length;j++) mu[j]*=in[j];
                                for(int i=0;i<mu.length;i++) munt+=mu[i];
                                return munt;
                        }
public double func(double x[])
{
double ff1=0;
int n1=x.length;
double ff[]=new double[n1];
double G;
double x1[]=new double[NG];
double x2[]=new double[l];
for(int i=0;i<NG;i++)
{x1[i]=x[i];}
for(int i=NG;i<(NG+l);i++) {x2[i-NG]=x[i];}
calMu(x1);
for(int i=0;i<NG;i++) {ff1+=mu[i]*x1[i];}
b=getb(x1);
for(int i=0;i<l;i++) {ff1+=C*x2[i]*(b[i]-b0[i]);}
return ff1;
}
}
```

Example program minimizing gibbs freee energy equation by using continuity and Newton-Raphson method.

Program 6.3.8 example gibbs free energy minization , by gibbs equation function f1_gibbstest

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

```
public class f1_gibbstest
{
 public static void main(String arg[])
 {     String iLK[]={ "co","o2","co2" };
            double[] n0={1,1,0};
                        double Te[]={298.15,298.15,298.15};
                        double T1=2500;
                        double R=8.3145;
                        double C=R*T1;
                        double xx=500000.0/C;
                        double P1=1.01325;
                        double [] x0={0.5,0.5,0.5,xx,xx};
            f1_gibbs f=new f1_gibbs(iLK,n0,Te,T1,P1);
            double [] r= NA34F.newton_continuationRK4(f,x0,5);
            fxb f2 =new fxb(f);
            double [] r1= NA40.newtond(f2,x0);
    String s="continuity method with  RK4 : \n"+Matrix.toStringT(r)+"function = "+f.func(r)+"\n";
    s+="Newton-Raphson method  : \n"+Matrix.toStringT(r1)+"function = "+f.func(r1)+"\n";
    String s1="gibbs free energy minimisation  : ";
    JOptionPane.showMessageDialog(null,s,s1,JOptionPane.PLAIN_MESSAGE);
 }
}
```



6.4 Chemical Reaction Kinetics

In the previous subchapter, it is explained how the equilibrium composition of a reactive gas mixture can be predicted by the application of thermodynamics. But the reactions are also time dependent processes and equilibrium describes the potential of the reaction, but it does not give the rate at which the reaction proceeds, or how quickly equilibrium is approached. If a combustion or reaction vessel to be designed. Reaction rate is also an important parameter. Chemical reactions may ocur when two or more molecules or atoms approach one another. The type of reaction may ocur depends on intermolecular forces existed during a collisional encounter. A reaction which takes place by such a collusion process is called an elementary reaction. As en end result of many elementary reaction a global reaction occurs. For example if a global reaction is

$CO + 1/2O_2 \rightarrow CO_2$ Actual elementary reactions might be:

$CO + O_2 \rightarrow CO_2 + O$ and $CO + O + M \rightarrow CO_2 + M$where M is a third body such as N_2 or O_2
For a methane reaction

$$CH_4 + 2O_2 \rightarrow CO_2 + H_2O$$

Actual elementary reactions might be:
$CH_4 + O_2 \rightarrow CH_3 + HO_2$ or $NO + M \rightarrow N + O + M$ or $CH_4 + O \rightarrow CH_3 + OH$
For a set of elementary reactions
$AB + C \rightarrow BC + A$ reaction (1) (6.3.12)
$A + B + M \rightarrow AB + M$ reaction (2) (6.3.13)
$AB + M \rightarrow A + B + M$ reaction (3) (6.3.14)
The reaction rate is depends on the rate constant times concentration of each of the reactants
$-\frac{d[AB]}{dt} = -\frac{d[C]}{dt} = \frac{d[BC]}{dt} = \frac{d[A]}{dt} = k_1[AB][C]$ reaction (1) (6.3.15)
$-\frac{d[A]}{dt} = -\frac{d[B]}{dt} = \frac{d[AB]}{dt} = \frac{d[A]}{dt} = k_2[A][B][M]$ reaction (2) (6.3.16)
$-\frac{d[AB]}{dt} - \frac{d[A]}{dt} = \frac{d[B]}{dt} = \frac{d[A]}{dt} = k_3[AB][M]$ reaction (3) (6.3.17)
Where the factors k_1, k_2, k_3 are the reaction rate constants for the respective reactions. In General reaction constants can be calculated by using Arhenius equation
$k = AT^n p^m \, exp\left(-\frac{E}{RT}\right) = k_0 \, exp\left(-\frac{E}{RT}\right)$
Where E is the activation energy required to bring the ractants to a reactive state. A,n and m are constants. İf for a given reaction n=m=0 then $k_0 = AT^n p^m$ term can be taken as a constant as well. Usually these coefficients can be obtain from experimental data taken from actual reactions takes place.
For a reaction of the generic form:
$aA + bB \rightarrow cC + dD$ (6.3.18) where a,b,c,d are stoichiometric coefficients, the rate of destruction of A and B and the rate of formation C and D are given by:
$-\frac{d[A]}{dt} = -ak_f[A]^a[B]^b$ (6.3.19)
$-\frac{d[B]}{dt} = -bk_f[A]^a[B]^b$ (6.3.20)
$\frac{d[C]}{dt} = ck_f[A]^a[B]^b$ (6.3.21)
$\frac{d[D]}{dt} = dk_f[A]^a[B]^b$ (6.3.22)
Elementary reactions are reversible, so that the back reaction is
$cC + dD \rightarrow aA + bB$ (6.3.23)
$\frac{d[A]}{dt} = ak_b[C]^c[D]^d$ (6.3.24)
Combining backward and forward reactions, the net reaction rate of A is
$\frac{d[A]}{dt} = a\left(k_b[C]^c[D]^d - k_f[A]^a[B]^b\right)$ (6.3.25)
At equilibrium state $\frac{d[A]}{dt} = 0$, and (6.3.26)
$K_c = \frac{k_f}{k_b} = \frac{[C]^c[D]^d}{[A]^a[B]^b}$ (6.3.27)
K_c is called concentration equilibrium constant and related to equilibrium constant defined in the previous chapter as
$K_c = K(RT)^{a+b-c-d}$ (6.3.28) so if one side of the reaction constants are known, it is possible to obtain reaction constants for the reverse direction by using equilibrium constant.

Now we can look at some actual combustion reaction kinetics A global reaction scheme for propane and other hydrocarbons(C_nH_{2n+2}) combustion as[35] :
$C_nH_{2n+2} \rightarrow \left(\frac{n}{2}\right)C_2H_4 + H_2$ (6.3.29)
$C_2H_4 + O_2 \rightarrow 2CO + 2H_2$ (6.3.30)
$CO + \frac{1}{2}O_2 \rightarrow CO_2$ (6.3.31)

$H_2 + \frac{1}{2}O_2 \rightarrow H_2O$ (6.3.32)

Global rate equations in units of cm^3,gmol,s , cal and K have been developed for each reaction species. R=82.05 cm^3atm/K-gmol

$\frac{d[C_nH_{2n+2}]}{dt} = -10^{17.32}\, exp\left(\frac{-49600}{RT}\right)[C_nH_{2n+2}]^{0.5}[O_2]^{1.07}[C_2H_4]^{0.4}$ (6.3.33)

$\frac{d[C_2H_4]}{dt} = -10^{14.70}\, exp\left(\frac{-50000}{RT}\right)[C_2H_4]^{0.9}[O_2]^{1.18}[C_nH_{2n+2}]^{-0.37}$ (6.3.34)

$\frac{d[CO]}{dt} = -10^{14.60}\, exp\left(\frac{-40000}{RT}\right)[CO]^{1.0}[O_2]^{0.25}[H_2O]^{0.5} + 5 * 10^{8}\, exp\left(\frac{-40000}{RT}\right)[CO_2]^{1.0}$ (6.3.35)

$\frac{d[H_2]}{dt} = -10^{13.52}\, exp\left(\frac{-41000}{RT}\right)[H_2]^{0.85}[O_2]^{1.42}[C_2H_4]^{-0.56}$ (6.3.36)

Since this equations are experimental curve fit equations, limits of equations should be considered in use. This equation is valid for pressure 1 to 9 atm and temperatures 960 to 1540 K, and equivalance ratio F 0.12 to 2

For simpler analysis a single step equation is given in the same source, fort he reaction $Fuel + \alpha O_2 \rightarrow \beta CO_2 + \gamma H_2O$ (6.3.37) where α,β,γ are stoichiometric coefficients. The associated global kinetic rate equation is:

$\frac{d[fuel]}{dt} = -AT^n p^m\, exp\left(\frac{-E}{RT}\right)[fuel]^a[O_2]^b$ (6.3.38) for most of the constants n=m=0 and E , a and b coefficients for some common hydrocarbons are given in the table below.

A better accuracy can be obtained by using a two step equation as

$Fuel + \alpha O_2 \rightarrow \beta CO_2 + \gamma H_2O$ (6.3.39)

$CO + \frac{1}{2}O_2 \rightarrow CO_2$ (6.3.40)

Table is listed coefficients for both cases of equations

Table 6.4.1 Global one and two step kinetic reaction constants[

Fuel	A(one step)	A(two step)	E(kcal/gmol)	a	b
CH_4	1.3e6	2.8e9	48.4	-0.3	1.3
C_2H_6	1.1e12	1.3e12	30	0.1	1.65
C_3H_8	8.6e11	1.0e12	30	0.1	1.65
C_4H_{10}	7.4e11	8.8e11	30	0.15	1.6
C_5H_{12}	6.4e11	7.8e11	30	0.25	1.5
C_6H_{14}	5.7e11	7.0e11	30	0.25	1.5
C_7H_{16}	5.1e11	6.3e11	30	0.25	1.5
C_8H_{18}	4.6e11	5.7e11	30	0.25	1.5
C_9H_{20}	4.2e11	5.2e11	30	0.25	1.5
$C_{10}H_{22}$	3.8e11	4.7e11	30	0.25	1.5
CH_3OH	3.2e11	3.7e12	30	0.25	1.5
C_2H_5OH	1.5e11	1.8e12	30	0.25	1.6
C_6H_6	2.0e11	2.4e11	30	-0.1	1.85
C_2H_6	1.6e11	1.9e11	30	-0.1	1.85

By using these data chemical kinetics of the fuels can be calculated. Note that equations are differential equations, so they should be solved by using numerical differential equation solving methods. In this particular example &th order Runge-Kutta method is used.

Program 6.4.1 Chemical kinetics of C3H8 combustion (two step equation)

```
//=====================================================
// 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 T=1500.0;//degree K;
double nfi;
public  fm1(double nfi1,double Ti)
{nfi=nfi1;T=Ti;}

//multivariable function
 public double func(double x[],int x_ref)
 {
   //x[0] is x, x[1] is nfuel  x[2] is nCO2
   double y=1.0;
   double nCO=3.0*(nfi-x[1])-x[2];
   double nO2=(3.5*x[1])+1.5*nfi-0.5*x[2];
   double nH2O=4.0*(nfi-x[1]);
   double nCO2f=3.0*nfi;
   if(x_ref==0) {y=-1.0e12*Math.exp(-
30000.0/(1.987*T))*Math.pow(x[1],0.1)*Math.pow(nO2,1.65);return y;}
   else if(x_ref==1) {y=pow(10,14.6)*exp(-40000.0/(1.987*T))*nCO*pow(nO2,0.25)*pow(nH2O,0.5)-
   5.0e8*exp(-40000.0/(1.987*T))*x[2];return y;}
   else return 0;
 }}
```

class diferansiyel_kinetics

```
{
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;
}


  public static void main(String args[]) throws IOException
  {
  double x[];
  x=new double[1];
  double p=1.0; //atm;
  double R=82.05; // (cm^3 atm/K/gmol)
  double T=1500.0;
  double n=p/(R*T);

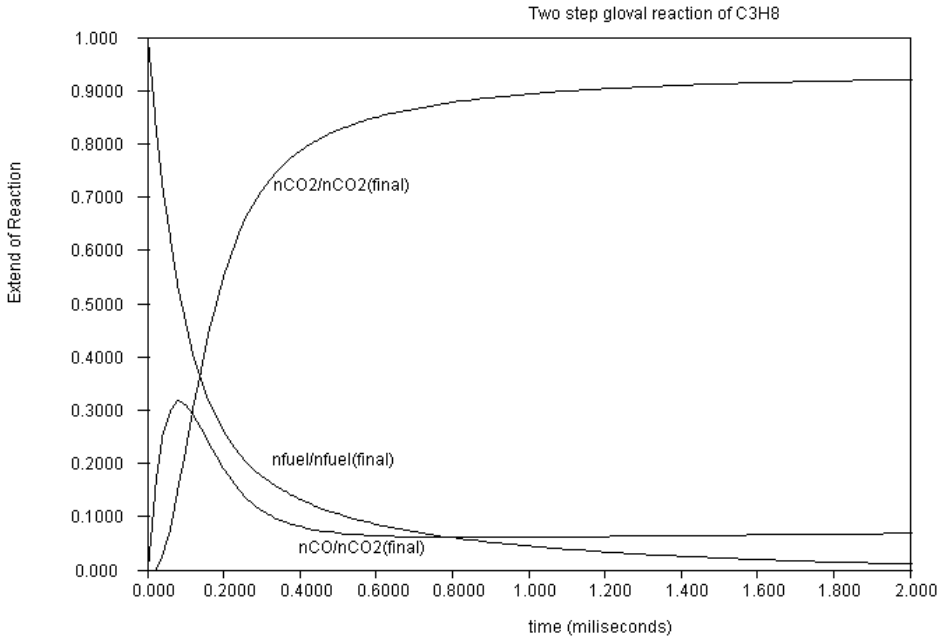
  double y[]=new double[2];
  y[0]=1./24.8*n;//initial amount of fuel
  y[1]=0.0;      //initial amount of CO2
  double nCO=0;
  double nCO2f=3.0*y[0];
  fm1 b3=new fm1(y[0],T);
  int N=100;
  double a[][]=RK6(b3,0.0,2.0e-3,y,N);
  double z[][]=new double[4][a[0].length];
  for(int i=0;i<a[0].length;i++)
  {z[0][i]=a[0][i]*1e3;//milisecond
   z[1][i]=a[1][i]/y[0];
   z[2][i]=a[2][i]/nCO2f;
   nCO=3.0*(y[0]-a[1][i])-a[2][i];
   z[3][i]=nCO/nCO2f;
  }
```

```
 String s="C3H8 reaction";
 String s1[]={"time miliseconds","nfuel/nfuel(initial)","nCO2/nCO2(final)","nCO/nCO2(final)"};
 Text.printT(z,s1,s);
 Plot pp=new Plot(z[0],z[1]);
 pp.addData(z[0],z[2]);
 pp.addData(z[0],z[3]);
 pp.plot();
 }
}
```

C3H8 reaction

time miliseconds	nfuel/nfuel(initial)	nCO2/nCO2(final)	nCO/nCO2(final)
0.0	1.0	0.0	0.0
0.02	0.8499213276548222	0.0	0.1500786723451778
0.04	0.7260237235497475	0.0248162771714069	0.24915999927884558
0.06000000000000001	0.622265416984699	0.07709127890116606	0.300643304114135
0.08	0.5357167789379452	0.14724177494564047	0.31704144611641427
0.1	0.46424748941134525	0.22519132401786732	0.3105611865707874
0.12000000000000001	0.40567089452061017	0.30325973457374605	0.2910693709056438
0.14	0.35780185389565666	0.3766289197674227	0.26556922633692065
0.16	0.3186197501910662	0.4428378476043715	0.23854240220456233
0.18000000000000002	0.2863764268264579	0.5010463705984641	0.21257720257507798
0.2	0.25963000876311904	0.5513888529867126	0.18898113825016835
0.22	0.2372282609115129	0.5945081763510455	0.1682635627374417
0.24000000000000002	0.21826801714700075	0.6312580442830378	0.15047393856996155
0.26	0.20204860614866055	0.662530437902172	0.1354209559491674
0.28	0.18802855895134887	0.689165543711067	0.12280589733758412
0.30000000000000004	0.17578916990873708	0.7119117057075685	0.11229912438369449
0.32	0.1650054419052346	0.7314136320633756	0.10358092603138962
0.34	0.15542362281954383	0.7482153355491645	0.09636104163129158
0.36000000000000004	0.14684413530805207	0.7627698803052391	0.09038598438670879
0.38	0.13910871895136454	0.7754515275648937	0.08543975348374176
0.4	0.13209077777289566	0.7865679786159747	0.08134124361112971
0.42000000000000004	0.12568813223201236	0.7963716224425613	0.07794024532542619
0.44	0.1198175622031585	0.8050693622713343	0.07511307552550725
0.46	0.11441068073948812	0.8128309449453257	0.07275837431518623
0.48000000000000004	0.10941079727348914	0.8197958868004008	0.07079331592611009
0.5	0.1047705184017614	0.8260791605267525	0.06915032107148603
0.52	0.10044990070245546	0.8317758260819303	0.06777427321561412
0.5400000000000001	0.09641501873595083	0.8369647814454706	0.06662019981857863
0.5600000000000002	0.09263684702251465	0.8417117903736836	0.06565136260380172
0.5800000000000002	0.08909038086239551	0.8460719220422871	0.06483769709531734
0.6000000000000003	0.08575393996335737	0.8500915154459475	0.06415454459069507
0.6200000000000003	0.08260861287300228	0.8538097614384377	0.0635816256885601
0.6400000000000003	0.07963781056185192	0.857259977993613	0.06310221144453514
0.6600000000000005	0.07682690516860113	0.8604706397132466	0.06270245511815235
0.6800000000000005	0.07416293562401223	0.8634662105950897	0.062370853780898135
0.7000000000000005	0.07163436613724299	0.8662678192856574	0.06209781457709953
0.7200000000000006	0.0692308867368103	0.8688938081360214	0.06187530512716839
0.7400000000000007	0.06694324748332994	0.8713601810328305	0.06169657148383963
0.7600000000000007	0.06476311981369515	0.8736809699050839	0.061555910281220864
0.7800000000000007	0.0626829798838221	0.8758685357659739	0.06144848435020403
0.8000000000000008	0.06069600985798753	0.8779338169352869	0.06137017320672554
0.8200000000000008	0.05879601392733913	0.8798865345349158	0.061317451537745024
0.8400000000000009	0.0569773464879761	0.8817353633231539	0.06128729018887
0.860000000000001	0.05523485041456008	0.8834880743241921	0.061277075261247926
0.880000000000001	0.053563803762078026	0.8851516544310472	0.06128454180687475
0.900000000000001	0.05195987354126111	0.8867324071440783	0.06130771931466057
0.9200000000000012	0.05041907546127787	0.8882360377986536	0.06134488674006863
0.9400000000000012	0.04893773873111448	0.8896677259911553	0.061394535277730304
0.9600000000000012	0.047512475169565145	0.8910321873981516	0.06145533743228328
0.9800000000000013	0.0461401520014439	0.8923337267722061	0.06152612122634995

aaa



6.5 Power cycles with actual combustion process

6.5.1 Brayton (Gas turbine) cycle

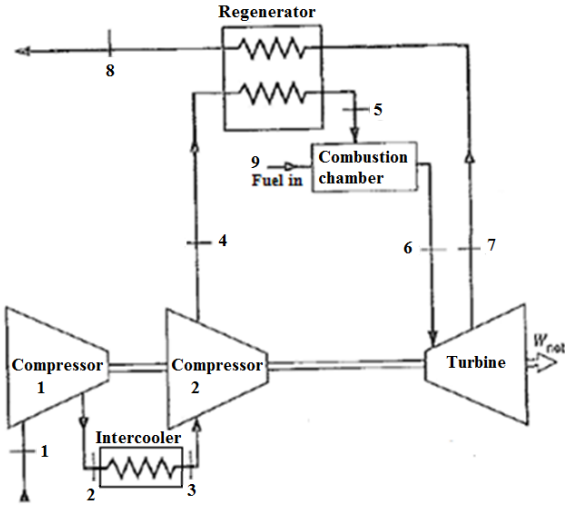


Figure 6.5.1.1 Brayton cycle given in the example problem

In actual gas turbine cycle, air combust with a fuel to generate the required heat for the cycle, exit of combustion chamber is combustion products. Therefore in the remaining process , combustion products should be taken into account as working fluid of the cycle. Since combustion products are a mixture, in order to evaluate gas properties. Gmix (gas mixture) class is used. In order to simulate combustion process, combustion class is used. The basic cycle equations are as follows:

$m = m_{air}$ (6.5.1)

$W_{lowpressurecompressor} = m(h_2 - h_1)$ (6.5.2)

$W_{lowpressureisentropiccompressor} = m(h_{2i} - h_1) \qquad s_{2i} = s_1$ (6.5.3)

$W_{highpressurecompressor} = m(h_4 - h_3)$ (6.5.4)

$W_{highpressureisentropiccompressor} = m(h_{4i} - h_3) \qquad s_{4i} = s_3$ (6.5.5)

$\eta_{isentropiccompressor} = \frac{W_{isentropiccompressor}}{W_{compressor}}$ (6.5.6)

$Q_{intercooler} = m(h_2 - h_3)$ (6.5.7)

$Q_{regenerator} = m(h_5 - h_4) = m_{product}(h_9 - h_{10})$ (6.5.8)

$Q_{combustionchamber} = m_{product}h_6 - (mh_5 + m_{fuel}h_{fuel})$ $P_6 = P_5 = P_4$ (6.5.9)

$W_{turbine} = m_{product}(h_6 - h_7)$ (6.5.10)

$W_{isentropicturbine} = m_{product}(h_6 - h_{7i})$ (6.5.11)

$\eta_{isentropicturbine} = \frac{W_{turbine}}{W_{isentropicturbine}}$ (6.5.12)

$\eta_{cycle} = \frac{W_{turbine} - W_{compressor}}{Q_{combustionchamber}}$ (6.5.13)

Program 6.5.1.1 Brayton cycle with combustion in combustion chamber

```
 public class brayton4
{
Gas g;
double PP1[],vv1[];
double PP2[],vv2[];
double TT1[],ss1[];
double TT2[],ss2[];
double
Wturbine,Wcompressor1,Wcompressor2,Wcompressor,Wnet,Q1,Qregenerator,Qintercooler,Qnet,eta;
double T[],P[],h[],s[];
double m; // air mass flow rate;
combustion cc;
double a_f; //air to fuel ratio
double mair;
double mratio;
double mproduct; //proct mass flow rate kg/s
double mfuel;
double ds5p; //entropy difference of two air and product gases
double dh5p; //enthalpy difference of two air and product gases
public brayton4(String fueli[],double Nfueli[],double m,double T1,double P1,double Tfuel,double
P2,double P4,double T3,double T6,double dT,double etacompressor1,double etacompressor2,double
etaturbine1,double etaturbine2)
{
  g=new Gas("air");
  g.base("mass");
  T=new double[11];
  P=new double[11];
  h=new double[11];
  s=new double[11];
  TT1=new double[25];ss1=new double[25];
  TT2=new double[2];ss2=new double[2];
// 1 compressor 1 inlet
P[1]=P1;T[1]=T1;
P[2]=P2;
P[3]=P2;T[3]=T3;
P[4]=P4;
P[5]=P4;
P[6]=P4;T[6]=T6;
P[7]=P1;
P[8]=P1;
// 1 inlet
double a[]=g.property(T[1],P[1]);
h[1]=a[3];
s[1]=a[5];
```

```
TT1[0]=T[1];ss1[0]=s[1];
double pr1=a[17];
// 2i = 11 isentropic outlet of compressor 1
double pr2=P2/P1*pr1;
double T2i=g.Ti(pr2);
a=g.property(T2i,P2);
double h2i=a[3];
double s2i=s[1];//a[5];
double Cp2i=a[9];
T[9]=T2i;
s[9]=s2i;
TT1[1]=T2i;ss1[1]=s2i;
// 2 outlet of compressor 1
h[2]=(h2i-h[1])/etacompressor1+h[1];
T[2]=(h[2]-h2i)/Cp2i+T2i;
a=g.property(T[2],P[2]);
s[2]=a[5];
// 3 compressor 2 inlet
a=g.property(T[3],P[3]);
h[3]=a[3];
s[3]=a[5];
double pr3=a[17];
// 4i = 12 isentropic process compressor 2 outlet
double pr4=P[4]/P[3]*pr3;
double T4i=g.Ti(pr4);
a=g.property(T4i,P[4]);
double h4i=a[3];
double s4i=a[5];
double Cp4i=a[9];
T[10]=T4i;
s[10]=s4i;
//4 outlet of compressor 2
h[4]=(h4i-h[3])/etacompressor2+h[3];
T[4]=(h[4]-h4i)/Cp4i+T4i;
a=g.property(T[4],P[4]);
s[4]=a[5];
double Cp4=a[9];
//6 combustion chamber outlet
  a_f=1.0;
  cc=new combustion("combustion chamber",fueli,Nfueli,a_f);
  a_f=cc.adiabatic_a_f_ratio(T1,Tfuel,T6,1.0,10.0);
  mair=cc.nair/28.97;//mass flow rate of air from reaction
  mratio=m/mair;//air mass flow rate correction ratio compare to input air flow rate
  mproduct=(cc.productmix.N/cc.productmix.M)*mratio; //proct mass flow rate kg/s
  mfuel=cc.fuelmix.N/cc.fuelmix.M*mratio;//fuel mass flow rate kg/s
  cc.fuelmix.base("mass");
  cc.productmix.base("mass");//productmix calculations will be mass bases kJ/kg..
  a=cc.productmix.property(T[6],P[6]);
  h[6]=a[3];
  s[6]=a[5];
  double Cp6=a[9];
  double pr6=a[17];
   a=g.property(T[6],P[6]);
  //fuel inlet enthalpy
  a=cc.fuelmix.property(Tfuel,P[6]);
  double hfuel=a[3];
  //7i isentropic turbine outlet
  double pr7=P[7]/P[6]*pr6;
```

```
  double T7i=cc.productmix.Ti(pr7);
  a=cc.productmix.property(T7i,P[7]);
  double h7i=a[3];
  double s7i=a[5];
  double Cp7i=a[9];
  T[0]=T7i;
  s[0]=s7i;
  //7 turbine outlet
  h[7]=h[6]-etaturbine1*(h[6]-h7i);
  T[7]=(h[7]-h7i)/Cp7i+T7i;
  a=cc.productmix.property(T[7],P[7]);
  s[7]=a[5];
  double Cp7=a[9]/cc.productmix.M;
  //8 heat exchanger outlet
  T[8]=T[4]+dT;
  a=cc.productmix.property(T[8],P[8]);
  h[8]=a[3];
  s[8]=a[5];
  double Cp8=a[8]/cc.productmix.M;
  //5 heat exchanger exit
  Qregenerator=mproduct*(h[7]-h[8]);
  T[5]=T[8]+Qregenerator/mproduct/Cp8;
  a=g.property(T[5],P[5]);
  h[5]=a[3];
  s[5]=a[5];
  //5' heat exchanger exit calculated as product gases
  //normalisation of two gas data
  a=cc.productmix.property(T[5],P[5]);
  double h5p=a[3];
  double s5p=a[5];
  ds5p=s5p-s[5];
  //System.out.println("ds5p="+ds5p);
  s[6]-=ds5p;
  s[7]-=ds5p;
  s[8]-=ds5p;
  s[0]-=ds5p;
Wturbine=mproduct*(h[6]-h[7]);
Wcompressor1=m*(h[2]-h[1]);
Wcompressor2=m*(h[4]-h[3]);
Wcompressor=Wcompressor1+Wcompressor2;
Q1=mproduct*h[6]-m*h[5]-mfuel*hfuel;
double Qregenerator1=m*(h[5]-h[4]);
Qintercooler=m*(h[2]-h[3]);
Qnet=Q1;
Wnet=Wturbine-Wcompressor;
eta=Wnet/Qnet;
}
public double[][] line(double T1,double T2,double Pi,int n)
{
double TT1[]=new double[n];
double ss1[]=new double[n];
double aa[];
for(int i=0;i<n;i++)
{TT1[i]=T1+i*(T2-T1)/(n-1);
 aa=g.property(TT1[i],Pi);
 ss1[i]=aa[5];
}
double a[][]={TT1,ss1};
```

```
return a;
}
public double[][] line1(double T1,double T2,double Pi,int n)
{
double TT1[]=new double[n];
double ss1[]=new double[n];
double aa[];
for(int i=0;i<n;i++)
{TT1[i]=T1+i*(T2-T1)/(n-1);
 aa=cc.productmix.property(TT1[i],Pi);
 ss1[i]=aa[5]-ds5p;
}
double a[][]={TT1,ss1};
return a;
}
public void plot()
{double a[][]=new double[2][61];
 double c1[][]=new double[2][3];
 double c3[][]=new double[2][3];
 double c6[][]=new double[2][3];
 c1[0][0]=T[1];
 c1[1][0]=s[1];
 c1[0][1]=T[9];
 c1[1][1]=s[9];
 c1[0][2]=T[2];
 c1[1][2]=s[2];

 c3[0][0]=T[3];
 c3[1][0]=s[3];
 c3[0][1]=T[10];
 c3[1][1]=s[10];
 c3[0][2]=T[4];
 c3[1][2]=s[4];

 c6[0][0]=T[6];
 c6[1][0]=s[6];
 c6[0][1]=T[0];
 c6[1][1]=s[0];
 c6[0][2]=T[7];
 c6[1][2]=s[7];

 int i=0;
 a[0][0]=T[1];
 a[1][0]=s[1];
 double b[][]=line(T[2],T[3],P[2],15);
 for(i=0;i<15;i++)
 {a[0][i+1]=b[0][i];
  a[1][i+1]=b[1][i];
 }
 b=line(T[4],T[5],P[4],15);
 for(i=0;i<15;i++)
 {a[0][i+16]=b[0][i];
  a[1][i+16]=b[1][i];
 }
 b=line1(T[5],T[6],P[4],15);
 for(i=0;i<15;i++)
 {a[0][i+31]=b[0][i];
  a[1][i+31]=b[1][i];
```

```
 }
 b=line1(T[7],T[8],P[7],15);
 for(i=0;i<15;i++)
 {a[0][i+46]=b[0][i];
  a[1][i+46]=b[1][i];
 }

 Plot p=new Plot(a[1],a[0]);
 p.addData(c1[1],c1[0],1);
 p.addData(c3[1],c3[0],1);
 p.addData(c6[1],c6[0],1);
 p.addData(s,T,22);
 p.setPlabel("Brayton (Gas turbine) cycle with combustion process");
 p.setXlabel("s entropy kJ/(kgK)") ;
 p.setYlabel("T degree C");
 p.plot();
}

public String toString()
{
String ss=" Brayton cycle \n";
ss+="Combustion Reaction : \n";
ss+=cc.toString()+"\n";
ss+="mair ="+m+"kg/s mproduct="+mproduct+" kg/s mfuel\n";
ss+="Turbine power output W turbine = "+Wturbine+" kW \n";
ss+="Low pressure compressor power input W turbine = "+Wcompressor1+" kW \n";
ss+="High pressure compressor power input W turbine = "+Wcompressor2+" kW \n";
ss+="Compressor power input W turbine = "+Wcompressor+" kW \n";
ss+="Net power output  Wnet = "+Wnet+" kW\n";
ss+="Combustion chamber heat input Q combustion = "+Qnet+" kW\n";
ss+="Cycle efficiency eta  = "+eta+"\n";
ss+="Equivalent temperature carnot cycle efficiency eta  = "+(1.0-T[1]/T[6])+"\n";
ss+="Regenerator heat transfer   = "+Qregenerator+" kW\n";
ss+="Termodynamic states : \n";
for(int i=1;i<9;i++)
{ss+="P["+i+"] = "+P[i]+"bar T["+i+"] = "+(T[i]-273.15)+" degree C h["+i+"] = "+h[i]+" kJ/kg s["+i+"]
= "+s[i]+" kJ/kgK\n";}
return ss;
}
}
```

In the example program Natural gas with the following gas combination is taken

Gas name	**amount**
CH_4 methane	**0.906000**
C_2H_6 ethane	0.056000
C_3H_8 propane	0.000800
C_4H_{10} butane	0.000200
CO_2 carbondioxide	0.017000
N_2 nitrogen	0.011000

Air and gas input is 300 K. There is no heat transfer in combustion chamber, so the product will be in adiabatic flame temperature. Of course in order to obtain required turbine inlet temperature, amount of fuel should be adjusted. It is simulated by using an air to fuel ratio adjusted adiabatic flame temperature calculation method in combustion (Curve fitting by

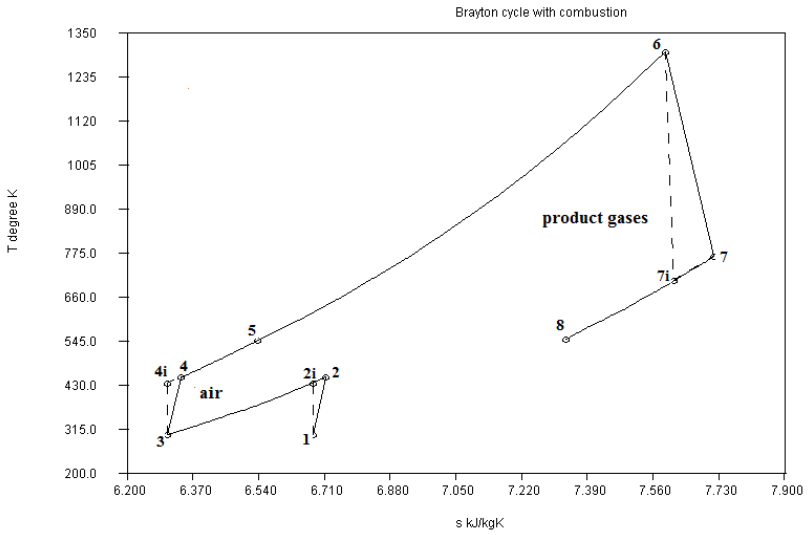
bisection method). Once amount of required fuel is calculated, the remaining of cycle can be treated just as normal cycle, with exception of using product gases instead of air.

Program 6.5.1.2 Brayton cycle with combustion in combustion chamber test program

```
public class braytontest4
{
 public static void main(String arg[])
 {
            double mhava=1; // kg/s air inlet temperature
            double T1=300.0; //degree K air inlet temperature
            double T3=300.0; //degree K air inlet temperature
            double P1=1.1;// bar air inlet pressure
            double P2=4.062019202;//bar air inlet pressure
            double P4=15.0;// compressed air pressure
            double T6=1300.0; //degree K turbine inlet temperature
            //natural gas fuel
            // ch4 0.906
    // c2h6 0.056
    // c3h8 8e-4
    // c4h10 2e-4
    // co2 0.017
    // n2  0.011
    String s[]={"ch4","c2h6","c3h8","c4h10","co2","n2"};
    double N[]={0.906,0.056,8.0e-4,2.0e-4,0.017,0.011};
    double a_f=1.0;
    double T8=1500.0; //degree K turbine inlet temperature
            double dT=100.0;// degree K heat exchanger temperature difference
            double etakompressor=0.9; //compressor isentropic efficiency
            double etaturbine=0.9; //turbine isentropic efficiency
    brayton4 bry1=new
brayton4(s,N,mhava,T1,P1,T1,P2,P4,T3,T6,dT,etakompressor,etakompressor,etaturbine,etaturbine);
    System.out.println(bry1.toString());
    bry1.plot();
 }
}
```

```
---------- Capture Output ----------
> "C:\java\bin\javaw.exe" braytontest4
Brayton cycle
Combustion Reaction :
[  0.9060 CH4 +   0.0560 C2H6 +   0.0008 C3H8 +   0.0002 C4H10 +   0.0170 CO2 +   0.0110 N2] +
4.9245*[ O2 + (79/21) *N2] →   1.0382 CO2 +   1.9842 H2O +  18.5365 N2 +   2.9112 O2
mair =1.0 kg/s  mproduct=1.0661129068583868 kg/s
Turbine power output W turbine = 717.3634398398311 kW
Low pressure compressor power input W turbine = 151.45004280335826 kW
High pressure compressor power input W turbine = 151.45004282508165 kW
Compressor power input W turbine = 302.9000856284399 kW
Net power output  Wnet = 414.4633542113912 kW
Combustion chamber heat input Q combustion = 1028.3480180935708 kW
Cycle efficiency eta  = 0.403038024986672
Equivalent temperature carnot cycle efficiency eta  = 0.7692307692307692
Regenerator heat transfer   = 265.35145370760745 kW
Termodynamic states :
P[1] = 1.1bar T[1] = 26.850000000000023 degree C h[1] = 300.47241417394997 kJ/kg s[1] =
6.680027277763032 kJ/kgK
P[2] = 4.062019202bar T[2] = 176.76142627605628 degree C h[2] = 451.92245697730823 kJ/kg s[2] =
6.714303344306349 kJ/kgK
```

P[3] = 4.062019202bar T[3] = 26.850000000000023 degree C h[3] = 300.47241417394997 kJ/kg s[3] = 6.305019006221799 kJ/kgK
P[4] = 15.0bar T[4] = 176.7614262973688 degree C h[4] = 451.9224569990316 kJ/kg s[4] = 6.339295072768477 kJ/kgK
P[5] = 15.0bar T[5] = 271.9600652210031 degree C h[5] = 549.8121169311926 kJ/kg s[5] = 6.536561114130103 kJ/kgK
P[6] = 15.0bar T[6] = 1026.85 degree C h[6] = 1519.7930629146838 kJ/kg s[6] = 7.903886582600744 kJ/kgK
P[7] = 1.1bar T[7] = 492.0016930813757 degree C h[7] = 846.9155138061639 kJ/kg s[7] = 8.027493063476864 kJ/kgK
P[8] = 1.1bar T[8] = 276.76142629736876 degree C h[8] = 598.0193115366096 kJ/kg s[8] = 7.645187250991324 kJ/kgK



6.6 Fuel cells systems

Fuel cells are electrochemical energy conversion devices, which directly converts chemical energy of a hydrocarbon into electrical energy and heat energy. The fuel cells are, like batteries, are consist of an anode, a cothode and an eloctrolyte. But unlike batteries fule cells generate electricty continuously as long as fuel an oxidant (usually air) is supplied to the fule cell. There are several different type of electrochemical reactions used in fuel cells, so there are several different fuel cell types according to the main chemical reaction type and system structure to allow this particular reaction. Alkaline(AFC) , Phosphoric acid(PAFC), Polymer electrode membrane(PEM) and solid oxide fuel (SOFC) type are the most common ones. Alkaline fuel cells are used in apollo program to power the systems. More recent two are PEM and SOFC fuel cells. Fuel cells are isothermal devices which operate almost constant temperatures, as much as system design allows it. The basic difference of fuel cell electrochemical reaction to combustion reaction is the generation of electrical energy (Work) out of fuel cell reaction. Work(electricity) generation is function of gibbs free energy generation. Energy balance on the control volume around the fuel cell can be given as $dQ + dW = dh + dKE + dPE$In operation of fuel cells, kinetic and potential energies can be neglected due to relatively small values compare to chemical energies. For heat transfer of isothermal process $dQ = TdS$can also be given so $dW = dh - TdS = dG$. In ideal conditions work output of a fuel cell is equal to chemial reaction gibbs free energy change. Ideal efficiency, or effectiveness of a fuel cell can be specified as

$$\eta_i = \frac{\Delta h}{\Delta g}$$

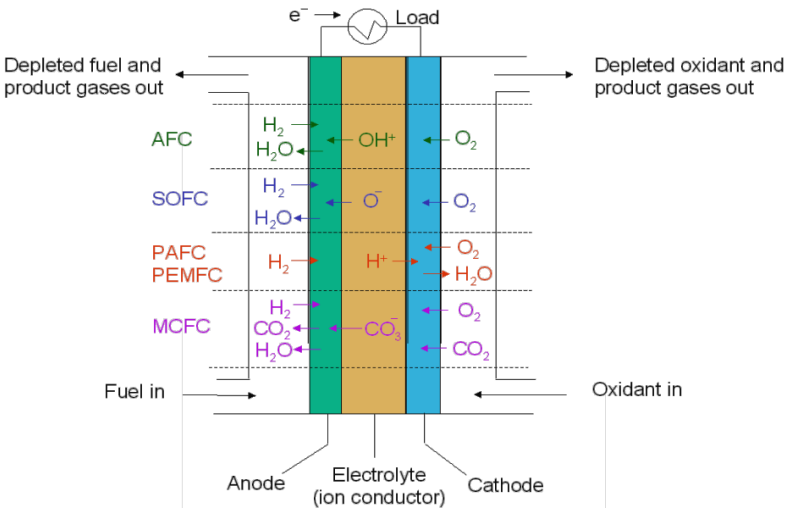


Figure 6.6.1 Operation principles of several type of fuel cells

In additional to ideal efficiency of a fuel cell, another important parameter is the ideal open-circuit voltage developed by the cell, which is $W_{elec} = -N_e F V_i = \Delta g$ where F is called faraday constant and its value is F=96487 kJ/(V kgmole elctrons) and N_e, number of electrons exchanged in reaction, is determined from a knowledge of specific reaction of the fuel cell. For example PEM type of fuel cell reactions:

Anode: $H_2 \rightarrow 2e^- + 2H^+$

Cathode: $2e^- + 2H^+ + 0.5O_2 \rightarrow H_2O$

Total reaction: $H_2 + 0.5O_2 \rightarrow H_2O$

So for this reaction N_e=2 since two electrons are liberatedat the anodefor every molecule of hydrogen consumed. Basic reactions of different type of fuel cells are shown in the figure. Cell voltage in an actual fuel cell is much lower than the ideal open circuit voltage calculated above. When some current started to drain from the circuit coltage drops due irreversibilities of the system. A typical voltage-current density diagram is as shown in the next figure.

Hydrogen is the simplest fuel and most of the fuel cells are used hydrogen as a fuel. Unfortunately, hydrogen is not existed as a free molecule in nature, so required hydrogen or other light hydrocarbons are obtained by converting hydrocarbons to hydrogen by using several different reaction in a chemical reactor called fuel reformer. One type of fuel reformer is called steam reformer. In this type of reformer steam and fuel heated to high temperatures and react with each other. Reaction is endothermic so if it is done adiabatiacly, gases will be cooled down through the reaction. For better fuel conversion efficiencies reactor should be heated up so that reaction could ocur isothermally. In the example program natural gas (assumed as 92 % methane and 8 % ethane) is reacted with steam to carbon ratio of 3.5(3.5 kmol of H2O per kmol of C in fuel). Inlet temperature of reactor was 1100 K and pressure is 2 atm. In adiabatic process temperature dropped to 767 K and 1.42 kmol of hydrogen generated for each kmol of natural gas. In isothermal process temperature is kept constant at 1100 K. In this case 3.54 kmol of hydrogen generated, but for this purpose 229050 kJ/kmol fuel of heat transfer is required to add to the process.

Program 6.6.1 Steam reformer to generate hydrogen from natural gas

```
public class gibbstest2 {
   public static void main(String[] arg)
   {    String s0[]={ "ch4","c2h6","co","co2","h2","h2o"};
            double[] n0={0.92,0.08,0,0,0,3.5};
```

```
                        double Te[]={1100,1100,1100,1100,1100,1100};
                        double T1=1100;//degree K
                        double P1=1.01325*2;//bar
                        int nmax1=500;//iterasyon sayısı
                        System.out.println("---------------------------------");
                        gibbs g1=new gibbs(s0,n0,Te,T1,P1);
                        g1.getEq();
                        for(int i=0;i<g1.n.length;i++) System.out.println(g1.gs[i].Formula+"   "+g1.n[i][0]);

                        System.out.println("Q= "+g1.delQ());
                         System.out.println("---------------------------------");
                        T1=g1.set_Q(s0,n0,Te,0.0,P1);
                        System.out.println("T1="+T1);
                        g1.getEq();
                        for(int i=0;i<g1.n.length;i++) System.out.println(g1.gs[i].Formula+"   "+g1.n[i][0]);
                        System.out.println("Q= "+g1.delQ());
                        System.out.println("---------------------------------");
            }
}
```

```
---------- Capture Output ----------
> "C:\java\bin\java.exe" gibbstest2
---------------------------------
CH4   0.004437243308236965
C2H6   2.2054011986170583E-7
CO   0.678456144854299
CO2   0.39710617075722027
H2   3.5437933381319064
H2O   2.0273315136312515
Q= 229050.03662771435
---------------------------------
T1=766.9857610446043
CH4   0.690073839575674
C2H6   0.0029669275977683863
CO   0.03260244276066198
CO2   0.351389862468137
H2   1.4263337057522907
H2O   2.7646178323030544
Q= -5.688723467756063E-4
---------------------------------

> Terminated with exit code 0.
```

A second way to design fuel reformer is to combusting fuel partially (partial oxidation) with a small amount of oxygen or air to create a gas mixture. Since this reaction is exothermic (gives heat to outside) system will generate the required hydrogen. In the next example a partial oxidation reformer values are given. There are only 0.38 kmol hydrogen generated per kmol of fuel and reaction rised the temperature to 1151 K.

Program 6.6.2 Partial oxidation reformer to generate hydrogen from natural gas

```
public class gibbstest2A {
   public static void main(String[] arg)
   {    String s0[]={ "ch4","c2h6","o2","n2","co","co2","h2","h2o"};
            double[] n0={0.92,0.08,0.1,0.376,0,0,0,0};
                        double Te[]={1100,1100,1100,1100,1100,1100,1100,1100};
                        double T1=1100;//degree K
```

```
                    double P1=1.01325*2;//bar
                    int nmax1=500;//iterasyon sayısı
                    System.out.println("---------------------------------");
                    gibbs g1=new gibbs(s0,n0,Te,T1,P1);
                    g1.getEq();
                    for(int i=0;i<g1.n.length;i++) System.out.println(g1.gs[i].Formula+"   "+g1.n[i][0]);

                    System.out.println("Q= "+g1.delQ());
                     System.out.println("---------------------------------");
                    T1=g1.set_Q(s0,n0,Te,0.0,P1);
                    System.out.println("T1="+T1);
                    g1.getEq();
                    for(int i=0;i<g1.n.length;i++) System.out.println(g1.gs[i].Formula+"   "+g1.n[i][0]);
                    System.out.println("Q= "+g1.delQ());
                    System.out.println("---------------------------------");
            }
}
```

```
---------- Capture Output ----------
> "C:\java\bin\java.exe" gibbstest2A
---------------------------------
CH4   0.7595653769400248
C2H6   0.06026372795107228
O2   1.730587548110011E-219
N2   0.3759999999999998
CO   0.19987468890113855
CO2   3.2478256693280153E-5
H2   0.38001770768126064
H2O   6.035458547586149E-5
Q= -4706.579566238721
---------------------------------
T1=1151.275694976136
CH4   0.7539817568405258
C2H6   0.06302411271183837
O2   4.4745674265526995E-208
N2   0.37600000000000006
CO   0.19996065954635073
CO2   9.358189445199061E-6
H2   0.3829435241086663
H2O   2.062407475861162E-5
Q= 0.001238806500059475
---------------------------------

> Terminated with exit code 0.
```

There are a third type of reformer which combines steam reformer with a partial oxidation reformer. They are called autothermal reformers. In autothermal reformer, heat supply to endothermic steam reaction by exorthermic partial oxidation reaction, so that total system is still adiabatic. As it is seen from the result, hydrogen generation is not as good as isentropic steam reformer, but it is given next best solution to fuel conversion process. There is one catch though in partial oxidation and autothermal reforming carbonmonoxide is generated as well, and this additional gas is poisonous for some type of fuel cells, such as PEM, therefore should be removed by using other chemical processes.

Program 6.6.3 autothermal reformer to generate hydrogen from natural gas

```
public class gibbstest2A {
    public static void main(String[] arg)
    {    String s0[]={ "ch4","c2h6","o2","n2","co","co2","h2","h2o"};
            double[] n0={0.92,0.08,0.452595,0.452595*3.76,0,0,0,3.5};
                    double Te[]={1100,1100,1100,1100,1100,1100,1100,1100};
                    double T1=1100;//degree K
                    double P1=1.01325*2;//bar
                    int nmax1=500;//iterasyon sayısı
                    System.out.println("---------------------------------");
                    gibbs g1=new gibbs(s0,n0,Te,T1,P1);
                    g1.getEq();
                    for(int i=0;i<g1.n.length;i++) System.out.println(g1.gs[i].Formula+"   "+g1.n[i][0]);

                    System.out.println("Q= "+g1.delQ());
                     System.out.println("---------------------------------");
                    T1=g1.set_Q(s0,n0,Te,0.0,P1);
                    System.out.println("T1="+T1);
                    g1.getEq();
                    for(int i=0;i<g1.n.length;i++) System.out.println(g1.gs[i].Formula+"   "+g1.n[i][0]);
                    System.out.println("Q= "+g1.delQ());
                    System.out.println("---------------------------------");
            }
}
```

```
---------- Capture Output ----------
> "C:\java\bin\java.exe" gibbstest2A
---------------------------------
CH4   7.932144841714599E-4
C2H6   8.930137454580886E-9
O2   7.054507293493372E-18
N2   1.7017572000000003
CO   0.5348935920946577
CO2   0.5443131755608981
H2   2.7967434874576873
H2O   2.781670056783569
Q= -0.8948624420445412
---------------------------------
T1=1100.0026848913144
CH4   7.931674024865107E-4
C2H6   8.929121761493364E-9
O2   7.055462587966727E-18
N2   1.7017571999999987
CO   0.5348956722944691
CO2   0.5443111424447892
H2   2.7967415955916763
H2O   2.781672042815991
Q= 0.013248269096948206
---------------------------------

> Terminated with exit code 0.
```

Now system thermodynamic analysis of a fuel cell system can be investigated more closely. In order to do this investigation a real fuel cell system will be considered. We will take a solid oxide fuel cell system as the basic system for the investigation. SOFC fuel cell uses a solid electrolite of zirconiumoxide(ZrO_2) stabilised by yttrium oxide(Y_2O_3). These metal oxides are usually called as ceramics, therefore SOFC's are also called ceramic fuel cells.

Anodes are usually made of nickel doped (for electrical activation) zirconia ceramics, and cathode are made of ceramic structures such as $LaCo_{1-x}Mn_xO_3$, x = 0.2-0.3 or $LaSrMnO_3$. Electrodes can be applied to the surface of ceramic elecrolytes or in more recent SOFC systems elecrolyte and cathode are applied as a paint layer into a thicker anode layer. In SOFC oxygen ion passes through the electrolyte. Since ionised gas is oxygen theroretically any fuel can be used in anod site. In the katod, oxygen ionised by using electrical current coming through the external connection. Oxygen ion is passed through the eleoctrolyte to anot site. $O_2 + 4e^- \rightarrow 2O^{2-}$ In the onot site fuel combines with the oxygen ion g water and electron current.

$2H_2 + 2O^{2-} \rightarrow H_2O + 4e^-$

$2CO + 2O^{2-} \rightarrow 2CO_2 + 4e^-$

Electrons are carried through the external circuit to the katot site.

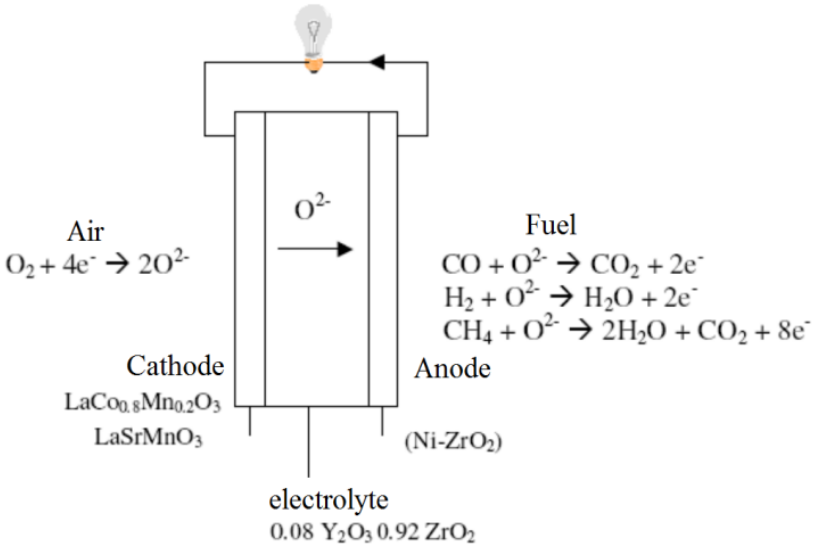


Figure 6.6.2 SOFC fule cell operation

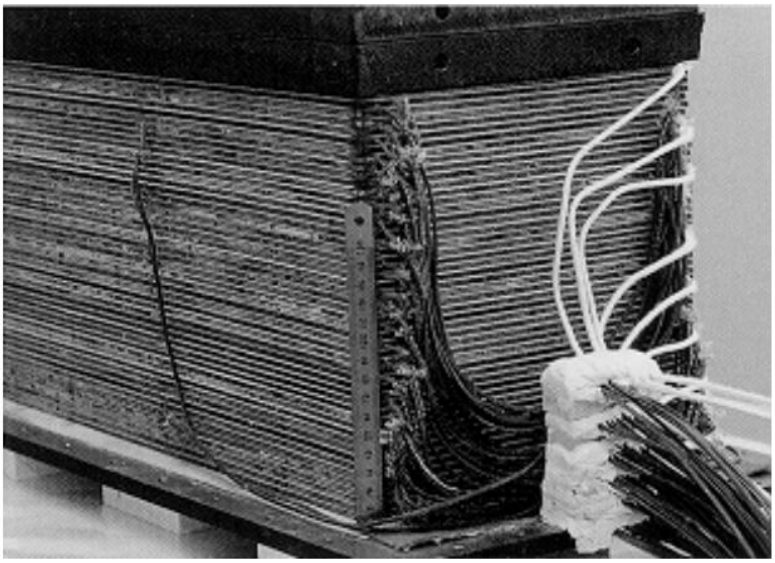


Figure 6.6.3 SOFC Fuel Cell stack

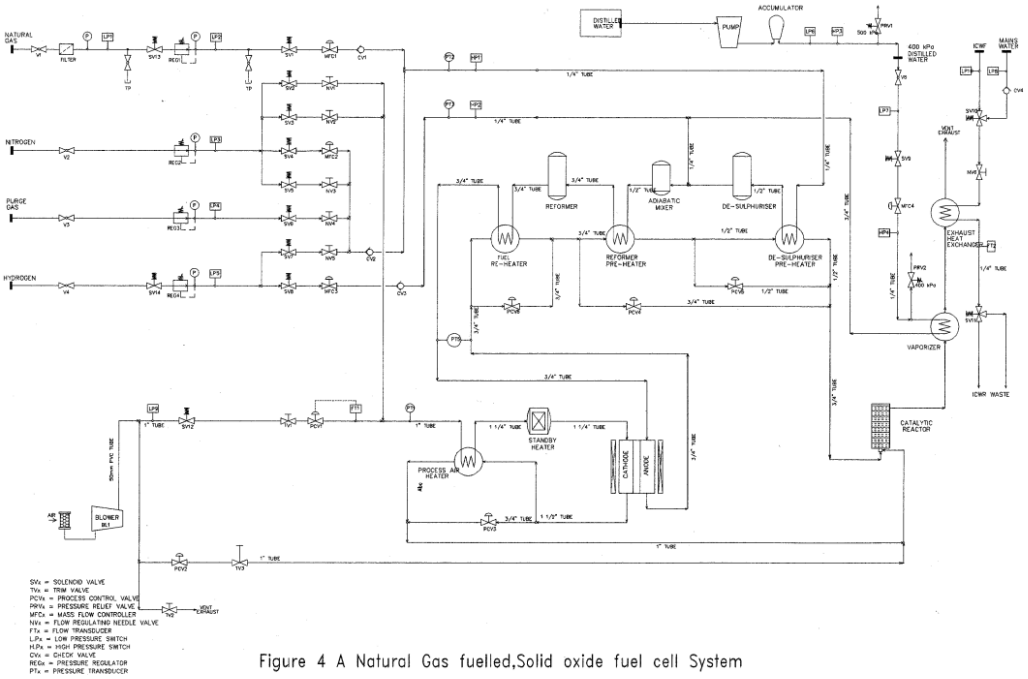


Figure 4 A Natural Gas fuelled,Solid oxide fuel cell System

Figure 6.6.4 Natural gas fuelled solid oxide fuel cell system

A natural gas fuelled solid oxide fuel cell system flow diagram is shown in Figure 4. As seen from the flow diagram, two main gas streams exist in the flow diagram. The first stream is the air stream. The air stream enters to the system through a blower. Then the air goes through a process air heat exchanger. In normal operation conditions, air stream should be heated up about 700°C at the exit of the heat exchanger. At the exit, an electrical heater is located. The main duty of this heater is to supply starting energy and temperatures for the heater. Once operating conditions of the fuel cell is reached, electrical heater is turned off, and fuel cell is used as the main heat source. Gas stream exit the fuel cell at 930°C and input the process air heater as hot gas stream. In this stage, some of the oxygen in the original air is already spent in the fuel cell. Therefore this gas stream is called depleted air stream. When depleted air passed through process air heat exchanger, it is cooled down to about 200°C. Depleted air is then sent to the catalytic reactor. Natural gas stream with main gas pipeline pressure of 40 kPa is used; therefore no additional fan or compressor is needed in fuel stream. Natural gas is first enters de-sulfuriser preheater. In this heat exchanger, it is heated to 300°C. It is then entered to desulphuriser. Sulphur and sulphur compounds are chemically bonded to de-sulphurization catalysts inside of the de-sulphuriser. Desulphurised natural gas entered to an adiabatic mixer and mixed up with steam and entered to catalytic reformer. Natural gas mix is heated up to 500°C, in another heat exchanger, Then it is entered an adiabatic reformer. In this reformer, high carbon number hydrocarbons are broken to methane and hydrogen by water shift reaction on nickel based catalyst surfaces. This process is called adiabatic fuel reforming. At the exit of the adiabatic reformer, mixture is cooled down to about 300°C because of the energy taken by reforming process. This mixture is went through another heat exchanger and heated to about 700°C before going to the solid oxide fuel cell stack. In the first stage of the stack, the rest of the methane is also converted to CO and hydrogen by internal reforming process. Internal reforming process is similar to reforming process, but instead of catalysts electrochemical reactions forced the mixture into equilibrium condition. Converted fuel then undergoes an electrochemical cell reaction and creates heat and electricity. 85% of the fuel is converted into the cell through electrochemical reaction. The rest of the gas together with combustion gases comes out of the fuel cell at 930°C. This stream, called depleted fuel stream, goes through the three stages of heat exchangers and heating up incoming streams. Relatively cooled depleted fuel finally enters to catalytic reactor and reacts with depleted air and additional fresh air. Catalytic reactor is made of platinum embedded metal surface. Small quantities of hydrocarbons can react on these surfaces. Reheated final combustion gases pass through a water

evaporator to supply system steam requirements for reforming process. A further heat exchanger is taken any heat left in the exhaust stream and cooled exhaust is released to the atmosphere. As another stream water is purified then pressurized through a pump and send to the water evaporator. The fuel cell itself can be compare to the cyclinder of an engine, the system around it makes it a whole power generation system.

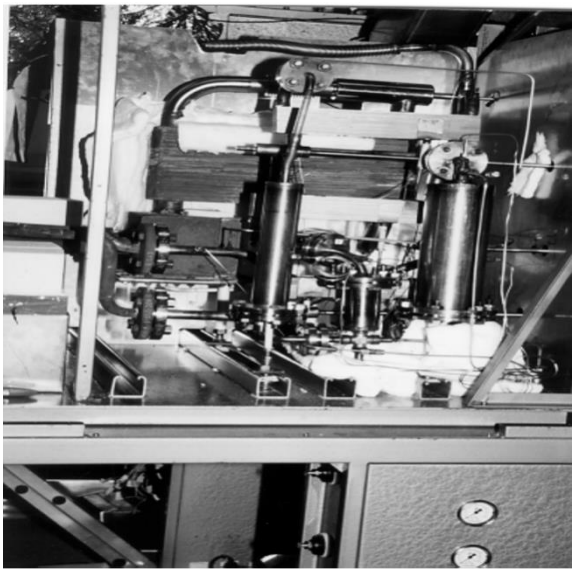


Figure 6.6.5 view of Natural gas fuelled solid oxide fuel cell

PROBLEMS

PROBLEM 1: The following reaction is given: Temperature of reactants is T=300 K

$CH_4 + 2.2O_2 + 8.72619N_2 \rightarrow n_1CH_4 + n_2O_2 + n_3N_2 + n_3CO_2 + n_3CO + n_4N_2O + n_5NO + n_5NO_2$

a) Calculate equilibrium of gases when outside temperature is T=1200 K
b) Calculate equilibrium of gases when outside temperature is T=2000 K
c) Calculate equilibrium of gases when outside temperature is T=3000 K
d) Calculate equilibrium of gases when outside temperature is at adiabatic flame temperature (Q=0)

You can use gibbs.java, gibstest2.java, gibstest3.java, gibsTableEN.java. apply Gibbs free energy minimisation method

PROBLEM 2: Repeat the same problem, but this time use equilibrium equations

$H_4 + 2.2O_2 + 8.72619N_2 \rightarrow n_1CH_4 + n_2O_2 + n_3N_2 + n_3CO_2 + n_3CO + n_4N_2O + n_5NO + n_5NO_2$

$CO + 0.5O_2 = CO_2 \quad 0.5N_2 + 0.5O_2 = NO \quad N_2 + 0.5O_2 = N_2O \quad 0.5N_2 + O_2 = NO_2$

a) Calculate equilibrium of gases when outside temperature is T=1200 K
b) Calculate equilibrium of gases when outside temperature is T=2000 K
c) Calculate equilibrium of gases when outside temperature is T=3000 K

6.4 Chemical Reaction Kinetics

In the previous subchapter, it is explained how the equilibrium composition of a reactive gas mixture can be predicted by the application of thermodynamics. But the reactions are also time dependent processes and equilibrium describes the potential of the reaction, but it does

not give the rate at which the reaction proceeds, or how quickly equilibrium is approached. If a combustion or reaction vessel to be designed. Reaction rate is also an important parameter. Chemical reactions may ocur when two or more molecules or atoms approach one another. The type of reaction may ocur depends on intermolecular forces existed during a collisional encounter. A reaction which takes place by such a collusion process is called an elementary reaction. As en end result of many elementary reaction a global reaction occurs. For example if a global reaction is

$CO + 1/2O_2 \rightarrow CO_2$ Actual elementary reactions might be:

$CO + O_2 \rightarrow CO_2 + O$ and $CO + O + M \rightarrow CO_2 + M$ where M is a third body such as N_2 or O_2

For a methane reaction

$CH_4 + 2O_2 \rightarrow CO_2 + H_2O$

Actual elementary reactions might be:

$CH_4 + O_2 \rightarrow CH_3 + HO_2$ or $NO + M \rightarrow N + O + M$ or $CH_4 + O \rightarrow CH_3 + OH$

For a set of elementary reactions

$AB + C \rightarrow BC + A$ reaction (1) (6.3.12)

$A + B + M \rightarrow AB + M$ reaction (2) (6.3.13)

$AB + M \rightarrow A + B + M$ reaction (3) (6.3.14)

The reaction rate is depends on the rate constant times concentration of each of the reactants

$$-\frac{d[AB]}{dt} = -\frac{d[C]}{dt} = \frac{d[BC]}{dt} = \frac{d[A]}{dt} = k_1[AB][C] \quad \text{reaction (1)} \quad (6.3.15)$$

$$-\frac{d[A]}{dt} = -\frac{d[B]}{dt} = \frac{d[AB]}{dt} = \frac{d[A]}{dt} = k_2[A][B][M] \quad \text{reaction (2)} \quad (6.3.16)$$

$$-\frac{d[AB]}{dt} - \frac{d[A]}{dt} = \frac{d[B]}{dt} = \frac{d[A]}{dt} = k_3[AB][M] \quad \text{reaction (3)} \quad (6.3.17)$$

Where the factors k_1, k_2, k_3 are the reaction rate constants for the respective reactions. In General reaction constants can be calculated by using Arhenius equation

$$k = AT^n p^m \exp\left(-\frac{E}{RT}\right) = k_0 \exp\left(-\frac{E}{RT}\right)$$

Where E is the activation energy required to bring the ractants to a reactive state. A,n and m are constants. İf for a given reaction n=m=0 then $k_0 = AT^n p^m$ term can be taken as a constant as well. Usually these coefficients can be obtain from experimental data taken from actual reactions takes place.

For a reaction of the generic form:

$aA + bB \rightarrow cC + dD$ (6.3.18) where a,b,c,d are stoichiometric coefficients, the rate of destruction of A and B and the rate of formation C and D are given by:

$$-\frac{d[A]}{dt} = -ak_f[A]^a[B]^b \quad (6.3.19)$$

$$-\frac{d[B]}{dt} = -bk_f[A]^a[B]^b \quad (6.3.20)$$

$$\frac{d[C]}{dt} = ck_f[A]^a[B]^b \quad (6.3.21)$$

$$\frac{d[D]}{dt} = dk_f[A]^a[B]^b \quad (6.3.22)$$

Elementary reactions are reversible, so that the back reaction is

$cC + dD \rightarrow aA + bB$ (6.3.23)

$$\frac{d[A]}{dt} = ak_b[C]^c[B]^d \quad (6.3.24)$$

Combining backward and forward reactions, the net reaction rate of A is

$$\frac{d[A]}{dt} = a\left(k_b[C]^c[B]^d - k_f[A]^a[B]^b\right) \quad (6.3.25)$$

At equilibrium state $\frac{d[A]}{dt} = 0$, and (6.3.26)

$$K_c = \frac{k_f}{k_b} = \frac{[C]^c[B]^d}{[A]^a[B]^b} \quad (6.3.27)$$

K_c is called concentration equilibrium constant and related to equilibrium constant defined in the previous chapter as

$K_c = K(RT)^{a+b-c-d}$ (6.3.28) so if one side of the reaction constants are known, it is possible to obtain reaction constants for the reverse direction by using equilibrium constant.

Now we can look at some actual combustion reaction kinetics A global reaction scheme for propane and other hydrocarbons(C_nH_{2n+2}) combustion as[35] :

$$C_nH_{2n+2} \rightarrow \left(\frac{n}{2}\right)C_2H_4 + H_2 \quad (6.3.29)$$

$$C_2H_4 + O_2 \rightarrow 2CO + 2H_2 \quad (6.3.30)$$

$$CO + \frac{1}{2}O_2 \rightarrow CO_2 \quad (6.3.31)$$

$$H_2 + \frac{1}{2}O_2 \rightarrow H_2O \quad (6.3.32)$$

Global rate equations in units of cm^3,gmol,s , cal and K have been developed for each reaction species. R=82.05 cm^3atm/K-gmol

$$\frac{d[C_nH_{2n+2}]}{dt} = -10^{17.32}\exp\left(\frac{-49600}{RT}\right)[C_nH_{2n+2}]^{0.5}[O_2]^{1.07}[C_2H_4]^{0.4} \quad (6.3.33)$$

$$\frac{d[C_2H_4]}{dt} = -10^{14.70}\exp\left(\frac{-50000}{RT}\right)[C_2H_4]^{0.9}[O_2]^{1.18}[C_nH_{2n+2}]^{-0.37} \quad (6.3.34)$$

$$\frac{d[CO]}{dt} = -10^{14.60}\exp\left(\frac{-40000}{RT}\right)[CO]^{1.0}[O_2]^{0.25}[H_2O]^{0.5} + 5*10^{8}\exp\left(\frac{-40000}{RT}\right)[CO_2]^{1.0}$$

(6.3.35)

$$\frac{d[H_2]}{dt} = -10^{13.52}\exp\left(\frac{-41000}{RT}\right)[H_2]^{0.85}[O_2]^{1.42}[C_2H_4]^{-0.56} \quad (6.3.36)$$

Since this equations are experimental curve fit equations, limits of equations should be considered in use. This equation is valid for pressure 1 to 9 atm and temperatures 960 to 1540 K, and equivalance ratio F 0.12 to 2

For simpler analysis a single step equation is given in the same source, fort he reaction

$Fuel + \alpha O_2 \rightarrow \beta CO_2 + \gamma H_2O$ (6.3.37) where α,β,γ are stoichiometric coefficients. The associated global kinetic rate equation is:

$$\frac{d[fuel]}{dt} = -AT^n p^m \exp\left(\frac{-E}{RT}\right)[fuel]^a[O_2]^b \quad (6.3.38)$$ for most of the constants n=m=0

and E , a and b coefficients for some common hydrocarbons are given in the table below. A better accuracy can be obtained by using a two step equation as

$Fuel + \alpha O_2 \rightarrow \beta CO_2 + \gamma H_2O$ (6.3.39)

$$CO + \frac{1}{2}O_2 \rightarrow CO_2 \quad (6.3.40)$$

Table is listed coefficients for both cases of equations

Table 6.4.1 Global one and two step kinetic reaction constants[

Fuel	A(one step)	A(two step)	E(kcal/gmol)	a	b
CH_4	1.3e6	2.8e9	48.4	-0.3	1.3
C_2H_6	1.1e12	1.3e12	30	0.1	1.65
C_3H_8	8.6e11	1.0e12	30	0.1	1.65
C_4H_{10}	7.4e11	8.8e11	30	0.15	1.6
C_5H_{12}	6.4e11	7.8e11	30	0.25	1.5
C_6H_{14}	5.7e11	7.0e11	30	0.25	1.5
C_7H_{16}	5.1e11	6.3e11	30	0.25	1.5
C_8H_{18}	4.6e11	5.7e11	30	0.25	1.5
C_9H_{20}	4.2e11	5.2e11	30	0.25	1.5
$C_{10}H_{22}$	3.8e11	4.7e11	30	0.25	1.5
CH_3OH	3.2e11	3.7e12	30	0.25	1.5
C_2H_5OH	1.5e11	1.8e12	30	0.25	1.6
C_6H_6	2.0e11	2.4e11	30	-0.1	1.85
C_2H_6	1.6e11	1.9e11	30	-0.1	1.85

By using these data chemical kinetics of the fuels can be calculated. Note that equations are differential equations, so they should be solved by using numerical differential equation solving methods. In this particular example &th order Runge-Kutta method is used.

Program 6.4.1 Chemical kinetics of C3H8 combustion (two step equation)

```
//=======================================================
// 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 T=1500.0;//degree K;
double nfi;
public  fm1(double nfi1,double Ti)
{nfi=nfi1;T=Ti;}

//multivariable function
 public double func(double x[],int x_ref)
 {
  //x[0] is x, x[1] is nfuel  x[2] is nCO2
  double y=1.0;
  double nCO=3.0*(nfi-x[1])-x[2];
  double nO2=(3.5*x[1])+1.5*nfi-0.5*x[2];
  double nH2O=4.0*(nfi-x[1]);
  double nCO2f=3.0*nfi;
  if(x_ref==0) {y=-1.0e12*Math.exp(-
30000.0/(1.987*T))*Math.pow(x[1],0.1)*Math.pow(nO2,1.65);return y;}
  else if(x_ref==1) {y=pow(10,14.6)*exp(-40000.0/(1.987*T))*nCO*pow(nO2,0.25)*pow(nH2O,0.5)-
  5.0e8*exp(-40000.0/(1.987*T))*x[2];return y;}
  else return 0;
 }}
```

```
class diferansiyel_kinetics
{
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];
  }
```

```
 }

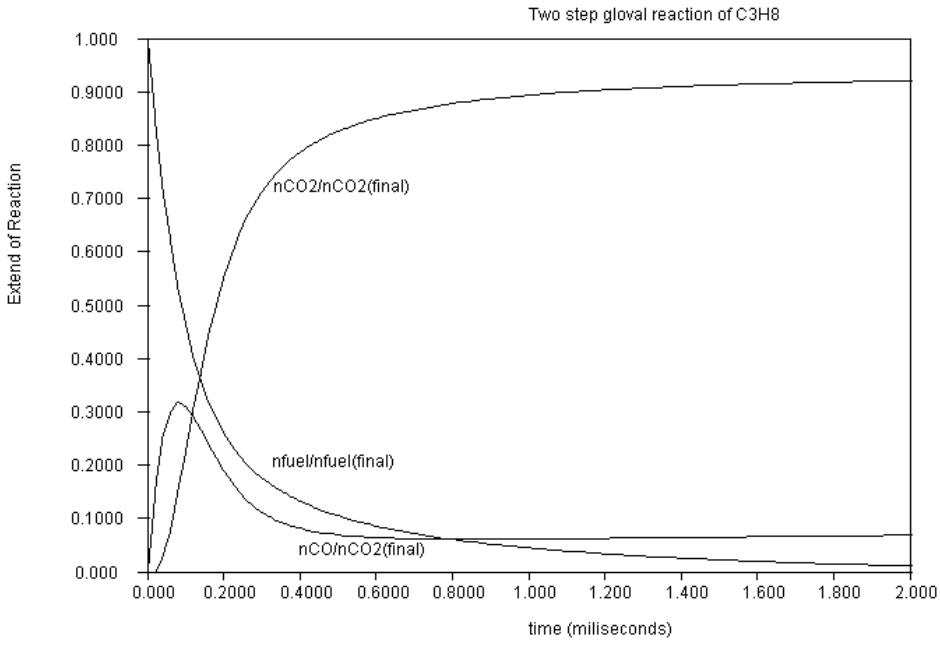
return a;
 }


  public static void main(String args[]) throws IOException
  {
  double x[];
  x=new double[1];
  double p=1.0; //atm;
  double R=82.05; // (cm^3 atm/K/gmol)
  double T=1500.0;
  double n=p/(R*T);

  double y[]=new double[2];
  y[0]=1./24.8*n;//initial amount of fuel
  y[1]=0.0;      //initial amount of CO2
  double nCO=0;
  double nCO2f=3.0*y[0];
  fm1 b3=new fm1(y[0],T);
  int N=100;
  double a[][]=RK6(b3,0.0,2.0e-3,y,N);
  double z[][]=new double[4][a[0].length];
  for(int i=0;i<a[0].length;i++)
  {z[0][i]=a[0][i]*1e3;//milisecond
   z[1][i]=a[1][i]/y[0];
   z[2][i]=a[2][i]/nCO2f;
   nCO=3.0*(y[0]-a[1][i])-a[2][i];
   z[3][i]=nCO/nCO2f;
  }
  String s="C3H8 reaction";
  String s1[]={"time miliseconds","nfuel/nfuel(initial)","nCO2/nCO2(final)","nCO/nCO2(final)"};
  Text.printT(z,s1,s);
  Plot pp=new Plot(z[0],z[1]);
  pp.addData(z[0],z[2]);
  pp.addData(z[0],z[3]);
  pp.plot();
 }
}
```

C3H8 reaction

time miliseconds	nfuel/nfuel(initial)	nCO2/nCO2(final)	nCO/nCO2(final)
0.0	1.0	0.0	0.0
0.02	0.8499213276548222	0.0	0.1500786723451778
0.04	0.7260237235497475	0.0248162771714069	0.24915999927884558
0.06000000000000001	0.622265416984699	0.07709127890116606	0.300643304114135
0.08	0.5357167789379452	0.14724177494564047	0.31704144611641427
0.1	0.46424748941134525	0.22519132401786732	0.3105611865707874
0.12000000000000001	0.40567089452061017	0.30325973457374605	0.2910693709056438
0.14	0.35780185389565666	0.3766289197674227	0.26556922633692065
0.16	0.3186197501910662	0.4428378476043715	0.23854240220456233
0.18000000000000002	0.2863764268264579	0.5010463705984641	0.21257720257507798
0.2	0.25963000876311904	0.5513888529867126	0.18898113825016835
0.22	0.2372282609115129	0.5945081763510455	0.1682635627374417
0.24000000000000002	0.21826801714700075	0.6312580442830378	0.15047393856996155
0.26	0.20204860614866055	0.662530437902172	0.1354209559491674
0.28	0.18802855895134887	0.689165543711067	0.12280589733758412
0.30000000000000004	0.17578916990873708	0.7119117057075685	0.11229912438369449
0.32	0.1650054419052346	0.7314136320633756	0.10358092603138962
0.34	0.15542362281954383	0.7482153355491645	0.09636104163129158
0.36000000000000004	0.14684413530805207	0.7627698803052391	0.09038598438670879
0.38	0.13910871895136454	0.7754515275648937	0.08543975348374176
0.4	0.13209077777289566	0.7865679786159747	0.08134124361112971
0.42000000000000004	0.12568813223201236	0.7963716224425613	0.07794024532542619
0.44	0.1198175622031585	0.8050693622713343	0.07511307552550725
0.46	0.11441068073948812	0.8128309449453257	0.07275837431518623
0.48000000000000004	0.10941079727348914	0.8197958868004008	0.07079331592611009
0.5	0.1047705184017614	0.8260791605267525	0.06915032107148603
0.52	0.10044990070245546	0.8317758260819303	0.06777427321561412
0.5400000000000001	0.09641501873595083	0.8369647814454706	0.06662019981857863
0.5600000000000002	0.09263684702251465	0.8417117903736836	0.06565136260380172
0.5800000000000002	0.08909038086239551	0.8460719220422871	0.06483769709531734
0.6000000000000003	0.08575393996335737	0.8500915154459475	0.06415454459069507
0.6200000000000003	0.08260861287300228	0.8538097614384377	0.0635816256885601
0.6400000000000003	0.07963781056185192	0.857259977993613	0.06310221144453514
0.6600000000000005	0.07682690516860113	0.8604706397132466	0.06270245511815235
0.6800000000000005	0.07416293562401223	0.8634662105950897	0.062370853780898135
0.7000000000000005	0.07163436613724299	0.8662678192856574	0.06209781457709953
0.7200000000000006	0.0692308867368103	0.8688938081360214	0.06187530512716839
0.7400000000000007	0.06694324748332994	0.8713601810328305	0.06169657148383963
0.7600000000000007	0.06476311981369515	0.8736809699050839	0.061555910281220864
0.7800000000000007	0.0626829798838221	0.8758685357659739	0.06144848435020403
0.8000000000000008	0.06069600985798753	0.8779338169352869	0.06137017320672554
0.8200000000000008	0.05879601392733913	0.8798865345349158	0.061317451537745024
0.8400000000000009	0.0569773464879761	0.8817353633231539	0.06128729018887
0.860000000000001	0.05523485041456008	0.8834880743241921	0.061277075261247926
0.880000000000001	0.053563803762078026	0.8851516544310472	0.06128454180687475
0.900000000000001	0.05195987354126111	0.8867324071440783	0.06130771931466057
0.9200000000000012	0.05041907546127787	0.8882360377986536	0.06134488674006863
0.9400000000000012	0.04893773873111448	0.8896677259911553	0.061394535277730304
0.9600000000000012	0.047512475169565145	0.8910321873981516	0.06145533743228328
0.9800000000000013	0.0461401520014439	0.8923337267722061	0.06152612122634995



6.5 Power cycles with actual combustion process

6.5.1 Brayton (Gas turbine) cycle

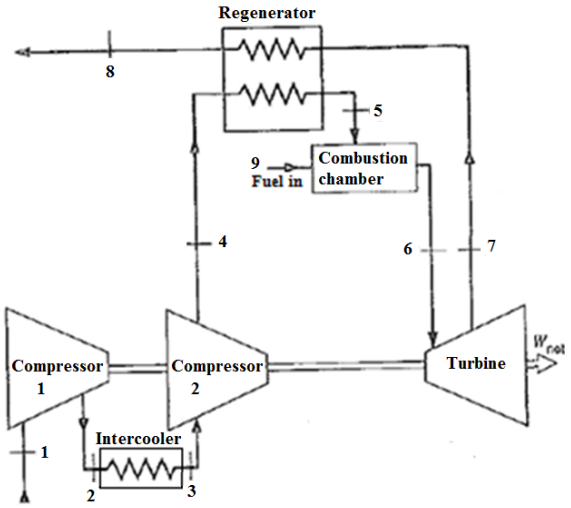


Figure 6.5.1.1 Brayton cycle given in the example problem

In actual gas turbine cycle, air combust with a fuel to generate the required heat for the cycle, exit of combustion chamber is combustion products. Therefore in the remaining process , combustion products should be taken into account as working fluid of the cycle. Since combustion products are a mixture, in order to evaluate gas properties. Gmix (gas mixture) class is used. In order to simulate combustion process, combustion class is used. The basic cycle equations are as follows:

$$m = m_{air} \quad (6.5.1)$$

$$W_{low\ pressure\ compressor} = m(h_2 - h_1) \quad (6.5.2)$$

$$W_{low\ pressure isentropic compressor} = m(h_{2i} - h_1) \qquad s_{2i} = s_1 \quad (6.5.3)$$

$$W_{high\ pressure\ compressor} = m(h_4 - h_3) \quad (6.5.4)$$

$$W_{high\ pressure isentropic compressor} = m(h_{4i} - h_3) \qquad s_{4i} = s_3 \quad (6.5.5)$$

$$\eta_{isentropic compressor} = \frac{W_{isentropic compressor}}{W_{compressor}} \quad (6.5.6)$$

$$Q_{intercooler} = m(h_2 - h_3) \quad (6.5.7)$$

$$Q_{regenerator} = m(h_5 - h_4) = m_{product}(h_9 - h_{10}) \quad (6.5.8)$$

$$Q_{combustion\ chamber} = m_{product}h_6 - \left(mh_5 + m_{fuel}h_{fuel}\right) \qquad P_6 = P_5 = P_4 \quad (6.5.9)$$

$$W_{turbine} = m_{product}(h_6 - h_7) \quad (6.5.10)$$

$$W_{isentropic\ turbine} = m_{product}(h_6 - h_{7i}) \quad (6.5.11)$$

$$\eta_{isentropic turbine} = \frac{W_{turbine}}{W_{isentropic turbine}} \quad (6.5.12)$$

$$\eta_{cycle} = \frac{W_{turbine} - W_{compressor}}{Q_{combustionchamber}} \quad (6.5.13)$$

Program 6.5.1.1 Brayton cycle with combustion in combustion chamber

```
 public class brayton4
{
Gas g;
double PP1[],vv1[];
double PP2[],vv2[];
double TT1[],ss1[];
double TT2[],ss2[];
double
Wturbine,Wcompressor1,Wcompressor2,Wcompressor,Wnet,Q1,Qregenerator,Qintercooler,Qnet,eta;
double T[],P[],h[],s[];
double m; // air mass flow rate;
combustion cc;
double a_f; //air to fuel ratio
double mair;
double mratio;
double mproduct; //proct mass flow rate kg/s
double mfuel;
double ds5p; //entropy difference of two air and product gases
double dh5p; //enthalpy difference of two air and product gases
public brayton4(String fueli[],double Nfueli[],double m,double T1,double P1,double Tfuel,double
P2,double P4,double T3,double T6,double dT,double etacompressor1,double etacompressor2,double
etaturbine1,double etaturbine2)
{
  g=new Gas("air");
  g.base("mass");
  T=new double[11];
  P=new double[11];
  h=new double[11];
  s=new double[11];
  TT1=new double[25];ss1=new double[25];
  TT2=new double[2];ss2=new double[2];
// 1 compressor 1 inlet
P[1]=P1;T[1]=T1;
P[2]=P2;
P[3]=P2;T[3]=T3;
P[4]=P4;
P[5]=P4;
P[6]=P4;T[6]=T6;
P[7]=P1;
P[8]=P1;
// 1 inlet
double a[]=g.property(T[1],P[1]);
h[1]=a[3];
s[1]=a[5];
TT1[0]=T[1];ss1[0]=s[1];
double pr1=a[17];
// 2i = 11 isentropic outlet of compressor 1
double pr2=P2/P1*pr1;
double T2i=g.Ti(pr2);
a=g.property(T2i,P2);
double h2i=a[3];
double s2i=s[1];//a[5];
double Cp2i=a[9];
```

```
T[9]=T2i;
s[9]=s2i;
TT1[1]=T2i;ss1[1]=s2i;
// 2 outlet of compressor 1
h[2]=(h2i-h[1])/etacompressor1+h[1];
T[2]=(h[2]-h2i)/Cp2i+T2i;
a=g.property(T[2],P[2]);
s[2]=a[5];
// 3 compressor 2 inlet
a=g.property(T[3],P[3]);
h[3]=a[3];
s[3]=a[5];
double pr3=a[17];
// 4i = 12 isentropic process compressor 2 outlet
double pr4=P[4]/P[3]*pr3;
double T4i=g.Ti(pr4);
a=g.property(T4i,P[4]);
double h4i=a[3];
double s4i=a[5];
double Cp4i=a[9];
T[10]=T4i;
s[10]=s4i;
//4 outlet of compressor 2
h[4]=(h4i-h[3])/etacompressor2+h[3];
T[4]=(h[4]-h4i)/Cp4i+T4i;
a=g.property(T[4],P[4]);
s[4]=a[5];
double Cp4=a[9];
//6 combustion chamber outlet
  a_f=1.0;
  cc=new combustion("combustion chamber",fueli,Nfueli,a_f);
  a_f=cc.adiabatic_a_f_ratio(T1,Tfuel,T6,1.0,10.0);
  mair=cc.nair/28.97;//mass flow rate of air from reaction
  mratio=m/mair;//air mass flow rate correction ratio compare to input air flow rate
  mproduct=(cc.productmix.N/cc.productmix.M)*mratio; //proct mass flow rate kg/s
  mfuel=cc.fuelmix.N/cc.fuelmix.M*mratio;//fuel mass flow rate kg/s
  cc.fuelmix.base("mass");
  cc.productmix.base("mass");//productmix calculations will be mass bases kJ/kg..
  a=cc.productmix.property(T[6],P[6]);
  h[6]=a[3];
  s[6]=a[5];
  double Cp6=a[9];
  double pr6=a[17];
   a=g.property(T[6],P[6]);
  //fuel inlet enthalpy
  a=cc.fuelmix.property(Tfuel,P[6]);
  double hfuel=a[3];
  //7i isentropic turbine outlet
  double pr7=P[7]/P[6]*pr6;
  double T7i=cc.productmix.Ti(pr7);
  a=cc.productmix.property(T7i,P[7]);
  double h7i=a[3];
  double s7i=a[5];
  double Cp7i=a[9];
  T[0]=T7i;
  s[0]=s7i;
  //7 turbine outlet
  h[7]=h[6]-etaturbine1*(h[6]-h7i);
```

```
  T[7]=(h[7]-h7i)/Cp7i+T7i;
  a=cc.productmix.property(T[7],P[7]);
  s[7]=a[5];
  double Cp7=a[9]/cc.productmix.M;
  //8 heat exchanger outlet
  T[8]=T[4]+dT;
  a=cc.productmix.property(T[8],P[8]);
  h[8]=a[3];
  s[8]=a[5];
  double Cp8=a[8]/cc.productmix.M;
  //5 heat exchanger exit
  Qregenerator=mproduct*(h[7]-h[8]);
  T[5]=T[8]+Qregenerator/mproduct/Cp8;
  a=g.property(T[5],P[5]);
  h[5]=a[3];
  s[5]=a[5];
  //5' heat exchanger exit calculated as product gases
  //normalisation of two gas data
  a=cc.productmix.property(T[5],P[5]);
  double h5p=a[3];
  double s5p=a[5];
  ds5p=s5p-s[5];
  //System.out.println("ds5p="+ds5p);
  s[6]-=ds5p;
  s[7]-=ds5p;
  s[8]-=ds5p;
  s[0]-=ds5p;
Wturbine=mproduct*(h[6]-h[7]);
Wcompressor1=m*(h[2]-h[1]);
Wcompressor2=m*(h[4]-h[3]);
Wcompressor=Wcompressor1+Wcompressor2;
Q1=mproduct*h[6]-m*h[5]-mfuel*hfuel;
double Qregenerator1=m*(h[5]-h[4]);
Qintercooler=m*(h[2]-h[3]);
Qnet=Q1;
Wnet=Wturbine-Wcompressor;
eta=Wnet/Qnet;
}
public double[][] line(double T1,double T2,double Pi,int n)
{
double TT1[]=new double[n];
double ss1[]=new double[n];
double aa[];
for(int i=0;i<n;i++)
{TT1[i]=T1+i*(T2-T1)/(n-1);
 aa=g.property(TT1[i],Pi);
 ss1[i]=aa[5];
}
double a[][]={TT1,ss1};
return a;
}
public double[][] line1(double T1,double T2,double Pi,int n)
{
double TT1[]=new double[n];
double ss1[]=new double[n];
double aa[];
for(int i=0;i<n;i++)
{TT1[i]=T1+i*(T2-T1)/(n-1);
```

```
 aa=cc.productmix.property(TT1[i],Pi);
 ss1[i]=aa[5]-ds5p;
}
double a[][]={TT1,ss1};
return a;
}
public void plot()
{double a[][]=new double[2][61];
 double c1[][]=new double[2][3];
 double c3[][]=new double[2][3];
 double c6[][]=new double[2][3];
 c1[0][0]=T[1];
 c1[1][0]=s[1];
 c1[0][1]=T[9];
 c1[1][1]=s[9];
 c1[0][2]=T[2];
 c1[1][2]=s[2];

 c3[0][0]=T[3];
 c3[1][0]=s[3];
 c3[0][1]=T[10];
 c3[1][1]=s[10];
 c3[0][2]=T[4];
 c3[1][2]=s[4];

 c6[0][0]=T[6];
 c6[1][0]=s[6];
 c6[0][1]=T[0];
 c6[1][1]=s[0];
 c6[0][2]=T[7];
 c6[1][2]=s[7];

 int i=0;
 a[0][0]=T[1];
 a[1][0]=s[1];
 double b[][]=line(T[2],T[3],P[2],15);
 for(i=0;i<15;i++)
 {a[0][i+1]=b[0][i];
  a[1][i+1]=b[1][i];
 }
 b=line(T[4],T[5],P[4],15);
 for(i=0;i<15;i++)
 {a[0][i+16]=b[0][i];
  a[1][i+16]=b[1][i];
 }
 b=line1(T[5],T[6],P[4],15);
 for(i=0;i<15;i++)
 {a[0][i+31]=b[0][i];
  a[1][i+31]=b[1][i];
 }
 b=line1(T[7],T[8],P[7],15);
 for(i=0;i<15;i++)
 {a[0][i+46]=b[0][i];
  a[1][i+46]=b[1][i];
 }

 Plot p=new Plot(a[1],a[0]);
 p.addData(c1[1],c1[0],1);
```

```
 p.addData(c3[1],c3[0],1);
 p.addData(c6[1],c6[0],1);
 p.addData(s,T,22);
 p.setPlabel("Brayton (Gas turbine) cycle with combustion process");
 p.setXlabel("s entropy kJ/(kgK)") ;
 p.setYlabel("T degree C");
 p.plot();
 }

 public String toString()
 {
 String ss=" Brayton cycle \n";
 ss+="Combustion Reaction : \n";
 ss+=cc.toString()+"\n";
 ss+="mair ="+m+"kg/s mproduct="+mproduct+" kg/s mfuel\n";
 ss+="Turbine power output W turbine = "+Wturbine+" kW \n";
 ss+="Low pressure compressor power input W turbine = "+Wcompressor1+" kW \n";
 ss+="High pressure compressor power input W turbine = "+Wcompressor2+" kW \n";
 ss+="Compressor power input W turbine = "+Wcompressor+" kW \n";
 ss+="Net power output  Wnet = "+Wnet+" kW\n";
 ss+="Combustion chamber heat input Q combustion = "+Qnet+" kW\n";
 ss+="Cycle efficiency eta  = "+eta+"\n";
 ss+="Equivalent temperature carnot cycle efficiency eta  = "+(1.0-T[1]/T[6])+"\n";
 ss+="Regenerator heat transfer   = "+Qregenerator+" kW\n";
 ss+="Termodynamic states : \n";
 for(int i=1;i<9;i++)
 {ss+="P["+i+"] = "+P[i]+"bar T["+i+"] = "+(T[i]-273.15)+" degree C h["+i+"] = "+h[i]+" kJ/kg s["+i+"]
 = "+s[i]+" kJ/kgK\n";}
 return ss;
 }
 }
```

In the example program Natural gas with the following gas combination is taken

Gas name	**amount**
CH_4 methane	**0.906000**
C_2H_6 ethane	0.056000
C_3H_8 propane	0.000800
C_4H_{10} butane	0.000200
CO_2 carbondioxide	0.017000
N_2 nitrogen	0.011000

Air and gas input is 300 K. There is no heat transfer in combustion chamber, so the product will be in adiabatic flame temperature. Of course in order to obtain required turbine inlet temperature, amount of fuel should be adjusted. It is simulated by using an air to fuel ratio adjusted adiabatic flame temperature calculation method in combustion (Curve fitting by bisection method). Once amount of required fuel is calculated, the remaining of cycle can be treated just as normal cycle, with exception of using product gases instead of air.

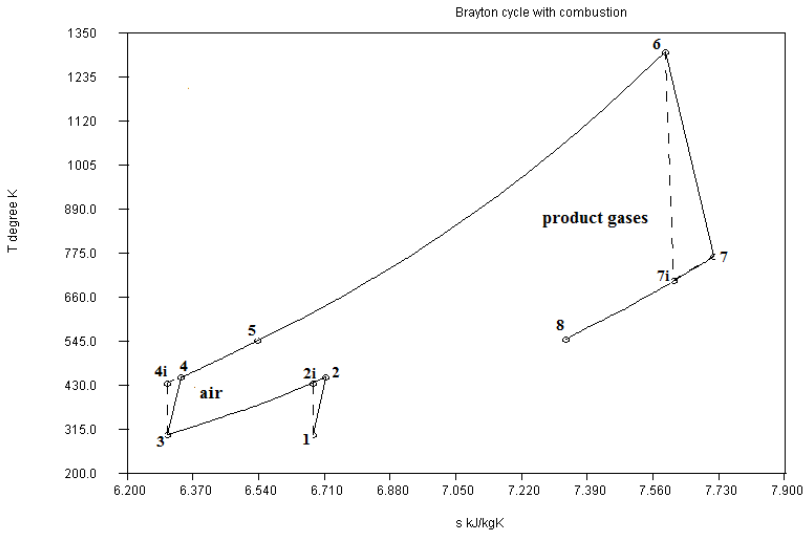
Program 6.5.1.2 Brayton cycle with combustion in combustion chamber test program

```
public class braytontest4
{
 public static void main(String arg[])
 {
```

```
            double mhava=1; // kg/s air inlet temperature
            double T1=300.0; //degree K air inlet temperature
            double T3=300.0; //degree K air inlet temperature
            double P1=1.1;// bar air inlet pressure
            double P2=4.062019202;//bar air inlet pressure
            double P4=15.0;// compressed air pressure
            double T6=1300.0; //degree K turbine inlet temperature
            //natural gas fuel
            // ch4 0.906
    // c2h6 0.056
    // c3h8 8e-4
    // c4h10 2e-4
    // co2 0.017
    // n2  0.011
    String s[]={"ch4","c2h6","c3h8","c4h10","co2","n2"};
    double N[]={0.906,0.056,8.0e-4,2.0e-4,0.017,0.011};
    double a_f=1.0;
    double T8=1500.0; //degree K turbine inlet temperature
            double dT=100.0;// degree K heat exchanger temperature difference
            double etakompressor=0.9; //compressor isentropic efficiency
            double etaturbine=0.9; //turbine isentropic efficiency
    brayton4 bry1=new
brayton4(s,N,mhava,T1,P1,T1,P2,P4,T3,T6,dT,etakompressor,etakompressor,etaturbine,etaturbine);
    System.out.println(bry1.toString());
    bry1.plot();
  }
}
```

```
---------- Capture Output ----------
> "C:\java\bin\javaw.exe" braytontest4
Brayton cycle
Combustion Reaction :
[  0.9060 CH4 +   0.0560 C2H6 +   0.0008 C3H8 +   0.0002 C4H10 +   0.0170 CO2 +   0.0110 N2] +
4.9245*[ O2 + (79/21) *N2] →   1.0382 CO2 +   1.9842 H2O +  18.5365 N2 +   2.9112 O2
mair =1.0 kg/s  mproduct=1.0661129068583868 kg/s
Turbine power output W turbine = 717.3634398398311 kW
Low pressure compressor power input W turbine = 151.45004280335826 kW
High pressure compressor power input W turbine = 151.45004282508165 kW
Compressor power input W turbine = 302.9000856284399 kW
Net power output  Wnet = 414.4633542113912 kW
Combustion chamber heat input Q combustion = 1028.3480180935708 kW
Cycle efficiency eta  = 0.403038024986672
Equivalent temperature carnot cycle efficiency eta  = 0.7692307692307692
Regenerator heat transfer   = 265.35145370760745 kW
Termodynamic states :
P[1] = 1.1bar T[1] = 26.850000000000023 degree C h[1] = 300.47241417394997 kJ/kg s[1] =
6.680027277763032 kJ/kgK
P[2] = 4.062019202bar T[2] = 176.76142627605628 degree C h[2] = 451.92245697730823 kJ/kg s[2] =
6.714303344306349 kJ/kgK
P[3] = 4.062019202bar T[3] = 26.850000000000023 degree C h[3] = 300.47241417394997 kJ/kg s[3] =
6.305019006221799 kJ/kgK
P[4] = 15.0bar T[4] = 176.7614262973688 degree C h[4] = 451.9224569990316 kJ/kg s[4] =
6.339295072768477 kJ/kgK
P[5] = 15.0bar T[5] = 271.9600652210031 degree C h[5] = 549.8121169311926 kJ/kg s[5] =
6.536561114130103 kJ/kgK
P[6] = 15.0bar T[6] = 1026.85 degree C h[6] = 1519.7930629146838 kJ/kg s[6] = 7.903886582600744
kJ/kgK
```

P[7] = 1.1bar T[7] = 492.0016930813757 degree C h[7] = 846.9155138061639 kJ/kg s[7] = 8.027493063476864 kJ/kgK
P[8] = 1.1bar T[8] = 276.76142629736876 degree C h[8] = 598.0193115366096 kJ/kg s[8] = 7.645187250991324 kJ/kgK



6.6 Fuel cells systems

Fuel cells are electrochemical energy conversion devices, which directly converts chemical energy of a hydrocarbon into electrical energy and heat energy. The fuel cells are, like batteries, are consist of an anode, a cothode and an eloctrolyte. But unlike batteries fule cells generate electricty continuously as long as fuel an oxidant (usually air) is supplied to the fule cell. There are several different type of electrochemical reactions used in fuel cells, so there are several different fuel cell types according to the main chemical reaction type and system structure to allow this particular reaction. Alkaline(AFC) , Phosphoric acid(PAFC), Polymer electrode membrane(PEM) and solid oxide fuel (SOFC) type are the most common ones. Alkaline fuel cells are used in apollo program to power the systems. More recent two are PEM and SOFC fuel cells. Fuel cells are isothermal devices which operate almost constant temperatures, as much as system design allows it. The basic difference of fuel cell electrochemical reaction to combustion reaction is the generation of electrical energy (Work) out of fuel cell reaction. Work(electricity) generation is function of gibbs free energy generation. Energy balance on the control volume around the fuel cell can be given as $dQ + dW = dh + dKE + dPE$ In operation of fuel cells, kinetic and potential energies can be neglected due to relatively small values compare to chemical energies. For heat transfer of isothermal process $dQ = TdS$ can also be given so $dW = dh - TdS = dG$. In ideal conditions work output of a fuel cell is equal to chemial reaction gibbs free energy change. Ideal efficiency, or effectiveness of a fuel cell can be specified as

$$\eta_i = \frac{\Delta h}{\Delta g}$$

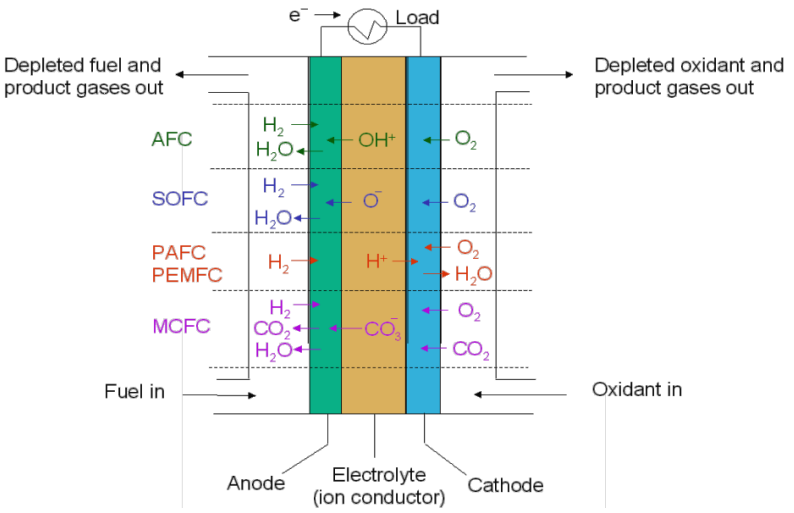


Figure 6.6.1 Operation principles of several type of fuel cells

In additional to ideal efficiency of a fuel cell, another important parameter is the ideal open-circuit voltage developed by the cell, which is $W_{elec} = -N_e F V_i = \Delta g$ where F is called faraday constant and its value is F=96487 kJ/(V kgmole elctrons) and N_e, number of electrons exchanged in reaction, is determined from a knowledge of specific reaction of the fuel cell. For example PEM type of fuel cell reactions:

Anode: $H_2 \rightarrow 2e^- + 2H^+$

Cathode: $2e^- + 2H^+ + 0.5O_2 \rightarrow H_2O$

Total reaction: $H_2 + 0.5O_2 \rightarrow H_2O$

So for this reaction N_e =2 since two electrons are liberatedat the anodefor every molecule of hydrogen consumed. Basic reactions of different type of fuel cells are shown in the figure. Cell voltage in an actual fuel cell is much lower than the ideal open circuit voltage calculated above. When some current started to drain from the circuit coltage drops due irreversibilities of the system. A typical voltage-current density diagram is as shown in the next figure.

Hydrogen is the simplest fuel and most of the fuel cells are used hydrogen as a fuel. Unfortunately, hydrogen is not existed as a free molecule in nature, so required hydrogen or other light hydrocarbons are obtained by converting hydrocarbons to hydrogen by using several different reaction in a chemical reactor called fuel reformer. One type of fuel reformer is called steam reformer. In this type of reformer steam and fuel heated to high temperatures and react with each other. Reaction is endothermic so if it is done adiabatiacly, gases will be cooled down through the reaction. For better fuel conversion efficiencies reactor should be heated up so that reaction could ocur isothermally. In the example program natural gas (assumed as 92 % methane and 8 % ethane) is reacted with steam to carbon ratio of 3.5(3.5 kmol of H2O per kmol of C in fuel). Inlet temperature of reactor was 1100 K and pressure is 2 atm. In adiabatic process temperature dropped to 767 K and 1.42 kmol of hydrogen generated for each kmol of natural gas. In isothermal process temperature is kept constant at 1100 K. In this case 3.54 kmol of hydrogen generated, but for this purpose 229050 kJ/kmol fuel of heat transfer is required to add to the process.

Program 6.6.1 Steam reformer to generate hydrogen from natural gas

```
public class gibbstest2 {
   public static void main(String[] arg)
   {    String s0[]={ "ch4","c2h6","co","co2","h2","h2o"};
```

```
            double[] n0={0.92,0.08,0,0,0,3.5};
                        double Te[]={1100,1100,1100,1100,1100,1100};
                        double T1=1100;//degree K
                        double P1=1.01325*2;//bar
                        int nmax1=500;//iterasyon sayısı
                        System.out.println("---------------------------------");
                        gibbs g1=new gibbs(s0,n0,Te,T1,P1);
                        g1.getEq();
                        for(int i=0;i<g1.n.length;i++) System.out.println(g1.gs[i].Formula+"   "+g1.n[i][0]);

                        System.out.println("Q= "+g1.delQ());
                         System.out.println("---------------------------------");
                        T1=g1.set_Q(s0,n0,Te,0.0,P1);
                        System.out.println("T1="+T1);
                        g1.getEq();
                        for(int i=0;i<g1.n.length;i++) System.out.println(g1.gs[i].Formula+"   "+g1.n[i][0]);
                        System.out.println("Q= "+g1.delQ());
                        System.out.println("---------------------------------");
            }
}
```

```
---------- Capture Output ----------
> "C:\java\bin\java.exe" gibbstest2
---------------------------------
CH4   0.004437243308236965
C2H6   2.2054011986170583E-7
CO   0.678456144854299
CO2   0.39710617075722027
H2   3.5437933381319064
H2O   2.0273315136312515
Q= 229050.03662771435
---------------------------------
T1=766.9857610446043
CH4   0.690073839575674
C2H6   0.0029669275977683863
CO   0.03260244276066198
CO2   0.351389862468137
H2   1.4263337057522907
H2O   2.7646178323030544
Q= -5.688723467756063E-4
---------------------------------

> Terminated with exit code 0.
```

A second way to design fuel reformer is to combusting fuel partially (partial oxidation) with a small amount of oxygen or air to create a gas mixture. Since this reaction is exothermic (gives heat to outside) system will generate the required hydrogen. In the next example a partial oxidation reformer values are given. There are only 0.38 kmol hydrogen generated per kmol of fuel and reaction rised the temperature to 1151 K.

Program 6.6.2 Partial oxidation reformer to generate hydrogen from natural gas

```
public class gibbstest2A {
   public static void main(String[] arg)
   {    String s0[]={ "ch4","c2h6","o2","n2","co","co2","h2","h2o"};
            double[] n0={0.92,0.08,0.1,0.376,0,0,0,0};
                        double Te[]={1100,1100,1100,1100,1100,1100,1100,1100};
```

```
                    double T1=1100;//degree K
                    double P1=1.01325*2;//bar
                    int nmax1=500;//iterasyon sayısı
                    System.out.println("----------------------------------");
                    gibbs g1=new gibbs(s0,n0,Te,T1,P1);
                    g1.getEq();
                    for(int i=0;i<g1.n.length;i++) System.out.println(g1.gs[i].Formula+"   "+g1.n[i][0]);

                    System.out.println("Q= "+g1.delQ());
                     System.out.println("----------------------------------");
                    T1=g1.set_Q(s0,n0,Te,0.0,P1);
                    System.out.println("T1="+T1);
                    g1.getEq();
                    for(int i=0;i<g1.n.length;i++) System.out.println(g1.gs[i].Formula+"   "+g1.n[i][0]);
                    System.out.println("Q= "+g1.delQ());
                    System.out.println("----------------------------------");
            }
}
```

```
---------- Capture Output ----------
> "C:\java\bin\java.exe" gibbstest2A
---------------------------------
CH4   0.7595653769400248
C2H6   0.06026372795107228
O2   1.730587548110011E-219
N2   0.3759999999999998
CO   0.19987468890113855
CO2   3.2478256693280153E-5
H2   0.38001770768126064
H2O   6.035458547586149E-5
Q= -4706.579566238721
---------------------------------
T1=1151.275694976136
CH4   0.7539817568405258
C2H6   0.06302411271183837
O2   4.4745674265526995E-208
N2   0.37600000000000006
CO   0.19996065954635073
CO2   9.358189445199061E-6
H2   0.3829435241086663
H2O   2.062407475861162E-5
Q= 0.001238806500059475
---------------------------------

> Terminated with exit code 0.
```

There are a third type of reformer which combines steam reformer with a partial oxidation reformer. They are called autothermal reformers. In autothermal reformer, heat supply to endothermic steam reaction by exorthermic partial oxidation reaction, so that total system is still adiabatic. As it is seen from the result, hydrogen generation is not as good as isentropic steam reformer, but it is given next best solution to fuel conversion process. There is one catch though in partial oxidation and autothermal reforming carbonmonoxide is generated as well, and this additional gas is poisonous for some type of fuel cells, such as PEM, therefore should be removed by using other chemical processes.

Program 6.6.3 autothermal reformer to generate hydrogen from natural gas

```
public class gibbstest2A {
   public static void main(String[] arg)
   {    String s0[]={ "ch4","c2h6","o2","n2","co","co2","h2","h2o"};
            double[] n0={0.92,0.08,0.452595,0.452595*3.76,0,0,0,3.5};
                        double Te[]={1100,1100,1100,1100,1100,1100,1100,1100};
                        double T1=1100;//degree K
                        double P1=1.01325*2;//bar
                        int nmax1=500;//iterasyon sayısı
                        System.out.println("---------------------------------");
                        gibbs g1=new gibbs(s0,n0,Te,T1,P1);
                        g1.getEq();
                        for(int i=0;i<g1.n.length;i++) System.out.println(g1.gs[i].Formula+"
"+g1.n[i][0]);

                        System.out.println("Q= "+g1.delQ());
                         System.out.println("---------------------------------");
                        T1=g1.set_Q(s0,n0,Te,0.0,P1);
                        System.out.println("T1="+T1);
                        g1.getEq();
                        for(int i=0;i<g1.n.length;i++) System.out.println(g1.gs[i].Formula+"
"+g1.n[i][0]);
                        System.out.println("Q= "+g1.delQ());
                        System.out.println("---------------------------------");
            }
}
```

```
---------- Capture Output ----------
> "C:\java\bin\java.exe" gibbstest2A
---------------------------------
CH4   7.932144841714599E-4
C2H6   8.930137454580886E-9
O2   7.054507293493372E-18
N2   1.7017572000000003
CO   0.5348935920946577
CO2   0.5443131755608981
H2   2.7967434874576873
H2O   2.781670056783569
Q= -0.8948624420445412
---------------------------------
T1=1100.0026848913144
CH4   7.931674024865107E-4
C2H6   8.929121761493364E-9
O2   7.055462587966727E-18
N2   1.7017571999999987
CO   0.5348956722944691
CO2   0.5443111424447892
H2   2.7967415955916763
H2O   2.781672042815991
Q= 0.013248269096948206
---------------------------------

> Terminated with exit code 0.
```

Now system thermodynamic analysis of a fuel cell system can be investigated more closely. In order to do this investigation a real fuel cell system will be considered. We will take a solid oxide fuel cell system as the basic system for the investigation. SOFC fuel cell uses a

solid electrolite of zirconiumoxide(ZrO_2) stabilised by yttrium oxide(Y_2O_3). These metal oxides are usually called as ceramics, therefore SOFC's are also called ceramic fuel cells. Anodes are usually made of nickel doped (for electrical activation) zirconia ceramics, and cathode are made of ceramic structures such as $LaCo_{1-x}Mn_xO_3$, x = 0.2-0.3 or $LaSrMnO_3$. Electrodes can be applied to the surface of ceramic elecrolytes or in more recent SOFC systems elecrolyte and cathode are applied as a paint layer into a thicker anode layer. In SOFC oxygen ion passes through the electrolyte. Since ionised gas is oxygen theroretically any fuel can be used in anod site. In the katod, oxygen ionised by using electrical current coming through the external connection. Oxygen ion is passed through the eleoctrolyte to anot site. $O_2 + 4e^- \rightarrow 2O^{2-}$ In the onot site fuel combines with the oxygen ion g water and electron current.

$2H_2 + 2O^{2-} \rightarrow H_2O + 4e^-$

$2CO + 2O^{2-} \rightarrow 2CO_2 + 4e^-$

Electrons are carried through the external circuit to the katot site.

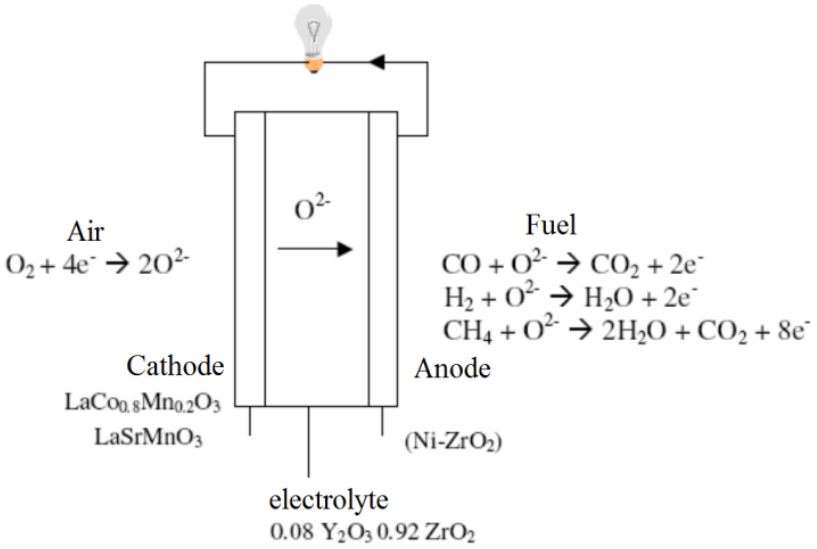


Figure 6.6.2 SOFC fule cell operation

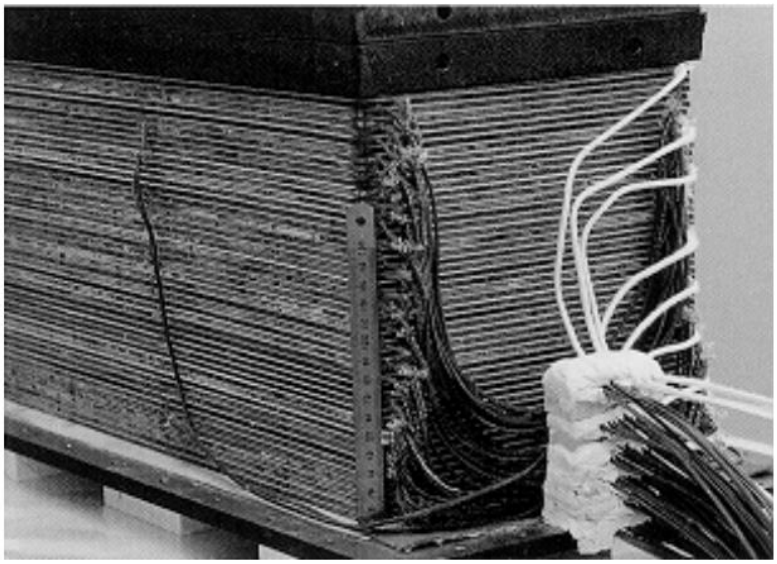


Figure 6.6.3 SOFC Fuel Cell stack

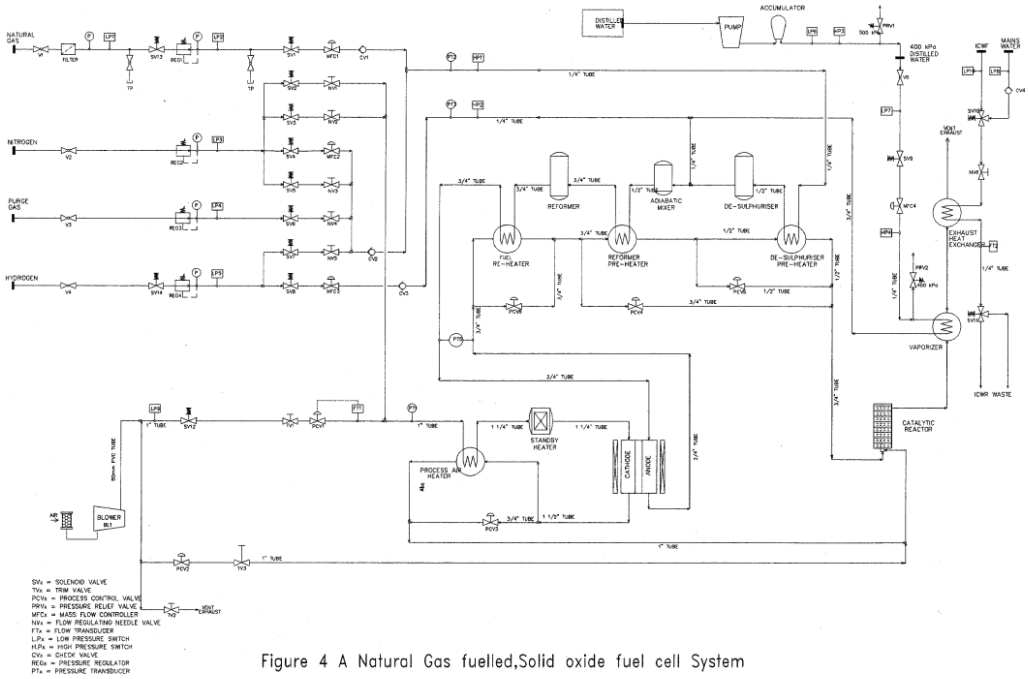


Figure 4 A Natural Gas fuelled,Solid oxide fuel cell System

Figure 6.6.4 Natural gas fuelled solid oxide fuel cell system

A natural gas fuelled solid oxide fuel cell system flow diagram is shown in Figure 4. As seen from the flow diagram, two main gas streams exist in the flow diagram. The first stream is the air stream. The air stream enters to the system through a blower. Then the air goes through a process air heat exchanger. In normal operation conditions, air stream should be heated up about 700°C at the exit of the heat exchanger. At the exit, an electrical heater is located. The main duty of this heater is to supply starting energy and temperatures for the heater. Once operating conditions of the fuel cell is reached, electrical heater is turned off, and fuel cell is used as the main heat source. Gas stream exit the fuel cell at 930°C and input the process air heater as hot gas stream. In this stage, some of the oxygen in the original air is already spent in the fuel cell. Therefore this gas stream is called depleted air stream. When depleted air passed through process air heat exchanger, it is cooled down to about 200°C. Depleted air is then sent to the catalytic reactor. Natural gas stream with main gas pipeline pressure of 40 kPa is used; therefore no additional fan or compressor is needed in fuel
stream. Natural gas is first enters de-sulfuriser preheater. In this heat exchanger, it is heated to 300°C. It is then entered to desulphuriser. Sulphur and sulphur compounds are chemically bonded to de-sulphurization catalysts inside of the de-sulphuriser. Desulphurised natural gas entered to an adiabatic mixer and mixed up with steam and entered to catalytic reformer. Natural gas mix is heated up to 500°C, in another heat exchanger, Then it is entered an adiabatic reformer. In this reformer, high carbon number hydrocarbons are broken to methane and hydrogen by water shift reaction on nickel based catalyst surfaces. This process is called adiabatic fuel reforming. At the exit of the adiabatic reformer, mixture is cooled down to about 300°C because of the energy taken by reforming process. This mixture is went through another heat exchanger and heated to about 700°C before going to the solid oxide fuel cell stack. In the first stage of the stack, the rest of the methane is also converted to CO and hydrogen by internal reforming process. Internal reforming process is similar to reforming process, but instead of catalysts electrochemical reactions forced the mixture into equilibrium condition. Converted fuel then undergoes an electrochemical cell reaction and creates heat and electricity. 85% of the fuel is converted into the cell through electrochemical reaction. The rest of the gas together with combustion gases comes out of the fuel cell at 930°C. This stream, called depleted fuel stream, goes through the three stages of heat exchangers and heating up incoming streams. Relatively cooled depleted fuel finally enters to catalytic reactor and reacts with depleted air and additional fresh air. Catalytic reactor is made of platinum embedded metal surface. Small quantities of

hydrocarbons can react on these surfaces. Reheated final combustion gases pass through a water evaporator to supply system steam requirements for reforming process. A further heat exchanger is taken any heat left in the exhaust stream and cooled exhaust is released to the atmosphere. As another stream water is purified then pressurized through a pump and send to the water evaporator. The fuel cell itself can be compare to the cyclinder of an engine, the system around it makes it a whole power generation system.

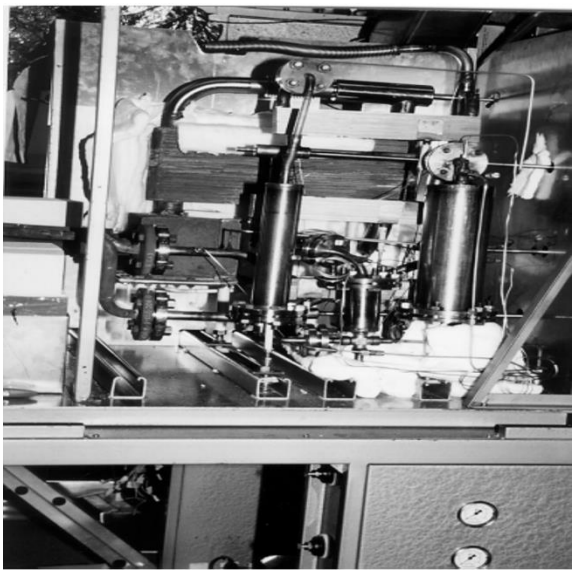


Figure 6.6.5 view of Natural gas fuelled solid oxide fuel cell

7. COMPRESSORS

7.1 Reciprocating ideal gas compressors

A typical one stage reciprocating air compressor shown in the figure. The compressor consist of a piston and a cylinder, induction and delivery valves, water ora ir cooling systems, If it is air cooled, a fin structure is usually added outside of the cylinder. We usually have some additional pressure drop both in induction and delivery of the stream. For ideal case the mass in the cylinder should be emptied completely, but due to some clearance distance left, some air will remain in the cyclinder after the delivery of the pressurised gas. This gas will expand back to low pressure and reduce the total amount of compressor output. Geometry and Pressure-Volume diagram of such a compressor shown in the following figures.

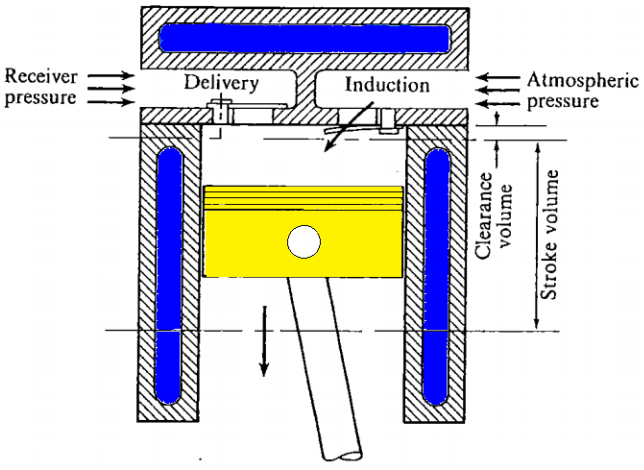


Figure 7.1-1 Reciprocating compressor geometry

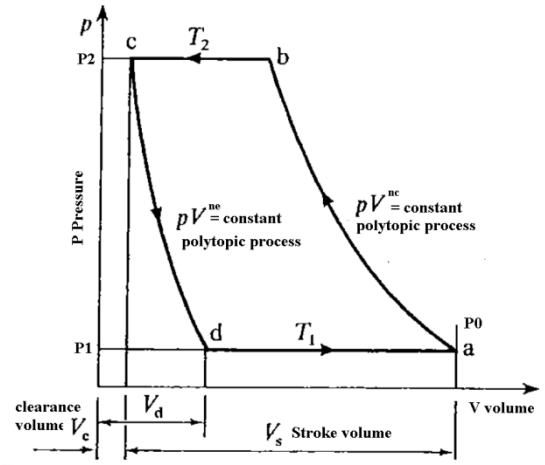


Figure 7.1-2Reciprocating compressor Pressure-Volume diagram

Let us consider now such a compressor, an investigate the basic equations:
Assuming induced mass flow rate of the compressor to be m kg/s, induced volume can be calculated from induced mass flow rate by using thermodynamic realtions of ideal gas

$$P_1 * V_I = R * T_1$$

Induced volume : $V_I = \frac{R * T_1}{P_1} = V_a - V_d$

Stroke volume: $V_s = V_a - V_c$

Clearance volume: $V_c = cvr * V_c$ where cvr is the clearance ratio $cvr = \frac{V_c}{V_c}$

$V_d = V_c \left(\frac{P_2}{P_1} \right)^{1/ne}$ where ne polytropic expansion coefficient

$V_a = (1 + crv) * V_s$

2

$V_b = V_a \left(\frac{P_1}{P_2} \right)^{1/nc}$ where nc polytropic compression coefficient

Polytropic compression work:

$$W_1 = \frac{nc}{nc-1}(P_2 V_b - P_1 V_a) = \frac{nc}{nc-1} m_a R(T_2 - T_1)$$

Polytropic compression work:

$$W_2 = \frac{nc}{nc-1}(P_2 V_b - P_1 V_a) = \frac{nc}{nc-1} m_a R(T_2 - T_1)$$

And

$$W = W_1 - W_2$$

If the compressor were isothermal, it could operate with a minimum power requairement. The required power fort his case would be:

$$W_{isothermal} = (m_a - m_d)RT_1 \ln(\frac{P_2}{P_1}) = mRT_1 \ln(\frac{P_2}{P_1})$$

Isothermal efficiency of the pump is the raio of the isothermal work to actual work required to operate the compressor.

$$\eta_{isothermal} = \frac{W_{isothermal}}{W}$$

Volumetric efficiency is the ratio of induced volume to swept volume

$$\eta_{volumetric} = \frac{V_a - V_d}{V_s}$$

A program is given below to calculate these set of equations

Program 7.1-1 Single stage gas compressor

```
import java.awt.Color;
public class single_stage_gas_compressor
{ public  Gas g;
  public double Vs;// Swept volume m^3/s
  public double Vc;//Clerance volume
  public double VI; //indicated volume : Va-Vd
  public double cvr;// clearance volume ratio;
  public double ne,nc;//expansion and compression polytropic expansion coefficients
  public double m; //mass flow rate kg/s
  public double N; //compressor speed rpm (revolution per minute)
  public double T0,T1,T2; //inlet temperature
  public double P0; //Atmospheric pressure
  public double P1; //inlet pressure
  public double P2; //outlet pressure
  public double Vs_cycle;// swept volume m^3
  public String gas;
  public double Va,Vb,Vd;
  public double va,vb,vc,vd,vs,vi;
  public double W,W1,W2,W3;
  public double ma,md;
  public double etav;
  public double Wisothermal;
  public double etaisothermal;
  public single_stage_gas_compressor(String gasi,double mi,double cvri,double T0i,double P0i,double T1i,double
P1i,double P2i,double nei,double nci,double Ni)
  { m=mi;cvr=cvri;T0=T0i;P0=P0i;T1=T1i;P1=P1i;P2=P2i;g=new Gas(gasi);g.mole=false;
    ne=nei;nc=nci;N=Ni;
    double a[]=g.property(T1,P1);
    VI=m*a[2];
    Vs=VI/((1+cvr)-cvr*Math.pow((P2/P1),(1.0/nc)));
    Vc=cvr*Vs;
    Va=(1.0+cvr)*Vs;
    Vb=Va*Math.pow((P1/P2),(1.0/ne));
    Vd=Vc*Math.pow((P2/P1),(1.0/nc));
    va=a[2];
    ma=Va/va;
    md=ma-m;
    vb=Vb/ma;
    vc=Vc/md;
    vd=Vd/md;
    etav=(Va-Vd)/Vs;
    T2=g.T(P2,vb);
    double Tc=T2;
```

```
  double Td=g.T(P1,vd);
  W1=ne/(ne-1)*ma*(g.R/g.M)*(T2-T1);
  W2=nc/(nc-1)*md*(g.R/g.M)*(Tc-Td);
  W=W1-W2;
  Wisothermal=m*(g.R/g.M)*T1*Math.log(P2/P1);
  Vs_cycle=Vs/(N/60.0);  // m^3/rev
  etaisothermal=Wisothermal/W;
 }
 public double[][] plot_data()
 {double a[][]=new double[2][46];
 a[0][0]=Va;
 a[1][0]=P0;
 a[0][1]=Va;
 a[1][1]=P1;
 double dP=(P2-P1)/21;
 double P=P1;
 double V=Va;
 for(int i=0;i<=20;i++)
 { P+=dP;
  V=Va*Math.pow((P1/P),(1.0/ne));
  a[0][i+2]=V;
  a[1][i+2]=P;
 }
 a[0][23]=Vc;
 a[1][23]=P2;
 P=P2;
 V=Vc;
 for(int i=0;i<=20;i++)
 { P-=dP;
  V=Vc*Math.pow((P2/P),(1.0/ne));
  a[0][i+24]=V;
  a[1][i+24]=P;
 }
 a[0][44]=Vd;
 a[1][44]=P1;
 a[0][45]=Va;
 a[1][45]=P1;
 return a;
 }
 public void plot_Pv()
 {
 double a[][]=plot_data();
 Text.printT(a);
 Plot pp=new Plot(a);
 pp.setPlabel("Single stage reciprocating air compressor");
 pp.setXlabel("V m^3/s");
 pp.setYlabel("P bar");
 pp.setPlotType(0,5);
 pp.setColor(0,Color.blue);
 pp.plot();
 }

 public String toString()
 {String s="work required W = "+W+" kW\n";
  s+="induced mass flow rate m="+m+" kg/s\n";
  s+="compression mass flow rate m="+ma+" kg/s\n";
  s+="expansion mass flow rate m="+md+" kg/s\n";
  s+="polytropic expansion coefficient ne="+ne+"\n";
  s+="polytropic compression coefficient nc="+nc+"\n";
  s+="outside (inlet) gas condition T0 = "+T0+" degree K P0 = "+P0+" bar \n";
  s+="compressor inlet gas condition T1 = "+T1+" degree K P1 = "+P1+" bar \n";
  s+="compressor inlet gas condition T2 = "+T2+" degree K P2 = "+P2+" bar \n";
  s+="Va="+Va+" m^3/s Vb="+Vb+" m^3/s Vc="+Vc+" m^3/s Vd="+Vd+" m^3/s Vs="+Vs+"VI="+VI+" m^3/s \n";
  s+="Vs_cycle = "+Vs_cycle+" m^3\n";
  s+="volumetric efficiency = "+etav+"\n";
  s+="isothermal (minimum possible) work output = "+Wisothermal+"\n";
```

```
   s+="isothermal efficiency = "+etaisothermal+"\n";
   return s;
  }

  public static void main(String arg[])
  { single_stage_gas_compressor ssc=new
single_stage_gas_compressor("air",0.286034892,0.05,288.0,1.013,305.0,0.95,7.0,1.3,1.3,300.0);
    System.out.println(ssc.toString());
    ssc.plot_Pv();
  }
}
```

```
---------- Capture Output ----------
> "C:\java\bin\javaw.exe" single_stage_gas_compressor
work required W = 63.53862446172352 kW
induced mass flow rate m=0.286034892 kg/s
compression mass flow rate m=0.36732610467088783 kg/s
expansion mass flow rate m=0.08129121267088785 kg/s
polytropic expansion coefficient ne=1.3
polytropic compression coefficient nc=1.3
outside (inlet) gas condition T0 = 288.0 degree K P0 = 1.013 bar
compressor inlet gas condition T1 = 305.0 degree K P1 = 0.95 bar
compressor inlet gas condition T2 = 483.5758824133243 degree K P2 = 7.0 bar
Va=0.3385342959534208 m^3/s Vb=0.07284362815538707 m^3/s Vc=0.016120680759686708 m^3/s
Vd=0.07491943289300315 m^3/s Vs=0.3224136151937341VI=0.2636148630604177 m^3/s
Vs_cycle = 0.06448272303874683 m^3
volumetric efficiency = 0.8176294382047575
isothermal (minimum possible) work output = 50.01662022638588 kW
isothermal efficiency = 0.7871844984072095
```

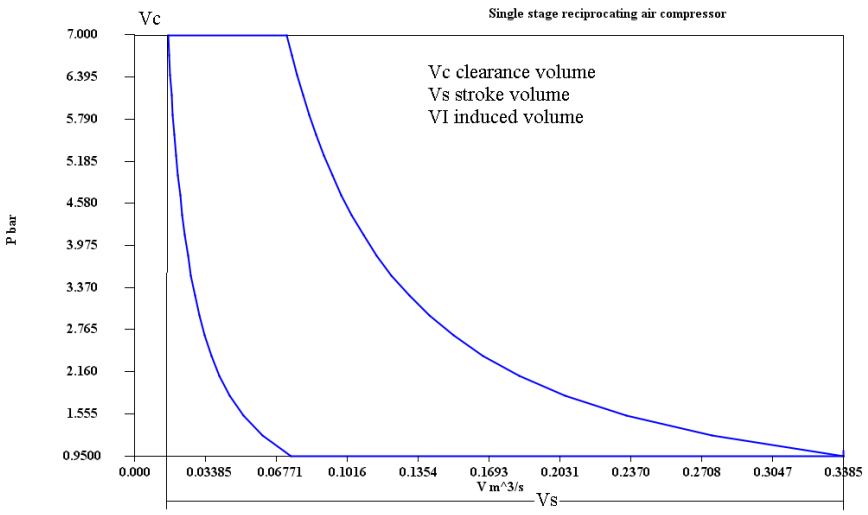


Figure 7.1-3Reciprocating compressor Pressure-Volume diagram of example problem

Isothermal process is ideal process for compression of gases, but even aprroaching isothermal process with only one piston-cylinder system is difficult to achieve, instead of this if more than one stages is used and intercooling is applied between the stages, a better cooling of the gas can be achieved, therefore total process can be closer to the isothermal one. Not that cooling to environmental temperature is limit, but by using cooling towers temperatures close to or even lower than environmental temperatures can be achieved in the second stage inlet.

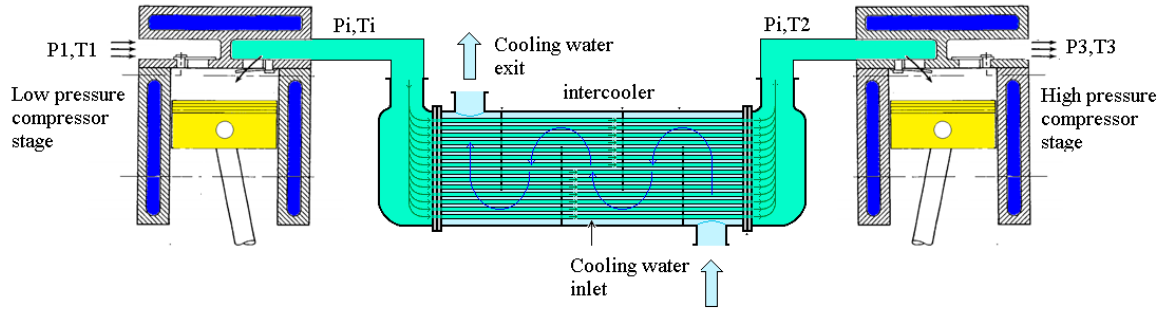


Figure 7.1-4 Two stage Reciprocating compressor with intercooling

In Figure 7.1-5 PV diagram of two stage reciprocating compressor is shown. Crosses area is the work saved due to cooling effect.

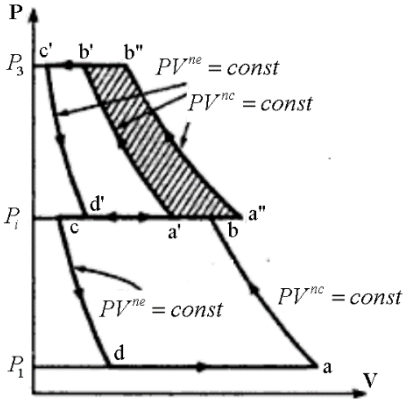


Figure 7.1-5 Two stage reciprocating compressor with intercooling PV

```
import java.awt.Color;
public class two_stage_gas_compressor
{ public Gas g;
  public double Vs1,Vs2;// Swept volume m^3/s
  public double Vc1,Vc2;//Clerance volume
  public double VI1,VI2; //indicated volume : Va-Vd
  public double cvr1,cvr2;// clearance volume ratio;
  public double ne1,nc1,ne2,nc2;//expansion and compression polytropic expansion coefficients
  public double m; //mass flow rate kg/s
  public double N; //compressor speed rpm (revolution per minute)
  public double T0,T1,T2,T3; //inlet temperature
  public double P0; //Atmospheric pressure
  public double P1; //inlet pressure
  public double P2; //outlet pressure first stage, inlet pressure second stage
  public double P3; //outlet pressure second stage
  public double Vs_cycle;// swept volume m^3
  public String gas;
  public double Va1,Vb1,Vd1,Va2,Vb2,Vd2;
  public double W,W1,W2,W3,W4;
  public double ma1,md1,ma2,md2;
  public double etav;
  public double Wisothermal;
  public double etaisothermal;
  public single_stage_gas_compressor ssc[];
  public two_stage_gas_compressor(String gasi,double mi,double cvr1i,double cvr2i,double T0i,double P0i,double
T1i,double P1i,double P2i,double P3i,double ne1i,double nc1i,double ne2i,double nc2i,double Ni)
  { m=mi;cvr1=cvr1i;cvr2=cvr2i;T0=T0i;P0=P0i;T1=T1i;P1=P1i;P2=P2i;P3=P3i;g=new Gas(gasi);g.mole=false;
    ne1=ne1i;nc1=nc1i;ne2=ne2i;nc2=nc2i;N=Ni;
    ssc=new single_stage_gas_compressor[2];
    ssc[0]=new single_stage_gas_compressor(gasi,m,cvr1,T0,P0,T1,P1,P2,ne1,nc1,N);
    double T2=ssc[0].T2;
    ssc[1]=new single_stage_gas_compressor(gasi,m,cvr2,T1,P2,T1,P2,P3,ne2,nc2,N);
```

```
   W=ssc[0].W+ssc[1].W;
   Wisothermal=m*(g.R/g.M)*T1*Math.log(P3/P1);
   etaisothermal=Wisothermal/W;
  }

  public void plot_Pv()
  {
  double a[][]=ssc[0].plot_data();
  double b[][]=ssc[1].plot_data();
  int n=a[0].length+b[0].length;
  int n1=a[0].length;
  double xi[][]=new double[3][a[0].length];
  double yi[][]=new double[3][a[0].length];
  for(int i=0;i<n1;i++)
  {xi[0][i]=a[0][i];yi[0][i]=a[1][i];}
  for(int i=0;i<n1;i++)
  {xi[1][i]=b[0][i];yi[1][i]=b[1][i];}
  Plot pp=new Plot(xi,yi);
  pp.setPlabel("Double stage reciprocating air compressor");
  pp.setXlabel("V m^3/s");
  pp.setYlabel("P bar");
  pp.setPlotType(0,5);
  pp.setPlotType(1,5);
  pp.setColor(0,Color.blue);
  pp.setColor(1,Color.blue);
  pp.plot();
  }

  public String toString()
  {String s="First stage Data : \n";
   s+=ssc[0].toString()+"\n";
   s+="second stage Data : \n";
   s+=ssc[1].toString()+"\n";
   s+="Combined data : ";
   s+="work required W = "+W+" kW\n";
   s+="isothermal (minimum possible) work output = "+Wisothermal+"\n";
   s+="isothermal efficiency = "+etaisothermal+"\n";

   s+="W="+W+"\n";
   return s;
  }

  public static void main(String arg[])
  { two_stage_gas_compressor ssc=new
two_stage_gas_compressor("air",0.286034892,0.05,0.05,288.0,1.013,305.0,0.95,2.578,7.0,1.3,1.3,1.3,1.3,300.0);
    System.out.println(ssc.toString());
    ssc.plot_Pv();
  }
}
```

```
---------- Capture Output ----------
> "C:\java\bin\javaw.exe" two_stage_gas_compressor
First stage Data :
work required W = 28.115774582942578 kW
induced mass flow rate m=0.286034892 kg/s
compression mass flow rate m=0.318749135391564 kg/s
expansion mass flow rate m=0.032714243391564035 kg/s
polytropic expansion coefficient ne=1.3
polytropic compression coefficient nc=1.3
outside (inlet) gas condition T0 = 288.0 degree K P0 = 1.013 bar
compressor inlet gas condition T1 = 305.0 degree K P1 = 0.95 bar
compressor inlet gas condition T2 = 384.01965208637057 degree K P2 = 2.578 bar
Va=0.2937648938188761 m^3/s Vb=0.1362990204550468 m^3/s Vc=0.013988804467565528 m^3/s
Vd=0.030150030758458363 m^3/s Vs=0.27977608935131054VI=0.2636148630604177 m^3/s
Vs_cycle = 0.05595521787026211 m^3
```

```
volumetric efficiency = 0.9422351412218097
isothermal (minimum possible) work output = 25.000934281788012
isothermal efficiency = 0.889213783103657

second stage Data :
work required W = 28.134349402297975 kW
induced mass flow rate m=0.286034892 kg/s
compression mass flow rate m=0.31876565856185696 kg/s
expansion mass flow rate m=0.03273076656185697 kg/s
polytropic expansion coefficient ne=1.3
polytropic compression coefficient nc=1.3
outside (inlet) gas condition T0 = 305.0 degree K P0 = 2.578 bar
compressor inlet gas condition T1 = 305.0 degree K P1 = 2.578 bar
compressor inlet gas condition T2 = 384.07185681125037 degree K P2 = 7.0 bar
Va=0.10825877260443238 m^3/s Vb=0.050206408467080355 m^3/s Vc=0.0051551796478301135 m^3/s
Vd=0.011115979777668667 m^3/s Vs=0.10310359295660226VI=0.09714279282676372 m^3/s
Vs_cycle = 0.020620718591320454 m^3
volumetric efficiency = 0.9421863006040194
isothermal (minimum possible) work output = 25.015685944597877
isothermal efficiency = 0.889151036936885

Combined data : work required W = 56.25012398524055 kW
isothermal (minimum possible) work output = 50.01662022638588
isothermal efficiency = 0.8891823996603052
W=56.25012398524055
```

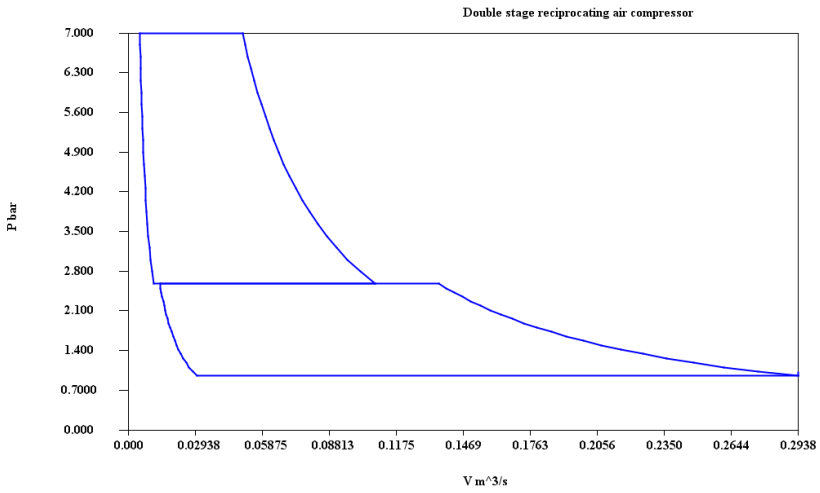


Figure 7.1-6 Two stage Reciprocating compressor Pressure-Volume diagram with intercooling

The single stage and double stage examples utilised the same mass flow rate and exit pressures, therefore results are comparable. While single stage compressor required 63.53 kW input power, double stage compressor with intercooling required only 56.25 kW. If Intercooling is not applied, Performance will be same as single stage compressor.

```
---------- Capture Output ----------
> "C:\java\bin\javaw.exe" two_stage_gas_compressor2
First stage Data :
work required W = 28.115774582942578 kW
induced mass flow rate m=0.286034892 kg/s
compression mass flow rate m=0.318749135391564 kg/s
expansion mass flow rate m=0.032714243391564035 kg/s
polytropic expansion coefficient ne=1.3
polytropic compression coefficient nc=1.3
outside (inlet) gas condition T0 = 288.0 degree K P0 = 1.013 bar
```

```
compressor inlet gas condition T1 = 305.0 degree K P1 = 0.95 bar
compressor inlet gas condition T2 = 384.01965208637057 degree K P2 = 2.578 bar
Va=0.2937648938188761 m^3/s Vb=0.1362990204550468 m^3/s Vc=0.013988804467565528 m^3/s
Vd=0.030150030758458363 m^3/s Vs=0.27977608935131054VI=0.2636148630604177 m^3/s
Vs_cycle = 0.05595521787026211 m^3
volumetric efficiency = 0.9422351412218097
isothermal (minimum possible) work output = 25.000934281788012
isothermal efficiency = 0.889213783103657

second stage Data :
work required W = 35.42341989884214 kW
induced mass flow rate m=0.286034892 kg/s
compression mass flow rate m=0.318765658561857 kg/s
expansion mass flow rate m=0.03273076656185703 kg/s
polytropic expansion coefficient ne=1.3
polytropic compression coefficient nc=1.3
outside (inlet) gas condition T0 = 384.01965208637057 degree K P0 = 2.578 bar
compressor inlet gas condition T1 = 384.01965208637057 degree K P1 = 2.578 bar
compressor inlet gas condition T2 = 483.5775109141728 degree K P2 = 7.0 bar
Va=0.13630654488803814 m^3/s Vb=0.06321392626896528 m^3/s Vc=0.006490787851811341 m^3/s
Vd=0.01399591700596542 m^3/s Vs=0.1298157570362268VI=0.12231062788207273 m^3/s
Vs_cycle = 0.02596315140724536 m^3
volumetric efficiency = 0.9421863006040193
isothermal (minimum possible) work output = 31.49677053490619
isothermal efficiency = 0.8891510369368854

Combined data : work required W = 63.53919448178472 kW
isothermal (minimum possible) work output = 50.01662022638588
isothermal efficiency = 0.7871774364518351
W=63.53919448178472
```

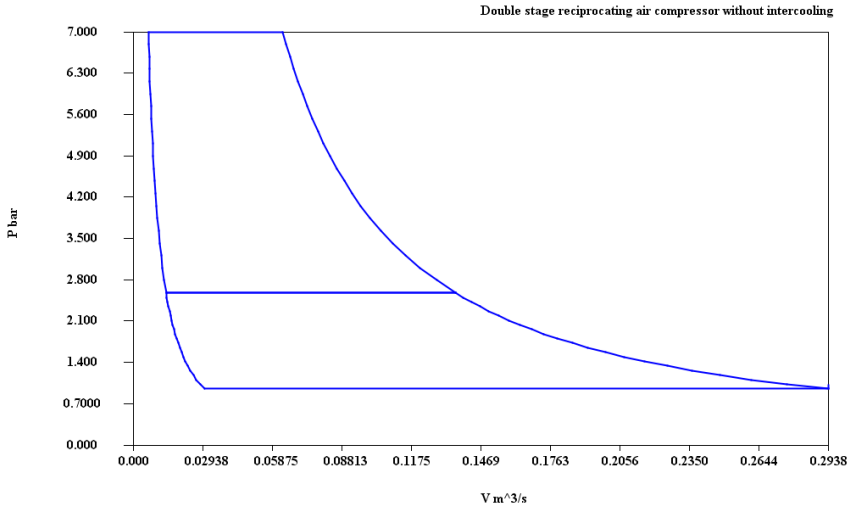


Figure 7.1-7 Two stage Reciprocating compressor Pressure-Volume diagram without intercooling

Compressor used in refrigerators and heat pumps utilises refrigerants and refrigerants can not be considered as ideal gas. But if equation of state for the refrigerants are available, compressors can be simulated in similar manners. As the basic equation of state, ISO 17025 equation of state for refrigerants are used.

```
 import java.awt.Color;
public class single_stage_refrigeration_compressor
{ public  refISO g;
  public double Vs;// Swept volume m^3/s
```

```
  public double Vc;//Clerance volume
  public double VI; //indicated volume : Va-Vd
  public double cvr;// clearance volume ratio;
  public double ne,nc;//expansion and compression polytropic expansion coefficients
  public double m; //mass flow rate kg/s
  public double N; //compressor speed rpm (revolution per minute)
  public double T0,T1,T2,Td; //inlet temperature
  public double P0; //Atmospheric pressure
  public double P1; //inlet pressure
  public double P2; //outlet pressure
  public double Vs_cycle;// swept volume m^3
  public String gas;
  public double Va,Vb,Vd;
  public double va,vb,vc,vd,vs,vi;
  public double W,W1,W2,W3;
  public double ma,md;
  public double etav;
  public double Wisothermal;
  public double Qisothermal;
  public double etaisothermal;
  public double ha,hb,hc,hd;
  public double sa,sb,sc,sd;
  public single_stage_refrigeration_compressor(String gasi,double mi,double cvri,double T0i,double P0i,double T1i,double
P1i,double P2i,double nei,double nci,double Ni)
  { m=mi;cvr=cvri;T0=T0i;P0=P0i;T1=T1i;P1=P1i;P2=P2i;gas=gasi;g=new refISO(gas);g.setMass();
    ne=nei;nc=nci;N=Ni;
    double a[]=g.property("tp",T1,P1);
    VI=m*a[2];

    Vs=VI/((1+cvr)-cvr*Math.pow((P2/P1),(1.0/nc)));
    Vc=cvr*Vs;
    Va=(1.0+cvr)*Vs;
    Vb=Va*Math.pow((P1/P2),(1.0/ne));
    Vd=Vc*Math.pow((P2/P1),(1.0/nc));
    va=a[2];
    ma=Va/va;
    md=ma-m;
    ha=a[3];
    sa=a[5];
    vb=Vb/ma;
    vc=Vc/md;
    vd=Vd/md;
    etav=(Va-Vd)/Vs;
    double a1[]=g.property("pv",P2,vb);
    T2=a1[1];
    double a1A[]=g.property("tp",T1,P2);
    hb=a1A[3];
    sb=a1A[5];
    double Tc=T2;
    double a2[]=g.property("pv",P1,vd);
    Td=a2[1];

    double a3[]=g.property("pv",P2,vc);
    Tc=a3[1];
    hc=md*a3[3];
    sd=md*a3[5];

    W1=ne/(ne-1)*(P2*Vb-P1*Va);
    W2=nc/(nc-1)*(P2*Vc-P1*Vd);
    Qisothermal=(ma-md)*T1*(sb-sa);
    Wisothermal=(ma-md)*(hb-ha)-Qisothermal;
    W=W1-W2;
    etaisothermal=Wisothermal/W;
    Vs_cycle=Vs/(N/60.0);  // m^3/rev
  }
  public double[][] plot_data()
```

```
 {double a[][]=new double[2][46];
 a[0][0]=Va;
 a[1][0]=P0;
 a[0][1]=Va;
 a[1][1]=P1;
 double dP=(P2-P1)/21;
 double P=P1;
 double V=Va;
 for(int i=0;i<=20;i++)
 { P+=dP;
   V=Va*Math.pow((P1/P),(1.0/ne));
   a[0][i+2]=V;
   a[1][i+2]=P;
 }
 a[0][23]=Vc;
 a[1][23]=P2;
 P=P2;
 V=Vc;
 for(int i=0;i<=20;i++)
 { P-=dP;
   V=Vc*Math.pow((P2/P),(1.0/ne));
   a[0][i+24]=V;
   a[1][i+24]=P;
 }
 a[0][44]=Vd;
 a[1][44]=P1;
 a[0][45]=Va;
 a[1][45]=P1;
 return a;
 }
 public void plot_Pv()
 {
 double a[][]=plot_data();
 Text.printT(a);
 Plot pp=new Plot(a);
 pp.setPlabel("Single stage reciprocating Refrigeration("+gas+") compressor");
 pp.setXlabel("V m^3/s");
 pp.setYlabel("P bar");
 pp.setPlotType(0,5);
 pp.setColor(0,Color.blue);
 pp.plot();
 }

 public String toString()
 {String s="work required W = "+W+" kW\n";
  s+="induced mass flow rate m="+m+" kg/s\n";
  s+="compression mass flow rate m="+ma+" kg/s\n";
  s+="expansion mass flow rate m="+md+" kg/s\n";
  s+="polytropic expansion coefficient ne="+ne+"\n";
  s+="polytropic compression coefficient nc="+nc+"\n";
  s+="outside (inlet) gas condition T0 = "+T0+" degree K P0 = "+P0+" bar \n";
  s+="compressor inlet gas condition T1 = "+T1+" degree K P1 = "+P1+" bar \n";
  s+="compressor inlet gas condition T2 = "+T2+" degree K P2 = "+P2+" bar \n";
  s+="Va="+Va+" m^3/s Vb="+Vb+" m^3/s Vc="+Vc+" m^3/s Vd="+Vd+" m^3/s Vs="+Vs+"VI="+VI+" m^3/s \n";
  s+="Vs_cycle = "+Vs_cycle+" m^3\n";
  s+="volumetric efficiency = "+etav+"\n";
  s+="isothermal work Wisothermal = "+Wisothermal+"\n";
  s+="isothermal efficiency etaisothermal = "+etaisothermal;
  return s;
 }

 public static void main(String arg[])
 { single_stage_refrigeration_compressor ssc=new
single_stage_refrigeration_compressor("R134a",1.0,0.02,273.0,3.0,273.0,2.95,15.0,1.2,1.2,300.0);
   System.out.println(ssc.toString());
   ssc.plot_Pv();
```

```
  }
}
```

```
 ---------- Capture Output ----------
> "C:\java\bin\javaw.exe" single_stage_refrigeration_compressor
work required W = 41.5219367987685 kW
induced mass flow rate m=1.0 kg/s
compression mass flow rate m=1.0822867414486952 kg/s
expansion mass flow rate m=0.08228674144869519 kg/s
polytropic expansion coefficient ne=1.2
polytropic compression coefficient nc=1.2
outside (inlet) gas condition T0 = 273.0 degree K P0 = 3.0 bar
compressor inlet gas condition T1 = 273.0 degree K P1 = 2.95 bar
compressor inlet gas condition T2 = 358.25104240259975 degree K P2 = 15.0 bar
Va=8.15506872984526 m^3/s Vb=2.1031484243870993 m^3/s Vc=0.15990330842833844 m^3/s
Vd=0.6200334960870695 m^3/s Vs=7.995165421416922VI=7.535035233758191 m^3/s
Vs_cycle = 1.5990330842833844 m^3
volumetric efficiency = 0.9424489471567198
isothermal work Wisothermal = 36.10413428812925
isothermal efficiency etaisothermal = 0.8695195135791466
```

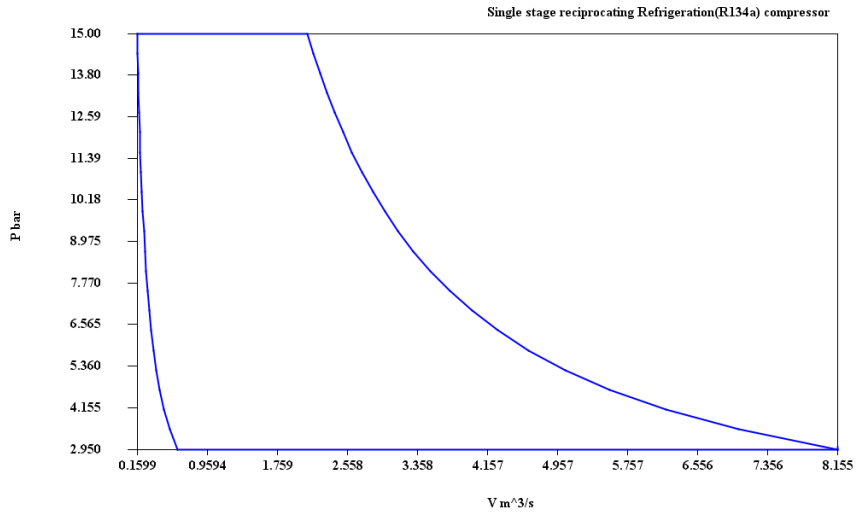


Figure 7.1-8 single stage Reciprocating refrigerant (R134a) compressor Pressure-Volume diagram

Let us look at rotating piston compressors now. Figure 7.1-9 and 71-10 showing geometry and operating principles of such a machine. Inlet and exit is seperated by a spring pushed valve plateLeft hand side space is used for compression while the right hand side is the suction side. Therefore compressor is a double effect type.

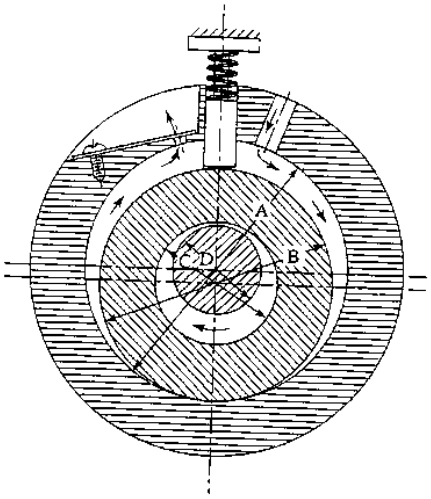


Figure 7.1-9 rotating piston compressor

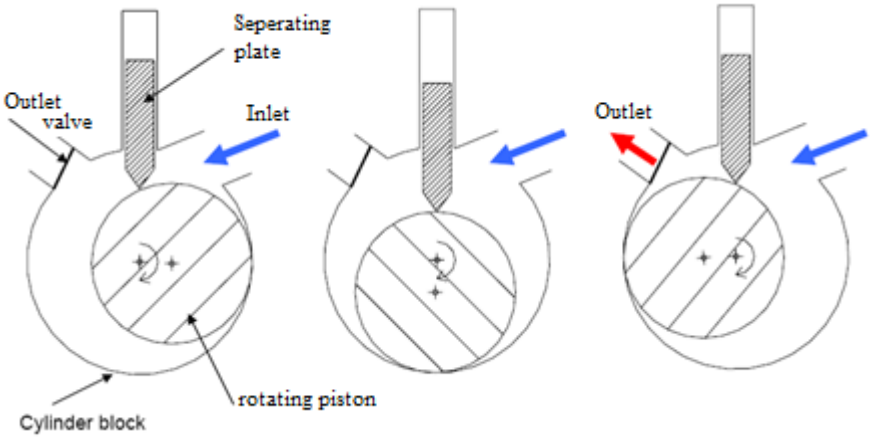


Figure 7.1-10 operating of a rotating piston compressor

Swept volume of the compressor can be written as:

$$V_s = \frac{\pi W}{4}(A^2 - B^2)$$

In this equation, W is the width of the seperation plate and compressor cylinder block. A is the diameter of the compressor body, and B is the diameter of the piston. Compressor efficiency is function of gas volume remaining inside of the compressor while the gas is exhausting from the compressor. Compressor work output:

$w_{compressor} = w_{shaft} / \left[\left(\frac{A}{B} - 1\right)\frac{A}{B}\right]$. Swept volume of the compressor can be calculated as:

$$C.D. = \frac{\pi W}{4}(A^2 - B^2)(w_{shaft})\left[\left(\frac{A}{B} - 1\right)\frac{A}{B}\right]$$

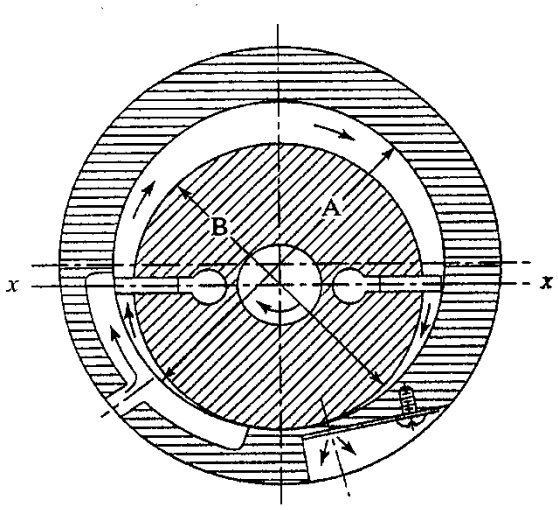


Figure 7.1-10 rotating compressor with sliding vane

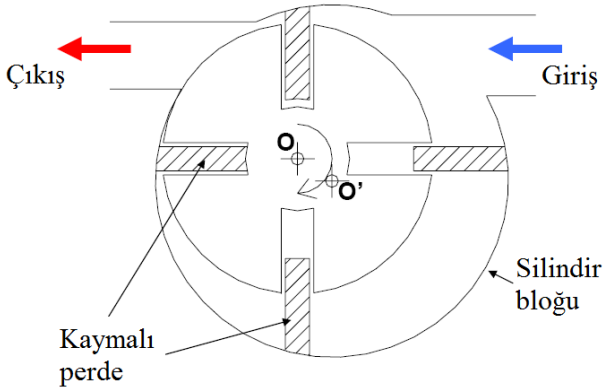


Figure 7.1-11 rotating compressor with sliding vane operation

Rotating compressor with swept type separating plates are similar to rotating compressors.
In this compressors sliding vanes are located on the piston. Vanes move forward by an internal spring push to close space between piston and compressor body. Maximum volumetric volume happens at x-x center point.

$$V_{x-x} = W\left[\frac{A^2}{4}\cos^{-1}\left(\frac{A-B}{A}\right) + \frac{C}{4}(B-A)\right]$$

In this equation:

$$C = \sqrt{B(2A-B)}$$

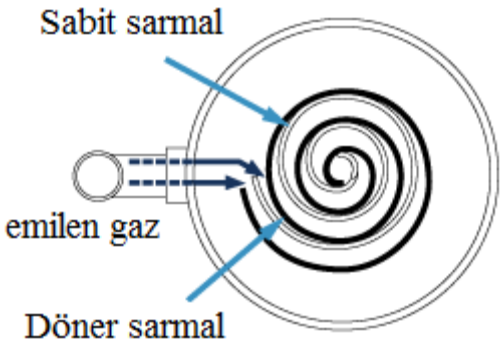
Swept volume:

$$V_s = \frac{W}{4}\left[\pi(A^2 - \frac{B^2}{2})\cos^{-1}\left(\frac{A-B}{A}\right) + \frac{C}{4}(B-A) - 2t(C-B)\right]$$

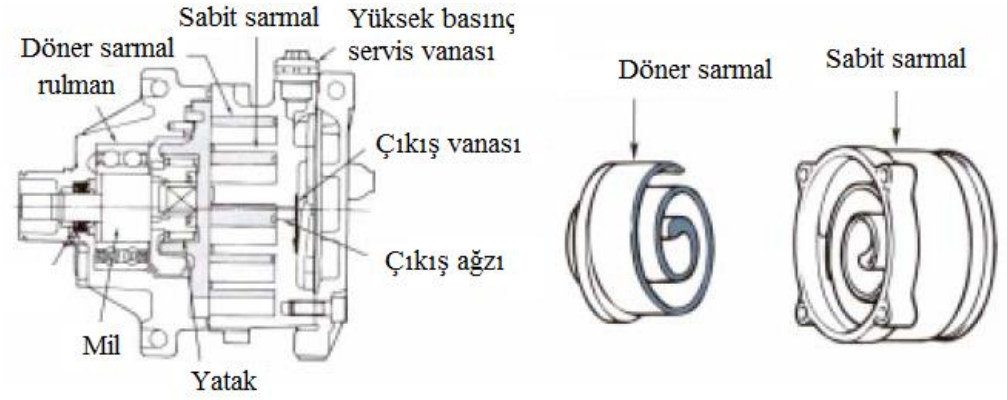
Where t is channel thickness.

C.D=$2V_s(W_{mil})$

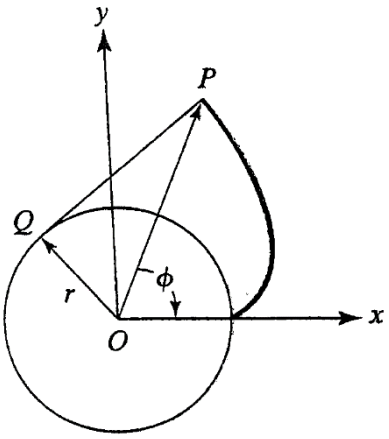
Döner kompresörler gibi sarmal kompresörler de sabit hacim aktarım makinalarıdır. Sarmal kompresördeki temel geometri bir sipiral şeklindedir. Şekil 3.1.13 de sipiralin hareket geometrisi görülmektedir.



Şekil 5.12a Sarmal kompresör



Şekil 5.12b Sarmal kompresör

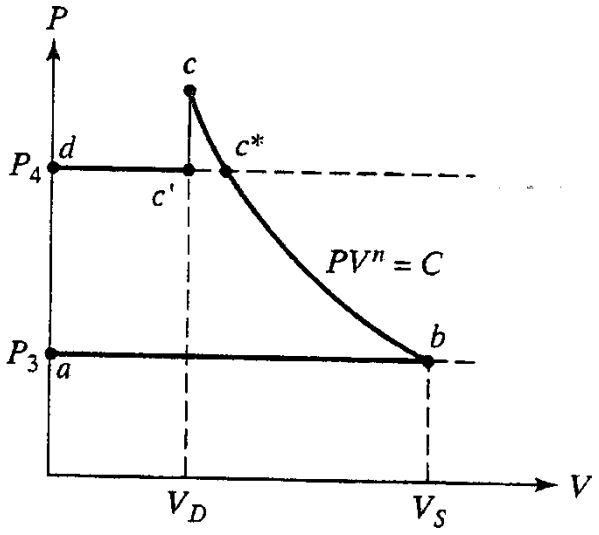


Şekil 5.13 Sarmal kompresörün sipiral hareketi

P noktasının x ve y koordinatı hareket denklemlerini yazacak olursak:

$x_p = r(\cos\phi + \phi\sin\phi)$

$y_p = r(\sin\phi - \phi\cos\phi)$ olur. Bu denklemlerde r temel dönme çemberinin çapı, ϕ merkez noktasıyla sipiral yayın uç noktası arasındaki açıdır. Sipirallerin kalınlığı t, yüksekliği h dir.



Şekil 5.14 Sarmal kompresörün P-V diyagramı

Şekil 3.1.14 de sarmal kompresörün P-V diyagramı görülmektedir. a nokrası kompresörün giriş noktasıdır. Akış a-b arasında çok az bir basınç düşümü olmaktadır, bu yüzden bu prosesde basınç düşümü olmadığını kabul edebiliriz. Sıkıştırma sonucu systemde istenilen basıncın üzerinde bir basınç oluşması mümkündür.

$\frac{P_c}{P_b} = \frac{P_c}{P_3} = \left(\frac{V_b}{V_c}\right)^n = \left(\frac{V_s}{V_d}\right)^n = V_r^n$ buradaki V_r kompresörün hacim oranıdır.

Bu kompresörün hacimsel verimini teorik denklemlerle vermek oldukça zordur, genellikle imlatçıların deneysel verilerini kullanmamız gerekir. Ancak bir yaklaşım olarak

$\eta_v = (1 - L_{uç} - L_{yan})\frac{v_3}{v_b}$ yaabiliriz. Burada $L_{uç}$ sarmal kompresörün ucundan sızan(kaçan) gaz yüzdesi, $L_{uç}$ sarmal kompresörün sipiralinin yan yüzeyinden sızan(kaçan) gaz yüzdesidir.

Kompresörde pompalanan kütlesel debi

$$m = \frac{\eta_v C.D.}{v_3} = (1 - L_{uç} - L_{yan})\frac{C.D.}{v_b}$$

Çevrim başına iş gereksinimi :

$$W/çevrim = \int_b^c VdP - \int_{c'}^c VdP$$

$$W/çevrim = \frac{n}{n-1} P_3 V_s \left[\left(\frac{P_c}{P_3}\right)^{(n-1)/n} - 1\right] - (P_c - P_4)V_d$$

Aynı denklemi hacim oranı V_r cinsinden yazarsak :

$$W/çevrim = \frac{n}{n-1} P_3 V_s \left[V_r^{(n-1)} - 1\right] - (P_3 V_r^n - P_4)\frac{V_s}{V_r}$$

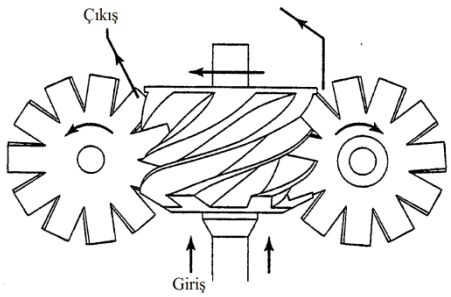
Tüm çevrim için iş girişi:

$$w = \frac{n}{n-1} P_3 v_b \left[V_r^{(n-1)} - 1 \right] - (P_3 V_r^n - P_4) \frac{v_b}{V_r}$$

Elektrik motoru iş gereksinimi:

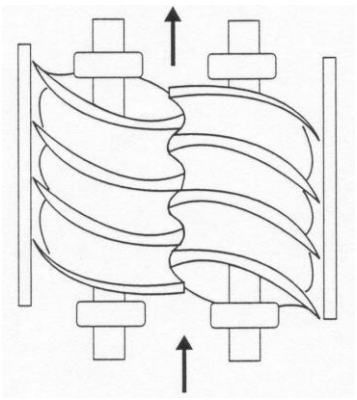
$$W_{elektrik} = \frac{\eta_v C.D.}{\eta_m \eta_e} \left[\frac{n}{n-1} P_3 \left[V_r^{(n-1)} - 1 \right] - \frac{(P_3 V_r^n - P_4)}{V_r} \right]$$

Vidalı kompresörler temel olarak iki türlü imal edilebilen pozitif hacim süpürme kompresörleridir. Tek vida veya çift vida türlerinde olurlar. Tek vidalı türü şekil 3.1.15 de gösterilmiştir. Helical bir vida iki adet yıldız tipi çıkış ağzı olan mile hareket verir.

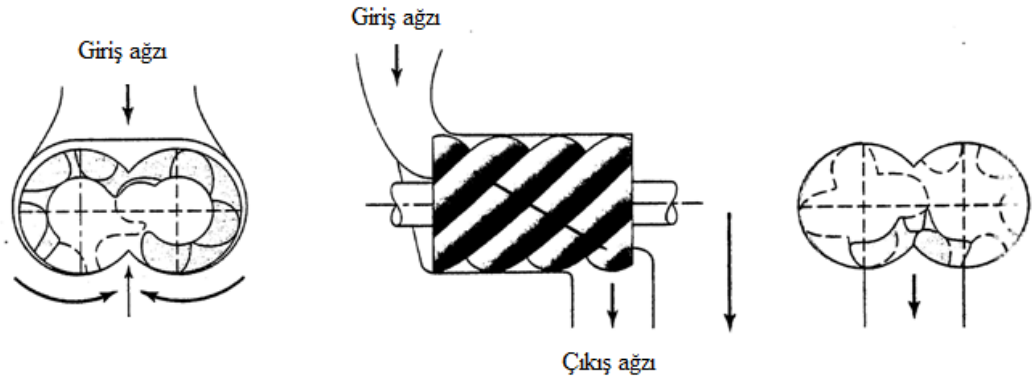


Şekil 5.15 Tek vidalı kompresör

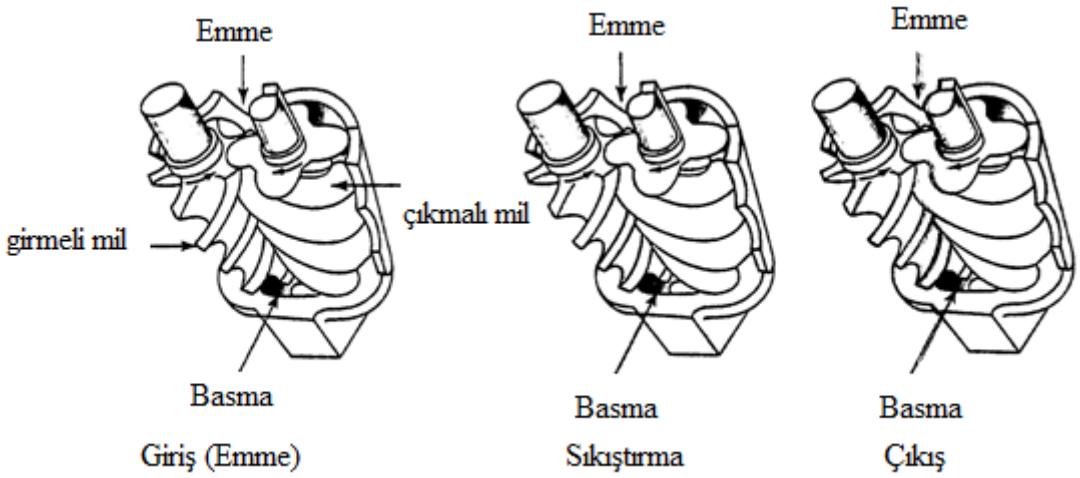
Çift vidalı kompresörde yıldız tipi mil yerine ikinci bir helical vida alır, ikinci vidanın yüzeyi birinci ile aradaki hacimleri kapatacak şekilde ters büküme sahiptir.



Şekil 5. 16a Çift vidalı kompresör



Şekil 5.16b Çift vidalı kompresör



Şekil 5.17 Çift vidalı kompresörün temel proseslerinin oluşması



Şekil 5.18 Çift vidalı kompresörün vidalarının görünümü

8.0 OPTIMISATION OF THERMODYNAMIC CYCLES

10.1 DEFINITIONS OF OPTIMIZATION

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))

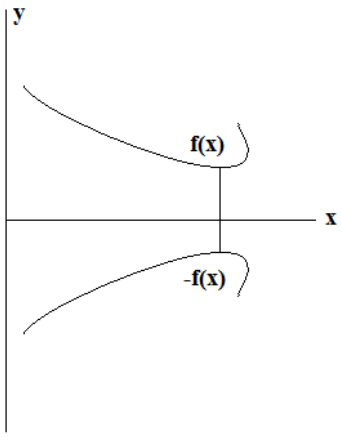


Figure 8.1-1 Relation between maximum and minimum of the same function

When a thermodynamic process operating, adjustable parameters in the cycle can be set to the values that the best performance (Efficiency, coefficicent of performance etc.) can be obtained. As a first example let us look at a one dimensional optimisation of a flash chamber refrigeration cycle we already investigated. As an optimisation method one of the simplest, golden search (fibonnachi search) method will be utilised.

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.

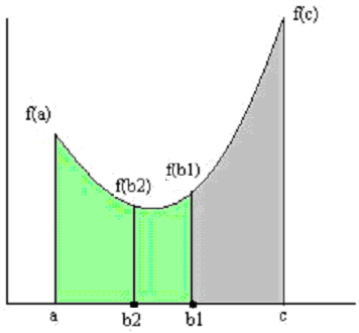


Figure 8.1-2 Golden ratio search

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 .
A java code for golden search is given below.

Program 10.1-1 Golden search or Fibbonachi method

```
/ 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);
     }
}
```

As another one dimensional optimisation method Newton- Raphson method will be investigated. This method is actually a root finding method, but the root of derivative of a function is the optimum point, therefore, the root of the derivative is an optimum point. This method will not directly determine if the optimum is a minimum or maximum.
Minimum of f(x)

Roots of $g(x) = \frac{df(x)}{dx} = 0$

Newton raphson formula

$$x_{n+1} = x_n - \frac{g(x)}{\frac{dg(x)}{dx}} = x_n - \frac{\frac{df(x)}{dx}}{\frac{d^2 f(x)}{dx^2}} = x_n - \frac{f'(x)}{f''(x)}$$

Program 10.1-2 Newton search

```
// OPTIMISATION

// Newton-Raphson 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)

  {return x*x*x-3*x;}

}


public class newton_opt

{

public static double newton(f_x f,double x)

{

// root finding newton-raphson program

int nmax=100;

double tolerance=1.0e-8;

double dfx,d2fx;

for(int i=0;i<nmax;i++)
```

```
{
dfx=f.dfunc(x);
d2fx=f.dfunc2(x);
x-=dfx/d2fx;
if(Math.abs(dfx)<tolerance)  {return x;}
}
return x;
}
    public static void main (String args[])
    {
            fb f=new fb();
    double x0;
    x0=Double.parseDouble(JOptionPane.showInputDialog(" enter guess value  x0 : "));
    double r;
    r= newton(f,x0);
    JOptionPane.showMessageDialog(null," minimum value  : "+r,
    "Newton optimisation method : ",JOptionPane.PLAIN_MESSAGE);
    System.exit(0);
    }
}
```

Optimisation algorithms are basically scholastic or geometric type. For the example problem, A scholastic optimisation method, bat algorithm will be used. 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 10.1-1 Bat Search algorithm

Objective function $f(x) =, \; x = (x_1, x_2, ..., x_d)^T$
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 *fmin*, 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 10.1-3 Bath search algorithm

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

abstract class f_xj
{
  // single function  multi independent variable
  // a single value is returned indiced to equation_ref
  // example f[0]=x[0]+sin(x[1])
  //         f[1]=x[0]*x[0]-x[1]
  // func(x,1) returns the value of f[1]
  // func(x,0) returns the value of f[0]

  abstract double func(double x[]);
}

//Program to find minimum of the function
//for maximum finding change the sign of the function

public class bat
{
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
double ymin;
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 bat(f_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);
               //for(int i=0;i<n;i++)
               //{System.out.println(z[0][i]+"   "+z[1][i]);}
                        Plot pp=new Plot(z[0],z[1]);
                        pp.setPlotType(0,35);
                        pp.setMinMax(xxmin[0],xxmax[0],xxmin[1],xxmax[1]);
                        pp.plot();
           }
           }
}
```

Another example to the stohastic optimisation code is multiple 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.

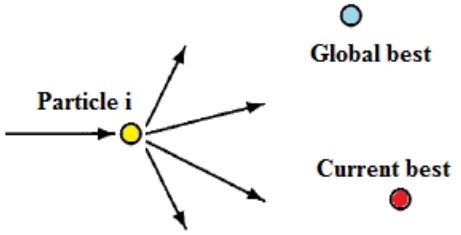


Figure 5.23.1 Particle will attract toward the global best, current best and also move randomly

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:

$$V_i^{t+1} = \theta V_i^t + \alpha * random() * (GP\min - x_i^t) + \beta * random() * (P\min - x_i^t)$$

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 $0 < \gamma < 1 \quad \alpha_0 \cong 0.5..1$ 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 is given below:

```
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[])

{

//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[]={0.0,0.0};

double xmax[]={5.0,5.0};

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);

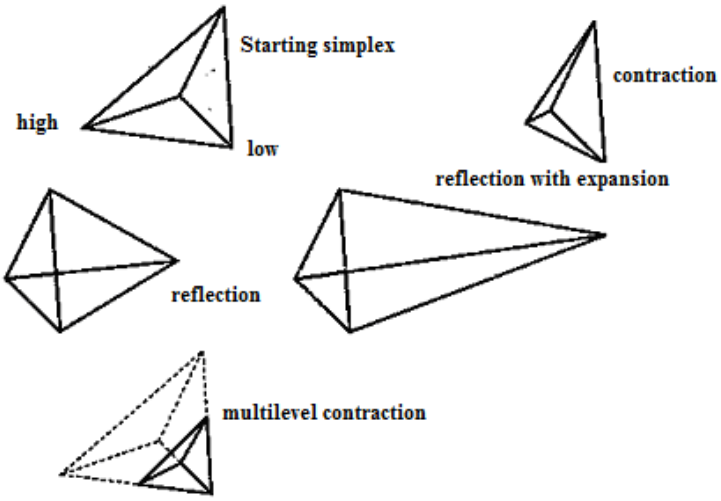
    String s2="Particle swarm-Nelder-Mead simplex optimization method ";

    JOptionPane.showMessageDialog(null,s,s2,JOptionPane.PLAIN_MESSAGE);
```

```
}

}
```

As an example of geometric Nelder & Mead simplex method can be given. 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:
  $f(x_1) \le f(x_2) \le \ldots \le f(x_{n+1})$
- **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.:

$f(x_1) \le f(x_r) < f(x_n)$

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, $f(x_r) < f(x_1)$

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, $f(x_e) < f(x_r)$

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 $f(x_r) < f(x_n)$

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. $f(x_c) < f(x_{n+1})$

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

$x_i = x_1 + \sigma(x_i - x_1) \quad for\ all\ i \in \{2,...,n+1\}$ g oto step 1.

Note: $\alpha, \gamma, \rho,$ and σ are respectively the reflection, the expansion, the contraction and the shrink coefficient. Standard values are $\alpha = 1, \gamma = 2, \rho = 1/2$ and $\sigma = 1/2$

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

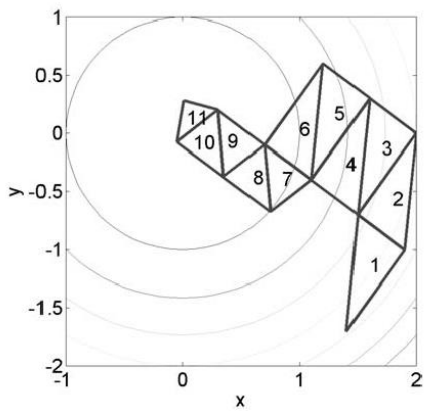
$x_{i-1} = P_i + \lambda dx_j$ where if i=j λ=1 else if i ≠ j λ=0 .

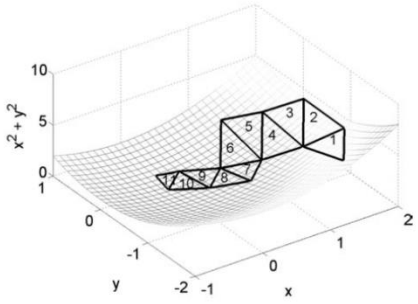
For example:

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 8.1-1 Movements of the nelder-Mead simplex element to find the optimum point





Program 10.1-4 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.
//_____________________________________________________________________
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;
```

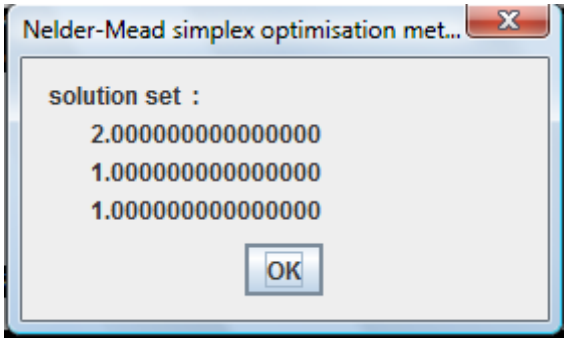
```
                }
                else   /////// shrink
                {
                   for (i=0; i<NPTS; i++)
                    if (i != ilo)
                    {
                       for (j=0; j<NDIMS; j++)
                         p[i][j] = 0.5*p[ilo][j] + 0.5*p[i][j];
                       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);
    }
  // 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);
    }
}
```



10.2 OPTIMISATION OF THERMODYNAMIC CYCLES

As the first example case, a two stage flash chamber type refrigeration cycle will be investigated to see what intermediate pressure(flash chamber pressure) will give us the best COP value. Please refer to the refrigeration section for detailed thermodynamics of the cycle. As the first method, graphic method will be employed. In this method equal distance independent variable arrays are prepared and searched for all the given array. Array dimension is depends on number of independent variables in the object function. An example program and results are given below for one dimensional case.

Program 10.2-1 Graphic search method applied to flash chamber two stage refrigeration cycle

```
// 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
{  refrigerantEN r;
   ref_cycle4 re;
```

```
  double m;
  String refrigerant;
  double P1,P2,P6;
  double dT1i; //degree C
  double etaisent2i;
  double etaisent6i;
  double dT7i; //degree C
  double dPevap1i;
  double dPevap2i;
  double dPcomp1i;
  double dPcond1i;
  double dPcond2i;
  double dPcond3i;
  double dh1i;
  double dh2i;
  public fb(String refi)
  {m=1.0;  // kg/s çevrim R134a debisi
  refrigerant=refi;
  r=new refrigerantEN(refrigerant);
  P1=Pres(-5.0);
  P2=Pres(15.0);
  P6=Pres(45.0);
  dT1i=10.0; //degree C
  etaisent2i=0.8;
  etaisent6i=0.8;
  dT7i=3.0; //degree C
  dPevap1i=5.0;
  dPevap2i=5.0;
  dPcomp1i=5.0;
  dPcond1i=5.0;
  dPcond2i=5.0;
  dPcond3i=5.0;
  dh1i=1.0;
  dh2i=1.0;
  re=new ref_cycle4(refrigerant,m,dT1i,P1,P2,etaisent2i,P6,etaisent6i,dT7i,
  dPevap1i,dPevap2i,dPcomp1i,dPcond1i,dPcond2i,dPcond3i,dh1i,dh2i);
}
public  double Pres(double T)
{
double a[]=r.property("tx",T,0.0);
double P1a=a[0];
return P1a;
}
  public double func (double T)
  { P2=Pres(T);
            re=new ref_cycle4("R134a",m,dT1i,P1,P2,etaisent2i,P6,etaisent6i,dT7i,
   dPevap1i,dPevap2i,dPcomp1i,dPcond1i,dPcond2i,dPcond3i,dh1i,dh2i);
            re.cycle();
   return -re.COP_evaporator;
  }
}


public class ref_cycle4_opt1A
{


    public static void main (String args[])
```

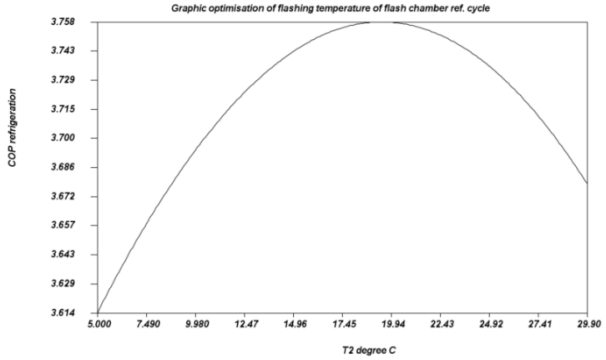
```
    {
            fb f=new fb("R134a");
    double a,b;
    double TT[]=new double[250];
    double CO[]=new double[250];
    //a=Double.parseDouble(JOptionPane.showInputDialog(" enter lower limit of the function a : "));
    //b=Double.parseDouble(JOptionPane.showInputDialog(" enter upper limit of the function b : "));
    double r;
    double y=0;
    double ymin=99.0e30;
    double Tmin=99.0e30;
    int i=0;
    for(double T=5;T<30;T+=0.1)
    {   y=f.func(T);
       if(y<ymin) {ymin=y;Tmin=T;}
              System.out.println("i="+i+"T = "+T+"COP = "+(-y)+"ymin="+ymin+"Tmin="+Tmin);
              TT[i]=T;CO[i]=-y;
              i++;
            }
     String s="Graphic search optimisation : \n";
     s+="T2 = "+Tmin+"\ncop = "+(-f.func(Tmin))+"\n";
     s+=f.re.toString();
     System.out.println(s);
     f.re.plot(-10.0);
     Plot p=new Plot(TT,CO);
     p.setPlabel("Graphic optimisation of flashing temperature of flsh chamber refrigeration cycle");
     p.setXlabel("T2 degree C");
     p.setYlabel("COP refrigeration");
     p.plot();
    }
}
```

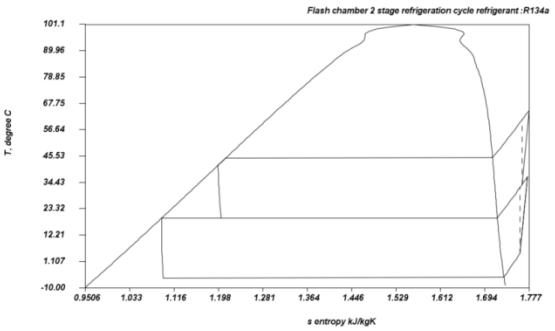
```
---------- Capture Output ----------
> "C:\java\bin\javaw.exe" ref_cycle4_opt1A
i=0T = 5.0COP = 3.614168439076277ymin=-3.614168439076277Tmin=5.0
i=1T = 5.1COP = 3.616069552879226ymin=-3.616069552879226Tmin=5.1
i=2T = 5.199999999999999COP = 3.6179594504231454ymin=-
3.6179594504231454Tmin=5.199999999999999
i=3T = 5.299999999999999COP = 3.6198380814301023ymin=-
3.6198380814301023Tmin=5.299999999999999
i=4T = 5.399999999999999COP = 3.6217053706371574ymin=-
3.6217053706371574Tmin=5.399999999999999
i=5T = 5.499999999999998COP = 3.6235612527098944ymin=-
3.6235612527098944Tmin=5.499999999999998
i=6T = 5.599999999999998COP = 3.625405654148372ymin=-
3.625405654148372Tmin=5.599999999999998
i=7T = 5.6999999999999975COP = 3.627238510374598ymin=-
3.627238510374598Tmin=5.6999999999999975
i=8T = 5.799999999999997COP = 3.629059745268473ymin=-
3.629059745268473Tmin=5.799999999999997
i=9T = 5.899999999999997COP = 3.6308693017797187ymin=-
3.6308693017797187Tmin=5.899999999999997
i=10T = 5.9999999999999964COP = 3.632667066295041ymin=-
3.632667066295041Tmin=5.9999999999999964
…….
i=130T = 17.99999999999999COP = 3.75635729715892ymin=-
3.75635729715892Tmin=17.99999999999999
```

i=131T = 18.09999999999999COP = 3.75656101219027ymin=-3.75656101219027Tmin=18.09999999999999
i=132T = 18.199999999999992COP = 3.7567499338224093ymin=-3.7567499338224093Tmin=18.199999999999992
i=133T = 18.299999999999994COP = 3.756924088663429ymin=-3.756924088663429Tmin=18.299999999999994
i=134T = 18.399999999999995COP = 3.757083497608288ymin=-3.757083497608288Tmin=18.399999999999995
i=135T = 18.499999999999996COP = 3.7572281588233944ymin=-3.7572281588233944Tmin=18.499999999999996
i=136T = 18.599999999999998COP = 3.757358108496234ymin=-3.757358108496234Tmin=18.599999999999998
i=137T = 18.7COP = 3.7574733551231496ymin=-3.7574733551231496Tmin=18.7
i=138T = 18.8COP = 3.7575739206478223ymin=-3.7575739206478223Tmin=18.8
i=139T = 18.900000000000002COP = 3.757659821965377ymin=-3.757659821965377Tmin=18.900000000000002
i=140T = 19.000000000000004COP = 3.7577310810574605ymin=-3.7577310810574605Tmin=19.000000000000004
i=141T = 19.100000000000005COP = 3.7577877202559886ymin=-3.7577877202559886Tmin=19.100000000000005
i=142T = 19.200000000000006COP = 3.7578297550968682ymin=-3.7578297550968682Tmin=19.200000000000006
i=143T = 19.300000000000008COP = 3.7578572111390076ymin=-3.7578572111390076Tmin=19.300000000000008
i=144T = 19.40000000000001COP = 3.7578701139188837ymin=-3.7578701139188837Tmin=19.40000000000001
i=145T = 19.50000000000001COP = 3.7578684790665564ymin=-3.7578701139188837Tmin=19.40000000000001
i=146T = 19.600000000000012COP = 3.757852337465667ymin=-3.7578701139188837Tmin=19.40000000000001
i=147T = 19.700000000000014COP = 3.757821716021588ymin=-3.7578701139188837Tmin=19.40000000000001
i=148T = 19.800000000000015COP = 3.757776631867818ymin=-3.7578701139188837Tmin=19.40000000000001
i=149T = 19.900000000000016COP = 3.75771711138564ymin=-3.7578701139188837Tmin=19.40000000000001
i=150T = 20.000000000000018COP = 3.7576431914065456ymin=-3.7578701139188837Tmin=19.40000000000001
i=151T = 20.10000000000002COP = 3.7575548810421013ymin=-3.7578701139188837Tmin=19.40000000000001
i=152T = 20.20000000000002COP = 3.757452196598752ymin=-3.7578701139188837Tmin=19.40000000000001
i=153T = 20.300000000000022COP = 3.7573351199034897ymin=-3.7578701139188837Tmin=19.40000000000001
i=154T = 20.400000000000023COP = 3.7572036512241382ymin=-3.7578701139188837Tmin=19.40000000000001
i=155T = 20.500000000000025COP = 3.7570577922421755ymin=-3.7578701139188837Tmin=19.40000000000001
i=156T = 20.600000000000026COP = 3.7568975584750213ymin=-3.7578701139188837Tmin=19.40000000000001
i=157T = 20.700000000000028COP = 3.756722875061401ymin=-3.7578701139188837Tmin=19.40000000000001
i=158T = 20.80000000000003COP = 3.756533813719189ymin=-3.7578701139188837Tmin=19.40000000000001
i=159T = 20.90000000000003COP = 3.7563303517510516ymin=-3.7578701139188837Tmin=19.40000000000001
……..

i=240T = 29.000000000000146COP = 3.6911666792177655ymin=-3.7578701139188837Tmin=19.40000000000001
i=241T = 29.100000000000147COP = 3.689765878172501ymin=-3.7578701139188837Tmin=19.40000000000001
i=242T = 29.20000000000015COP = 3.6883507901411434ymin=-3.7578701139188837Tmin=19.40000000000001
i=243T = 29.30000000000015COP = 3.686921356051609ymin=-3.7578701139188837Tmin=19.40000000000001
i=244T = 29.40000000000015COP = 3.6854775637540342ymin=-3.7578701139188837Tmin=19.40000000000001
i=245T = 29.500000000000153COP = 3.684019370026176ymin=-3.7578701139188837Tmin=19.40000000000001
i=246T = 29.600000000000154COP = 3.6825467382893717ymin=-3.7578701139188837Tmin=19.40000000000001
i=247T = 29.700000000000156COP = 3.6810596249625918ymin=-3.7578701139188837Tmin=19.40000000000001
i=248T = 29.800000000000157COP = 3.6795580009529365ymin=-3.7578701139188837Tmin=19.40000000000001
i=249T = 29.90000000000016COP = 3.6780418279821676ymin=-3.7578701139188837Tmin=19.40000000000001
Graphic search optimisation :
T2 = 19.40000000000001
cop = 3.7578701139188837
two stage flash point refrigeration cycle
Flash ratio : 0.1819461894361739
compressor1 work = 18.48252017878701 kW
compressor2 work = 20.324252448450352 kW
total compressor work = 38.806772627237365 kW
isentropic compressor 1 work = 14.786016143029618 kW
isentropic compressor 2 work = 16.259401958760293 kW
total isentropic compressor work = 31.045418101789913 kW
condenser heat output = 184.81952989021426 kW
evaporator heat input = 145.8308110735407 kW
COP evaporator = 3.7578701139188837
COP condenser = 4.76255863030709
COP evaporator carnot = 3.809162193191491
COP condenser carnot = 4.809162193191491
h1 compressor 1 inlet = 403.8909091817481 kJ/kg
T1 compressor 1 inlet = 4.452127613408595 derece C
P1 compressor 1 inlet = 243.34658575187083 kPa
s1 compressor 1 inlet = 1.7602021806315065 kJ/kgK
x1 compressor 1 inlet = 2.0 kgvapor/kgtotal
h2 compressor 1 exit = 426.48419087069755 kJ/kg
T2 compressor 1 exit = 36.888828569753954 derece C
P2 compressor 1 exit = 561.1803361692431 kPa
s2 compressor 1 exit = 1.7748879591320963 kJ/kgK
x2 compressor 1 exit = 0.0 kgvapor/kgtotal
h5 compressor 2 inlet = 423.3820723757278 kJ/kg
T5 compressor 2 inlet = 33.671168861137616 derece C
P5 compressor 2 inlet = 561.1803361692431 kPa
s5 compressor 2 inlet = 1.7648300700955235 kJ/kgK
x5 compressor 2 inlet = 0.0 kgvapor/kgtotal
h6 compressor 2 exit = 443.7063248241781 kJ/kg
T6 compressor 2 exit = 64.70435475451629 derece C
P6 compressor 2 exit = 1159.9066190539932 kPa
s6 compressor 2 exit = 1.776928390648843 kJ/kgK
x6 compressor 2 exit = 1.1408063654453964 kgvapor/kgtotal
h8 flash chamber input = 226.6253608885008 kJ/kg

```
T8 flash chamber input = 19.400001412683316 derece C
P8 flash chamber input = 561.1803361692431 kPa
s8 flash chamber input = 1.0933511979300417 kJ/kgK
x8 flash chamber input = 0.0 kgvapor/kgtotal
h4 expansion valve exit-evaporator inlet = 225.6253608885008 kJ/kg
T4 expansion valve exit-evaporator inlet = -6.104624271642147 derece C
P4 expansion valve exit-evaporator inlet = 233.34658575187083 kPa
s4 expansion valve exit-evaporator inlet = 1.096484627076248 kJ/kgK
x4 expansion valve exit-evaporator inlet = 0.16625532763755035 kgvapor/kgtotal
```





The same task can be achived by using a numerical optimisation algorithm, such as golden search method.

Program 10.2-2 Golden search method applied to flash chamber two stage refrigeration cycle

```
 // 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
{  refrigerantEN r;
   ref_cycle4 re;
   double m;
   String refrigerant;
   double P1,P2,P6;
   double dT1i; //degree C
   double etaisent2i;
   double etaisent6i;
   double dT7i; //degree C
   double dPevap1i;
   double dPevap2i;
   double dPcomp1i;
   double dPcond1i;
```

```
   double dPcond2i;
   double dPcond3i;
   double dh1i;
   double dh2i;
   public fb(String refi)
   {m=1.0;  // kg/s çevrim R134a debisi
   refrigerant=refi;
   r=new refrigerantEN(refrigerant);
   P1=Pres(-5.0);
   P2=Pres(15.0);
   P6=Pres(45.0);
   dT1i=10.0; //degree C
   etaisent2i=0.8;
   etaisent6i=0.8;
   dT7i=3.0; //degree C
   dPevap1i=5.0;
   dPevap2i=5.0;
   dPcomp1i=5.0;
   dPcond1i=5.0;
   dPcond2i=5.0;
   dPcond3i=5.0;
   dh1i=1.0;
   dh2i=1.0;
   re=new ref_cycle4(refrigerant,m,dT1i,P1,P2,etaisent2i,P6,etaisent6i,dT7i,
   dPevap1i,dPevap2i,dPcomp1i,dPcond1i,dPcond2i,dPcond3i,dh1i,dh2i);
}
public  double Pres(double T)
{
double a[]=r.property("tx",T,0.0);
double P1a=a[0];
return P1a;
}
   public double func (double T)
   { P2=Pres(T);
             re=new ref_cycle4("R134a",m,dT1i,P1,P2,etaisent2i,P6,etaisent6i,dT7i,
    dPevap1i,dPevap2i,dPcomp1i,dPcond1i,dPcond2i,dPcond3i,dh1i,dh2i);
             re.cycle();
    return -re.COP_evaporator;
   }
}


public class ref_cycle4_opt
{
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);}

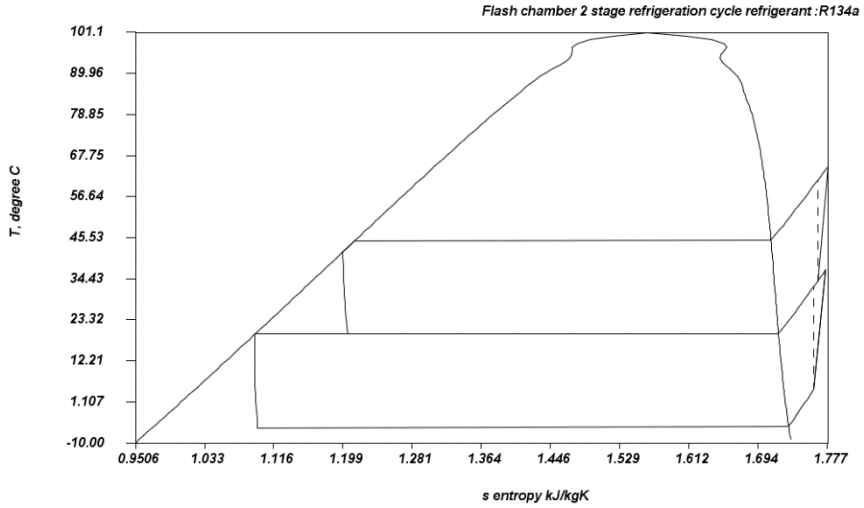
public static double golden(f_x f,double a,double b,double epsilon,double delta,int print)
{
//  find the minimum of the function
// note  maximum f(x) = minimum (-f(x))
    double r1 = (Math.sqrt(5.0)-1.0)/2.0; // golden ratio
    double r2 = r1*r1;
    double h = b - a;
    double ya = f.func(a);
    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("R134a");
     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;
     for(double T=5;T<30;T+=1)
     {f.func(T);}
      r= golden(f,10.0,30.0,1.0e-10,1.0e-5,0);
      String s="Fibonacchi (Golden search) optimisation : \n";
      s+="T2 = "+r+"\ncop = "+(-f.func(r));
      s+=f.re.toString();
      System.out.println(s);
```

```
    f.re.plot(-10.0);
    }
}
```

```
---------- Capture Output ----------
> "C:\java\bin\javaw.exe" ref_cycle4_opt
Fibonacchi (Golden search) optimisation :
T2 = 19.43946807215391
cop = 3.757871203828108
two stage flash point refrigeration cycle
Flash ratio : 0.18167692296202828
 compressor1 work = 18.516264771685925 kW
 compressor2 work = 20.291177231408597 kW
 total compressor work = 38.80744200309452 kW
 isentropic compressor 1 work = 14.813011817348752 kW
 isentropic compressor 2 work = 16.232941785126854 kW
 total isentropic compressor work = 31.045953602475606 kW
 condenser heat output = 184.82248772371486 kW
 evaporator heat input = 145.8333687976583 kW
 COP evaporator    = 3.757871203828108
 COP condenser     = 4.7625527008190085
 COP evaporator  carnot = 3.809014522613925
 COP condenser   carnot = 4.809014522613924
 h1 compressor 1 inlet = 403.8909091817481 kJ/kg
 T1 compressor 1 inlet = 4.452127613408595 derece C
 P1 compressor 1 inlet = 243.34658575187083 kPa
 s1 compressor 1 inlet = 1.7602021806315065 kJ/kgK
 x1 compressor 1 inlet = 2.0 kgvapor/kgtotal
 h2 compressor 1 exit = 426.5179928987416 kJ/kg
 T2 compressor 1 exit = 36.93818832020559 derece C
 P2 compressor 1 exit = 561.8684489855331 kPa
 s2 compressor 1 exit = 1.7749077205739434 kJ/kgK
 x2 compressor 1 exit = 0.0 kgvapor/kgtotal
 h5 compressor 2 inlet = 423.41810542627013 kJ/kg
 T5 compressor 2 inlet = 33.72338776914749 derece C
 P5 compressor 2 inlet = 561.8684489855331 kPa
 s5 compressor 2 inlet = 1.764858720106506 kJ/kgK
 x5 compressor 2 inlet = 0.0 kgvapor/kgtotal
 h6 compressor 2 exit = 443.7092826576787 kJ/kg
 T6 compressor 2 exit = 64.70707834270311 derece C
 P6 compressor 2 exit = 1159.9066190539932 kPa
 s6 compressor 2 exit = 1.7769371454077838 kJ/kgK
 x6 compressor 2 exit = 1.14082513680892 kgvapor/kgtotal
 h8 flash chamber input = 226.68089300388175 kJ/kg
 T8 flash chamber input = 19.439469392166934 derece C
 P8 flash chamber input = 561.8684489855331 kPa
 s8 flash chamber input = 1.0935386263910631 kJ/kgK
 x8 flash chamber input = 0.0 kgvapor/kgtotal
 h4 expansion valve exit-evaporator inlet = 225.68089300388175 kJ/kg
 T4 expansion valve exit-evaporator inlet = -6.104624271642147 derece C
 P4 expansion valve exit-evaporator inlet = 233.34658575187083 kPa
 s4 expansion valve exit-evaporator inlet = 1.0966925781238985 kJ/kgK
 x4 expansion valve exit-evaporator inlet = 0.166528687233415 kgvapor/kgtotal
```



R134a is used as the refrigeration fluid. As a next question, different refrigeration fluids can be compared with the same evaporation and condensation pressures (intermediate pressure will be optimised for each case to give the maximum COP for the given refrigeration fluid)

Program 10.2-3 graphic comparison applied to flash chamber two stage refrigeration cycle for different working fluids

```
§
```

COP of different refrigeration fluids

Refrigerant name	COP	T2
R12	4.210915644659164	19.064975670977535
R123	4.104217774498053	17.228572174111967
R124	4.026871371852699	18.72632651557373
R134a	3.757871203856632	19.43752381973284
R1234yf	4.017150001959193	19.290953253718726

Now let us apply the same one stage intermediate pressure optimisation to two stage air compressor investigated in the previous chapter.

Program 10.2-4 Two stage gas compressor intermediate pressure optimisation

```
import java.awt.Color;
class fc extends f_x
{  two_stage_gas_compressor ssc;
   String gasi;
   double mi;
   double cvr1i;
   double cvr2i;
   double T0i;
   double P0i;
   double T1i;
   double P1i;
   double P2i;
   double P3i;
   double ne1i;
   double nc1i;
   double ne2i;
   double nc2i;
   double Ni;
   public fc()
   {   gasi="air";
   mi=0.286034892;
   cvr1i=0.05;
```

```
            cvr2i=0.05;
            T0i=288.0; //degree K
            P0i=1.013; //bar first stage inlet pressure
            T1i=305.0; //degree K
            P1i=0.95;  //bar second stage inlet pressure
            P2i=4.0; //bar the value which should be optimised intermediate pressure
            P3i=7.0;   //bar exit pressure
            ne1i=1.3; //expansion polytropic coefficient first stage
            nc1i=1.3; //compression polytropic coefficient first stage
            ne2i=1.3; //expansion polytropic coefficient second stage
            nc2i=1.3; //compression polytropic coefficient second stage
            Ni=300.0; //compressor speed rpm (revolution per minute)
            ssc=new
          two_stage_gas_compressor(gasi,mi,cvr1i,cvr2i,T0i,P0i,T1i,P1i,P2i,P3i,ne1i,nc1i,ne2i,nc2i,Ni);
            System.out.println("P2="+P2i+"bar W="+ssc.W+" kW");
            }

            public double func (double P)
            { ssc=new
          two_stage_gas_compressor(gasi,mi,cvr1i,cvr2i,T0i,P0i,T1i,P1i,P,P3i,ne1i,nc1i,ne2i,nc2i,Ni);
              return ssc.W;
            }
          }


          public class two_stage_gas_compressor_opt1
          {
           public static void main(String arg[])
           {   fc f1=new fc();
           double a=2.0; //bar
           double b=6.5; //bar
           double P=ref_cycle4_opt.golden(f1,a,b,1.0e-5,1.0e-5,0);
           double W=f1.func(P);
           System.out.println("P= "+P+" bar"+" W = "+W+" kW");
           }
}
```

```
          ---------- Capture Output ----------
          > "C:\java\bin\javaw.exe" two_stage_gas_compressor_opt1
          P2=4.0bar W=57.653672837637075 kW
          P= 2.5787565792953444 bar W = 56.250123354011464 kW

> Terminated with exit code 0.
```

As it is seen from the results for P=4 bar intermediate pressure power requirement is W=57.653672837637075 kW The optimum intermediate pressure is P=2.578 bar and power requirement drops to W = 56.250123354011464 kW

In the next example, Multidimensional optimisation process will be investigated. In thermodynamic systems, usually more than one thermodynamic properties can be considered for optimisation process. In the flash chamber refrigeration cycle, let us consider condenser pressure and intermediate pressure to be independent variables, and investigate the process to obtain highest refrigeration site COP. The first consideration in such a problem is the searching space for each independent variable. For the example problem, it is assumed that condensation temperature is between 45 to 55 C and intermediate temperature is between 10 to 30 C. Of course it is clear to us that lowest condensation temperature is optimal as far as COP concern, but it is not and optimisation reality and it will be a good check point for us. We are also needed an optimisation algorithm. Bat algorithm will be used in this example.

Program 10.2-5 two stage flash chamber refrigeration cycle optimisation

```
          // OPTIMISATION bath algorithm multidimensiional method
          import java.util.*;
          import java.awt.*;
          import java.applet.Applet;
```

```
import java.awt.event.*;
import javax.swing.*;
class fd extends f_xj
{  refrigerantEN r;
   ref_cycle4 re;
   double m;
   double T,T6,dT1,dT7;
   String refrigerant;
   double P1,P2,P6;
   double dT1i; //degree C
   double etaisent2i;
   double etaisent6i;
   double dT7i; //degree C
   double dPevap1i;
   double dPevap2i;
   double dPcomp1i;
   double dPcond1i;
   double dPcond2i;
   double dPcond3i;
   double dh1i;
   double dh2i;
   public fd(String refi)
   {m=1.0;  // kg/s çevrim R134a debisi
   refrigerant=refi;
   r=new refrigerantEN(refrigerant);
   P1=Pres(-5.0);
   P2=Pres(15.0);
   P6=Pres(45.0);
   dT1i=10.0; //degree C
   etaisent2i=0.8;
   etaisent6i=0.8;
   dT7i=3.0; //degree C
   dPevap1i=5.0;
   dPevap2i=5.0;
   dPcomp1i=5.0;
   dPcond1i=5.0;
   dPcond2i=5.0;
   dPcond3i=5.0;
   dh1i=1.0;
   dh2i=1.0;
   re=new ref_cycle4(refrigerant,m,dT1i,P1,P2,etaisent2i,P6,etaisent6i,dT7i,
   dPevap1i,dPevap2i,dPcomp1i,dPcond1i,dPcond2i,dPcond3i,dh1i,dh2i);
}
public  double Pres(double T)
{
double a[]=r.property("tx",T,0.0);
double P1a=a[0];
return P1a;
}
   public double func (double x[])
   {   T=x[0];
      T6=x[1];
      //dT1=x[1];
      //dT7=x[2];
               P2=Pres(T);
               P6=Pres(T6);
               re=new ref_cycle4("R134a",m,dT1i,P1,P2,etaisent2i,P6,etaisent6i,dT7i,
      dPevap1i,dPevap2i,dPcomp1i,dPcond1i,dPcond2i,dPcond3i,dh1i,dh2i);
               re.cycle();
               double y=re.COP_evaporator;
               if(y<0.0) y=0.0;
               //System.out.println("T2="+x[0]+"T6="+T6+"COP="+y);
      return -y;
   }
}
```

```
public class ref_cycle4_opt2
{
    public static void main (String args[])
    {
                fd f=new fd("R134a");
        double[] xmin={10.0,45.0};
        double[] xmax={30.0,55.0};
                    bat  b=new bat(f,20,xmin,xmax,10);
        double x0[]=b.xmin();
        String s="bat search : \n"+b.toString()  ;
        System.out.println(s);
    }
}
```

```
---------- Capture Output ----------
> "C:\java\bin\javaw.exe" ref_cycle4_opt2
bat search :
ymin = -3.7578667917271122
xmin = 19.4545614408934  45.00004766510894

> Terminated with exit code 0.
```

Since the bat algorithm statistical in nature, the result will change slightly in each run.

```
---------- Capture Output ----------
> "C:\java\bin\javaw.exe" ref_cycle4_opt2
bat search :
ymin = -3.757870971919903
xmin = 19.447771511798642 45.00000191731657
> Terminated with exit code 1.
```

If the same result (exactly) is searched statistical method result can be smoothed by using a geometric optimisation method such as Nelder & Mead method. If 3 variable, evaporator temperature, condenser temperature and intermediate temperature is required to give the maximum refrigeration COP:

Program 10.2-6 two stage flash chamber refrigeration cycle optimisation with 3 variable, bat search method

```
// OPTIMISATION bat algorithm multidimensional method
import java.util.*;
import java.awt.*;
import java.applet.Applet;
import java.awt.event.*;
import javax.swing.*;
class fd extends f_xj
{  refrigerantEN r;
  ref_cycle4 re;
  double m;
  double T,T6,T1,dT1,dT7;
  String refrigerant;
  double P1,P2,P6;
  double dT1i; //degree C
  double etaisent2i;
  double etaisent6i;
  double dT7i; //degree C
  double dPevap1i;
  double dPevap2i;
  double dPcomp1i;
  double dPcond1i;
  double dPcond2i;
  double dPcond3i;
  double dh1i;
  double dh2i;
  public fd(String refi)
  {m=1.0;  // kg/s çevrim R134a debisi
  refrigerant=refi;
  r=new refrigerantEN(refrigerant);
  P1=Pres(-5.0);
```

```
            P2=Pres(15.0);
            P6=Pres(45.0);
            dT1i=10.0; //degree C
            etaisent2i=0.8;
            etaisent6i=0.8;
            dT7i=3.0; //degree C
            dPevap1i=5.0;
            dPevap2i=5.0;
            dPcomp1i=5.0;
            dPcond1i=5.0;
            dPcond2i=5.0;
            dPcond3i=5.0;
            dh1i=1.0;
            dh2i=1.0;
            re=new ref_cycle4(refrigerant,m,dT1i,P1,P2,etaisent2i,P6,etaisent6i,dT7i,
            dPevap1i,dPevap2i,dPcomp1i,dPcond1i,dPcond2i,dPcond3i,dh1i,dh2i);
        }
        public  double Pres(double T)
        {
        double a[]=r.property("tx",T,0.0);
        double P1a=a[0];
        return P1a;
        }
          public double func (double x[])
          {   T=x[0];
              T6=x[1];
              T1=x[2];
              //dT1=x[1];
              //dT7=x[2];
                      P2=Pres(T);
                      P6=Pres(T6);
                      P1=Pres(T1);
                      re=new ref_cycle4("R134a",m,dT1i,P1,P2,etaisent2i,P6,etaisent6i,dT7i,
             dPevap1i,dPevap2i,dPcomp1i,dPcond1i,dPcond2i,dPcond3i,dh1i,dh2i);
                      re.cycle();
                      double y=re.COP_evaporator;
                      if(y<0.0) y=0.0;
                      //System.out.println("T2="+x[0]+"T6="+T6+"COP="+y);
              return -y;
          }
        }


        public class ref_cycle4_opt2
        {
            public static void main (String args[])
            {
                        fd f=new fd("R134a");
                double[] xmin={10.0,45.0,-10.0};
                double[] xmax={30.0,55.0,-5.0};
                            bat  b=new bat(f,20,xmin,xmax,10);
                double x0[]=b.xmin();
                String s="bat search : \n"+b.toString()  ;
                System.out.println(s);
            }
}
```

```
---------- Capture Output ----------
> "C:\java\bin\javaw.exe" ref_cycle4_opt2
bat search :
ymin = -3.757806621095091
xmin = 19.698271944900583 45.00010535980363 -5.0000657778552595

> Terminated with exit code 0.
```

Different optimisation methods can be utilised to solve the problem, in the example particle swarm algorithm is used to solve the optimum point

Program 10.2-7 Two stage flash chamber refrigeration problem with particle swarm algorithm solution

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

class fd extends f_xj
{  refrigerantEN r;
  ref_cycle4 re;
  double m;
  double T,T6,T1,dT1,dT7;
  String refrigerant;
  double P1,P2,P6;
  double dT1i; //degree C
  double etaisent2i;
  double etaisent6i;
  double dT7i; //degree C
  double dPevap1i;
  double dPevap2i;
  double dPcomp1i;
  double dPcond1i;
  double dPcond2i;
  double dPcond3i;
  double dh1i;
  double dh2i;
  int i;
  public fd(String refi)
  {m=1.0;  // kg/s çevrim R134a debisi
  refrigerant=refi;
  r=new refrigerantEN(refrigerant);
  P1=Pres(-5.0);
  P2=Pres(15.0);
  P6=Pres(45.0);
  dT1i=10.0; //degree C
  etaisent2i=0.8;
  etaisent6i=0.8;
  dT7i=3.0; //degree C
  dPevap1i=5.0;
  dPevap2i=5.0;
  dPcomp1i=5.0;
  dPcond1i=5.0;
  dPcond2i=5.0;
  dPcond3i=5.0;
  dh1i=1.0;
  dh2i=1.0;
  re=new ref_cycle4(refrigerant,m,dT1i,P1,P2,etaisent2i,P6,etaisent6i,dT7i,
  dPevap1i,dPevap2i,dPcomp1i,dPcond1i,dPcond2i,dPcond3i,dh1i,dh2i);
  i=0;
}
public  double Pres(double T)
{
double a[]=r.property("tx",T,0.0);
double P1a=a[0];
return P1a;
}
  public double func (double x[])
```

```
  {   T=x[0];
      T6=x[1];
      //T1=x[2];
      //dT1=x[1];
      //dT7=x[2];
              P2=Pres(T);
              P6=Pres(T6);
              //P1=Pres(T1);
              //dT1i=x[3];
              re=new ref_cycle4("R134a",m,dT1i,P1,P2,etaisent2i,P6,etaisent6i,dT7i,
      dPevap1i,dPevap2i,dPcomp1i,dPcond1i,dPcond2i,dPcond3i,dh1i,dh2i);
              re.cycle();
              double y=re.COP_evaporator;
              if(y<0.0) y=0.0;

              i++;
              if(i%10==0) System.out.println("i = "+i+" T2 = "+x[0]+" T6 = "+T6+" COP = "+y);
      return -y;
  }
}

public class particleswarmtest
{
public static void main(String arg[])
{


fd ff=new fd("R134a");
       double[] xmin={10.0,45.0};
       double[] xmax={30.0,50.0};
particle_swarm1 ps=new particle_swarm1(ff,50,xmin,xmax,2.0,2.0,10);
String s="Particle swarm optimisation  : \n"+ps.toString();
    String s2="Particle swarm optimization method ";
    System.out.println(s);
}
}
```

```
---------- Capture Output ----------
> "C:\java\bin\javaw.exe" particleswarmtest
i = 10 T2 = 22.758458974575746 T6 = 47.515242185678254 COP = 3.5424074520554076
i = 20 T2 = 10.11876883682352 T6 = 49.033392163400435 COP = 3.341887937086461
i = 30 T2 = 25.8756265668614 T6 = 46.61485634131441 COP = 3.59762047530829
i = 40 T2 = 22.432473344237913 T6 = 48.85114881854379 COP = 3.4389672499808066
i = 50 T2 = 26.05916501567809 T6 = 46.42779354107809 COP = 3.61089093228496
i = 60 T2 = 23.12152563160804 T6 = 48.24573087454522 COP = 3.484302976180071
i = 70 T2 = 10.690273589145171 T6 = 49.07399476356091 COP = 3.3463714211302893
i = 80 T2 = 26.858696031820685 T6 = 47.473977039090094 COP = 3.523460843629756
i = 90 T2 = 22.690749546769243 T6 = 49.77084720469877 COP = 3.3694709422111537
i = 100 T2 = 26.160079260691738 T6 = 47.33464767241985 COP = 3.5395658407491433
i = 110 T2 = 13.824128984640769 T6 = 45.0 COP = 3.7350463804299774
i = 120 T2 = 22.47551236834652 T6 = 45.0 COP = 3.7512200116982064
i = 130 T2 = 15.227713559766013 T6 = 45.57890747585669 COP = 3.692326322680936
i = 140 T2 = 11.40674511615716 T6 = 47.698435884872765 COP = 3.470576839680356
i = 150 T2 = 20.30618694319371 T6 = 46.777827366689536 COP = 3.6052429747235895
i = 160 T2 = 10.0 T6 = 45.0 COP = 3.6947101110258136
i = 170 T2 = 26.912278145870197 T6 = 45.0 COP = 3.717181989236496
i = 180 T2 = 15.231564735632908 T6 = 45.0 COP = 3.744897011817181
i = 190 T2 = 10.935905541157554 T6 = 45.965741961707415 COP = 3.617337562751933
```

```
i = 200 T2 = 17.33633642999306 T6 = 45.0 COP = 3.7546332301460494
i = 210 T2 = 25.683143517560485 T6 = 45.0 COP = 3.729555008219542
i = 220 T2 = 22.845792076395437 T6 = 45.0 COP = 3.7494969013576056
i = 230 T2 = 17.493319253394304 T6 = 45.0 COP = 3.7550989882779926
i = 240 T2 = 23.07506028404878 T6 = 45.08420838725151 COP = 3.7411394366061548
i = 250 T2 = 17.91579574182776 T6 = 45.0 COP = 3.7561742969750265
i = 260 T2 = 27.861332659689257 T6 = 45.0 COP = 3.7061188723281777
i = 270 T2 = 18.763398660138154 T6 = 45.0 COP = 3.7575388141696617
i = 280 T2 = 22.39015930223401 T6 = 45.0 COP = 3.751588865530541
i = 290 T2 = 25.203393059550407 T6 = 45.0 COP = 3.733771632225613
i = 300 T2 = 19.23911863241534 T6 = 45.0 COP = 3.7578422335342916
i = 310 T2 = 28.55517501635207 T6 = 45.0 COP = 3.69722538144381
i = 320 T2 = 17.662382135122822 T6 = 45.0 COP = 3.7555606058458344
i = 330 T2 = 22.41009629584867 T6 = 45.0 COP = 3.751503656483436
i = 340 T2 = 20.750901330806155 T6 = 45.0 COP = 3.756628442202176
i = 350 T2 = 20.120207856857874 T6 = 45.0 COP = 3.7575352891017455
i = 360 T2 = 15.91649960755365 T6 = 45.0 COP = 3.7488320056340876
i = 370 T2 = 17.664428141865493 T6 = 45.0 COP = 3.7555659426491474
i = 380 T2 = 19.909178987991226 T6 = 45.0 COP = 3.757710926178213
i = 390 T2 = 17.86328013493673 T6 = 45.0 COP = 3.756054868695251
i = 400 T2 = 20.120926249314323 T6 = 45.0 COP = 3.757534586305722
i = 410 T2 = 15.241887281269735 T6 = 45.0 COP = 3.7449590145133667
i = 420 T2 = 18.02278912096939 T6 = 45.0 COP = 3.756405023380269
i = 430 T2 = 18.60310766784639 T6 = 45.0 COP = 3.757361911067386
i = 440 T2 = 17.021270281797698 T6 = 45.0 COP = 3.753591468743827
i = 450 T2 = 19.92603423806435 T6 = 45.0 COP = 3.7576992524479707
i = 460 T2 = 20.744981431286867 T6 = 45.0 COP = 3.756639615220462
i = 470 T2 = 19.11869927869384 T6 = 45.0 COP = 3.757796689811217
i = 480 T2 = 19.48882753799289 T6 = 45.0 COP = 3.7578693826202745
i = 490 T2 = 17.829362350883756 T6 = 45.0 COP = 3.7559755744122785
i = 500 T2 = 19.524041018532696 T6 = 45.0 COP = 3.7578659227431634
i = 510 T2 = 22.943669400286744 T6 = 45.0 COP = 3.7490079390648003
i = 520 T2 = 19.96636759881753 T6 = 45.0 COP = 3.7576696565962138
i = 530 T2 = 19.835168353257817 T6 = 45.0 COP = 3.757757342662764
i = 540 T2 = 19.104676999500835 T6 = 45.0 COP = 3.7577900116096856
i = 550 T2 = 19.194928073391537 T6 = 45.0 COP = 3.757827978998961
Particle swarm optimisation  :
ymin = -3.7578708598526034
xmin = 19.460467399174078 45.0
y global min = -3.7578711792561217
xmin = 19.433218645077105 45.0

> Terminated with exit code 0.
```

A simple stage supercritical CO2 refrigeration cycle optimisation problem, evaporator degree of superheat, one phase heat exchanger (same duty of condenser for subcritical refrigeration cycle) and heat exchanger pressure is optimised in this problem. Particle swarm optimisation is used as the basic optimisation method.

Program 10.2-8 one stage supercritical CO2 refrigeration problem with particle swarm algorithm solution

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

class fx extends f_xj
```

```
{  double m;       //  kg/s çevrim debisi
double P2;         //  kPa compressor exit pressure
double P1;         //  kPa compressor exit pressure = optimisation value
double dT1i;       //  evaporator superheatdegree K = optimisation value
double eta_isent1i; //  isentropic  efficiency
double dT6i;       //  condenser cooling degree K = optimisation variable
double dPevap1i;   //  evaporator two pahse section pressure drop kPa
double dPevap2i;   //  evaporator vapor section pressure drop kPa
double dPcond1i;   //  kPa vapor section pressure drop
double dPcond2i;   //  kPa two phase section pressure drop
double dPcond3i;   //  kPa liquid section pressure drop
double dhi;        //  kJ/kgK
int i;
refISO_cycle3 r;
  public fx(String refi)
  {m=0.057;          //  kg/s çevrim debisi
   P2=8000.0;        //  kPa pump exit pressure =optimisation value
P1=2000.0 ;        //  kPa pump inlet pressure
dT1i=10.0;         //  evaporator superheatdegree K
eta_isent1i=0.8;   //  isentropic  efficiency
dT6i=48;           //  condenser cooling degree K =optimisation value
dPevap1i=10.0;     //  evaporator two pahse section pressure drop kPa
dPevap2i=10.0;     //  evaporator vapor section pressure drop kPa
dPcond1i=10.0;     //  kPa vapor section pressure drop
dPcond2i=10.0;     //  kPa two phase section pressure drop
dPcond3i=10.0;     //  kPa liquid section pressure drop
dhi=10.0;          //  kJ/kgK
refISO_cycle3 r=new
refISO_cycle3("R744",m,dT1i,P1,eta_isent1i,P2,dT6i,dPevap1i,dPevap2i,dPcond1i,dPcond2i,dPcond3i,dhi);
r.cycle();
i=0;
}

  public double func (double x[])
  {   P2=x[0];
      dT1i=x[1];
      dT6i=x[2];
              r=new
refISO_cycle3("R744",m,dT1i,P1,eta_isent1i,P2,dT6i,dPevap1i,dPevap2i,dPcond1i,dPcond2i,dPcond3i,dhi);
      r.cycle();
              double y=r.COP_condenser;
              if(y<0.0) y=0.0;
              i++;
              if(i%10==0) System.out.println("i = "+i+" P2 = "+P2+" dT1 = "+dT1i+" dT6 = "+dT6i+" COP =
"+y);
      return -y;
  }
}

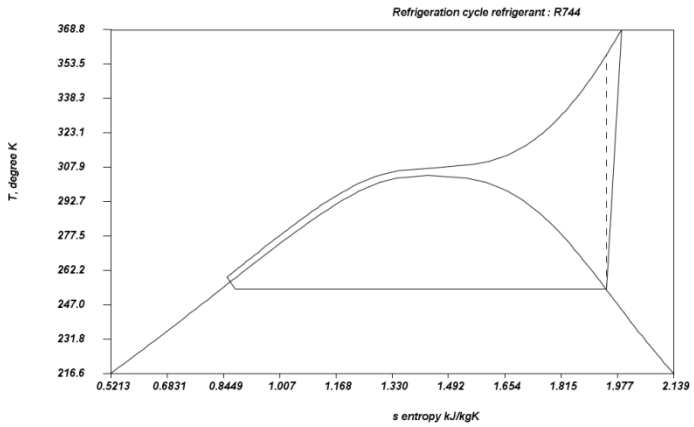
public class particleswarmtest2
{
public static void main(String arg[])
{


fx ff=new fx("R134a");
       double[] xmin={8000.0,0.0,5.0};
       double[] xmax={9000.0,50.0,45.0};
```

```
particle_swarm1 ps=new particle_swarm1(ff,50,xmin,xmax,2.0,2.0,10);
String s="Particle swarm optimisation  : \n"+ps.toString();
    String s2="Particle swarm optimization method ";
    System.out.println(s);
    ff.r.plot(50.0);
}
}
```

```
---------- Capture Output ----------
> "C:\java\bin\javaw.exe" particleswarmtest2
i = 10 P2 = 8133.602478707649 dT1 = 23.556949598019 45 dT6 = 39.809212254568955 COP =
4.124684348731806
i = 20 P2 = 8767.488767874336 dT1 = 20.415849896315 58 dT6 = 40.63953925907256 COP =
3.980338236983231
i = 30 P2 = 8429.491785661363 dT1 = 22.829629011057417 dT6 = 41.07948272359955 COP =
4.068530258415389
i = 40 P2 = 8057.721770233712 dT1 = 13.309182585062358 dT6 = 44.52606682581234 COP =
4.3415804223455785
i = 50 P2 = 8804.384288556223 dT1 = 48.27676102059103 dT6 = 39.74817187801134 COP =
3.8102558941548725
i = 60 P2 = 8133.794649011889 dT1 = 23.81310322663671 dT6 = 40.19421479631644 COP =
4.131843260448622
i = 70 P2 = 8767.843435058916 dT1 = 21.3320852609697 dT6 = 41.607507504199965 COP =
3.99524168906167
i = 80 P2 = 8430.143166003125 dT1 = 23.316069463153642 dT6 = 41.67555506520334 COP =
4.078405781193588
i = 90 P2 = 8058.377030884775 dT1 = 13.933183619820763 dT6 = 45.0 COP = 4.347177957909526
i = 100 P2 = 8804.643289748548 dT1 = 49.087647671986126 dT6 = 39.7739530326892 COP =
3.8075579286423458
i = 110 P2 = 8117.6331937339755 dT1 = 8.1514167681909 dT6 = 44.272820521252314 COP =
4.36500524377585
i = 120 P2 = 8000.0 dT1 = 0.0 dT6 = 45.0 COP = 4.52216598959036
i = 130 P2 = 8000.0 dT1 = 3.2182170149211977 dT6 = 45.0 COP = 4.480738295897946
i = 140 P2 = 8114.720860699665 dT1 = 3.9492738798676985 dT6 = 45.0 COP = 4.431082041464701
i = 150 P2 = 8663.63437833573 dT1 = 20.760486806556436 dT6 = 45.0 COP = 4.101884387862649
i = 160 P2 = 8049.170397835771 dT1 = 0.0 dT6 = 45.0 COP = 4.504239950426327
i = 170 P2 = 8000.0 dT1 = 0.0 dT6 = 45.0 COP = 4.52216598959036
i = 180 P2 = 8000.0 dT1 = 0.0 dT6 = 45.0 COP = 4.52216598959036
i = 190 P2 = 8074.698798590122 dT1 = 0.0 dT6 = 45.0 COP = 4.495037334276547
i = 200 P2 = 8000.0 dT1 = 0.0 dT6 = 45.0 COP = 4.52216598959036
i = 210 P2 = 8000.0 dT1 = 0.0 dT6 = 45.0 COP = 4.52216598959036
i = 220 P2 = 8000.0 dT1 = 0.0 dT6 = 45.0 COP = 4.52216598959036
i = 230 P2 = 8000.0 dT1 = 0.0 dT6 = 45.0 COP = 4.52216598959036
i = 240 P2 = 8000.0 dT1 = 0.0 dT6 = 45.0 COP = 4.52216598959036
i = 250 P2 = 8000.0 dT1 = 0.0 dT6 = 45.0 COP = 4.52216598959036
i = 260 P2 = 8000.0 dT1 = 0.0 dT6 = 45.0 COP = 4.52216598959036
i = 270 P2 = 8000.0 dT1 = 0.0 dT6 = 45.0 COP = 4.52216598959036
i = 280 P2 = 8000.0 dT1 = 0.0 dT6 = 45.0 COP = 4.52216598959036
i = 290 P2 = 8000.0 dT1 = 0.0 dT6 = 45.0 COP = 4.52216598959036
i = 300 P2 = 8000.0 dT1 = 0.0 dT6 = 45.0 COP = 4.52216598959036
i = 310 P2 = 8000.0 dT1 = 0.0 dT6 = 45.0 COP = 4.52216598959036
i = 320 P2 = 8000.0 dT1 = 0.0 dT6 = 45.0 COP = 4.52216598959036
i = 330 P2 = 8000.0 dT1 = 0.0 dT6 = 45.0 COP = 4.52216598959036
i = 340 P2 = 8000.0 dT1 = 0.0 dT6 = 45.0 COP = 4.52216598959036
i = 350 P2 = 8000.0 dT1 = 0.0 dT6 = 45.0 COP = 4.52216598959036
i = 360 P2 = 8000.0 dT1 = 0.0 dT6 = 45.0 COP = 4.52216598959036
i = 370 P2 = 8000.0 dT1 = 0.0 dT6 = 45.0 COP = 4.52216598959036
i = 380 P2 = 8000.0 dT1 = 0.0 dT6 = 45.0 COP = 4.52216598959036
```

```
i = 390 P2 = 8000.0 dT1 = 0.0 dT6 = 45.0 COP = 4.52216598959036
i = 400 P2 = 8000.0 dT1 = 0.0 dT6 = 45.0 COP = 4.52216598959036
i = 410 P2 = 8000.0 dT1 = 0.0 dT6 = 45.0 COP = 4.52216598959036
i = 420 P2 = 8000.0 dT1 = 0.0 dT6 = 45.0 COP = 4.52216598959036
i = 430 P2 = 8000.0 dT1 = 0.0 dT6 = 45.0 COP = 4.52216598959036
i = 440 P2 = 8000.0 dT1 = 0.0 dT6 = 45.0 COP = 4.52216598959036
i = 450 P2 = 8000.0 dT1 = 0.0 dT6 = 45.0 COP = 4.52216598959036
i = 460 P2 = 8000.0 dT1 = 0.0 dT6 = 45.0 COP = 4.52216598959036
i = 470 P2 = 8000.0 dT1 = 0.0 dT6 = 45.0 COP = 4.52216598959036
i = 480 P2 = 8000.0 dT1 = 0.0 dT6 = 45.0 COP = 4.52216598959036
i = 490 P2 = 8000.0 dT1 = 0.0 dT6 = 45.0 COP = 4.52216598959036
i = 500 P2 = 8000.0 dT1 = 0.0 dT6 = 45.0 COP = 4.52216598959036
i = 510 P2 = 8000.0 dT1 = 0.0 dT6 = 45.0 COP = 4.52216598959036
i = 520 P2 = 8000.0 dT1 = 0.0 dT6 = 45.0 COP = 4.52216598959036
i = 530 P2 = 8000.0 dT1 = 0.0 dT6 = 45.0 COP = 4.52216598959036
i = 540 P2 = 8000.0 dT1 = 0.0 dT6 = 45.0 COP = 4.52216598959036
i = 550 P2 = 8000.0 dT1 = 0.0 dT6 = 45.0 COP = 4.52216598959036
Particle swarm optimisation  :
ymin = -4.52216598959036
xmin = 8000.0 0.0 45.0
y global min = -4.52216598959036
xmin = 8000.0 0.0 45.0
```



As we can predict from our thermodynamic cycle knowledge, minimum heat exchanger pressure, maximum subcooling temperature and minimum superheat temperature came as the optimum solution. Note that cycle is solved 550 times! to get the optimal solution

If the same problem is searched by cuckoo's nest search method the results are

Program 10.2-9 one stage supercritical CO2 refrigeration problem with cuckoo's nest algorithm solution

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

public class cuckoosearchtest2
{
public static void main(String args[])
{
fx ff=new fx("R134a");
```

```
int n=500;//Number of nests
double pa=0.24546557654;// Discovery rate of alien eggs/solutions
double tol=1e-9;//minimum values of tolerance leads to better results
      double[] xmin={8000.0,0.0,5.0};
      double[] xmax={9000.0,50.0,45.0};
cuckoo_search1 cc=new cuckoo_search1(ff,n,pa,tol,xmin,xmax);
cc.solution();
}
```

```
---------- Capture Output ----------
> "C:\java\bin\javaw.exe" cuckoosearchtest2
i = 10 P2 = 8091.352983297842 dT1 = 47.71486525996571 dT6 = 13.586209253598112 COP =
3.412310625602257
i = 20 P2 = 8382.736006049632 dT1 = 8.80732522196892 dT6 = 15.664765847183094 COP =
3.507009341707381
i = 30 P2 = 8106.318860622454 dT1 = 11.745623100518287 dT6 = 34.13715552484419 COP =
4.078220226451497
i = 40 P2 = 8223.279125422721 dT1 = 28.77493954186242 dT6 = 18.573364912133364 COP =
3.554887510176952
i = 50 P2 = 8335.693121989167 dT1 = 36.208770235080074 dT6 = 32.16944228210947 COP =
3.8259333957775583
i = 60 P2 = 8753.424291681307 dT1 = 9.82854747477771 dT6 = 15.384246278126511 COP =
3.415374677383789
i = 70 P2 = 8513.938221829065 dT1 = 2.4469702221994902 dT6 = 36.330803161911206 COP =
4.0948059706845
i = 80 P2 = 8391.707213080334 dT1 = 13.955375738206083 dT6 = 10.941385618906248 COP =
3.337867759729371
i = 90 P2 = 8304.598425897122 dT1 = 14.128453638518879 dT6 = 15.483650861956564 COP =
3.497698280611008
i = 100 P2 = 8505.661726416254 dT1 = 49.62517072316135 dT6 = 39.18601702950549 COP =
3.871674434297711
i = 110 P2 = 8353.522264672853 dT1 = 4.755221796850173 dT6 = 38.697878602617976 COP =
4.181027244315151
i = 120 P2 = 8830.990395420595 dT1 = 41.896089654870224 dT6 = 21.403389696993635 COP =
3.4509810505964413
i = 130 P2 = 8861.754438117903 dT1 = 41.13649182466317 dT6 = 33.14042113374475 COP =
3.694133722902154
i = 140 P2 = 8625.721412261862 dT1 = 39.44575139435075 dT6 = 28.937942003257447 COP =
3.6688732275060714
i = 150 P2 = 8745.693703392417 dT1 = 30.052424904993853 dT6 = 23.639666731411776 COP =
3.555165267367716
i = 160 P2 = 8396.967741061637 dT1 = 20.855050002904413 dT6 = 32.76825620050268 COP =
3.8984504033317147
i = 170 P2 = 8046.839740169595 dT1 = 32.64762191185036 dT6 = 12.116844492776151 COP =
3.4051409647138784
i = 180 P2 = 8867.006137034845 dT1 = 40.25548297751337 dT6 = 16.122217258820093 COP =
3.326300327544658
i = 190 P2 = 8566.928471249015 dT1 = 11.845828818332338 dT6 = 18.240596459736768 COP =
3.528314552167801
i = 200 P2 = 8893.244920205247 dT1 = 28.671495131258386 dT6 = 28.851847035686717 COP =
3.6453854328422444
i = 210 P2 = 8736.482206683982 dT1 = 32.26675023649302 dT6 = 14.574997044976321 COP =
3.329729773612047
i = 220 P2 = 8323.251445223696 dT1 = 42.04751558568511 dT6 = 13.576366051291139 COP =
3.3696177802354823
i = 230 P2 = 8475.77695140316 dT1 = 49.75282326346137 dT6 = 30.15534402441364 COP =
3.6980688558404173
```

i = 240 P2 = 8251.361285588395 dT1 = 20.045080275198224 dT6 = 33.64109503480928 COP = 3.9654047118436027
i = 250 P2 = 8276.956280978833 dT1 = 18.678244282634644 dT6 = 5.840111825782275 COP = 3.1717685063915977
i = 260 P2 = 8847.4928955619 dT1 = 41.22862017194432 dT6 = 41.72885180358472 COP = 3.8669881276303038
i = 270 P2 = 8924.497857657023 dT1 = 9.714515440505561 dT6 = 22.951513486770843 COP = 3.5876601473436884
i = 280 P2 = 8851.643240845746 dT1 = 30.229229138946224 dT6 = 25.49959449492732 COP = 3.57364661051039
i = 290 P2 = 8178.464362774559 dT1 = 18.666210683099017 dT6 = 13.631192876280803 COP = 3.4547120772318536
i = 300 P2 = 8969.141212180035 dT1 = 40.88388346500875 dT6 = 12.923082134995836 COP = 3.229129989182223
i = 310 P2 = 8370.948476587342 dT1 = 8.367627251807026 dT6 = 37.00161946144086 COP = 4.098656668774356
i = 320 P2 = 8762.08721013313 dT1 = 6.1933428157739225 dT6 = 15.008441450490139 COP = 3.417955740443847
i = 330 P2 = 8210.8467590501 dT1 = 10.445440467971007 dT6 = 29.63781982385095 COP = 3.938445550089155
i = 340 P2 = 8211.509099534835 dT1 = 11.22511168006275 dT6 = 7.423527745740501 COP = 3.2561870903094037
i = 350 P2 = 8155.575001220082 dT1 = 22.004413008518192 dT6 = 12.018153281112735 COP = 3.4006391867368198
i = 360 P2 = 8922.616092831397 dT1 = 8.808577349571467 dT6 = 25.036083018591185 COP = 3.647301722550468
i = 370 P2 = 8490.563356357185 dT1 = 13.982711863107628 dT6 = 14.536894743808517 COP = 3.4290964687064505
i = 380 P2 = 8962.133480137365 dT1 = 10.027645088743897 dT6 = 44.69773315418461 COP = 4.100140953659381
i = 390 P2 = 8758.34661561745 dT1 = 23.050268266891283 dT6 = 37.325448983480705 COP = 3.8932283506561705
i = 400 P2 = 8552.053269689302 dT1 = 42.11186119504537 dT6 = 41.630918019374406 COP = 3.93885866126458
i = 410 P2 = 8398.791175015152 dT1 = 27.488139637526533 dT6 = 17.54308143942262 COP = 3.490373259581237
i = 420 P2 = 8298.236843322033 dT1 = 25.360261131567302 dT6 = 23.56794934481707 COP = 3.677562287575247
i = 430 P2 = 8400.477321298831 dT1 = 9.488734904992269 dT6 = 5.962127048424812 COP = 3.170552402399478
i = 440 P2 = 8305.157351320802 dT1 = 28.202851587109702 dT6 = 14.330400759851742 COP = 3.4216269092950515
i = 450 P2 = 8420.481328583941 dT1 = 31.756247518796854 dT6 = 31.586681466221588 COP = 3.8093776081904736
i = 460 P2 = 8419.671605639596 dT1 = 15.412851392910449 dT6 = 34.82614714426349 COP = 3.975988133556496
i = 470 P2 = 8205.779097811053 dT1 = 5.163574954277039 dT6 = 18.707850451080002 COP = 3.6612281095456636
i = 480 P2 = 8957.061936581124 dT1 = 8.828513370080854 dT6 = 34.48040001839983 COP = 3.873417595284618
i = 490 P2 = 8244.219198860153 dT1 = 46.536468694027825 dT6 = 15.212707468771027 COP = 3.4221047310135044
i = 500 P2 = 8591.691059357192 dT1 = 32.72656986587529 dT6 = 19.906158370719748 COP = 3.4920525028450853
Optimum value of a function = -4.347336617784968
x[0] = 8096.31907817711
x[1] = 5.720689732974033
x[2] = 42.29184137726958

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

If the same problem is solved bybat's search algoritma:

Program 10.2-10 one stage supercritical CO2 refrigeration problem with bat search algorithm solution

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


public class batsearchtest2
{
public static void main(String args[])
{
fx ff=new fx("R134a");
int n=500;//Number of nests
double pa=0.24546557654;// Discovery rate of alien eggs/solutions
double tol=1e-9;//minimum values of tolerance leads to better results
      double[] xmin={8000.0,0.0,5.0};
      double[] xmax={9000.0,50.0,45.0};
      bat  b=new bat(ff,20,xmin,xmax,10);
         double x0[]=b.xmin();
         String s="bat search : \n"+b.toString()  ;
         System.out.println(s);
}
```

```
---------- Capture Output ----------
> "C:\java\bin\javaw.exe" batsearchtest2
i = 10 P2 = 8754.93863892307 dT1 = 37.782203418647356 dT6 = 17.936933422691467 COP =
3.3969390337869547
i = 20 P2 = 8696.366638634187 dT1 = 42.272256279048655 dT6 = 5.837028623337082 COP =
3.082480459626122
i = 30 P2 = 8512.008777048779 dT1 = 28.727592425252663 dT6 = 13.754536247803404 COP =
3.3605686402778114
i = 40 P2 = 8372.2365095891 dT1 = 26.03209919615003 dT6 = 5.172836675099516 COP =
3.1232830998033005
i = 50 P2 = 8207.701978058398 dT1 = 7.78988354949682 dT6 = 44.29201256581437 COP =
4.3387856111814935
i = 60 P2 = 8207.674335547616 dT1 = 7.1709657963024025 dT6 = 44.9376500972994 COP =
4.361440652168533
i = 70 P2 = 8207.701978058398 dT1 = 7.78988354949682 dT6 = 44.29201256581437 COP =
4.3387856111814935
i = 80 P2 = 8207.674335547616 dT1 = 7.1709657963024025 dT6 = 44.9376500972994 COP =
4.361440652168533
i = 90 P2 = 8239.256107102965 dT1 = 30.076910195133834 dT6 = 14.711895441624655 COP =
3.442487717082138
i = 100 P2 = 8675.523916694196 dT1 = 45.58999604777041 dT6 = 40.81984568648201 COP =
3.8745909404351933
i = 110 P2 = 8181.773290237398 dT1 = 6.2809673082484485 dT6 = 44.87347373641724 COP =
4.378201161565979
i = 120 P2 = 8195.086927440194 dT1 = 6.965265373598003 dT6 = 44.99980945240853 COP =
4.369444085399163
i = 130 P2 = 8177.303581097368 dT1 = 6.196834580477567 dT6 = 44.74580898031095 COP =
4.377413539253536
```

i = 140 P2 = 8418.911671982056 dT1 = 30.53462837577763 dT6 = 13.109386798669224 COP = 3.3579126454568087
i = 150 P2 = 8002.522625082559 dT1 = 2.394847784663459 dT6 = 5.489185609957204 COP = 3.2319962345554147
i = 160 P2 = 8012.863228367918 dT1 = 0.0019715075184847976 dT6 = 44.32008471988804 COP = 4.498988046163064
i = 170 P2 = 8008.413371222153 dT1 = 0.42370026596320787 dT6 = 44.77875291842441 COP = 4.507409875923624
i = 180 P2 = 8004.17522760248 dT1 = 0.0018254716090565328 dT6 = 44.99832539899887 COP = 4.520563333250339
i = 190 P2 = 8001.857808411476 dT1 = 0.00150010371899884 dT6 = 44.99869032616035 COP = 4.521428002383452
i = 200 P2 = 8649.987451285977 dT1 = 32.2981759788759 dT6 = 8.13530579060955 COP = 3.167773252391448
i = 210 P2 = 8207.412384771089 dT1 = 36.951240594645036 dT6 = 44.79309387394948 COP = 4.130201867160264
i = 220 P2 = 8000.007029438815 dT1 = 1.3732407345638732E-4 dT6 = 44.9993240507917 COP = 4.522143217975028
i = 230 P2 = 8009.442935303091 dT1 = 7.368534777449553E-4 dT6 = 44.84586806706962 COP = 4.514516357095029
i = 240 P2 = 8007.191206492038 dT1 = 0.001415669474959673 dT6 = 44.999155282680185 COP = 4.519485557210995
i = 250 P2 = 8134.208764289048 dT1 = 3.8763525018640443 dT6 = 16.618565281779293 COP = 3.619831757639393
i = 260 P2 = 8427.813375793286 dT1 = 44.19214703733014 dT6 = 18.445626383349392 COP = 3.4650563916331745
i = 270 P2 = 8000.016543727253 dT1 = 0.8532640751811917 dT6 = 44.99903464453306 COP = 4.510734024002571
i = 280 P2 = 8003.427067802323 dT1 = 5.833559226874962E-4 dT6 = 44.74548967201574 COP = 4.51399762981148
i = 290 P2 = 8000.042542455404 dT1 = 0.001336857686150618 dT6 = 44.948430927058496 COP = 4.520733750980706
i = 300 P2 = 8127.967246327756 dT1 = 49.642237613585756 dT6 = 44.152963514368466 COP = 4.078755294182301
i = 310 P2 = 8815.80895295915 dT1 = 3.733158995242719 dT6 = 23.685294833573856 COP = 3.6681544393737053
i = 320 P2 = 8907.027728757006 dT1 = 33.986592037466764 dT6 = 43.727549223982805 COP = 3.9267400110036106
i = 330 P2 = 8000.0193852456505 dT1 = 0.4992813239244565 dT6 = 44.98657682974813 COP = 4.515087489402474
i = 340 P2 = 8481.504208488683 dT1 = 1.2362184649953034 dT6 = 12.510831213255138 COP = 3.4177858939384826
i = 350 P2 = 8252.470763228452 dT1 = 48.4180021753349 dT6 = 16.95796561444383 COP = 3.4606427111069875
i = 360 P2 = 8453.545309370877 dT1 = 16.485075242144475 dT6 = 28.786060736712436 COP = 3.8113412417184933
i = 370 P2 = 8436.775647884508 dT1 = 23.896807196214912 dT6 = 10.716604168240144 COP = 3.2981599108787356
i = 380 P2 = 8001.858175618812 dT1 = 0.0023247005688067767 dT6 = 44.99813237569851 COP = 4.521401573337739
i = 390 P2 = 8000.011873175442 dT1 = 0.11381525588498939 dT6 = 44.999626836480026 COP = 4.520612497506967
i = 400 P2 = 8000.0336590764955 dT1 = 0.0013903113661808703 dT6 = 44.99822015059659 COP = 4.522086529566625
i = 410 P2 = 8504.810942375107 dT1 = 46.99871736474353 dT6 = 44.8506506409673 COP = 3.9926415720253705
i = 420 P2 = 8602.26796828321 dT1 = 0.43829598788129864 dT6 = 14.366908964588267 COP = 3.4584411218244706

```
i = 430 P2 = 8150.190776298264 dT1 = 8.47596769227945 dT6 = 15.028198182746667 COP =
3.5420161107535955
i = 440 P2 = 8000.036596954752 dT1 = 0.12146520147905511 dT6 = 44.99801972165909 COP =
4.52045665081331
i = 450 P2 = 8000.01839904496 dT1 = 0.2008906269835522 dT6 = 44.96732376754992 COP =
4.518562051389481
i = 460 P2 = 8414.674669188389 dT1 = 31.54954199283896 dT6 = 42.69240087245995 COP =
4.053969803928414
i = 470 P2 = 8766.413185687396 dT1 = 15.58199404564548 dT6 = 10.697804315419823 COP =
3.2548851788713162
i = 480 P2 = 8274.975844632252 dT1 = 8.763710843845363 dT6 = 38.58818624787916 COP =
4.164465067921379
i = 490 P2 = 8000.012196779647 dT1 = 0.2276215056400747 dT6 = 44.99965534654935 COP =
4.519079996685731
i = 500 P2 = 8000.035827273441 dT1 = 4.333512213115612E-4 dT6 = 44.999394511107226 COP =
4.522130541250826
i = 510 P2 = 8666.85198642515 dT1 = 8.866874873429392 dT6 = 29.021548062774883 COP =
3.810362457886978
i = 520 P2 = 8448.870724535846 dT1 = 24.466257411541537 dT6 = 39.24020415396255 COP =
4.011303312662225
i = 530 P2 = 8743.307374818261 dT1 = 15.616492080405076 dT6 = 39.12075943563516 COP =
3.986421095845116
i = 540 P2 = 8002.0456633580025 dT1 = 1.408832925706871E-5 dT6 = 44.87757310450371 COP =
4.518094812360777
i = 550 P2 = 8008.538932269252 dT1 = 0.001881266946187439 dT6 = 44.998835578585926 COP =
4.518976774444599
i = 560 P2 = 8355.557796394627 dT1 = 28.099144802966748 dT6 = 8.738564468049866 COP =
3.2447908638456755
i = 570 P2 = 8008.538932269252 dT1 = 0.001881266946187439 dT6 = 44.998835578585926 COP =
4.518976774444599
bat search :
ymin = -4.522151988402749
xmin = 8000.012472185228 2.1557191974610646E-4 44.99976029888722

> Terminated with exit code 0.
```

As another example, **Rankine cycle with closed feedwater heater** was investigated previously in power cycle section, if the problem treated to find second stage turbine inlet pressure to maximise cycle efficiency with bat algorithm

Program 10.2-11 Rankine power cycle optimisation with, bat search method

```
import java.awt.Color;
class fd extends f_xj
{  double m; //cycle mass flow rate kg/s
double T3; //first stage turbine inlet temperature C
double P3; //first stage turbine inlet pressure bar
double P4; //second stage turbine inlet pressure bar
double P5; //second stage turbine exit pressure bar
double etat1; //first stage turbine isentropic efficiency
double etap1; //first stage pump isentropic efficiency
double etap2; //second stage turbine isentropic efficiency
double dT;
rankine3 r;
  public fd()
  {   m=1.0; //cycle mass flow rate kg/s
T3=600.0; //first stage turbine inlet temperature C
P3=30.0; //first stage turbine inlet pressure bar
P4=5.0; //second stage turbine inlet pressure bar optimisation variable
```

```
P5=0.1; //second stage turbine exit pressure bar
etat1=1.0; //first stage turbine isentropic efficiency
etap1=0.8; //first stage pump isentropic efficiency
etap2=0.8; //second stage turbine isentropic efficiency
dT=5.0;
r=new rankine3(m,T3,P3,P4,P5,etat1,etap1,etap2,dT);
  }

  public double func (double x[])
  {   P4=x[0];
              r=new rankine3(m,T3,P3,P4,P5,etat1,etap1,etap2,dT);
              r.cycle();
     return -r.etacycle;
  }
}


public class rankine_opt1
{
 public static void main(String arg[])
 {        fd f=new fd();
          double[] xmin={1.0};
          double[] xmax={10.0};
                        bat  b=new bat(f,20,xmin,xmax,20);
          double x0[]=b.xmin();
          String s="bat search : \n"+b.toString()  ;
          System.out.println(s);
 }
}
```

```
---------- Capture Output ----------
> "C:\java\bin\javaw.exe" rankine_opt1
bat search :
ymin = -0.390556299567963
xmin = 3.5282084698296585

> Terminated with exit code 0.
```

When we originally looked at this problem (Program 3.3.6 Rankine cycle with closed feedwater heater test program) second stage turbine inlet(steam extraction) pressure was taken as 5 bar and total system efficiency was 31%. By changing extraction presure to 3.53 bar, system efficiency become 39%.

Of course as one variable optimisation problem, it can easily be solved by using one of the multi-variable algorithms

Program 10.2-12 Rankine power cycle optimisation with, golden search method

```
import java.awt.Color;
class fe extends f_x
{  double m; //cycle mass flow rate kg/s
double T3; //first stage turbine inlet temperature C
double P3; //first stage turbine inlet pressure bar
double P4; //second stage turbine inlet pressure bar
double P5; //second stage turbine exit pressure bar
double etat1; //first stage turbine isentropic efficiency
double etap1; //first stage pump isentropic efficiency
double etap2; //second stage turbine isentropic efficiency
double dT;
rankine3 r;
```

```
  public fe()
  {   m=1.0; //cycle mass flow rate kg/s
T3=600.0; //first stage turbine inlet temperature C
P3=30.0; //first stage turbine inlet pressure bar
P4=5.0; //second stage turbine inlet pressure bar optimisation variable
P5=0.1; //second stage turbine exit pressure bar
etat1=1.0; //first stage turbine isentropic efficiency
etap1=0.8; //first stage pump isentropic efficiency
etap2=0.8; //second stage turbine isentropic efficiency
dT=5.0;
r=new rankine3(m,T3,P3,P4,P5,etat1,etap1,etap2,dT);
  }

  public double func (double x)
  {   P4=x;
              r=new rankine3(m,T3,P3,P4,P5,etat1,etap1,etap2,dT);
     return -r.etacycle;
  }
}


public class rankine_opt2
{
 public static void main(String arg[])
 {        fe f=new fe();
          double xmin=1.0;
          double xmax=10.0;
          double P4=ref_cycle4_opt.golden(f,xmin,xmax,1.0e-3,1.0e-3,0);
                        double efficiency=-f.func(P4);
          String s="P4 = "+P4+" efficiency = "+efficiency;
          System.out.println(s);
 }
}
```

```
---------- Capture Output ----------
> "C:\java\bin\javaw.exe" rankine_opt2
P4 = 3.5331943279354787 efficiency = 0.3905562430461128

> Terminated with exit code 0.
```

In the next example, we will try to optimised flash temperatures of a double flash geothermal power plant so that exergetic efficiency will be maximum. Particle swarm optimisation will be used.

```
// OPTIMISATION Graphic search
import java.util.*;
import java.awt.*;
import java.applet.Applet;
import java.awt.event.*;
import javax.swing.*;
class fg extends f_xj
{  double mi; //mass flow rate
   double xi; //dryness fraction
   double P1; //first stage flash pressure (inlet)
   double P2; //second flash pressure optimisation parameter
   double P3; //third flash pressure optimisation parameter
   double P4;
   double etait1i;
   double etait2i;
```

```
   double etaipi;
   double T0;
   double Preinjection;
            flash2 rc;
            int i;
   public fg()
   {mi=6.0;
   xi=0.0;
   P1=20.0;
   P2=10.0;
   P3=3.0;
   P4=0.8;
   etait1i=0.9;
   etait2i=0.9;
   etaipi=0.9;
   T0=25.0;
   Preinjection=20.0;
   rc=new flash2(mi,xi,P1,P2,P3,P4,etait1i,etait2i,etaipi,T0,Preinjection);
   rc.cycle();
   i=0;
   }
   public double func (double x[])
   {
   P2=x[0];
   P3=x[1];
   rc=new flash2(mi,xi,P1,P2,P3,P4,etait1i,etait2i,etaipi,T0,Preinjection);
   rc.cycle();
   i++;
   if(i%10==0) System.out.println("i = "+i+" P2 = "+P2+" P3 = "+P3+" eta exergy = "+rc.etaexergy+"
efficiency = "+rc.etacycle);
   return -rc.etaexergy; //exergetic efficiency
   }
}


public class geothermal_flash_opt1
{


    public static void main (String args[])
    {
             fg f=new fg();
    double[] xmin={7.1,1.1};
    double[] xmax={19.0,7.0};
    particle_swarm1 ps=new particle_swarm1(f,50,xmin,xmax,2.0,2.0,10);
    String s="Particle swarm optimisation  : \n"+ps.toString();
    String s2="Particle swarm optimization method ";
    System.out.println(s);
    }
}
```

```
---------- Capture Output ----------
> "C:\java\bin\javaw.exe" geothermal_flash_opt1
i = 10 P2 = 8.922695079789056 P3 = 2.1965069926468255 eta exergy = 0.23975658751785658 efficiency =
0.08515786163866979
i = 20 P2 = 11.117364167615396 P3 = 4.295121796429263 eta exergy = 0.23666352765225462 efficiency =
0.08405925422686954
i = 30 P2 = 13.428277181733103 P3 = 3.346286045011228 eta exergy = 0.22867642126767773 efficiency =
0.08122235657399283
```

i = 40 P2 = 10.085778480896298 P3 = 5.6953034953734605 eta exergy = 0.22188257589899701 efficiency = 0.07880928692743916
i = 50 P2 = 11.755595910744905 P3 = 1.4045309255478617 eta exergy = 0.1900968024885872 efficiency = 0.06751946785641909
i = 60 P2 = 9.290161053474897 P3 = 2.48873869874415 eta exergy = 0.24211425282127508 efficiency = 0.08599526818410559
i = 70 P2 = 11.806009798758685 P3 = 4.4944503233412885 eta exergy = 0.23376734985182518 efficiency = 0.0830305763041343
i = 80 P2 = 14.281787443305936 P3 = 3.9409180094230463 eta exergy = 0.22559279936639545 efficiency = 0.08012710138232552
i = 90 P2 = 10.931075839513234 P3 = 6.558459504748374 eta exergy = 0.21077203591678928 efficiency = 0.07486299357912693
i = 100 P2 = 12.318889634590104 P3 = 1.5890056429343151 eta exergy = 0.19584449564021267 efficiency = 0.06956096028511491
i = 110 P2 = 9.881284996667299 P3 = 2.8375618240052676 eta exergy = 0.24268801833492862 efficiency = 0.08619906089207909
i = 120 P2 = 7.514818794919537 P3 = 4.089293979361365 eta exergy = 0.2350298390331469 efficiency = 0.08347899309274623
i = 130 P2 = 7.1 P3 = 3.0273024515059657 eta exergy = 0.24301961250107082 efficiency = 0.0863168380527105
i = 140 P2 = 11.554932914620409 P3 = 1.1 eta exergy = 0.16666219795841197 efficiency = 0.05919585585143518
i = 150 P2 = 7.1 P3 = 4.5460863593511185 eta exergy = 0.226193687369243 efficiency = 0.08034052758235859
i = 160 P2 = 7.1 P3 = 3.2288638420969433 eta exergy = 0.241730050671667 efficiency = 0.0858588054748373
i = 170 P2 = 7.1 P3 = 2.1129436634608503 eta exergy = 0.241748470332494 efficiency = 0.08586534785577615
i = 180 P2 = 7.1 P3 = 2.6172948678499033 eta exergy = 0.2441802874616896 efficiency = 0.08672909199212095
i = 190 P2 = 7.1 P3 = 1.1695919412584415 eta exergy = 0.21264294869186168 efficiency = 0.07552751309405274
i = 200 P2 = 7.1 P3 = 5.72333296335013 eta exergy = 0.20660032035029557 efficiency = 0.07338126421066526
i = 210 P2 = 7.383767566062751 P3 = 2.9525551947358726 eta exergy = 0.2442699196961002 efficiency = 0.08676092798668245
i = 220 P2 = 7.2984914390347715 P3 = 2.163153775978621 eta exergy = 0.24233502545906105 efficiency = 0.0860736832380435
i = 230 P2 = 7.1 P3 = 2.798807474251481 eta exergy = 0.24394047234410202 efficiency = 0.08664391334149946
i = 240 P2 = 7.1 P3 = 3.97964361317835 eta exergy = 0.2340513585457384 efficiency = 0.08313145183506686
i = 250 P2 = 8.154838086712594 P3 = 5.121070938248167 eta exergy = 0.22406751043795947 efficiency = 0.07958534215530502
i = 260 P2 = 7.62268328322836 P3 = 2.553824349319185 eta exergy = 0.24490115292876893 efficiency = 0.08698513234680386
i = 270 P2 = 7.440245799085582 P3 = 2.6468048649443734 eta exergy = 0.24486925240400967 efficiency = 0.08697380176981419
i = 280 P2 = 7.1 P3 = 2.8922403490779565 eta exergy = 0.243640437918064 efficiency = 0.0865373457163764
i = 290 P2 = 7.1 P3 = 3.1997451992828814 eta exergy = 0.24194093521507254 efficiency = 0.0859337084293504
i = 300 P2 = 8.565079960647367 P3 = 3.997829608265964 eta exergy = 0.23962939477635864 efficiency = 0.08511268472822882
i = 310 P2 = 8.415727094221273 P3 = 2.7596601446365683 eta exergy = 0.24537377347802058 efficiency = 0.08715299991514655
i = 320 P2 = 8.987214101648142 P3 = 2.683025594533742 eta exergy = 0.2443088508522301 efficiency = 0.08677475573607388
i = 330 P2 = 8.661197213777234 P3 = 2.7432647968206068 eta exergy = 0.2450939311849346 efficiency = 0.08705360422586861
i = 340 P2 = 8.695634719816391 P3 = 2.5900194421919758 eta exergy = 0.24438145587411766 efficiency = 0.08680054392597175
i = 350 P2 = 8.597197851332773 P3 = 2.5033439130343678 eta exergy = 0.24400980752580212 efficiency = 0.08666854013432727
i = 360 P2 = 8.71382388219697 P3 = 2.866410553952239 eta exergy = 0.24529402852811794 efficiency = 0.08712467573235559
i = 370 P2 = 9.55953144733351 P3 = 2.7566271416363635 eta exergy = 0.24325499615149274 efficiency = 0.0864004427141805
i = 380 P2 = 8.893327461228655 P3 = 3.019041281081366 eta exergy = 0.2451872356555144 efficiency = 0.08708674454237945
i = 390 P2 = 9.43524263312795 P3 = 2.381038471611845 eta exergy = 0.240428081284655 efficiency = 0.08539636592285339
i = 400 P2 = 8.46290947259319 P3 = 1.7710926997194256 eta exergy = 0.2324368692294798 efficiency = 0.08255800999876768
i = 410 P2 = 8.364865554599119 P3 = 2.902104410734401 eta exergy = 0.24552058707902413 efficiency = 0.08720514585386868
i = 420 P2 = 8.18659281553023 P3 = 3.0019490382273837 eta exergy = 0.24537661843773398 efficiency = 0.08715401040119053

```
i = 430 P2 = 8.757631302878057 P3 = 2.9642370184772364 eta exergy = 0.24531753744572313 efficiency =
0.08713302573922473
i = 440 P2 = 8.313825275923763 P3 = 2.8119219365441803 eta exergy = 0.24549969987516693 efficiency =
0.08719772703950156
i = 450 P2 = 8.263671788356309 P3 = 2.9276867709997036 eta exergy = 0.24550761640471092 efficiency =
0.08720053886934341
i = 460 P2 = 7.97635593304737 P3 = 2.8997727183182307 eta exergy = 0.24540182646046127 efficiency =
0.08716296390413207
i = 470 P2 = 7.557274415709548 P3 = 3.062945998562059 eta exergy = 0.24426233093309396 efficiency =
0.08675823257448632
i = 480 P2 = 8.646515008119033 P3 = 2.858597017375886 eta exergy = 0.24534902421560625 efficiency =
0.08714420935683005
i = 490 P2 = 7.746424154317776 P3 = 3.0770359915794017 eta exergy = 0.24461321343408532 efficiency =
0.0868828606557424
i = 500 P2 = 8.143434424065887 P3 = 3.4991831528262494 eta exergy = 0.24299753442692817 efficiency =
0.08630899625948782
i = 510 P2 = 7.873930005503491 P3 = 2.917039022444921 eta exergy = 0.24527790835901017 efficiency =
0.08711895009559756
i = 520 P2 = 7.775177534591211 P3 = 2.9853081194416307 eta exergy = 0.24498397148618026 efficiency =
0.08701454822782786
i = 530 P2 = 8.429637228687632 P3 = 2.8245024925308986 eta exergy = 0.24546496912027696 efficiency =
0.0871853912081897
i = 540 P2 = 8.082816911847678 P3 = 2.883824890120465 eta exergy = 0.24548527890878877 efficiency =
0.08719260493348462
i = 550 P2 = 8.183217685631815 P3 = 2.8381318784723906 eta exergy = 0.24552715183355542 efficiency =
0.08720747755396441
Particle swarm optimisation  :
ymin = -0.24553012714189582
xmin = 8.213221425003312 2.843958161560117
y global min = -0.24553012714189582
xmin = 8.213221425003312 2.843958161560117

> Terminated with exit code 0.
```

Exergetic efficiency is 24.55 % now. In the original problem we have solved(P2= , P3=) exergetic efficiency was:

9.0 THERMODYNAMIC SYSTEM SIMULATIONS AND CONTROL

9.1 BASICS OF SYSTEM SIMULATIONS AND CONTROL

In real modern thermodynamic systems, data such as temperature, pressure, flow rates, humidty etc. read from digital or analog sensors to control unir, it could be a datalogger, microcontrollers or programmable logic controller (PLC). Data is transferred from there to the main computer and stored usually into the database programs and then processed to calculate cycle performances such as efficiency ,COP or other properties and system input and output data can be processed by also using visual graphic user interface programs. Through the GUI programs, user can also send signals back to the system to control several aspects such as openin and closing valves, adjusting a frequency controlled motor to drive a compressor or pump. Recently by adding futures such as optimisation, neural networks and several control modes self governed control programs can also be created. In this chapter, some of the basics of such systems will be introduced. As control microprocessor, ARDUINO will be used due to simplicity of programming plus availability of sensor connections. As data base programs, SQLite will be used. This flexible database system is the basic database in android operating systems. And because of programs operating in control systems are real time programming, we will give information here for real time programming concepts as well.

9.1.1 Database programming

9.1.2 Real time systems

If all the real world processes carried out in a sequential order, some good results might be obtained out of it. Unfortunately real world processes seldomly sequential, usually happend and operates paralel to each other. Most of the natural systems including human body Works in paralel orders. When one process continues, another action goes on paralel to it. Modern computers start to utilised parallel processing nowadays. It is possible to have the tasks in between different processors. Another way to paralel use of the programs is time sharing in one processor. Multithreading can be defined as using one processor (Central Processing Unit) by several programs in the same time, by dividing CPU time into small process time packages and assign each one several package to one program opratation while assigning other small process time packages(Threads) into other tasks or programs. If only one processor in operations when one thread operates, the others waits in a row, but if the computer speed is fast enough, it could give the impresion of paralel processing. While the CPU process one thread, the other program parts waits either in cash memory or other memory units. Multithreading usualy controlled through operating systems. A programming language can achive multithreading through using operation system facilities, but in this case program can functions only for one operating system. Java is an operating system free language, therefore all the multithreading programming features are included inside the language. Java can operate multithreading through Thread class or Runnable interface. In table an example of running different programs in CPU threads (time sharing parts)

TABLE 11.1.2-1 A thread structure example

Thread1	Thread2	Thread3	Thread4	Thread5	Thread6	Thread7	Thread8	Thread9
PR1	PR1	PR2	PR3	PR1	PR2	PR2	PR3	PR3

An example program utilising threads are given as the following program. Class worker is a Thread program (extended from the class Thread). Two object of this class is created, and started by using start() method. In this one all the methods has the same priority so threads are shared equally between the objects

PROGRAM 11.1.2-1 Thread class example

```
class worker extends Thread
{
public void run()
{ for(int i=0;i<5;i++) {System.out.println(this.getName()+"-->"+i);}}}
public class C18E1
{public static void main(String args[])
{worker i1=new worker();
worker i2=new worker();
i1.start();
i2.start();}}
```

```
---------- Capture Output ----------
                    "C:\turhan\java\bin\javaw.exe" C18E1
Thread-0-->0
Thread-1-->0
Thread-0-->1
Thread-1-->1
Thread-0-->2
Thread-1-->2
```

```
Thread-0-->3
Thread-1-->3
Thread-0-->4
Thread-1-->4
> Terminated with exit code 0.
```

Methods of Thread class is given in the following table

TABLE 11.1.2-2 Field and methods of Thread class

Field Summary	
static int	MAX_PRIORITY The maximum priority that a thread can have.
static int	MIN_PRIORITY The minimum priority that a thread can have.
static int	NORM_PRIORITY The default priority that is assigned to a thread.

Constructor Summary	
Thread()	Allocates a new Thread object.
Thread(Runnable target)	Allocates a new Thread object.
Thread(Runnable target, String name)	Allocates a new Thread object.
Thread(String name)	Allocates a new Thread object.
Thread(ThreadGroup group, Runnable target)	Allocates a new Thread object.
Thread(ThreadGroup group, Runnable target, String name)	Allocates a new Thread object so that it has target as its run object, has the specified name as its name, and belongs to the thread group referred to by group.
Thread(ThreadGroup group, Runnable target, String name, long stackSize)	Allocates a new Thread object so that it has target as its run object, has the specified name as its name, belongs to the thread group referred to by group, and has the specified *stack size*.
Thread(ThreadGroup group, String name)	Allocates a new Thread object.

Method Summary	
static int	activeCount() Returns the number of active threads in the current thread's thread group.
void	checkAccess() Determines if the currently running thread has permission to modify this thread.
int	countStackFrames()**Deprecated.** *The definition of this call depends on* suspend(), *which is deprecated. Further, the results of this call were never well-defined.*
static Thread	currentThread() Returns a reference to the currently executing thread object.
void	destroy() Destroys this thread, without any cleanup.
static void	dumpStack() Prints a stack trace of the current thread.
static int	enumerate(Thread[] tarray) Copies into the specified array every active thread in the current thread's thread group and its subgroups.
ClassLoader	getContextClassLoader() Returns the context ClassLoader for this Thread.
String	getName() Returns this thread's name.
int	getPriority() Returns this thread's priority.
ThreadGroup	getThreadGroup() Returns the thread group to which this thread belongs.
static boolean	holdsLock(Object obj) Returns true if and only if the current thread holds the monitor lock on the specified object.
void	interrupt() Interrupts this thread.
static boolean	interrupted() Tests whether the current thread has been interrupted.
boolean	isAlive() Tests if this thread is alive.
boolean	isDaemon() Tests if this thread is a daemon thread.
boolean	isInterrupted() Tests whether this thread has been interrupted.
void	join() Waits for this thread to die.

void	join(long millis) Waits at most millis milliseconds for this thread to die.
void	join(long millis, int nanos) Waits at most millis milliseconds plus nanos nanoseconds for this thread to die.
void	resume() **Deprecated.** *This method exists solely for use with* suspend(), *which has been deprecated because it is deadlock-prone. For more information, see* Why are Thread.stop, Thread.suspend and Thread.resume Deprecated?.
void	run() If this thread was constructed using a separate Runnable run object, then that Runnable object's run method is called; otherwise, this method does nothing and returns.
void	setContextClassLoader(ClassLoader cl) Sets the context ClassLoader for this Thread.
void	setDaemon(boolean on) Marks this thread as either a daemon thread or a user thread.
void	setName(String name) Changes the name of this thread to be equal to the argument name.
void	setPriority(int newPriority) Changes the priority of this thread.
static void	sleep(long millis) Causes the currently executing thread to sleep (temporarily cease execution) for the specified number of milliseconds.
static void	sleep(long millis, int nanos) Causes the currently executing thread to sleep (cease execution) for the specified number of milliseconds plus the specified number of nanoseconds.
void	start() Causes this thread to begin execution; the Java Virtual Machine calls the run method of this thread.
void	stop() **Deprecated.** *This method is inherently unsafe. Stopping a thread with Thread.stop causes it to unlock all of the monitors that it has locked (as a natural consequence of the unchecked ThreadDeath exception propagating up the stack). If any of the objects previously protected by these monitors were in an inconsistent state, the damaged objects become visible to other threads, potentially resulting in arbitrary behavior. Many uses of stop should be replaced by code that simply modifies some variable to indicate that the target thread should stop running. The target thread should check this variable regularly, and return from its run method in an orderly fashion if the variable indicates that it is to stop running. If the target thread waits for long periods (on a condition variable, for example), the interrupt method should be used to interrupt the wait. For more information, see* Why are Thread.stop, Thread.suspend and Thread.resume Deprecated?.
void	stop(Throwable obj) **Deprecated.** *This method is inherently unsafe. See* stop() *(with no arguments) for details. An additional danger of this method is that it may be used to generate exceptions that the target thread is unprepared to handle (including checked exceptions that the thread could not possibly throw, were it not for this method). For more information, see* Why are Thread.stop, Thread.suspend and Thread.resume Deprecated?.
void	suspend() **Deprecated.** *This method has been deprecated, as it is inherently deadlock-prone. If the target thread holds a lock on the monitor protecting a critical system resource when it is suspended, no thread can access this resource until the target thread is resumed. If the thread that would resume the target thread attempts to lock this monitor prior to calling resume, deadlock results. Such deadlocks typically manifest themselves as "frozen" processes. For more information, see* Why are Thread.stop, Thread.suspend and Thread.resume Deprecated?.
String	toString()Returns a string representation of this thread, including the thread's name, priority, and thread group.
static void	yield()Causes the currently executing thread object to temporarily pause and allow other threads to execute.

In the second example Thread.sleep method is used. When thread sleeps it is inactivated and waits to be activate again for a designated time(in microseconds)

PROGRAM 11.1.2-2 Thread class example

```
class worker1 extends Thread {
String isim;
public worker1(String isim1)
{isim=isim1; setName(isim);}

public void run()
{ for(int i=0;i<20;i++) {try{if(isim.equals("ali")) Thread.sleep(1000);
                          if(isim.equals("veli")) Thread.sleep(2000);} catch(InterruptedException ex) {System.out.println("error...");}
                         System.out.println(this.getName()+"-->"+i);}}}
public class C18E2
{public static void main(String args[])
{worker1 i0=new worker1("ali");
worker1 i1=new worker1("veli");
worker1 i2=new worker1("deli");
i0.start();
i1.start();
i2.start();
```

```
}}
```

```
---------- Capture Output ----------
                        "d:\co\java\bin\java.exe" C18E2
deli-->0
deli-->1
deli-->2
deli-->3
deli-->4
deli-->5
deli-->6
deli-->7
deli-->8
deli-->9
deli-->10
deli-->11
deli-->12
deli-->13
deli-->14
deli-->15
deli-->16
deli-->17
deli-->18
deli-->19
ali-->0
ali-->1
veli-->0
ali-->2
veli-->1
ali-->3
ali-->4
veli-->2
ali-->5
ali-->6
veli-->3
ali-->7
ali-->8
veli-->4
ali-->9
ali-->10
veli-->5
ali-->11
ali-->12
veli-->6
ali-->13
ali-->14
veli-->7
ali-->15
ali-->16
veli-->8
ali-->17
ali-->18
veli-->9
ali-->19
veli-->10
veli-->11
veli-->12
veli-->13
veli-->14
veli-->15
veli-->16
veli-->17
veli-->18
veli-->19
```

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

In the next example different thread priorities are tried. MIN_PRORITY gives slower Access.

PROGRAM 1811.1.2-3Thread class example with changing thread priorities

```
class Robot extends Thread
{
public Robot(String isim) {super(isim);}
public void run()
{ try{
            for(int i=0;i<10;i++)
{System.out.println("Priority: "+this.getPriority()+this.getName()+"-->"+i);}
} catch(Exception ex) {System.out.println("error..");}
}
}

public class C18E3
{
public static void main(String args[])
{Robot r1=new Robot("A");
Robot r2=new Robot("B");
Robot r3=new Robot("C");
Robot r4=new Robot("D");
r1.setPriority(Thread.MIN_PRIORITY);
r2.setPriority(Thread.NORM_PRIORITY);
r3.setPriority(Thread.MAX_PRIORITY);
r4.setPriority(Thread.MAX_PRIORITY-2);

r1.start();
r2.start();
r3.start();
r4.start();
}
}
```

```
---------- Capture Output ----------
                    "C:\turhan\java\bin\javaw.exe" C18E3
Priority: 1A-->0
Priority: 5B-->0
Priority: 5B-->1
Priority: 10C-->0
Priority: 1A-->1
Priority: 10C-->1
Priority: 8D-->0
Priority: 5B-->2
Priority: 8D-->1
Priority: 10C-->2
Priority: 1A-->2
Priority: 10C-->3
Priority: 8D-->2
Priority: 5B-->3
Priority: 8D-->3
Priority: 10C-->4
Priority: 1A-->3
Priority: 10C-->5
Priority: 8D-->4
Priority: 5B-->4
Priority: 8D-->5
Priority: 10C-->6
Priority: 1A-->4
```

```
Priority: 10C-->7
Priority: 8D-->6
Priority: 5B-->5
Priority: 8D-->7
Priority: 10C-->8
Priority: 1A-->5
Priority: 10C-->9
Priority: 8D-->8
Priority: 5B-->6
Priority: 8D-->9
Priority: 1A-->6
Priority: 5B-->7
Priority: 1A-->7
Priority: 5B-->8
Priority: 1A-->8
Priority: 5B-->9
Priority: 1A-->9
> Terminated with exit code 0.
```

Now a computer clock can be programmed by using Thread.sleep method. If Thread wait 1000 microseconds in each period, a second counter is obtained.

PROGRAM 11.1.2-4 Thread class example a very simple clock

```
class clock extends Thread
{ int second;
  int minute;
  long counter;
  public clock()
 {counter=0;}

public void run()
{ while(true)
{ try{   second=(int)counter%60;
          minute=(int)counter/60;
         Thread.sleep(1000);
         counter++;
} catch(InterruptedException ex) {System.out.println("error...");}
 System.out.println("minute : "+minute+" second : "+second);
}}}

public class C18E5
{public static void main(String args[])  {clock c1=new clock();c1.start();}}
```

```
---------- Capture Output ----------
                        "C:\turhan\java\bin\javaw.exe" C18E5
minute : 0second : 0
minute : 0second : 1
minute : 0second : 2
minute : 0second : 3
minute : 0second : 4
minute : 0second : 5
minute : 0second : 6
minute : 0second : 7
minute : 0second : 8
```

A clock program can be made more sofisticated if time is read from the java Date class. System still waits 1000 microsecond before reactivating. Output is a graphic screen.

PROGRAM 11.1.2-5 Thread class example a digital clock with graphical output

```
import java.awt.*;
import java.awt.event.*;
import java.awt.geom.*;
import javax.swing.*;
import java.net.*;
import java.io.*;
import java.awt.Graphics;
import java.util.*;
import java.text.*;
import java.util.Locale;

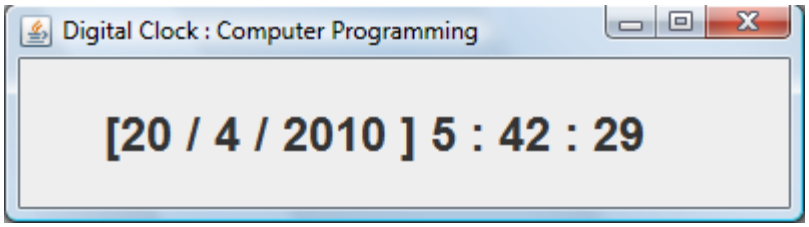
class clock1 extends Thread
{
JFrame a;
public clock1(JFrame ai)
{a=ai;}

public void run()
{
while(true)
{
a.validate();
a.repaint();
try {Thread.sleep(1000);}
catch(InterruptedException e)
{System.err.println(e.toString());}
}}}

public class C18E6 extends JFrame implements Serializable
{
JLabel jb;
Font F;
Date D;
clock1 r;
GregorianCalendar calender;
public C18E6()
{
super("Digital Clock : Computer Programming");
Container cc=getContentPane();
jb=new JLabel("");
F=new Font("TimesRoman",Font.BOLD,24);
jb.setFont(F);
// Türkiye standart (Greenwitch zamanına göre iki saat ileride)
String[] ids = TimeZone.getAvailableIDs(+3 * 60 * 60 * 1000);
SimpleTimeZone pdt = new SimpleTimeZone(+3 * 60 * 60 * 1000, ids[0]);
calender=new GregorianCalendar(pdt);
D=new Date();
calender.setTime(D);
r=new clock1(this);
cc.add(jb);
r.start();
}

public void paint(Graphics g)
{D=new Date();
calender.setTime(D);
String s="      ["+
calender.get(Calendar.DATE) +" / "+
```

```
(calender.get(Calendar.MONTH)+1)+" / "+
calender.get(Calendar.YEAR) +" ] "+
calender.get(Calendar.HOUR_OF_DAY)+" : "+
calender.get(Calendar.MINUTE)+" : "+
calender.get(Calendar.SECOND);
jb.setText(s);
}
public static void main(String s[])
{              C18E6 f = new C18E6();
               f.setDefaultCloseOperation( JFrame.EXIT_ON_CLOSE );
               f.setSize(400,110);
               f.setVisible(true);      }}
```



Another method of creating threads is through **Runnable interface**. Extends can be used only once in a class creation. If a second library addition is needed, implements is used. Runnable is the implemented version for Thread use. To compare and see the differences, A runnable version of some of the programs given earlier is listed here

PROGRAM 11.1.2.1A Runnable interface Thread class example

```
class worker implements Runnable
{
Thread t;
public worker()
{t=new Thread(this);t.start();}
public void run()
{ for(int i=0;i<5;i++) {System.out.println(t.getName()+"-->"+i);}}
}

public class C18E1A
{
public static void main(String args[])
{worker i1=new worker();
worker i2=new worker();
}
}
```

One advantage of Runnable Interface is the ability of starting threads by using Lambda expressions. Lambda version of the same program with Anonymous Thread classes is given as the next example:

PROGRAM 18.1B Runnable interface Thread class example with anonymous inner class Thread and Runnable class lambda expressions

```
public class C18E1B
{public static void main(String args[])
{ new Thread(() -> {for(int i=0;i<20;i++){System.out.println("thread 1 i = "+i);}}).start();
new Thread(() -> {for(int i=0;i<20;i++){System.out.println("thread 2 i = "+i);}}).start();
}}
```

```
---------- Capture Output ----------
                        "C:\java\bin\java.exe" C18E1B
thread 1 i = 0
thread 1 i = 1
thread 1 i = 2
thread 1 i = 3
thread 1 i = 4
thread 1 i = 5
thread 1 i = 6
thread 2 i = 0
thread 2 i = 1
thread 2 i = 2
thread 2 i = 3
thread 2 i = 4
thread 1 i = 7
thread 1 i = 8
thread 1 i = 9
thread 1 i = 10
thread 1 i = 11
thread 1 i = 12
thread 1 i = 13
thread 2 i = 5
thread 2 i = 6
thread 2 i = 7
thread 2 i = 8
thread 2 i = 9
thread 2 i = 10
thread 2 i = 11
thread 2 i = 12
thread 1 i = 14
thread 1 i = 15
thread 2 i = 13
thread 2 i = 14
thread 2 i = 15
thread 2 i = 16
thread 1 i = 16
thread 1 i = 17
thread 1 i = 18
thread 1 i = 19
thread 2 i = 17
thread 2 i = 18
thread 2 i = 19

> Terminated with exit code 0.
```

Another way of writing by using Lambda Expressions, but in this version inner classes is not anonymous.

PROGRAM 18.1C Runnable interface inner Thread classes t1 and t2 and and Runnable class lambda expressions

```
public class C18E1C
{
public static void main(String args[])
{ Runnable r1=() -> {for(int i=0;i<40;i++){System.out.println("thread r1 i = "+i);}};
Thread t1=new Thread(r1);
t1.start();
Runnable r2=() -> {for(int i=0;i<40;i++){System.out.println("thread r2 i = "+i);}};
Thread t2=new Thread(r2);
t2.start();
```

```
}}
```

Now the second example given with Thread class is turned into a Runnable interface example:

PROGRAM 11.1.2-2A Runnable interface example

```
class worker1 implements Runnable
{ Thread t;
String isim;
public worker1(String isim1)
{isim=isim1;t=new Thread(this);t.start();}

public void run()
{ for(int i=0;i<5;i++) {
                        try
                        {
                        if(isim.equals("ali")) Thread.sleep(1000);
                        if(isim.equals("veli")) Thread.sleep(2000);
} catch(InterruptedException ex) {System.out.println("error...");}
                        System.out.println(t.getName()+"-->"+i);}}
}

public class C18E2A
{
public static void main(String args[])
{worker1 i1=new worker1("ali");
worker1 i2=new worker1("veli");

}
}
```

A lambda expression version of the same Runnable interface program:

PROGRAM 11.1.2-2B Runnable interface example, lambda expressions version

```
class worker1
{ Thread t;
String isim;
public worker1(String isim1,Runnable r)
{isim=isim1;t=new Thread(r);t.start();}
}

public class C18E2B
{
public static void a(String isim)
{ for(int i=0;i<5;i++) {
                        try
                        {
                        if(isim.equals("ali")) Thread.sleep(1000);
                        if(isim.equals("veli")) Thread.sleep(2000);
} catch(InterruptedException ex) {System.out.println("error...");}
                        System.out.println(isim+"-->"+i);}}
public static void main(String args[])
{   Runnable r1=() -> a("ali");
            worker1 i1=new worker1("ali",r1);
            Runnable r2=() -> a("veli");
            worker1 i2=new worker1("veli",r2);}}
```

A Runnable version of the previous Digital clock is created as the next example.

PROGRAM 11.1.2-6 Thread class and Runnable interphase example a digital clock with graphical output

```
import java.awt.Graphics;
import java.awt.*;
import java.util.*;
import javax.swing.*;
import java.text.*;
import java.awt.event.*;

class clock2 extends Thread
{
JFrame a;
public clock2(JFrame ai)
{a=ai;}

public void run()
{
while(true)
{
a.validate();
a.repaint();
try {Thread.sleep(1000);}
catch(InterruptedException e)
{System.err.println(e.toString());}
}}}

public class C18E7 extends JFrame implements Runnable
{
JLabel jb;
Font F;
Date D;
clock2 r;
GregorianCalendar calender;
Thread runner;

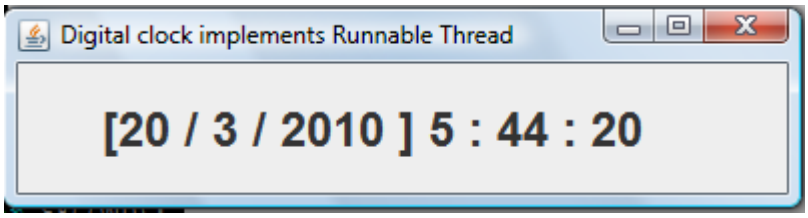
public C18E7()
{
super("Digital clock implements Runnable Thread");
Container cc=getContentPane();
jb=new JLabel("");
F=new Font("TimesRoman",Font.BOLD,24);
jb.setFont(F);
// Türkiye standart time + 1 hour( 3 hours advanced from Greenwitch)
String[] ids = TimeZone.getAvailableIDs(+3 * 60 * 60 * 1000);
SimpleTimeZone pdt = new SimpleTimeZone(+3 * 60 * 60 * 1000, ids[0]);
calender=new GregorianCalendar(pdt);
D=new Date();
calender.setTime(D);
r=new clock2(this);
cc.add(jb);
r.start();
}

public void start()
{
        if(runner==null)
        {
        runner=new Thread(this);
        runner.start();
        }}

public void run()
{
        while(true)
```

```
            {
            repaint();
            try {Thread.sleep(1000);}
            catch(InterruptedException e)
            {System.err.println(e.toString());}
            }
}


public void paint(Graphics g)
{
D=new Date();
calender.setTime(D);
String s="      ["+
calender.get(Calendar.DATE) +" / "+
calender.get(Calendar.MONTH)+" / "+
calender.get(Calendar.YEAR) +" ] "+
calender.get(Calendar.HOUR_OF_DAY)+" : "+
calender.get(Calendar.MINUTE)+" : "+
calender.get(Calendar.SECOND);
jb.setText(s);
}
public static void main(String s[])
{
            C18E7 f= new C18E7();
            f.setDefaultCloseOperation( JFrame.EXIT_ON_CLOSE );
            f.setSize(400,100);
            f.setVisible(true);
}
}
```



public interface **Runnable** has only one method

Method Summary	
void	run() When an object implementing interface Runnable is used to create a thread, starting the thread causes the object's run method to be called in that separately executing thread.

9.1.3 ARDUINO microcontroler

9.2 SIMULATION EXAMPLES

REFERENCES

[1] CHASE,M. W. Jr; DAVIES, C.A.;DOWNEY J. R.; FRUIP D. J. ; McDONALD R. A.; SYVERUD A.N., JANAF Thermochemical Tables, Third edition, J. Phys. Chem. Ref. Data, Vol 14, Suppl. 1, 1985

[2] Ihsan Barin, Thermochemical Data of Pure Substances, VCH publishing, 1989, ISBN 3-527 27812-5

[3] N.B. Vargaftick, Table of Thermophysical Properties of Liquids and Gases, 1975, Hemisphere Publishing

[4] Robert C. Reid , John M. Prausnitz ; The Properties of Gases & Liquids , Mc-Graw Hill, Fourth edition, ISBN 0-07-0511799-1

[5] LEMMON, E.W., MCLINDEN, M.O. and HUBER, M.L, NIST Standard Reference Database 23, NIST

[6] MARX, V., PRUß, A. and WAGNER, W., *Neue Zustandsgleichungen fur R 12, R 22, R 11 und R 113*,Beschreibung des thermodynamishchen Zustandsverhaltens bei Temperaturen bis 525 K und Drucken bis 200 MPa, VDI-Fortschritt-Ber. Series, **19**(1992), No. 57, Dusseldorf: VDI Verlag

[7] MCLINDEN, M.O. and WATANABE, K., *International collaboration on the thermophysical properties of alternative refrigerants*, Results of IEA Annex 18. 20th International Congress of Refrigeration, Sydney, Australia, September 19-24, 1999, International Institute of Refrigeration, pp 678-687

[8] OUTCALT, S.L. and MCLINDEN, M.O., *A modified Benedict–Webb–Rubin equation of state for the thermodynamic properties of R152a (1,1–difluoroethane)*, J. Phys. Chem. Ref. Data, **25**(1996), pp. 605-636

[9] SPAN, R. and WAGNER, W., *A new equation of state for carbon dioxide covering the fluid region from the triple–point temperature to 1 100 K at pressures up to 800 MPa*, J. Phys. Chem. Ref. Data, **26**(1996), pp. 1509-1596

[10] TILLNER-ROTH, R. and BAEHR, H.D., *An international standard formulation of the thermodynamic properties of 1,1,1,2-tetrafluoroethane (HFC-134a) covering temperatures from 170 K to 455 K at pressures up to 70 MPa*, J. Phys. Chem. Ref. Data, **23**(1994), pp. 657-729

[11] TILLNER-ROTH, R., HARMS–WATZENBERG, F. and BAEHR, H.D., *Eine neue Fundamentalgleichung fur Ammoniak*, DKV-Tagungsbericht **20**, II(1993), pp. 167-181

[12] TILLNER-ROTH, R., LI, J., YOKOZEKI, A., SATO, H. and WATANABE, K., *Thermodynamic Properties of Pure and Blended Hydrofluorocarbon (HFC) Refrigerants,* Tokyo: Japan Society of Refrigerating and Air Conditioning Engineers, (1998)

[13] TILLNER-ROTH, R. and YOKOZEKI, A., *An international standard equation of state for difluoromethane(R-32) for temperatures from the triple point at 136.34 K to 435 K and pressures up to 70 MPa,* J. Phys.Chem. Ref. Data, **26**(1997), pp. 1273-1328

[14] YOUNGLOVE, B.A. and MCLINDEN, M.O,. *An international standard equation-of-state formulation of the thermodynamic properties of refrigerant 123 (2,2-dichloro-1,1,1-trifluoroethane)*, J. Phys. Chem. Ref.Data, **23**(1994), pp. 731-779

[15] INTERNATIONAL STANDART ISO 17584 Refrigerant Properties, Reference numarası ISO 17584:2005(E)
[16] A.C. Bryant, Refrigeration Equipment, ISBN: 0750636882, Elsevier Science & Technology Books, Ağustos 1998

[17] W. P. Jones, Air Conditioning Engineering, ISBN: 0750650745, January 2001[18] ASHRAE 2000 HVAC Systems and Equipment Handbook

[17] M. Turhan Çoban, Soğutucu akışkanların termodinamik özellikleri, Martin-Hou hal denklemi, III. Ege Enerji Sempozyumu, Bildiriler Kitabı, Muğla Universitesi, Muğla, sayfa 521-528

[18] M. Turhan Çoban,"Kubik şerit ve b şerit interpolasyon yöntemi kullanarak soğutucu akışkanların doyma termofiziksel özelliklerinin hassas olarak oluşturulması", I Soğutma Teknolojileri Sempozyumu bildiri kitabı, 6-12 Ekim 2008, ISBN : 978-605-5771-00-3

[19] M. Turhan Çoban," Kısmi devamlı fonksiyonlar kullanarak soğutucu akışkanların doyma basınç eğrilerinin hassas olarak oluşturulması", I Soğutma Teknolojileri Sempozyumu bildiri kitabı, 6-12 Ekim 2008, ISBN : 978-605-5771-00-3

[20] M. Turhan Çoban, **"**İdeal gazların termodinamik ve termofiziksel özelliklerinin modellenmesi", 17. Isı Bilimi ve Tekniği Kongresi, 24-27 Haziran 2009, Cumhuriyet Universitesi, Sivas

[21] M. Turhan Çoban, **"**Gerçek gazların termodinamik ve termofiziksel özelliklerinin Lee-Kesler hal denklemi kullanılarak modellenmesi", 17. Isı Bilimi ve Tekniği Kongresi, 24-27 Haziran 2009, Cumhuriyet Universitesi, Sivas

[22] M. Turhan Çoban, Sayısal Metodların Soğutma Dunyasına uygulanması :Soğutucu akışkanların termodinamik özellikleri, Martin-Hou hal denklemi. Soğutma Dunyası ISSN 1304-1908

[23] M. Turhan Çoban, Sayısal Metodların Soğutma Dunyasına uygulanması :Soğutucu akışkanların termodinamik özellikleri, değiştirilmiş Benedict- Webb-Rubin hal denklemi. Soğutma Dunyası ISSN 1304-1908

[24] M. Turhan Çoban, Halil Atalay, Soğutucu akışkanların gerçek gaz denklemlerinin tanımlanması ve Peng-Robınson-Stryjek-Vera (PRSV) denkleminin java programlama dili ile hesaplanmasına yönelik örnek çalışma, Tesisat Kongresi (Teskon), 13 Nisan 2011, Tepekule kongre merkezi, İZMİR

[25] Herbert B. Callen, Thermodynamics, ISBN-13: 978-0471130352

[26] M. Turhan Çoban, Java 2 Programlama Kılavuzu, ALFA yayınevi, ticarethane sok no 41/1 34410 cagaloglu Istanbul, ISBN 975-316-631-1

[27] Kenneth Wark, Jr. Thermodynamics, Mc-Graw Hill International Editions, 5inci baskı, 1989, ISBN 0-07-068286-0

[28] Thomas H. Kuehn, Jams W. Ramsey, James L. Threlkeld, Thermal Environmental Engineering, Prentice Hall, 3rd edition, 1998, ISBN 0-13-917220-3 [34] Reference Fluid Thermodynamic and Transport Properties-REFPROP, version 7.0. Standard Reference Data Program, National Institute of Standards and Technology (2002)

[29] Wolfrang Wagner, Hans-Joachim Kretzchmar, International Steam Tables, Prentice Hall, 3rd edition, 1998, ISBN 0-13-917220-3 [34] Reference Fluid Thermodynamic and Transport Properties-REFPROP, version 7.0. Standard Reference Data Program, National Institute of Standards and Technology (2002)

[30] Richard B. Stewart, Richard T. Jacobsen, Steven G. Penoncello, ASHRAE Thermodynamic Properties of Refrigerants, Center of Applied Thermodynamics Studies, University of Idaho.

[31] Martin J. J. And Hou, Y. C., Development of an Equation of Statefor Gases, AIche Journal, 1, No 2, 142-155 (1955)

[32] Wilbert F. Stoecker, Jerold W. Jones, Refrigeration & Air Conditioning, Second Edition, McGraw-Hill publication, ISBN 0-07-066591-5

[33] J. Patek, J. Klomfar, A computationally effective formulation of the thermodynamic properties of LiBr-H_2O solutions from 273 to 500 K over full composition range. International Journal Of Refrigeration 29(2006) 566-578

[34] Sanford Gordon, Bonnie J. McBride, A computer Program for Calculating of Complex Chemical Equilibrium Compositions and Applications., NASA Reference Publication 1311, 1994

[35] Gary L. Borman, Kenneth W. Ragland, Combustion Engineering, McGraw-Hill publications ISBN 0-07-006567-5 1998

[36] H. Scott Fogler, Elements of Chemical Reaction Engineering, Pearson International Edition, ISBN 0-13-127839-8 2008

[37] William R. Simith, Ronald W. Missen, Chemical Reaction Equilibrium Analysis theory and Algorithms, Willey InterScience, ISBN 0-471-09347-5, 1982

[38] Irvin Glassman, Richard A. Yetter, Combustion, Fourth edition, Elsevier, ISBN: 978-0-12-088573-2, 2008

[39] Westbook, C.K, and Dryer, F.L., Chemical Kinetic Modelling of Hydrocarbon Combustion, “Prog. Energy Combustion Sci.”, vol 10, pp 1-57, 1984

[40] Prof. Dr.-Ing. Peter Stephan et al.,VDI Atlas., Second edition, Springer, ISBN 978-3-540-77876-9

[41] Aly FA, Lee LL (1981) Self consistent equations for calculating the ideal gas heat capacity, enthalpy and entropy. Fluid Phase Equilibria 6:169–179

[42] KAMEI, A., BEYERLEIN, S.W. and JACOBSEN, R.T., Application of nonlinear regression in the development of a wide range formulation for HCFC-22. Int. J. Thermophysics, **16**(1995), pp. 1155-1164

[43] LEMMON, E.W. and JACOBSEN, R.T., Equations of state for mixtures of R-32, R-125, R-134a, R-143a, and R-152a, J. Phys. Chem. Ref. Data (in press)

[44] LEMMON, E.W. and JACOBSEN, R.T., A new functional form and new fitting techniques for equations of state with application to pentafluoroethane (HFC-125), J. Phys. Chem. Ref. Data (submitted for publication)

[45] LEMMON, E.W. and JACOBSEN, R.T., An international standard formulation for the thermodynamic
properties of 1,1,1-trifluoroethane (HFC-143a) for temperatures from 161 to 500 K and pressures to
50 MPa. J. Phys. Chem. Ref. Data, **29**(2001), pp. 521-552

[46] LEMMON, E.W., MCLINDEN, M.O. and HUBER, M.L, NIST Standard Reference Database 23, NIST
Reference Fluid Thermodynamic and Transport Properties-REFPROP, version 7.0. Standard Reference Data Program, National Institute of Standards and Technology (2002)

[47] MARX, V., PRUß, A. and WAGNER, W., Neue Zustandsgleichungen fur R 12, R 22, R 11 und R 113,
Beschreibung des thermodynamishchen Zustandsverhaltens bei Temperaturen bis 525 K und Drucken
bis 200 MPa, VDI-Fortschritt-Ber. Series, **19**(1992), No. 57, Dusseldorf: VDI Verlag

[48] MCLINDEN, M.O. and WATANABE, K., International collaboration on the thermophysical properties
of alternative refrigerants, Results of IEA Annex 18. 20th International Congress of Refrigeration, Sydney,
Australia, September 19-24, 1999, International Institute of Refrigeration, pp 678-687

[49] OUTCALT, S.L. and MCLINDEN, M.O., A modified Benedict–Webb–Rubin equation of state for the
thermodynamic properties of R152a (1,1–difluoroethane), J. Phys. Chem. Ref. Data, **25**(1996),
pp. 605-636

[50] SPAN, R. and WAGNER, W., A new equation of state for carbon dioxide covering the fluid region from the triple–point temperature to 1 100 K at pressures up to 800 MPa, J. Phys. Chem. Ref. Data, **26**(1996),
pp. 1509-1596

[51] TILLNER-ROTH, R. and BAEHR, H.D., An international standard formulation of the thermodynamic
properties of 1,1,1,2-tetrafluoroethane (HFC-134a) covering temperatures from 170 K to 455 K at
pressures up to 70 MPa, J. Phys. Chem. Ref. Data, **23**(1994), pp. 657-729

[52] TILLNER-ROTH, R., HARMS–WATZENBERG, F. and BAEHR, H.D., Eine neue Fundamentalgleichung fur Ammoniak, DKV-Tagungsbericht **20**, II(1993), pp. 167-181

[53] TILLNER-ROTH, R., LI, J., YOKOZEKI, A., SATO, H. and WATANABE, K., Thermodynamic Properties of Pure and Blended Hydrofluorocarbon (HFC) Refrigerants, Tokyo: Japan Society of Refrigerating and Air Conditioning Engineers, (1998)

[54] TILLNER-ROTH, R. and YOKOZEKI, A., An international standard equation of state for difluoromethane (R-32) for temperatures from the triple point at 136.34 K to 435 K and pressures up to 70 MPa, J. Phys. Chem. Ref. Data, **26**(1997), pp. 1273-1328

[55] YOUNGLOVE, B.A. and MCLINDEN, M.O,. An international standard equation-of-state formulation of the thermodynamic properties of refrigerant 123 (2,2-dichloro-1,1,1-trifluoroethane), J. Phys. Chem. Ref.
Data, **23**(1994), pp. 731-779

[56] K. Kadoya, N. Matsunaga, and A. Nagashima, Viscosity and Thermal Conductivity of Dry Air in the Gaseous Phase, Journal of Physical and Chemical Reference Data 14, 947 (1985)

[57] E. W. Lemmon & R. T. Javobsen, Viscosity and Thermal Conductivity Equations for Nitrogen, Oxygen, Argon, and Air, International Journal of Thermophysics, Vol. 25, No. 1, January 2004

[58] The International Association for the Properties of Water and Steam, Revised Relase on the IAPWS Formulation 1995 for the Thermodynamic Properties of Ordinary Water Substance for General and Scientific Use, Prague, Check Republic 2018, IAPWS R6-95(2018)

[59] The International Association for the Properties of Water and Steam, Revised Relase on the IAPWS Industrial Formulation 1997 for the Thermodynamic Properties of Water and Steam, Lucerne, Switzerland, August 2007, IAPWS R7(2012)

[60] J.R. Cooper, R. B. Dooley, The International Association for the Properties of Water and Steam, Guideline on an Equation of State for Humid Air in Contact with SeaWater and Ice, Consistent with the IAPWS Formulation 2008 for the Thermodynamic Properties of SeaWater

[61] Robert C. Reid , John M. Prausnitz ; The Properties of Gases & Liquids , Mc-Graw Hill, Fourth edition, ISBN 0-07-0511799-1

[62] Bruce E. Poling, John M Prausnitz, John P. O'connel, The Properties of Gases & Liquids, 5th edition, 2004, McGraw-Hill ISBN 0-07-051799-1, ISBN-10: 0070116822

[63] Ian H. Bell, Jorrit Wronski, Sylvain Quailin, and Vincent Lemort, Pure and Pseudo-pure Fluid Thermophysical Property Evaluation and the Open-Source Thermophysical Property Library CoolProp, Ind Eng Chem Res 2014 Feb 12;53(6) 2498-2508 DOI 10.1021/ie4033999

[64] S. G. S. Beirao, A.P.C. Ribeiro, M.J.V. Lourenço, F.J.V. Santos, C.A. Nieto de Castro, Thermal Conductivity of Humid Air, Int J Thermopys (2012) 33:1686-1703, DOI 10.1007/s10765-012-1245-5

[65] Xiaoyan Ji, Thermodynamic properties of Humid air and their application in advanced power generation cycles, Doctaral Thesis in Chemical Engineering, KTH Royal Institute of Technology, Energy Processes, SE-100 44 Stockholm, Sweeden, ISBN 91-7178-437-3

[66] R. Feistel, D.G. Wright, D.R. Jackett, K.Miyagawa, J.H. Reissmann, W. Wagner, U. Overhoff, C. Guder, A. Feistal, G.M. Marion, Numerical implementation and ocenographic application oft he thermodynamic potentials of liquid water, water vapor, ice, sewater and humid air- Part 1: Background and equations, Ocean Sci., 6, 663-677,2010 DOI 10.5194/os-6-633-2010

[67] R. Feistel, D.G. Wright, D.R. Jackett, K.Miyagawa, J.H. Reissmann, W. Wagner, U. Overhoff, C. Guder, A. Feistal, G.M. Marion, Numerical implementation and ocenographic application oft he thermodynamic potentials of liquid water, water vapor, ice, sewater and humid air- Part 2: The library routines, Ocean Sci., 6, 695-718,2010 DOI 10.5194/os-6-695-2010

[68] Simeen Sattar, Thermodynamics of Mixing Real Gases, Jornal of Chemical Education Vol. 77 No 10 October 2000

[69] Wolgang Wagner, Hans-Joachim Kretzschmar, International Steam Tables, Properties of water and steam based on industrial formulation IAPWS-IF97, Springer, ISBN 978-3-540-21419-9, DOI 10.1007/978-3-540-74234-0

[70] Joseph H. Keenan, Frederick G. Keyes, Philip G. Hill, Joan G. Moore, Steam Tables Thermodynamic properties of Water incuding vapor, liquid and solid phases, Wiley-Interscience, ISBN 0-471-04210-2

[71] E. W. Lemmon1, and R. T Jacobsen , "Viscosity and Thermal Conductivity Equations for Nitrogen, Oxygen, Argon, and Air", International Journal of Thermophysics, Vol. 25, No. 1, January 2004 (© 2004)

[72] Harstad, K.G., Miller, R.S. & Bellan, J. 2000 Efficient high pressure state equations. Amer. Inst. Chem. Eng. J. 43 , 1675–1706.

[73] Cutrone, L.Ihme, M and Herrmann M, Modeling of high-pressure mixing and combustion in liquid rocket injectors, Center for Turbulance research, proceedings of the summer Program 2006, pp 269-281

[74] Thermodynamic and Transport Combustion Properties of Hydrocarbons With Air, Properties in SI units, NASA Technical Paper 1906, July 1982, https://ntrs.nasa.gov/archive/nasa/casi.ntrs.nasa.gov/19820024310.pdf

[75] Byung IK Lee, Michael G. Kesler, A generalized Thermodynamic Correlation Based on Three-Parameter Corresponding States, AICHE Journal, 21(3), 510-527 doi:10.1002/aic.690210313

[76] Malcolm W. Chase Jr., NIST-JANAF Thermochemical Tables Fourth Edition, Journal of Physical and Chemical Reference Data, Monograph no. 9, ISBN 1-56396-831-2

[77] Tien, Chang L., Lienhard, John H., Statistical Thermodynamics, Hemisphere Publishing Corporation, Mc Graw Hill Book Company, 1979, ISBN 0-07-064570-1

[78] D.-Y. Peng, D.B. Robinson, Indus. Eng. Chem. Fundam. 15 (1976) 59e64, http://dx.doi.org/10.1021/i160057a011.

[79] G. Soave, Chem. Eng. Sci. 27 (1972) 1197e1203, http://dx.doi.org/10.1016/ 0009-2509(72)80096-4.

[80] D.B. Robinson, D.-Y. Peng, S.Y.-K. Chung, Fluid Phase Equilibria 24 (1985) 25e41, http://dx.doi.org/10.1016/0378-3812(85)87035-7.

[81] Reflections on the motive power of Heat, From the original French of N.L.S. Carnot. New York : John Wiley & sons, London: Chapman & Hall Limited. 1897

[82] Schreiber, Donald R. and Pitzer, Kenneth S., Selected equation of State in the Acentric Factor System, International Journal of Thermodynamics, Vol 9, No. 6, 1988

[83] M. Binnewies, E. Milke, Thermochemical data of Elemnts and Compounds, Willey-VHC, ISBN 3-527-30524-6

APPENDIX

APPENDIX I

Specific Heat Cp of gases

$$Cp(T) = a_0 + a_1 T + a_2 T^2 + a_3 T^3 \quad kJ/(kmolK)$$

Reference: Robert C. Reid , John M. Prausnitz ; The Properties of Gases & Liquids , Mc-Graw Hill, Fourth edition, ISBN 0-07-0511799-1

Program: Gas_Data

Cp	formula	name	a0	a1	a2	a3	hf kJ/kmol	gf kJ/kmol
0	NH3	ammonia	2.731E+01	2.383E-02	1.707E-05	-1.185E-08	-45720	-16160
1	Ar	argon	2.080E+01	0.000E+00	0.000E+00	0.000E+00	0	0
2	C6H5Cl	chlorobenzene	-3.389E+01	5.631E-01	-4.522E-04	1.426E-07	51870	99230
3	NO	nitricoxide	2.935E+01	-9.378E-04	9.747E-06	-4.187E-09	90430	86750
4	NO2	nitrogendioxide	2.423E+01	4.836E-02	-2.081E-05	2.930E-10	33870	52000
5	N2	nitrogene	3.115E+01	-1.357E-02	2.680E-05	-1.168E-08	0	0
6	N2O	nitrousoxide	2.162E+01	7.281E-02	-5.778E-05	1.830E-08	81600	103700
7	O2	oxygen	2.811E+01	-3.680E-06	1.746E-05	-1.065E-08	0	0
8	O3	ozone	2.054E+01	8.009E-02	-6.243E-05	1.697E-08	142800	162900
9	H2	hydrogen	2.714E+01	9.274E-03	-1.381E-05	7.645E-09	0	0
10	H2O	water	3.224E+01	1.924E-03	1.055E-05	-3.596E-09	-242000	-228800
11	CH4	methane	1.925E+01	5.213E-02	1.197E-05	-1.132E-08	-74900	-50870
12	CH4O	methanol	2.115E+01	7.092E-02	2.587E-05	-2.852E-08	-201300	-162600
13	C2H6	ethane	5.409E+00	1.781E-01	-6.938E-05	8.713E-09	-84740	-32950
14	C3H8	propane	-4.224E+00	3.063E-01	-1.586E-04	3.215E-08	-103900	-23490
15	C3H6O	propionaldehyde	1.172E+01	2.614E-01	-1.300E-04	2.126E-08	-192200	-130500
16	C4H10	n-butane	9.487E+00	3.313E-01	-1.108E-04	-2.822E-09	-126200	-16100
17	C4H10	isobutane	-1.390E+00	3.847E-01	-1.846E-04	2.895E-08	-134600	-20900
18	C5H12	n-pentane	-3.626E+00	4.873E-01	-2.580E-04	5.305E-08	-146500	-8307
19	C6H6	benzene	-3.392E+01	4.739E-01	-3.017E-04	7.130E-08	82980	129700
20	C6H14	n-hexane	-4.413E+00	5.820E-01	-3.119E-04	6.494E-08	-167300	-167
21	CCl2F2	dichlorodiflоromethane	3.160E+01	1.782E-01	-1.509E-04	4.342E-08	-481500	-442500
22	C7H16	n-heptane	-5.146E+00	6.762E-01	-3.651E-04	7.658E-08	-187900	8000
23	C8H10	ethylbenzene	-4.310E+01	7.072E-01	-4.811E-04	1.301E-07	29810	130700
24	C8H18	n-octane	-6.096E+00	7.712E-01	-4.195E-04	8.855E-08	-208600	16400
25	C9H20	n-nonane	-8.347E+00	8.729E-01	-4.823E-04	1.031E-07	-229200	24830
26	C10H8	naphtalene	-6.880E+01	8.499E-01	-6.506E-04	1.981E-07	151100	223700
27	C11H10	1-methylnaphtalene	-6.482E+01	9.387E-01	-6.942E-04	2.016E-07	116900	217800
28	C11H10	2-methylnaphtalene	-5.652E+01	8.997E-01	-6.469E-04	1.840E-07	116200	216300
29	C12H12	1-ethylnaphtalene	-6.382E+01	1.040E+00	-8.000E-04	2.000E-07	96719.7	226139.87
30	C13H14	1-propylnaphtalene	-6.024E+01	1.116E+00	-8.000E-04	2.000E-07	74734.95	232797.2
31	C14H16	1-butylnaphtalene	1.507E+01	8.659E-01	4.000E-04	6.000E-08	53091.16	240250.06
32	C10H22	n-decane	-7.913E+00	9.609E-01	-5.288E-04	1.131E-07	-249800	33240
33	C11H24	n-undecane	-8.395E+00	1.054E+00	-5.799E-04	1.237E-07	-270500	41620
34	C12H26	n-dodecane	-9.328E+00	1.149E+00	-6.347E-04	1.359E-07	-291100	50070
35	C13H28	n-tridecane	-1.046E+01	1.245E+00	-6.912E-04	1.490E-07	-311700	58490
36	C14H30	n-tetradecane	-1.098E+01	1.338E+00	-7.423E-04	1.598E-07	-332300	66860
37	C15H32	n-pentadecane	-1.192E+01	1.433E+00	-7.972E-04	1.720E-07	-353000	75280
38	C16H34	n-hexadecane	-1.302E+01	1.529E+00	-8.537E-04	1.850E-07	-373600	83740
39	C17H36	n-heptadecane	-1.397E+01	1.624E+00	-9.081E-04	1.972E-07	-394200	92150
40	C18H38	n-octadecane	-1.447E+01	1.717E+00	-9.592E-04	2.078E-07	-414800	100600
41	C19H40	n-nonadecane	-1.549E+01	1.812E+00	-1.015E-03	2.205E-07	-435400	109000
42	C20H42	n-eicosane	-2.238E+01	1.939E+00	-1.117E-03	2.528E-07	-456100	117400
43	C3H6	propylene	3.710E+00	2.345E-01	-1.160E-04	2.205E-08	20430	62760
44	C11H16	penthylbenzene	-5.740E+01	1.036E+00	-7.000E-04	2.090E-07	-33780	153034.85
45	C12H18	hexylbenzene	-9.436E-01	9.028E-01	-5.000E-04	9.000E-08	-55059	161450.72
46	C13H20	heptylbenzene	1.510E+00	9.848E-01	-5.000E-04	1.000E-07	65819.64	-75700.96
47	C14H22	octylbenzene	3.759E+00	1.068E+00	-6.000E-04	1.000E-07	-96301	178324.33

48	C15H24	nonylbenzene	-2.874E+01	1.293E+00	-8.000E-04	2.000E-07	-116942.91	186698.33
49	C16H26	decylbenzene	9.150E+00	1.230E+00	-6.000E-04	1.000E-07	-137584.82	195114.2
50	C17H28	undecylbenzene	-3.162E+01	1.494E+00	-9.000E-04	2.000E-07	-158184.84	203571.94
51	C18H30	dodecylbenzene	3.366E+01	1.307E+00	-6.000E-04	9.000E-08	-178826.77	211945.94
52	CO	carbonmonoxide	3.087E+01	-1.285E-02	2.789E-05	-1.272E-08	-110600	-137400
53	CO2	carbondioxide	1.980E+01	7.344E-02	-5.602E-05	1.715E-08	-393800	-394600
54	C8H18	2,3,3-trimethylpentane	-9.215E+00	7.859E-01	-4.400E-04	9.697E-08	-216600	18920
55	C8H18	2,3,4-trimethylpentane	-9.215E+00	7.859E-01	-4.400E-04	9.697E-08	-217600	18920
56	C8H18	2-methyl-3-ethylpentane	-9.215E+00	7.859E-01	-4.400E-04	9.697E-08	-211300	21270
57	C8H18	3-methyl-3-ethylpentane	-9.215E+00	7.859E-01	-4.400E-04	9.697E-08	-215100	19930
59	C6H12	cyclohexane	-5.454E+01	6.113E-01	-2.523E-04	1.321E-08	-123200	31780
60	C6H12	methylcyclopentane	-5.011E+01	6.381E-01	-3.642E-04	8.014E-08	-106800	35800
61	C6H12	1-hexene	-1.746E+00	5.309E-01	-2.903E-04	6.054E-08	-41700	87500
62	C6H12	2-hexene,cis	-9.810E+00	5.309E-01	-2.717E-04	4.827E-08	-52380	76280
63	C6H12	2-hexene,trans	-3.292E+01	6.929E-01	-5.619E-04	2.005E-07	-53930	76490
64	C6H12	3-hexene,cis	-2.173E+01	5.811E-01	-3.362E-04	7.457E-08	-47650	83070
65	C6H12	3-hexene,trans	-4.338E+00	5.510E-01	-3.282E-04	8.047E-08	-54470	77670
66	C6H12	2-methyl-2-pentene	-1.475E+01	5.669E-01	-3.341E-04	7.963E-08	-66530	71260
67	C6H12	3-methyl-2-pentene,cis	-1.475E+01	5.669E-01	-3.341E-04	7.963E-08	-62220	73270
68	C6H12	3-methyl-2-pentene,trans	-1.475E+01	5.669E-01	-3.341E-04	7.963E-08	-63140	71340
69	C6H12	4-methyl-2-pentene,cis	-1.675E+00	5.376E-01	-3.044E-04	6.753E-08	-57480	82190
70	C6H12	4-methyl-2-pentene,trans	1.263E+01	5.154E-01	-3.007E-04	7.327E-08	-61500	79670
71	C6H12	2,3-dimethyl-1-butene	7.025E+00	5.585E-01	-3.696E-04	1.063E-07	-66360	79090
72	C6H12	2,3-dimethyl-2-butene	2.294E+00	4.827E-01	-2.199E-04	3.042E-08	-69840	75910
73	C6H12	3,3-dimethyl-1-butene	-1.256E+01	5.485E-01	-2.915E-04	5.208E-08	-61550	98220
74	C6H12O	cyclohexanol	-5.553E+01	7.214E-01	-4.086E-04	8.235E-08	-294800	-118000
76	C6H12O	methylisobutylketone	3.894E+00	5.656E-01	-3.318E-04	8.231E-08	-284000	0
77	C6H12O2	n-butylacetate	1.362E+01	5.489E-01	-2.278E-04	7.910E-10	-486800	0
78	C6H12O2	isobutylacetate	7.310E+00	5.740E-01	-2.576E-04	1.101E-08	-495500	0
79	C6H12O2	ethylbutyrate	2.151E+01	4.928E-01	-1.938E-04	3.559E-09	0	0
82	C6H14	2-methylpentane	-1.057E+01	6.184E-01	-3.573E-04	8.058E-08	-174400	-5020
83	C6H14	3-methylpentane	-2.386E+00	5.690E-01	-2.870E-04	5.033E-08	-171700	-2140
84	C6H14	2,2-dimethylbutane	-1.663E+01	6.293E-01	-3.481E-04	6.850E-08	-185700	-9630
85	C6H14	2,3-dimethylbutane	-1.461E+01	6.150E-01	-3.376E-04	6.820E-08	-177900	-4100
86	C6H14O	1-hexanol	4.811E+00	5.891E-01	-3.010E-04	5.426E-08	-317800	-135700
87	C6H14O	ethylbutylether	2.363E+01	5.367E-01	-2.528E-04	4.157E-08	0	0
88	C6H14O	dipropylether	1.862E+01	5.335E-01	-2.285E-04	2.442E-08	-293100	-105600
89	C6H14O	diisopropylether	7.505E+00	5.849E-01	-3.027E-04	5.845E-08	-319000	122000
90	C6H15N	dipropylamine	6.460E+00	6.293E-01	-3.390E-04	7.072E-08	0	0
92	C6H15N	triethylamine	-1.843E+01	7.155E-01	-4.392E-04	1.092E-07	-99650	110400
96	C7H5N	benzonitrile	-2.605E+01	5.732E-01	-4.430E-04	1.349E-07	219000	261000
97	C7H6O	benzaldehyde	-1.214E+01	4.961E-01	-2.845E-04	5.167E-08	-36800	22400
98	C7H6O2	benzoicacid	-5.129E+01	6.293E-01	-4.237E-04	1.062E-07	-290400	-210600
99	C7H8	toluene	-2.435E+01	5.125E-01	-2.765E-04	4.911E-08	50030	122100
101	C7H8O	benzylalcohol	-7.398E+00	5.481E-01	-3.357E-04	7.771E-08	-94080	0
102	C7H8O	o-cresol	-3.228E+01	7.055E-01	-5.924E-04	2.124E-07	-128700	-33000
103	C7H8O	m-cresol	-4.501E+01	7.264E-01	-6.029E-04	2.077E-07	-132400	-40570
104	C7H8O	p-cresol	-4.063E+01	7.055E-01	-5.757E-04	1.967E-07	-125500	-30900
107	C7H9N	m-toluidine	-1.599E+01	5.681E-01	-3.033E-04	4.643E-08	0	0
108	C7H14	cycloheptane	-7.619E+01	7.867E-01	-4.204E-04	7.561E-08	-119400	63050
109	C7H14	1,1-dimethylcyclopentane	-5.789E+01	7.670E-01	-4.501E-04	1.010E-07	-138400	39060
110	C7H14	1,2-dimethylcyclopentane	-5.564E+01	7.616E-01	-4.484E-01	1.014E-07	-129600	45760
111	C7H14	1,2-dimethylcyclopentane	-5.452E+01	7.591E-01	-4.480E-04	1.017E-07	-136800	38390
112	C7H14	ethylcyclopentane	-5.531E+01	7.511E-01	-4.396E-04	1.004E-07	-127200	44590
113	C7H14	methylcyclohexane	-6.192E+01	7.842E-01	-4.438E-04	9.366E-08	-154900	27300
114	C7H14	1-heptene	-3.303E+00	6.297E-01	-3.512E-04	7.607E-08	-62340	95880
119	C7H16	3-methylhexane	-7.046E+00	6.837E-01	-3.734E-04	7.834E-08	-192400	4600
120	C7H16	2,2-dimethylpentane	-5.010E+01	8.956E-01	-6.360E-04	1.736E-07	-206300	840
121	C7H16	2,3-dimethylpentane	-7.046E+00	6.837E-01	-3.734E-04	7.834E-08	-199400	670
122	C7H16	2,4-dimethylpentane	-7.046E+00	6.837E-01	-3.734E-04	7.834E-08	-202100	3100
123	C7H16	3,3-dimethylpentane	-7.046E+00	6.837E-01	-3.734E-04	7.834E-08	-201700	2640

124	C7H16	3-ethylpentane	-7.046E+00	6.837E-01	-3.734E-04	7.834E-08	-189800	11000
125	C7H16	2,2,3-trimethylbutane	-2.294E+01	7.519E-01	-4.421E-04	1.005E-07	-204900	4270
126	C7H16O	1-heptanol	4.907E+01	6.778E-01	-3.447E-04	6.046E-08	-332000	-121000
127	C8H4O3	phthalicanhydride	-4.455E+00	6.540E-01	-4.283E-04	1.009E-07	-371800	0
128	C8H8	styrene	-2.825E+01	6.159E-01	-4.023E-04	9.935E-08	145700	213900
129	C8H8O	methylphenylketone	-2.958E+01	6.410E-01	-4.071E-04	9.722E-08	-86920	1840
130	C8H8O2	methylbenzoate	-2.121E+01	5.501E-01	-1.799E-04	4.425E-08	-254100	0
131	C8H10	o-xylene	-1.585E+01	5.962E-01	-3.443E-04	7.528E-08	19000	122200
132	C8H10	m-xylene	-2.917E+01	6.297E-01	-3.747E-04	8.478E-08	17250	118900
133	C8H10	p-xylene	-2.509E+01	6.042E-01	-3.374E-04	6.820E+00	17960	121200
136	C8H16	1,1-dimethylcyclohexane	-7.211E+01	8.997E-01	-5.020E-04	1.030E-07	-181100	35250
137	C8H16	1,2dimethylcyclohexane-cis	-6.837E+01	8.972E-01	-5.137E-04	1.099E-07	-172300	41240
138	C8H16	1,2dimethylcyclohexane-trans	-6.848E+01	9.123E-01	-5.355E-04	1.181E-07	-180100	34500
139	C8H16	1,3-dimethylcyclohexane-cis	-6.516E+01	8.838E-01	-4.932E-04	1.020E-07	-184900	29850
140	C8H16	1,3dimethylcyclohexane-trans	-6.415E+01	8.826E-01	-5.016E-04	1.068E-07	-176700	36340
141	C8H16	1,4dimethylcyclohexane-cis	-6.415E+01	8.826E-01	-5.016E-04	1.068E-07	-176800	37970
142	C8H16	1,4dimethylcyclohexane-trans	-7.036E+01	9.131E-01	-5.309E-04	1.155E-07	-184700	31740
143	C8H16	ethylcyclohexane	-6.389E+01	8.893E-01	-5.108E-04	1.103E-07	-171900	39270
149	C8H16	n-propylcyclopentane	-5.597E+01	8.449E-01	-4.924E-04	1.117E-07	-148200	52630
152	C8H16	1-octene	-4.099E+00	7.239E-01	-4.036E-04	8.675E-08	-82980	104300
153	C8H16	2-octene-trans	-1.282E+01	7.532E-01	-4.442E-04	1.050E-07	-94580	92740
157	C8H18	2-methylheptane	-8.970E+01	1.242E+00	-1.176E-03	4.618E-07	-215600	12770
158	C8H18	3-methylheptane	-9.215E+00	7.859E-01	-4.400E-04	9.697E-08	-212800	13730
159	C8H18	4-methylheptane	-9.215E+00	7.859E-01	-4.400E-04	9.697E-08	-212200	16750
160	C8H18	2,2-dimethylhexane	-9.215E+00	7.859E-01	-4.400E-04	9.697E-08	-224900	10720
161	C8H18	2,3-dimethylhexane	-9.215E+00	7.859E-01	-4.400E-04	9.697E-08	-214100	17710
162	C8H18	2,4-dimethylhexane	-9.215E+00	7.859E-01	-4.400E-04	9.697E-08	-219600	11720
163	C8H18	2,5-dimethylhexane	-9.215E+00	7.859E-01	-4.400E-04	9.697E-08	-222800	10470
164	C8H18	3,3-dimethylhexane	-9.215E+00	7.859E-01	-4.400E-04	9.697E-08	-220300	13270
165	C8H18	3,4-dimethylhexane	-9.215E+00	7.859E-01	-4.400E-04	9.697E-08	-213100	17330
166	C8H18	3-ethylhexane	-9.215E+00	7.859E-01	-4.400E-04	9.697E-08	-211000	16940
167	C8H18	2,2,3-trimethylpentane	-9.215E+00	7.859E-01	-4.400E-04	9.697E-08	-220300	17120
168	C8H18	2,2,4-trimethylpentane	-7.461E+00	-7.779E-01	-4.287E-04	9.173E-08	-224300	13690
170	C8H18O	1-octanol	6.171E+00	7.607E-01	-3.797E-04	6.263E-08	-360100	-120200
171	C8H18O	2-octanol	2.588E+01	7.641E-01	-4.224E-04	9.064E-08	0	0
174	C8H18O	2-ethyl-1-hexanol	-1.499E+01	8.645E-01	-5.280E-04	1.285E-07	-365500	0
175	C8H18O	dibutylether	6.054E+00	7.729E-01	-4.085E-04	8.085E-08	-334100	-88590
177	C8H19N	dibutylamin	9.764E+00	8.081E-01	-4.329E-04	9.249E-08	0	0
182	C9H10	alpha-methylstyrene	-2.433E+01	6.933E-01	-4.530E-04	1.181E-07	0	0
183	C9H10O2	ethylbenzoate	2.067E+01	6.887E-01	-3.608E-04	5.062E-08	0	0
184	C9H12	n-propylbenzene	-3.129E+01	7.468E-01	-4.601E-04	1.081E-07	7830	137300
185	C9H12	isopropylbenzene	-3.936E+01	7.842E-01	-5.078E-04	1.291E-07	3940	137100
186	C9H12	1-methyl-3-ethylbenzene	-2.900E+01	7.293E-01	-4.363E-04	9.998E-08	-1930	126500
187	C9H12	1-methyl-4-ethylbenzene	-2.731E+01	7.176E-01	-4.224E-04	9.542E-08	-2050	126800
188	C9H12	1,2,3-trimethylbenzene	-6.942E+00	6.335E-01	-3.326E-04	6.611E-08	-9590	124600
189	C9H12	1,2,4-trimethylbenzene	-4.668E+00	6.238E-01	-3.263E-04	6.376E-08	-13940	117000
190	C9H12	1,3,5-trimethylbenzene	-1.959E+01	6.724E-01	-3.629E-04	7.700E-08	-16080	118000
192	C9H18	n-propylcyclohexane	-6.252E+01	9.889E-01	-5.795E-04	1.291E-07	-193400	47350
195	C9H18	1-nonene	-3.718E+00	8.122E-01	-4.509E-04	9.705E-08	-103600	112800
196	C9H18O	dibutylketone	0.000E+00	0.000E+00	0.000E+00	0.000E+00	0	0
197	C9H20	2-methyloctane	-1.011E+01	8.805E-01	-4.936E-04	1.083E-07	-229200	24830
198	C9H20	2,2-dimethylheptane	-2.089E+01	9.668E-01	-6.120E-04	1.570E-07	-247000	16750
199	C9H20	2,2,3-trimethylhexane	-4.563E+01	1.055E+00	-7.172E-04	1.987E-07	-241400	24530
200	C9H20	2,2,4-trimethylhexane	-6.031E+01	1.104E+00	-7.712E-04	2.188E-07	-243400	22520
201	C9H20	2,2,5-trimethylhexane	-5.411E+01	1.095E+00	-7.746E-04	2.255E-07	-252400	13440
202	C9H20	3,3-diethylpentane	-6.727E+01	1.126E+00	-7.988E-04	2.306E-07	-232100	35090
203	C9H20	2,2,3,3-tetramethylpentane	-5.458E+01	1.089E+00	-7.570E-04	2.142E-07	-237400	34330

204	C9H20	2,2,3,4-tetramethylpentane	-5.458E+01	1.089E+00	-7.570E-04	2.142E-07	-237100	32660
205	C9H20	2,2,4,4-[illegible]	-6.740E+01	1.168E+00	-8.612E-04	2.574E-07	-242100	34040
206	C9H20	2,3,3,4-[illegible]	-5.492E+01	1.091E+00	-7.603E-04	2.158E-07	-236400	34120
207	C9H20O	1-nonanol	1.280E+00	8.817E-01	-4.791E-04	9.801E-08	-387200	-118300
211	C10H14	n-butylbenzene	-2.299E+01	7.934E-01	-4.396E-04	8.570E-08	-13820	144800
213	C10H14	sec-butylbenzene	-6.515E+01	9.893E-01	-7.214E-04	2.152E-07	-17460	0
214	C10H14	tert-butylbenzene	-8.600E+01	1.102E+00	-8.746E-04	2.827E-07	-22690	0
216	C10H14	1-methyl-3-isopropylbenzene	-4.876E+01	9.064E-01	-6.054E-04	1.627E-07	-29310	0
218	C10H14	1,4-diethylbenzene	-3.742E+01	8.671E-01	-5.560E-04	1.411E-07	-22270	138000
219	C10H14	1,2,3,5-tetramethylbenzene	3.923E+00	7.131E-01	-3.711E-04	6.840E-08	-44840	118800
220	C10H14	1,2,4,5-tetramethylbenzene	1.625E+01	6.519E-01	-2.879E-04	3.257E-08	-45300	119500
222	C10H15N	n-butylaniline	-3.407E+01	9.144E-01	-5.560E-04	1.287E-07	0	0
223	C10H18	cis-decalin	-1.125E+02	1.118E+00	-6.607E-04	1.437E-07	-169100	85870
224	C10H18	trans-decalin	-9.767E+01	1.045E+00	-5.476E-04	8.981E-08	-182400	73480
227	C10H20	butylcyclohexane	-6.296E+01	1.081E+00	-6.305E-04	1.400E-07	-213300	564800
231	C10H20	1-decene	-4.664E+00	9.077E-01	-5.058E-04	1.095E-07	-124200	121100
233	C10H22	n-decane	-7.913E+00	9.609E-01	-5.288E-04	1.131E-07	-249800	33240
234	C10H22	3,3,5-trimethylheptane	-7.037E+01	1.232E+00	-8.646E-04	2.455E-07	-258700	33580
235	C10H22	2,2,3,3-tetramethylhexane	-5.883E+01	1.231E+00	-8.834E-04	2.585E-07	0	0
236	C10H22	2,2,5,5-tetramethylhexane	-6.234E+01	1.245E+00	-8.956E-04	2.618E-07	0	0
237	C10H22O	1-decanol	1.457E+01	8.947E-01	-3.921E-04	3.451E-08	-401900	-104300
238	C11H14O2	butylbenzoate	-1.737E+01	8.675E-01	-4.610E-04	7.235E-08	0	0
239	C11H22	n-hexylcyclopentane	-5.832E+01	1.128E+00	-6.536E-04	1.473E-07	-209600	78250
240	C11H22	1-undecene	-5.585E+00	1.003E+00	-5.602E-04	1.216E-07	-144900	129500
241	C11H24	n-undecane	-8.395E+00	1.054E+00	-5.799E-04	1.237E-07	-270500	41620
242	C12H10	diphenyl	-9.707E+01	1.106E+00	-8.855E-04	2.790E-07	182200	280300
243	C12H10O	diphenylether	-6.073E+01	9.282E-01	-5.870E-04	1.359E-07	49990	0
244	C12H24	n-heptylcyclopentane	-5.926E+01	1.223E+00	-7.084E-04	1.596E-07	-230300	86670
245	C12H24	1-dodecene	-6.544E+00	1.098E+00	-6.155E-04	1.341E-07	-165500	138000
246	C12H26	dodecane	-9.328E+00	1.149E+00	-6.347E-04	1.359E-07	-291100	50070
247	C12H26O	dihexylether	3.354E+01	1.073E+00	-5.535E-04	1.678E-07	0	0
248	C12H26O	dodecanol	9.224E+00	1.103E+00	-5.338E-04	7.779E-08	-443100	-87130
249	C12H27N	tributylamine	7.993E+00	1.198E+00	-6.703E-04	1.449E-07	0	0
251	C13H26	n-octylcyclopentane	-5.995E+01	1.317E+00	-7.612E-04	1.708E-07	-250900	91520
252	C13H26	1-tridecene	-7.118E+00	1.191E+00	-6.674E-04	1.451E-07	-186100	146400
253	C13H28	1-tridecane	-1.046E+01	1.245E+00	-6.912E-04	1.490E-07	-311700	58490
254	C14H10	anthracane	-5.898E+01	1.006E+00	-6.594E-04	1.606E-07	224800	0
255	C14H10	phenanthrene	-5.898E+01	1.006E+00	-6.594E-04	1.606E-07	0	0
256	C14H28	n-nonylcyclopentane	-6.081E+01	1.412E+00	-8.156E-04	1.830E-07	-271500	103500
257	C14H28	1-tetradecene	-7.967E+00	1.286E+00	-7.210E-04	1.569E-07	-206700	154900
258	C14H30	n-tetradecane	-1.098E+01	1.338E+00	-7.423E-04	1.598E-07	-332300	66860
259	C15H30	n-decyclopentane	-6.192E+01	1.508E+00	-8.717E-04	1.959E-07	-292200	111900
260	C15H30	1-pentadecene	-9.203E+00	1.382E+00	-7.783E-04	1.703E-07	-227400	163200
261	C15H32	n-pentadecane	-1.192E+00	1.433E+00	-7.972E-04	1.720E-07	-353000	75280
262	C16H22O4	dibutyl-o-phthalate	1.880E+00	1.254E+00	-6.121E-04	6.971E-08	0	0
263	C16H32	n-decyclohexane	-6.902E+01	1.654E+00	-9.613E-04	2.143E-07	0	0
264	C16H32	1-hexadecene	-9.705E+00	1.475E+00	-8.298E-04	1.810E-07	-248000	171600
265	C16H34	hexadecane	-1.302E+01	1.529E+00	-8.537E-04	1.850E-07	-373600	83740
266	C17H34	n-dodecyclopentane	-6.326E+01	1.695E+00	-9.768E-04	2.186E-07	-336100	126000
267	C17H36O	heptadecanol	-7.792E+00	1.653E+00	-9.345E-04	2.044E-07	-546300	-44670
271	C18H36	1-octadecene	-1.133E+01	1.664E+00	-9.374E-04	2.049E-07	-289200	188400
272	C18H36	n-tridecyclopentane	-6.421E+01	1.790E+00	-1.032E-03	2.309E-07	-354000	137100
273	C18H38	octadecane	-1.447E+01	1.717E+00	-9.592E-04	2.078E-07	-414800	100600
274	C18H38O	1-octadecanol	-8.704E+00	1.748E+00	-9.881E-04	2.157E-07	-566900	-36220
275	C19H38	1-cyclopentyltetradecane	-6.493E+01	1.884E+00	-1.085E-03	2.426E-07	-374600	145600
276	C19H40	n-nonadecane	-1.549E+01	1.812E+00	-1.015E-03	2.205E-07	-435400	109000
277	C20H40	1-cyclopentylpentadecane	-6.609E+01	1.980E+00	-1.140E-03	2.550E-07	-395300	154000

278	C20H42O	1-eicosanol	-1.258E+01	1.950E+00	-1.118E-03	2.516E-07	-608100	-19430
279	C6F6	perfluorobenzene	3.628E+01	5.267E-01	-4.547E-04	1.456E-07	-957300	-880000
290	C6H4Cl2	o-dichlorobenzene	-1.430E+01	5.506E-01	-4.513E-04	1.429E-07	30000	82730
291	C6H4F2	1,4-difluorobenzene	-2.596E+01	5.722E-01	-4.677E-04	1.475E-07	-307400	-253000
292	C6H5Br	bromobenzene	-2.881E+01	5.351E-01	-4.080E-04	1.212E-07	105100	138600
293	C6H5F	fluorobenzene	-3.873E+01	5.669E-01	-4.434E-04	1.355E-07	-116600	-69080
294	C6H5I	iodobenzene	-2.927E+01	5.564E-01	-4.509E-04	1.443E-07	162700	187900
295	C6H6	benzene	-3.392E+01	4.739E-01	-3.017E-04	7.130E-08	82980	129700
296	C6H6O	phenol	-3.584E+01	5.983E-01	-4.827E-04	1.527E-07	-96420	-32900
297	C6H7N	aniline	-4.052E+01	6.385E-01	-5.133E-04	1.633E-07	86920	166800
298	C6H7N	2-methylpyridine	-3.626E+01	5.584E-01	-3.704E-04	9.663E-08	99020	177200
299	C6H7N	3-methylpyridine	-3.709E+01	5.600E-01	-3.719E-04	9.685E-08	106200	184400
300	C6H7N	4-methylpyridine	-1.743E+01	4.882E-01	-2.798E-04	5.451E-08	102300	0
302	C6H10	cyclohexene	-6.865E+01	7.252E-01	-5.414E-04	1.644E-07	-5360	106900
303	C5H12O	2,2-dimethyl-1-propanol	1.215E+01	5.397E-01	-3.160E-04	7.122E-08	-293100	-125500
306	C5H12O	tert-butylmethylether	2.534E+00	5.136E-01	-2.596E-04	4.303E-08	-293100	-125500
311	C3H9N	n-propylamine	6.691E+00	3.498E-01	-1.822E-04	3.586E-08	-72430	39820
312	C3H9N	isopropylamine	-7.486E+00	4.175E-01	-2.826E-04	8.348E-08	-83820	0
313	C3H9N	trimethylamine	-8.206E+00	3.972E-01	-2.219E-04	4.622E-08	-23860	98980
316	C4H4	vinylacetylene	6.757E+00	2.841E-01	-2.265E-04	7.461E-08	304800	306200
317	C4H4O	furan	-3.553E+01	4.321E-01	-3.455E-04	1.047E-07	-34700	879
318	C4H4N	thiophene	-3.061E+01	4.480E-01	-3.772E-04	1.253E-07	115800	126900
319	C4H5N	allylcyanide	2.170E+01	2.572E-01	-1.192E-04	1.229E-08	0	0
321	C4H6	1-butyne	1.255E+01	2.744E-01	-1.545E-04	3.450E-08	165300	202200
322	C4H6	2-butyne	1.593E+01	2.381E-01	-1.070E-04	1.753E-08	146400	185600
323	C4H6	1,2-butadiene	1.120E+01	2.724E-01	-1.468E-04	3.089E-08	162300	198600
324	C4H6	1,3-butadiene	-1.687E+00	3.419E-01	-2.340E-04	6.335E-08	110200	150800
325	C4H6O2	vinylacetate	1.516E+01	2.795E-01	-8.805E-05	-1.660E-08	-316000	0
326	C4H6O3	aceticanhydride	-2.313E+01	5.087E-01	-3.580E-04	9.835E-08	-576100	-477000
328	C4H6O4	succinicacid	1.507E+01	4.689E-01	-3.143E-04	7.938E-08	0	0
329	C4H7N	butylonitrile	1.521E+01	3.206E-01	-1.638E-04	2.982E-08	34100	108700
330	C4H7O2	methylacrylate	1.516E+01	2.796E-01	-8.805E-05	-1.660E-08	0	0
331	C4H8	1-butene	-2.994E+00	3.532E-01	-1.990E-04	4.463E-08	-126	71340
332	C4H8	2-butene,cis	4.396E-01	2.953E-01	-1.108E-04	-6.160E-10	-6990	-65900
333	C4H8	2-butene,trans	1.832E+01	2.564E-01	-7.013E-05	-8.989E-09	-11180	63010
334	C4H8	cyclobutane	-5.025E+01	5.024E-01	-3.558E-04	1.047E-07	26670	110100
335	C4H8	isobutylene	1.605E+01	2.804E-01	-1.091E-04	9.098E-09	-16910	58110
336	C4H8O	n-butyraldehyde	1.408E+01	3.457E-01	-1.743E-04	2.887E-08	-205200	-114800
337	C4H8O	isobutyraldehyde	2.446E+01	3.356E-01	-2.057E-04	6.368E-08	-215900	-121400
338	C4H8O	methylethylketone	1.094E+01	3.559E-01	-1.900E-04	3.920E-08	-238500	-146200
339	C4H8O	tetrahydrofuran	1.910E+01	5.162E-01	-4.132E-04	1.454E-07	-184300	0
340	C4H8O	vinylethylether	1.728E+01	3.236E-01	-1.471E-04	2.150E-08	-140300	0
341	C4H8O2	n-butyricacid	1.174E+01	4.137E-01	-2.430E-04	5.531E-08	-476200	0
342	C4H8O2	isobutyricacid	9.814E+00	4.668E-01	-3.720E-04	1.350E-07	-484200	0
343	C4H8O2	1,4-dioxane	-5.357E+01	5.987E-01	-4.085E-04	1.062E-07	-315300	-180900
344	C4H8O2	ethylacetate	7.235E+00	4.072E-01	-2.092E-04	2.855E-08	-443200	-327600
345	C4H8O2	methylpropionate	1.820E+01	3.140E-01	-9.353E-05	-1.828E-08	0	0
348	C4H9Cl	1-chlorobutane	-2.613E+00	4.497E-01	-2.937E-04	8.018E-08	-147400	-38810
349	C4H9Cl	2-chlorobutane	-3.433E+00	4.559E-01	-2.981E-04	8.256E-08	-161600	-53510
350	C4H9Cl	tert-butylchloride	-3.931E+00	4.652E-01	-2.886E-04	7.871E-08	-183400	-64140
351	C4H9N	pyrrolidine	-5.153E+01	5.338E-01	-3.240E-04	7.528E-08	-3600	114800
352	C4H10	isobutane	-1.390E+00	3.847E-01	-1.846E-04	2.895E-08	-134600	-20900
353	C4H10O	n-butanol	3.266E+00	4.180E-01	-2.242E-04	4.685E-08	-274900	-150900
354	C4H10O	2-butanol	5.753E+00	4.245E-01	-2.328E-04	4.773E-08	-292800	-167700
355	C4H10O	isobutanol	-7.708E+00	4.689E-01	-2.884E-04	7.231E-08	-283400	-167400
356	C4H10O	tert-butanol	-4.681E+01	7.172E-01	-7.084E-04	2.920E-07	-312800	-177800
357	C4H10O	diethylether	2.142E+01	3.359E-01	-1.035E-04	-9.357E-09	-252400	-122400
358	C4H10O	methylpropylether	2.131E+01	3.390E-01	-1.127E-04	-2.855E-09	-237900	-110000
359	C4H10O	methylisopropylether	1.353E+01	3.697E-01	-1.481E-04	1.205E-08	-252200	-121000
360	C4H10O2	1,2-dimethoxyethane	3.223E+01	3.567E-01	-1.336E-04	8.399E-09	0	0
361	C4H10O3	diethyleneglycol	7.306E+01	3.461E-01	-1.468E-04	1.846E-08	-571500	0

362	C4H10S	diethylsulfide	1.359E+01	3.959E-01	-1.780E-04	2.649E-08	-83530	17800
363	C4H10S2	diethyldisulfide	2.690E+01	4.601E-01	-2.710E-04	5.970E-08	-74690	22270
364	C4H11N	n-butylamine	5.079E+00	4.476E-01	-2.407E-04	7.599E-08	-92110	49240
365	C4H11N	isobutylamine	9.491E+00	4.430E-01	-2.110E-04	2.333E-08	0	0
366	C4H11N	diethylamine	2.039E+00	4.430E-01	-2.183E-04	3.653E-08	-72430	72140
370	C5H5N	pyridine	3.979E+01	4.928E-01	-3.558E-04	1.004E-07	1.40E-05	190300
373	C5H8	cyclopentene	-4.151E+01	4.631E-01	-2.579E-04	5.434E-08	32950	110900
374	C5H8	1,2-pentadiene	8.826E+00	3.880E-01	-2.280E-04	5.246E-08	145700	210600
375	C5H8	1,3-pentadiene,trans	3.069E+01	2.811E-01	-6.711E-05	-2.352E-08	77870	146800
376	C5H8	1,4-pentadiene	6.996E+00	3.952E-01	-2.374E-04	5.598E-08	105500	170400
377	C5H8	pentyne	1.807E+01	3.511E-01	-1.913E-04	4.098E-08	144400	210400
378	C5H8	2-methyl-1,3-butadiene	-3.412E+00	4.585E-01	-3.337E-04	1.000E-07	75780	146000
379	C5H8	3-methyl-1,2-butadiene	1.469E+01	3.598E-01	-1.976E-04	4.262E-08	129800	198700
380	C5H8O	cyclopentanone	-4.064E+01	5.255E-01	-3.124E-04	7.130E-08	-192800	0
382	C5H8O2	ethylacrylate	1.681E+01	3.690E+00	-1.382E-04	-5.732E-09	0	0
383	C5H10	cyclopentane	-5.362E+01	5.426E-01	-3.031E-04	6.485E-08	-77290	38600
384	C5H10	1-pentene	-1.340E-01	4.329E-01	-2.317E-04	4.681E-08	-20930	79170
385	C5H10	2-pentene,cis	-1.429E+01	4.601E-01	-2.541E-04	5.455E-08	-28090	71890
386	C5H10	2-pentene,trans	1.947E+00	4.182E-01	-2.178E-04	4.405E-08	-31780	69960
387	C5H10	2-methyl-1-butene	1.057E+01	3.997E-01	-1.946E-04	3.314E-08	-36340	65650
388	C5H10	2-methyl-2-butene	1.180E+01	3.509E-01	-1.117E-04	-5.807E-09	-42580	59700
389	C5H10	3-methyl-1-butene	2.174E+01	3.890E-01	-2.007E-04	4.011E-08	-28970	74820
390	C5H10O	valeraldehyde	1.424E+01	4.329E-01	-2.107E-04	3.162E-08	-228000	-108400
391	C5H10O	methyl-n-propyl-ketone	1.147E+00	4.802E-01	-2.818E-04	6.661E-08	-258800	-137200
392	C5H10O	methyl	-2.914E+00	4.991E-01	-2.935E-04	6.665E-08	0	0
393	C5H10O	diethyl-ketone	3.001E+01	3.939E-01	-1.907E-04	3.398E-08	-258800	-135400
396	C5H10O2	n-valeric-acid	1.339E+01	5.033E-01	-2.931E-04	6.619E-08	-490700	-357400
398	C5H10O2	isobutyl-formate	1.985E+01	4.034E-01	-1.436E-04	-7.402E-09	0	0
399	C5H10O2	n-propyl-acetate	1.542E+01	4.501E-01	-1.686E-04	-1.439E-08	-466000	0
400	C5H10O2	ethyl-propionate	1.985E+01	4.034E-01	-1.437E-04	-7.394E-09	-470200	-323700
403	C5H11N	piperidine	-5.307E+01	6.289E-01	-3.358E-04	6.427E-08	-49030	0
404	C5H12	n-pentane	-3.626E+00	4.873E-01	-2.580E-04	5.305E-08	-146500	-8370
405	C5H12	2-methyl-butane	-9.525E+00	5.066E-01	-2.729E-04	5.723E-08	-154600	-14820
406	C5H12	2,2-dimethylpropane	-1.659E+01	5.552E-01	-3.306E-04	7.633E-08	-166100	-15240
407	C5H12O	1-pentanol	3.869E+00	5.045E-01	-2.639E-04	5.120E-08	-298900	-146100
408	C5H12O	2-methyl-1-butanol	-9.483E+00	5.677E-01	-3.481E-04	8.637E-08	-302700	-165700
409	C5H12O	3-methyl-1-butanol	-9.542E+00	5.681E-01	-3.485E-04	8.650E-08	-302300	0
410	C5H12O	2-methyl-2butanol	-1.209E+01	6.096E-01	-4.204E-04	1.228E-07	-329900	-165400
411	C5H12O	2,2-dimethyl-1-propanol	1.215E+01	5.397E-01	-3.160E-04	7.122E-08	-293100	-125500
412	C4H9N0	morpholine	-4.280E+01	5.388E-01	-2.666E-04	4.199E-08	0	0
413	AlBr3	aliminiumtribromide	6.494E+01	6.098E-02	-7.306E-05	2.978E-08	-423300	-452200
414	C6H10O	cyclohexanone	-3.781E+01	5.539E-01	-1.953E-04	-1.534E-08	-230300	-90810
429	C7H16	2-methylhexane	-3.939E+01	8.642E-01	-6.289E-04	1.836E-07	195100	3220

APPENDIX II

Thermodynamic properties of saturated water vapor (Steam)

Reference: [58] The International Association for the Properties of Water and Steam, Revised Relase on the IAPWS Formulation 1995 for the Thermodynamic Properties of Ordinary Water Substance for General and Scientific Use, Prague, Check Republic 2018, IAPWS R6-95(2018)

Program steam_IAPWS95

Temper.	Pressure	Spec. Volume		Internal energy		Enthalpy		Entropy	
T	P	Sat liq vl	Sat vapor	Sat liq ul	Sat vapor	Sat liq hl	Sat vapor	Sat liq sl	Sat vapor
degree C	kPa	m^3/kg	m^3/kg	kJ/kg	kJ/kg	kJ/kg	kJ/kg	kJ/kgK	kJ/kgK
0.01	0.611657	1.00021E-03	205.99746	-1.34E-10	2374.9112	6.12E-04	2500.911	-6.16E-08	9.1554915
1	0.657088	1.00015E-03	192.44475	4.1759929	2376.2768	4.1766501	2502.7299	0.01526	9.1290907
2	0.7059879	1.00011E-03	179.76359	8.3908959	2377.6556	8.3916019	2504.5666	0.0306065	9.1026693
3	0.7580824	1.00008E-03	168.0141	12.602783	2379.0339	12.603541	2506.4024	0.0458864	9.0764921
4	0.8135494	1.00007E-03	157.12133	16.811904	2380.4115	16.812717	2508.2375	0.061101	9.0505562
5	0.8725749	1.00008E-03	147.01686	21.018483	2381.7885	21.019356	2510.0717	0.0762517	9.0248587
6	0.9353531	1.00011E-03	137.63818	25.222728	2383.1648	25.223663	2511.9051	0.0913396	8.9993966
7	1.0020868	1.00014E-03	128.92813	29.424824	2384.5405	29.425826	2513.7377	0.1063659	8.974167
8	1.0729878	1.00020E-03	120.83443	33.624943	2385.9155	33.626017	2515.5693	0.1213316	8.9491672
9	1.1482767	1.00027E-03	113.30923	37.823241	2387.2897	37.82439	2517.4001	0.1362377	8.9243944
10	1.2281839	1.00035E-03	106.3087	42.01986	2388.6632	42.021089	2519.2298	0.1510851	8.8998459
11	1.3129491	1.00044E-03	99.792663	46.214929	2390.036	46.216243	2521.0586	0.1658747	8.875519
12	1.4028224	1.00055E-03	93.724291	50.408568	2391.4079	50.409972	2522.8864	0.1806073	8.851411
13	1.4980638	1.00067E-03	88.069781	54.600885	2392.7791	54.602384	2524.7132	0.1952838	8.8275193
14	1.5989441	1.00080E-03	82.798099	58.79198	2394.1494	58.79358	2526.5389	0.2099047	8.8038414
15	1.7057449	1.00095E-03	77.880738	62.981945	2395.5189	62.983652	2528.3636	0.2244709	8.7803747
16	1.818759	1.00110E-03	73.291491	67.170864	2396.8875	67.172684	2530.1871	0.2389831	8.7571168
17	1.9382909	1.00127E-03	69.006258	71.358815	2398.2552	71.360756	2532.0094	0.2534417	8.7340652
18	2.0646565	1.00145E-03	65.002858	75.54587	2399.6221	75.547937	2533.8307	0.2678476	8.7112174
19	2.1981843	1.00164E-03	61.260872	79.732095	2400.988	79.734297	2535.6507	0.2822012	8.6885712
20	2.3392148	1.00184E-03	57.761483	83.917553	2402.3529	83.919896	2537.4695	0.2965031	8.6661241
21	2.4881016	1.00205E-03	54.487349	88.1023	2403.7169	88.104793	2539.287	0.3107539	8.6438739
22	2.6452112	1.00228E-03	51.422477	92.286391	2405.0799	92.289042	2541.1033	0.3249542	8.6218183
23	2.8109238	1.00251E-03	48.552105	96.469874	2406.4419	96.472692	2542.9182	0.3391043	8.5999549
24	2.9856331	1.00275E-03	45.862606	100.6528	2407.8029	100.65579	2544.7318	0.3532049	8.5782817
25	3.1697469	1.00301E-03	43.341393	104.83521	2409.1628	104.83839	2546.5441	0.3672563	8.5567964
26	3.3636875	1.00327E-03	40.97683	109.01714	2410.5217	109.02052	2548.3549	0.3812591	8.5354968
27	3.567892	1.00354E-03	38.758158	113.19864	2411.8794	113.20222	2550.1643	0.3952137	8.5143809
28	3.7828126	1.00382E-03	36.675423	117.37974	2413.236	117.38354	2551.9723	0.4091206	8.4934466
29	4.0089169	1.00411E-03	34.719411	121.56048	2414.5915	121.56451	2553.7788	0.4229802	8.4726917
30	4.2466883	1.00441E-03	32.881588	125.7409	2415.9459	125.74516	2555.5837	0.4367929	8.4521142
31	4.4966264	1.00472E-03	31.154047	129.92101	2417.299	129.92553	2557.3871	0.4505592	8.4317122
32	4.7592471	1.00504E-03	29.529457	134.10087	2418.651	134.10565	2559.189	0.4642794	8.4114836
33	5.0350834	1.00536E-03	28.00102	138.28049	2420.0017	138.28555	2560.9892	0.4779539	8.3914265
34	5.3246852	1.00570E-03	26.562427	142.4599	2421.3512	142.46526	2562.7877	0.4915832	8.371539
35	5.6286201	1.00604E-03	25.20782	146.63914	2422.6993	146.6448	2564.5846	0.5051676	8.3518191
36	5.9474737	1.00639E-03	23.931759	150.81823	2424.0462	150.82421	2566.3797	0.5187075	8.3322649
37	6.2818494	1.00675E-03	22.72919	154.99719	2425.3918	155.00352	2568.1731	0.5322033	8.3128746

38	6.6323696	1.00712E-03	21.595411	159.17606	2426.736	159.18274	2569.9647	0.5456554	8.2936464
39	6.9996755	1.00749E-03	20.526052	163.35486	2428.0788	163.36191	2571.7545	0.559064	8.2745784
40	7.3844275	1.00788E-03	19.517044	167.5336	2429.4202	167.54105	2573.5424	0.5724296	8.2556689
41	7.7873057	1.00827E-03	18.5646	171.71233	2430.7602	171.72018	2575.3284	0.5857525	8.2369161
42	8.2090101	1.00867E-03	17.665195	175.89106	2432.0987	175.89934	2577.1125	0.5990331	8.2183181
43	8.6502612	1.00908E-03	16.81554	180.06981	2433.4358	180.07854	2578.8946	0.6122717	8.1998734
44	9.1118001	1.00949E-03	16.012572	184.24861	2434.7713	184.25781	2580.6746	0.6254686	8.1815802
45	9.5943888	1.00991E-03	15.253436	188.42748	2436.1053	188.43717	2582.4526	0.6386242	8.1634368
46	10.098811	1.01034E-03	14.535463	192.60645	2437.4377	192.61665	2584.2285	0.6517388	8.1454415
47	10.625872	1.01078E-03	13.856168	196.78553	2438.7684	196.79627	2586.0023	0.6648128	8.1275927
48	11.176398	1.01123E-03	13.213226	200.96475	2440.0976	200.97606	2587.7739	0.6778464	8.1098888
49	11.75124	1.01168E-03	12.604468	205.14414	2441.4251	205.15602	2589.5432	0.69084	8.0923282
50	12.35127	1.01214E-03	12.027864	209.3237	2442.7509	209.3362	2591.3103	0.7037939	8.0749092
51	12.977384	1.01260E-03	11.481519	213.50347	2444.0749	213.51661	2593.075	0.7167084	8.0576303
52	13.630501	1.01308E-03	10.96366	217.68346	2445.3972	217.69727	2594.8374	0.7295838	8.04049
53	14.311562	1.01356E-03	10.472626	221.8637	2446.7177	221.87821	2596.5973	0.7424205	8.0234867
54	15.021536	1.01404E-03	10.006865	226.04421	2448.0363	226.05945	2598.3548	0.7552186	8.0066188
55	15.761414	1.01454E-03	9.5649212	230.22502	2449.3531	230.24101	2600.1098	0.7679786	7.989885
56	16.532211	1.01504E-03	9.1454328	234.40613	2450.668	234.42291	2601.8622	0.7807007	7.9732837
57	17.334968	1.01555E-03	8.7471226	238.58757	2451.981	238.60518	2603.6121	0.7933853	7.9568134
58	18.170754	1.01606E-03	8.3687934	242.76937	2453.292	242.78783	2605.3592	0.8060325	7.9404727
59	19.040659	1.01658E-03	8.0093224	246.95154	2454.6009	246.9709	2607.1037	0.8186427	7.9242602
60	19.945802	1.01711E-03	7.6676563	251.13411	2455.9079	251.15439	2608.8454	0.8312163	7.9081744
61	20.887328	1.01765E-03	7.3428064	255.31709	2457.2127	255.33834	2610.5843	0.8437533	7.8922139
62	21.866409	1.01819E-03	7.0338445	259.50051	2458.5154	259.52277	2612.3203	0.8562543	7.8763774
63	22.884242	1.01874E-03	6.7398987	263.68438	2459.816	263.7077	2614.0534	0.8687193	7.8606635
64	23.942054	1.01929E-03	6.4601497	267.86874	2461.1143	267.89314	2615.7836	0.8811487	7.8450707
65	25.041098	1.01985E-03	6.1938278	272.05359	2462.4104	272.07912	2617.5107	0.8935428	7.8295979
66	26.182655	1.02042E-03	5.9402089	276.23896	2463.7043	276.26567	2619.2347	0.9059019	7.8142435
67	27.368035	1.02100E-03	5.698612	280.42486	2464.9958	280.45281	2620.9556	0.9182261	7.7990064
68	28.598576	1.02158E-03	5.4683965	284.61133	2466.2849	284.64055	2622.6733	0.9305158	7.7838852
69	29.875644	1.02216E-03	5.2489593	288.79838	2467.5717	288.82892	2624.3877	0.9427712	7.7688786
70	31.200636	1.02276E-03	5.0397327	292.98603	2468.856	293.01794	2626.0988	0.9549926	7.7539854
71	32.574976	1.02336E-03	4.8401818	297.17429	2470.1378	297.20763	2627.8066	0.9671803	7.7392043
72	34.000118	1.02396E-03	4.6498027	301.3632	2471.417	301.39801	2629.5109	0.9793345	7.724534
73	35.477549	1.02458E-03	4.4681205	305.55277	2472.6937	305.58912	2631.2117	0.9914554	7.7099734
74	37.008782	1.02520E-03	4.2946875	309.74301	2473.9678	309.78096	2632.909	1.0035433	7.6955211
75	38.595363	1.02582E-03	4.1290813	313.93396	2475.2392	313.97355	2634.6026	1.0155984	7.681176
76	40.238867	1.02645E-03	3.9709036	318.12563	2476.5079	318.16693	2636.2926	1.0276211	7.6669369
77	41.940901	1.02709E-03	3.8197783	322.31804	2477.7738	322.36112	2637.9788	1.0396114	7.6528025
78	43.703103	1.02773E-03	3.6753506	326.51121	2479.037	326.55612	2639.6612	1.0515698	7.6387718
79	45.527142	1.02838E-03	3.5372852	330.70516	2480.2973	330.75197	2641.3397	1.0634964	7.6248436
80	47.41472	1.02904E-03	3.4052654	334.8999	2481.5546	334.9487	2643.0143	1.0753914	7.6110166
81	49.367568	1.02970E-03	3.2789921	339.09547	2482.8091	339.14631	2644.6849	1.0872551	7.5972898
82	51.387453	1.03037E-03	3.1581825	343.29188	2484.0605	343.34483	2646.3515	1.0990878	7.5836621
83	53.47617	1.03105E-03	3.0425692	347.48915	2485.3089	347.54428	2648.0139	1.1108895	7.5701323
84	55.63555	1.03173E-03	2.9318993	351.68729	2486.5542	351.74469	2649.672	1.1226607	7.5566993
85	57.867455	1.03242E-03	2.8259335	355.88634	2487.7964	355.94608	2651.3259	1.1344015	7.543362
86	60.17378	1.03311E-03	2.7244453	360.0863	2489.0353	360.14847	2652.9755	1.1461121	7.5301194
87	62.556455	1.03381E-03	2.6272205	364.2872	2490.271	364.35188	2654.6206	1.1577927	7.5169703
88	65.017441	1.03451E-03	2.5340561	368.48906	2491.5034	368.55633	2656.2613	1.1694437	7.5039137
89	67.558733	1.03522E-03	2.4447598	372.6919	2492.7325	372.76184	2657.8973	1.1810651	7.4909486
90	70.182361	1.03594E-03	2.3591494	376.89574	2493.9581	376.96844	2659.5288	1.1926572	7.4780739
91	72.890387	1.03667E-03	2.2770523	381.10059	2495.1803	381.17616	2661.1555	1.2042203	7.4652886
92	75.68491	1.03740E-03	2.198305	385.30648	2496.399	385.385	2662.7775	1.2157545	7.4525916
93	78.568059	1.03813E-03	2.1227521	389.51343	2497.6141	389.595	2664.3946	1.2272601	7.439982
94	81.542002	1.03887E-03	2.0502465	393.72146	2498.8256	393.80617	2666.0068	1.2387372	7.4274587
95	84.608938	1.03962E-03	1.9806484	397.93058	2500.0334	398.01854	2667.6139	1.2501861	7.4150207
96	87.771103	1.04038E-03	1.9138255	402.14082	2501.2375	402.23213	2669.216	1.2616069	7.4026672
97	91.030767	1.04114E-03	1.8496517	406.35219	2502.4378	406.44697	2670.813	1.273	7.3903969
98	94.390235	1.04190E-03	1.7880077	410.56472	2503.6342	410.66307	2672.4047	1.2843655	7.3782092

99	97.851847	1.04268E-03	1.72878	414.77843	2504.8267	414.88045	2673.991	1.2957035	7.3661028
100	101.41798	1.04346E-03	1.6718606	418.99333	2506.0153	419.09915	2675.572	1.3070143	7.3540771
101	105.09104	1.04424E-03	1.6171472	423.20945	2507.1999	423.31919	2677.1475	1.3182982	7.3421308
102	108.87348	1.04503E-03	1.5645423	427.4268	2508.3803	427.54058	2678.7175	1.3295552	7.3302633
103	112.76778	1.04583E-03	1.5139533	431.64542	2509.5567	431.76335	2680.2818	1.3407856	7.3184734
104	116.77645	1.04663E-03	1.4652922	435.8653	2510.7288	435.98753	2681.8405	1.3519896	7.3067604
105	120.90206	1.04744E-03	1.4184751	440.08649	2511.8967	440.21313	2683.3933	1.3631673	7.2951234
106	125.14719	1.04826E-03	1.3734223	444.30899	2513.0603	444.44018	2684.9402	1.3743191	7.2835613
107	129.51446	1.04908E-03	1.3300579	448.53283	2514.2195	448.6687	2686.4812	1.385445	7.2720734
108	134.00654	1.04991E-03	1.2883096	452.75803	2515.3743	452.89872	2688.0162	1.3965452	7.2606587
109	138.62613	1.05074E-03	1.2481087	456.98461	2516.5245	457.13027	2689.545	1.40762	7.2493165
110	143.37597	1.05158E-03	1.2093896	461.21258	2517.6702	461.36335	2691.0676	1.4186695	7.2380458
111	148.25881	1.05243E-03	1.1720899	465.44198	2518.8113	465.59801	2692.584	1.4296939	7.2268458
112	153.27748	1.05328E-03	1.1361501	469.67281	2519.9477	469.83426	2694.0939	1.4406935	7.2157156
113	158.43481	1.05414E-03	1.1015134	473.90511	2521.0794	474.07213	2695.5975	1.4516683	7.2046545
114	163.73369	1.05500E-03	1.0681259	478.13889	2522.2062	478.31163	2697.0944	1.4626185	7.1936616
115	169.17704	1.05587E-03	1.0359358	482.37418	2523.3282	482.55281	2698.5848	1.4735445	7.182736
116	174.7678	1.05675E-03	1.0048942	486.61099	2524.4453	486.79568	2700.0684	1.4844462	7.171877
117	180.50897	1.05764E-03	0.9749539	490.84935	2525.5574	491.04026	2701.5453	1.495324	7.1610838
118	186.40358	1.05853E-03	0.9460703	495.08927	2526.6644	495.28659	2703.0153	1.5061779	7.1503555
119	192.45469	1.05942E-03	0.9182006	499.33079	2527.7663	499.53468	2704.4783	1.5170082	7.1396915
120	198.6654	1.06033E-03	0.891304	503.57392	2528.863	503.78457	2705.9342	1.527815	7.1290909
121	205.03886	1.06124E-03	0.8653414	507.81868	2529.9545	508.03627	2707.3831	1.5385986	7.1185529
122	211.57823	1.06215E-03	0.8402758	512.06509	2531.0406	512.28982	2708.8247	1.549359	7.1080768
123	218.28674	1.06307E-03	0.8160715	516.31319	2532.1214	516.54524	2710.259	1.5600966	7.0976619
124	225.16763	1.06400E-03	0.7926946	520.56298	2533.1968	520.80256	2711.686	1.5708113	7.0873073
125	232.22418	1.06494E-03	0.7701125	524.8145	2534.2667	525.06181	2713.1055	1.5815034	7.0770124
126	239.45973	1.06588E-03	0.7482942	529.06777	2535.331	529.323	2714.5174	1.5921732	7.0667765
127	246.87763	1.06683E-03	0.7272101	533.3228	2536.3898	533.58617	2715.9217	1.6028207	7.0565988
128	254.48128	1.06778E-03	0.7068316	537.57962	2537.4428	537.85135	2717.3182	1.6134461	7.0464786
129	262.27411	1.06874E-03	0.6871318	541.83826	2538.4901	542.11856	2718.707	1.6240495	7.0364151
130	270.25961	1.06971E-03	0.6680845	546.09874	2539.5316	546.38784	2720.0878	1.6346313	7.0264078
131	278.44127	1.07069E-03	0.6496648	550.36108	2540.5672	550.6592	2721.4607	1.6451914	7.0164559
132	286.82264	1.07167E-03	0.6318491	554.6253	2541.5969	554.93268	2722.8255	1.6557302	7.0065588
133	295.4073	1.07265E-03	0.6146144	558.89144	2542.6206	559.20831	2724.1822	1.6662477	6.9967157
134	304.19888	1.07365E-03	0.597939	563.15951	2543.6383	563.48611	2725.5307	1.6767441	6.9869259
135	313.20104	1.07465E-03	0.5818019	567.42953	2544.6498	567.76611	2726.8708	1.6872196	6.9771889
136	322.41746	1.07566E-03	0.5661832	571.70154	2545.6552	572.04836	2728.2026	1.6976743	6.967504
137	331.85187	1.07667E-03	0.5510635	575.97556	2546.6544	576.33286	2729.5258	1.7081085	6.9578705
138	341.50805	1.07770E-03	0.5364246	580.25162	2547.6472	580.61966	2730.8405	1.7185222	6.9482878
139	351.3898	1.07872E-03	0.5222488	584.52973	2548.6337	584.90878	2732.1466	1.7289157	6.9387553
140	361.50096	1.07976E-03	0.5085192	588.80993	2549.6138	589.20026	2733.4439	1.7392891	6.9292723
141	371.84541	1.08080E-03	0.4952195	593.09223	2550.5874	593.49412	2734.7325	1.7496425	6.9198382
142	382.42706	1.08185E-03	0.4823343	597.37668	2551.5544	597.79041	2736.0121	1.7599761	6.9104524
143	393.24986	1.08291E-03	0.4698486	601.66328	2552.5149	602.08914	2737.2828	1.7702901	6.9011143
144	404.31781	1.08397E-03	0.4577481	605.95208	2553.4687	606.39035	2738.5444	1.7805847	6.8918233
145	415.63492	1.08504E-03	0.446019	610.24309	2554.4158	610.69407	2739.7968	1.7908599	6.8825788
146	427.20526	1.08612E-03	0.4346483	614.53635	2555.3561	615.00035	2741.0401	1.801116	6.8733802
147	439.03293	1.08721E-03	0.4236231	618.83188	2556.2895	619.3092	2742.274	1.8113531	6.8642269
148	451.12205	1.08830E-03	0.4129315	623.1297	2557.2161	623.62066	2743.4986	1.8215714	6.8551184
149	463.4768	1.08940E-03	0.4025616	627.42986	2558.1356	627.93477	2744.7136	1.831771	6.8460541
150	476.10138	1.09050E-03	0.3925024	631.73237	2559.0482	632.25156	2745.9191	1.841952	6.8370333
151	489.00004	1.09162E-03	0.382743	636.03726	2559.9537	636.57107	2747.115	1.8521148	6.8280557
152	502.17706	1.09274E-03	0.3732731	640.34457	2560.852	640.89332	2748.3012	1.8622593	6.8191205
153	515.63674	1.09387E-03	0.3640827	644.65432	2561.7431	645.21836	2749.4775	1.8723857	6.8102272
154	529.38344	1.09501E-03	0.3551623	648.96654	2562.627	649.54622	2750.644	1.8824943	6.8013754
155	543.42155	1.09615E-03	0.3465026	653.28127	2563.5035	653.87694	2751.8005	1.8925851	6.7925644
156	557.75549	1.09730E-03	0.338095	657.59853	2564.3726	658.21055	2752.947	1.9026583	6.7837938
157	572.38971	1.09846E-03	0.3299307	661.91835	2565.2343	662.5471	2754.0833	1.9127141	6.7750629
158	587.32871	1.09963E-03	0.3220018	666.24076	2566.0885	666.88661	2755.2094	1.9227527	6.7663713
159	602.57702	1.10081E-03	0.3143002	670.5658	2566.9351	671.22912	2756.3251	1.9327741	6.7577185

160	618.1392	1.10199E-03	0.3068184	674.8935	2567.774	675.57468	2757.4305	1.9427785	6.7491038
161	634.01985	1.10318E-03	0.2995493	679.22388	2568.6053	679.92332	2758.5254	1.9527661	6.7405269
162	650.2236	1.10438E-03	0.2924856	683.55699	2569.4287	684.27508	2759.6098	1.9627371	6.7319872
163	666.75513	1.10559E-03	0.2856207	687.89285	2570.2444	688.63001	2760.6835	1.9726916	6.7234842
164	683.61914	1.10680E-03	0.2789482	692.2315	2571.0521	692.98813	2761.7464	1.9826297	6.7150173
165	700.82037	1.10802E-03	0.2724616	696.57297	2571.8519	697.3495	2762.7985	1.9925516	6.7065862
166	718.3636	1.10925E-03	0.2661551	700.9173	2572.6436	701.71415	2763.8398	2.0024574	6.6981902
167	736.25363	1.11049E-03	0.2600227	705.26451	2573.4273	706.08212	2764.87	2.0123474	6.6898289
168	754.49532	1.11174E-03	0.2540589	709.61465	2574.2028	710.45346	2765.8891	2.0222216	6.6815019
169	773.09353	1.11300E-03	0.2482584	713.96775	2574.9701	714.8282	2766.897	2.0320802	6.6732085
170	792.05318	1.11426E-03	0.2426158	718.32384	2575.729	719.2064	2767.8937	2.0419234	6.6649484
171	811.37923	1.11554E-03	0.2371262	722.68297	2576.4797	723.58809	2768.8789	2.0517513	6.6567211
172	831.07665	1.11682E-03	0.2317848	727.04516	2577.2218	727.97332	2769.8528	2.061564	6.648526
173	851.15046	1.11811E-03	0.2265869	731.41045	2577.9555	732.36213	2770.815	2.0713618	6.6403627
174	871.6057	1.11941E-03	0.221528	735.77889	2578.6806	736.75457	2771.7657	2.0811448	6.6322308
175	892.44748	1.12071E-03	0.2166038	740.15051	2579.397	741.15069	2772.7045	2.090913	6.6241297
176	913.6809	1.12203E-03	0.2118101	744.52534	2580.1048	745.55052	2773.6316	2.1006668	6.616059
177	935.31111	1.12336E-03	0.2071428	748.90343	2580.8037	749.95412	2774.5467	2.1104062	6.6080182
178	957.34331	1.12469E-03	0.2025981	753.28482	2581.4938	754.36154	2775.4498	2.1201314	6.6000069
179	979.78271	1.12604E-03	0.1981723	757.66954	2582.1749	758.77281	2776.3407	2.1298425	6.5920246
180	1002.6346	1.12739E-03	0.1938616	762.05764	2582.8471	763.188	2777.2194	2.1395397	6.5840708
181	1025.9042	1.12875E-03	0.1896626	766.44915	2583.5101	767.60714	2778.0858	2.1492231	6.5761451
182	1049.5968	1.13012E-03	0.185572	770.84412	2584.164	772.0303	2778.9397	2.158893	6.568247
183	1073.7179	1.13150E-03	0.1815864	775.24259	2584.8086	776.45751	2779.7811	2.1685494	6.5603761
184	1098.2728	1.13289E-03	0.1777027	779.64461	2585.4439	780.88883	2780.6099	2.1781925	6.5525319
185	1123.2669	1.13429E-03	0.1739178	784.0502	2586.0698	785.32432	2781.4259	2.1878225	6.544714
186	1148.7057	1.13570E-03	0.1702289	788.45943	2586.6862	789.76402	2782.2291	2.1974394	6.5369219
187	1174.5947	1.13712E-03	0.166633	792.87232	2587.293	794.20798	2783.0193	2.2070436	6.5291552
188	1200.9393	1.13855E-03	0.1631274	797.28893	2587.8902	798.65626	2783.7964	2.216635	6.5214134
189	1227.7452	1.13999E-03	0.1597096	801.7093	2588.4777	803.10892	2784.5603	2.226214	6.5136961
190	1255.0179	1.14144E-03	0.1563768	806.13348	2589.0553	807.56601	2785.311	2.2357806	6.5060028
191	1282.7631	1.14290E-03	0.1531266	810.56151	2589.6231	812.02758	2786.0483	2.2453349	6.4983332
192	1310.9863	1.14437E-03	0.1499567	814.99344	2590.1808	816.4937	2786.772	2.2548773	6.4906867
193	1339.6933	1.14585E-03	0.1468647	819.42932	2590.7285	820.96441	2787.4821	2.2644077	6.483063
194	1368.8898	1.14734E-03	0.1438483	823.86919	2591.266	825.43977	2788.1785	2.2739264	6.4754615
195	1398.5815	1.14885E-03	0.1409054	828.3131	2591.7933	829.91985	2788.861	2.2834334	6.4678819
196	1428.7742	1.15036E-03	0.138034	832.76111	2592.3102	834.40471	2789.5296	2.2929291	6.4603238
197	1459.4737	1.15188E-03	0.1352318	837.21326	2592.8167	838.8944	2790.184	2.3024135	6.4527866
198	1490.6859	1.15341E-03	0.1324971	841.6696	2593.3126	843.38898	2790.8242	2.3118868	6.44527
199	1522.4167	1.15496E-03	0.1298279	846.13019	2593.798	847.88851	2791.4501	2.3213492	6.4377735
200	1554.6719	1.15651E-03	0.1272223	850.59507	2594.2726	852.39307	2792.0616	2.3308007	6.4302968
201	1587.4574	1.15808E-03	0.1246786	855.0643	2594.7364	856.9027	2792.6584	2.3402416	6.4228393
202	1620.7794	1.15966E-03	0.1221951	859.53793	2595.1893	861.41748	2793.2405	2.3496721	6.4154006
203	1654.6437	1.16125E-03	0.11977	864.01602	2595.6312	865.93747	2793.8078	2.3590922	6.4079804
204	1689.0563	1.16285E-03	0.1174017	868.49862	2596.062	870.46274	2794.3601	2.3685022	6.4005782
205	1724.0235	1.16446E-03	0.1150888	872.98579	2596.4816	874.99335	2794.8974	2.3779023	6.3931935
206	1759.5512	1.16609E-03	0.1128296	877.47758	2596.8898	879.52937	2795.4194	2.3872925	6.385826
207	1795.6456	1.16773E-03	0.1106228	881.97404	2597.2867	884.07086	2795.926	2.396673	6.3784753
208	1832.3128	1.16937E-03	0.1084668	886.47525	2597.6721	888.61791	2796.4171	2.4060441	6.3711408
209	1869.5591	1.17104E-03	0.1063603	890.98125	2598.0458	893.17057	2796.8926	2.4154058	6.3638222
210	1907.3907	1.17271E-03	0.1043019	895.49211	2598.4078	897.72892	2797.3523	2.4247584	6.3565191
211	1945.8137	1.17439E-03	0.1022905	900.00788	2598.758	902.29303	2797.7962	2.4341019	6.349231
212	1984.8346	1.17609E-03	0.1003246	904.52863	2599.0962	906.86298	2798.2239	2.4434367	6.3419575
213	2024.4596	1.17781E-03	0.0984031	909.05441	2599.4224	911.43883	2798.6355	2.4527628	6.3346982
214	2064.6951	1.17953E-03	0.0965249	913.5853	2599.7364	916.02067	2799.0308	2.4620804	6.3274527
215	2105.5475	1.18127E-03	0.0946886	918.12135	2600.0381	920.60857	2799.4096	2.4713897	6.3202206
216	2147.0231	1.18302E-03	0.0928934	922.66263	2600.3275	925.2026	2799.7717	2.4806908	6.3130014
217	2189.1284	1.18478E-03	0.091138	927.20921	2600.6043	929.80285	2800.1171	2.489984	6.3057947
218	2231.87	1.18656E-03	0.0894214	931.76115	2600.8686	934.4094	2800.4455	2.4992694	6.2986001
219	2275.2542	1.18835E-03	0.0877426	936.31852	2601.12	939.02232	2800.7569	2.5085471	6.2914172
220	2319.2877	1.19016E-03	0.0861007	940.88138	2601.3587	943.6417	2801.051	2.5178174	6.2842455

221	2363.977	1.19198E-03	0.0844947	945.44982	2601.5843	948.26762	2801.3277	2.5270804	6.2770847
222	2409.3288	1.19381E-03	0.0829236	950.02388	2601.7968	952.90016	2801.5869	2.5363364	6.2699343
223	2455.3495	1.19566E-03	0.0813865	954.60366	2601.9961	957.53942	2801.8284	2.5455854	6.2627939
224	2502.046	1.19752E-03	0.0798826	959.18922	2602.1821	962.18547	2802.0521	2.5548277	6.2556631
225	2549.4249	1.19940E-03	0.0784111	963.78064	2602.3545	966.83841	2802.2577	2.5640634	6.2485415
226	2597.493	1.20129E-03	0.076971	968.37798	2602.5134	971.49832	2802.4451	2.5732927	6.2414286
227	2646.257	1.20320E-03	0.0755617	972.98133	2602.6585	976.1653	2802.6141	2.5825159	6.234324
228	2695.7237	1.20512E-03	0.0741823	977.59076	2602.7897	980.83943	2802.7646	2.591733	6.2272273
229	2745.8999	1.20706E-03	0.072832	982.20636	2602.9069	985.52081	2802.8964	2.6009444	6.2201381
230	2796.7925	1.20901E-03	0.0715102	986.82819	2603.01	990.20954	2803.0093	2.6101501	6.2130559
231	2848.4083	1.21098E-03	0.0702162	991.45634	2603.0988	994.90571	2803.1032	2.6193504	6.2059803
232	2900.7544	1.21297E-03	0.0689492	996.0909	2603.1731	999.60942	2803.1778	2.6285454	6.1989109
233	2953.8376	1.21497E-03	0.0677085	1000.7319	2603.2329	1004.3208	2803.233	2.6377353	6.1918473
234	3007.665	1.21699E-03	0.0664936	1005.3796	2603.278	1009.0399	2803.2686	2.6469204	6.184789
235	3062.2436	1.21902E-03	0.0653038	1010.0338	2603.3082	1013.7668	2803.2844	2.6561009	6.1777356
236	3117.5803	1.22108E-03	0.0641385	1014.6949	2603.3234	1018.5017	2803.2802	2.6652769	6.1706867
237	3173.6824	1.22315E-03	0.062997	1019.3627	2603.3234	1023.2446	2803.2559	2.6744486	6.1636418
238	3230.5569	1.22523E-03	0.0618788	1024.0375	2603.3081	1027.9957	2803.2112	2.6836162	6.1566005
239	3288.211	1.22734E-03	0.0607834	1028.7193	2603.2774	1032.755	2803.146	2.69278	6.1495624
240	3346.6519	1.22946E-03	0.0597101	1033.4082	2603.231	1037.5228	2803.06	2.7019402	6.142527
241	3405.8867	1.23160E-03	0.0586585	1038.1043	2603.1688	1042.299	2802.953	2.7110969	6.1354938
242	3465.9228	1.23376E-03	0.057628	1042.8077	2603.0907	1047.0838	2802.8249	2.7202503	6.1284626
243	3526.7675	1.23594E-03	0.0566181	1047.5185	2602.9964	1051.8774	2802.6754	2.7294008	6.1214327
244	3588.428	1.23814E-03	0.0556284	1052.2368	2602.8859	1056.6798	2802.5043	2.7385484	6.1144037
245	3650.9118	1.24035E-03	0.0546583	1056.9627	2602.7589	1061.4911	2802.3114	2.7476934	6.1073752
246	3714.2262	1.24259E-03	0.0537073	1061.6963	2602.6153	1066.3116	2802.0965	2.7568361	6.1003468
247	3778.3786	1.24485E-03	0.0527751	1066.4378	2602.4549	1071.1413	2801.8593	2.7659766	6.0933179
248	3843.3765	1.24712E-03	0.0518612	1071.1871	2602.2775	1075.9803	2801.5997	2.7751153	6.0862881
249	3909.2275	1.24942E-03	0.0509652	1075.9445	2602.0829	1080.8287	2801.3173	2.7842522	6.079257
250	3975.9391	1.25174E-03	0.0500866	1080.71	2601.8709	1085.6868	2801.0121	2.7933876	6.0722241
251	4043.5188	1.25408E-03	0.049225	1085.4837	2601.6415	1090.5546	2800.6836	2.8025219	6.0651889
252	4111.9742	1.25644E-03	0.0483801	1090.2659	2601.3942	1095.4323	2800.3318	2.8116551	6.0581509
253	4181.313	1.25882E-03	0.0475514	1095.0565	2601.1291	1100.32	2799.9563	2.8207876	6.0511096
254	4251.543	1.26122E-03	0.0467386	1099.8557	2600.8457	1105.2179	2799.5569	2.8299196	6.0440646
255	4322.6717	1.26365E-03	0.0459413	1104.6637	2600.5441	1110.126	2799.1333	2.8390514	6.0370154
256	4394.7071	1.26610E-03	0.0451592	1109.4805	2600.2239	1115.0446	2798.6853	2.8481831	6.0299614
257	4467.6568	1.26857E-03	0.0443919	1114.3063	2599.8849	1119.9738	2798.2126	2.8573151	6.0229022
258	4541.5288	1.27107E-03	0.043639	1119.1412	2599.527	1124.9138	2797.7149	2.8664476	6.0158372
259	4616.3309	1.27358E-03	0.0429003	1123.9854	2599.1498	1129.8647	2797.192	2.8755808	6.008766
260	4692.0711	1.27613E-03	0.0421755	1128.8389	2598.7533	1134.8266	2796.6436	2.8847152	6.001688
261	4768.7572	1.27870E-03	0.0414641	1133.702	2598.337	1139.7998	2796.0694	2.8938508	5.9946027
262	4846.3974	1.28129E-03	0.040766	1138.5747	2597.9009	1144.7844	2795.4691	2.902988	5.9875096
263	4924.9997	1.28391E-03	0.0400808	1143.4573	2597.4446	1149.7806	2794.8424	2.912127	5.9804081
264	5004.5722	1.28656E-03	0.0394082	1148.3498	2596.9679	1154.7885	2794.189	2.9212683	5.9732976
265	5085.1229	1.28923E-03	0.0387479	1153.2524	2596.4706	1159.8083	2793.5086	2.9304119	5.9661776
266	5166.6602	1.29193E-03	0.0380997	1158.1653	2595.9524	1164.8402	2792.8008	2.9395583	5.9590476
267	5249.1922	1.29466E-03	0.0374634	1163.0886	2595.4131	1169.8845	2792.0654	2.9487077	5.9519069
268	5332.7272	1.29741E-03	0.0368385	1168.0224	2594.8523	1174.9412	2791.302	2.9578605	5.944755
269	5417.2735	1.30019E-03	0.036225	1172.9671	2594.2697	1180.0106	2790.5103	2.9670169	5.9375913
270	5502.8395	1.30301E-03	0.0356224	1177.9226	2593.6652	1185.0928	2789.6899	2.9761774	5.9304151
271	5589.4336	1.30585E-03	0.0350307	1182.8892	2593.0385	1190.1882	2788.8404	2.9853421	5.9232259
272	5677.0643	1.30872E-03	0.0344496	1187.8671	2592.3891	1195.2968	2787.9615	2.9945114	5.916023
273	5765.74	1.31162E-03	0.0338787	1192.8565	2591.7168	1200.4189	2787.0528	3.0036857	5.9088058
274	5855.4695	1.31455E-03	0.033318	1197.8575	2591.0214	1205.5548	2786.114	3.0128653	5.9015736
275	5946.2612	1.31752E-03	0.0327672	1202.8703	2590.3024	1210.7046	2785.1446	3.0220506	5.8943258
276	6038.1238	1.32052E-03	0.032226	1207.8951	2589.5596	1215.8686	2784.1442	3.0312419	5.8870616
277	6131.0662	1.32355E-03	0.0316943	1212.9322	2588.7925	1221.047	2783.1124	3.0404396	5.8797804
278	6225.0969	1.32661E-03	0.0311719	1217.9817	2588.001	1226.24	2782.0488	3.049644	5.8724815
279	6320.225	1.32971E-03	0.0306585	1223.0438	2587.1845	1231.4479	2780.953	3.0588556	5.8651642
280	6416.4593	1.33285E-03	0.030154	1228.1189	2586.3428	1236.671	2779.8245	3.0680748	5.8578276
281	6513.8087	1.33602E-03	0.0296581	1233.2069	2585.4754	1241.9095	2778.6628	3.0773019	5.8504711

282	6612.2822	1.33922E-03	0.0291708	1238.3083	2584.582	1247.1636	2777.4674	3.0865373	5.8430939
283	6711.889	1.34247E-03	0.0286917	1243.4233	2583.6622	1252.4338	2776.238	3.0957815	5.8356952
284	6812.6382	1.34575E-03	0.0282208	1248.552	2582.7156	1257.7201	2774.9739	3.105035	5.8282742
285	6914.5389	1.34907E-03	0.0277579	1253.6948	2581.7417	1263.023	2773.6747	3.1142981	5.82083
286	7017.6004	1.35243E-03	0.0273027	1258.8518	2580.7401	1268.3426	2772.3398	3.1235712	5.8133619
287	7121.8321	1.35583E-03	0.0268552	1264.0234	2579.7104	1273.6794	2770.9687	3.132855	5.8058689
288	7227.2434	1.35928E-03	0.0264152	1269.2098	2578.6521	1279.0337	2769.5609	3.1421498	5.7983502
289	7333.8437	1.36276E-03	0.0259824	1274.4114	2577.5647	1284.4056	2768.1157	3.1514561	5.7908048
290	7441.6425	1.36629E-03	0.0255568	1279.6283	2576.4479	1289.7957	2766.6326	3.1607745	5.7832319
291	7550.6496	1.36987E-03	0.0251382	1284.8608	2575.301	1295.2042	2765.111	3.1701054	5.7756304
292	7660.8746	1.37349E-03	0.0247265	1290.1094	2574.1235	1300.6315	2763.5503	3.1794494	5.7679995
293	7772.3272	1.37715E-03	0.0243215	1295.3742	2572.915	1306.0779	2761.9497	3.188807	5.760338
294	7885.0174	1.38087E-03	0.0239231	1300.6557	2571.6749	1311.5438	2760.3087	3.1981788	5.752645
295	7998.955	1.38463E-03	0.0235311	1305.9541	2570.4026	1317.0297	2758.6266	3.2075653	5.7449194
296	8114.15	1.38844E-03	0.0231454	1311.2697	2569.0975	1322.5358	2756.9027	3.2169672	5.7371602
297	8230.6127	1.39231E-03	0.0227659	1316.603	2567.7591	1328.0626	2755.1361	3.226385	5.7293661
298	8348.3531	1.39622E-03	0.0223924	1321.9543	2566.3868	1333.6105	2753.3263	3.2358194	5.7215361
299	8467.3815	1.40020E-03	0.0220248	1327.3239	2564.9798	1339.1799	2751.4725	3.2452711	5.713669
300	8587.7083	1.40422E-03	0.0216631	1332.7123	2563.5377	1344.7713	2749.5737	3.2547406	5.7057636
301	8709.3441	1.40831E-03	0.021307	1338.1198	2562.0596	1350.3852	2747.6293	3.2642286	5.6978186
302	8832.2993	1.41245E-03	0.0209564	1343.5468	2560.5448	1356.022	2745.6384	3.2737359	5.6898327
303	8956.5846	1.41665E-03	0.0206114	1348.9939	2558.9928	1361.6822	2743.6001	3.2832632	5.6818047
304	9082.2109	1.42091E-03	0.0202716	1354.4613	2557.4026	1367.3663	2741.5135	3.2928112	5.673733
305	9209.189	1.42524E-03	0.019937	1359.9495	2555.7736	1373.0748	2739.3776	3.3023807	5.6656164
306	9337.5298	1.42963E-03	0.0196076	1365.4591	2554.1049	1378.8083	2737.1915	3.3119725	5.6574533
307	9467.2446	1.43409E-03	0.0192832	1370.9905	2552.3957	1384.5674	2734.9542	3.3215875	5.6492422
308	9598.3446	1.43861E-03	0.0189636	1376.5442	2550.6451	1390.3525	2732.6646	3.3312263	5.6409815
309	9730.8411	1.44321E-03	0.0186489	1382.1206	2548.8522	1396.1643	2730.3217	3.3408901	5.6326697
310	9864.7456	1.44788E-03	0.0183389	1387.7205	2547.0161	1402.0034	2727.9243	3.3505795	5.6243049
311	10000.07	1.45262E-03	0.0180334	1393.3442	2545.1358	1407.8705	2725.4713	3.3602957	5.6158855
312	10136.825	1.45744E-03	0.0177325	1398.9924	2543.2103	1413.7662	2722.9614	3.3700395	5.6074097
313	10275.024	1.46234E-03	0.0174359	1404.6656	2541.2386	1419.6912	2720.3933	3.379812	5.5988754
314	10414.678	1.46732E-03	0.0171437	1410.3646	2539.2194	1425.6463	2717.7657	3.3896142	5.5902806
315	10555.799	1.47239E-03	0.0168557	1416.0898	2537.1516	1431.6321	2715.0772	3.3994472	5.5816234
316	10698.4	1.47754E-03	0.0165718	1421.8421	2535.0341	1437.6494	2712.3264	3.4093121	5.5729015
317	10842.494	1.48278E-03	0.016292	1427.6221	2532.8655	1443.6991	2709.5116	3.4192103	5.5641126
318	10988.093	1.48811E-03	0.0160161	1433.4305	2530.6445	1449.782	2706.6314	3.4291427	5.5552543
319	11135.209	1.49354E-03	0.0157441	1439.2682	2528.3697	1455.899	2703.684	3.4391109	5.5463242
320	11283.856	1.49906E-03	0.0154759	1445.1358	2526.0396	1462.051	2700.6677	3.4491161	5.5373194
321	11434.047	1.50469E-03	0.0152114	1451.0343	2523.6526	1468.239	2697.5805	3.4591597	5.5282374
322	11585.795	1.51042E-03	0.0149505	1456.9645	2521.207	1474.464	2694.4206	3.4692433	5.5190752
323	11739.115	1.51627E-03	0.0146932	1462.9274	2518.7013	1480.727	2691.1859	3.4793685	5.5098296
324	11894.019	1.52222E-03	0.0144392	1468.9239	2516.1334	1487.0293	2687.8741	3.4895368	5.5004974
325	12050.522	1.52830E-03	0.0141887	1474.9552	2513.5016	1493.3719	2684.483	3.49975	5.4910752
326	12208.637	1.53449E-03	0.0139415	1481.0222	2510.8037	1499.7562	2681.0102	3.51001	5.4815594
327	12368.38	1.54081E-03	0.0136975	1487.1262	2508.0375	1506.1835	2677.4532	3.5203188	5.4719462
328	12529.765	1.54727E-03	0.0134566	1493.2683	2505.2009	1512.6553	2673.8091	3.5306785	5.4622315
329	12692.808	1.55386E-03	0.0132188	1499.45	2502.2913	1519.1729	2670.0751	3.5410912	5.4524111
330	12857.522	1.56060E-03	0.012984	1505.6726	2499.3063	1525.738	2666.2483	3.5515594	5.4424804
331	13023.924	1.56748E-03	0.0127521	1511.9376	2496.243	1532.3524	2662.3254	3.5620855	5.4324347
332	13192.029	1.57452E-03	0.012523	1518.2467	2493.0987	1539.0178	2658.3031	3.5726723	5.4222689
333	13361.854	1.58173E-03	0.0122967	1524.6014	2489.8703	1545.7363	2654.1776	3.5833228	5.4119778
334	13533.414	1.58911E-03	0.0120731	1531.0038	2486.5546	1552.5098	2649.9454	3.5940399	5.4015556
335	13706.727	1.59667E-03	0.0118522	1537.4556	2483.1481	1559.3407	2645.6023	3.6048271	5.3909964
336	13881.809	1.60442E-03	0.0116337	1543.9592	2479.6473	1566.2314	2641.1441	3.6156879	5.3802939
337	14058.677	1.61236E-03	0.0114177	1550.5167	2476.0484	1573.1844	2636.5664	3.6266263	5.3694415
338	14237.35	1.62052E-03	0.0112041	1557.1307	2472.3472	1580.2026	2631.8643	3.6376462	5.358432
339	14417.845	1.62890E-03	0.0109929	1563.8037	2468.5395	1587.2889	2627.0329	3.6487523	5.3472579
340	14600.181	1.63751E-03	0.0107838	1570.5386	2464.6208	1594.4466	2622.0667	3.6599494	5.3359116
341	14784.377	1.64637E-03	0.010577	1577.3384	2460.5863	1601.6791	2616.9604	3.6712425	5.3243845
342	14970.451	1.65550E-03	0.0103722	1584.2066	2456.4308	1608.9901	2611.7078	3.6826374	5.3126679

343	15158.424	1.66490E-03	0.0101695	1591.1466	2452.149	1616.3838	2606.3027	3.69414	5.3007526
344	15348.316	1.67459E-03	0.0099687	1598.1622	2447.7353	1623.8644	2600.7385	3.7057568	5.2886288
345	15540.148	1.68460E-03	0.0097698	1605.2575	2443.1836	1631.4365	2595.0082	3.7174947	5.2762861
346	15733.941	1.69494E-03	0.0095727	1612.4371	2438.4875	1639.1052	2589.1044	3.7293612	5.2637137
347	15929.716	1.70563E-03	0.0093774	1619.7055	2433.6404	1646.8757	2583.0193	3.7413643	5.2509001
348	16127.496	1.71670E-03	0.0091837	1627.0678	2428.6351	1654.754	2576.7448	3.7535125	5.2378332
349	16327.304	1.72817E-03	0.0089915	1634.5295	2423.4643	1662.7459	2570.2721	3.7658149	5.2245002
350	16529.164	1.74007E-03	0.0088009	1642.0963	2418.12	1670.8582	2563.592	3.7782813	5.2108878
351	16733.1	1.75253E-03	0.0086125	1649.8188	2412.6054	1679.144	2556.7187	3.7909888	5.1970208
352	16939.138	1.76545E-03	0.0084243	1657.6373	2406.8606	1687.5425	2549.5616	3.8038533	5.1827577
353	17147.303	1.77893E-03	0.0082373	1665.591	2400.8961	1696.0949	2542.1436	3.8169328	5.1681298
354	17357.621	1.79303E-03	0.0080512	1673.6901	2394.6965	1704.8128	2534.4467	3.8302453	5.1531123
355	17570.122	1.80780E-03	0.007866	1681.9461	2388.2439	1713.7093	2526.45	3.843811	5.1376756
356	17784.833	1.82330E-03	0.0076813	1690.3723	2381.518	1722.7995	2518.1294	3.8576526	5.1217842
357	18001.785	1.83963E-03	0.0074972	1698.984	2374.4952	1732.1007	2509.4574	3.871796	5.1053966
358	18221.009	1.85687E-03	0.0073132	1707.7991	2367.1483	1741.633	2500.4022	3.8862713	5.0884644
359	18442.537	1.87512E-03	0.0071292	1716.8385	2359.4454	1751.4204	2490.9264	3.9011132	5.070931
360	18666.403	1.89451E-03	0.0069449	1726.1272	2351.3491	1761.4909	2480.9862	3.9163638	5.0527293
361	18892.645	1.91521E-03	0.00676	1735.6952	2342.8145	1771.8785	2470.529	3.9320737	5.0337793
362	19121.298	1.93739E-03	0.006574	1745.5789	2333.787	1782.6244	2459.4911	3.9483058	5.0139843
363	19352.406	1.96131E-03	0.0063865	1755.8232	2324.1996	1793.7792	2447.7944	3.9651385	4.9932254
364	19586.011	1.98725E-03	0.0061969	1766.4845	2313.9683	1805.4068	2435.3407	3.9826712	4.971354
365	19822.162	2.01561E-03	0.0060044	1777.6355	2302.9857	1817.5893	2422.005	4.0010319	4.9481805
366	20060.914	2.04691E-03	0.0058079	1789.372	2291.1116	1830.435	2407.6241	4.0203893	4.9234564
367	20302.327	2.08187E-03	0.0056063	1801.825	2278.1568	1844.0918	2391.9777	4.040971	4.8968455
368	20546.474	2.12152E-03	0.0053976	1815.1804	2263.8563	1858.77	2374.7574	4.0630979	4.8678748
369	20793.448	2.16741E-03	0.005179	1829.7168	2247.818	1874.7846	2355.5073	4.0872475	4.8358417
370	21043.367	2.22209E-03	0.0049462	1845.8829	2229.4171	1892.6432	2333.5017	4.1141882	4.7996265
371	21296.405	2.29020E-03	0.0046914	1864.4808	2207.547	1913.2539	2307.457	4.1453013	4.7572517
372	21552.836	2.38170E-03	0.0043985	1887.2068	2179.895	1938.5391	2274.6946	4.1835379	4.7046055
373	21813.163	2.52643E-03	0.0040212	1919.027	2139.8312	1974.1365	2227.5467	4.2376474	4.629939

Thermodynamic properties of superheated steam

P=6.0	kPa			
Tsat=36.16025915111595degree	C			
T degree C	v m3/kg	u kJ/kg	h kJ/kg	s kJ/KgK
36.16026	23.734218	2424.262	2566.6673	8.3291465
60	25.585953	2458.5562	2612.0719	8.4705696
80	27.133344	2487.1794	2649.9795	8.5810706
100	28.677999	2515.8553	2687.9233	8.6855812
120	30.220911	2544.6401	2725.9656	8.7848907
140	31.762668	2573.5708	2764.1468	8.8796158
160	33.303635	2602.6736	2802.4954	8.9702574
180	34.844051	2631.9673	2841.0316	9.0572303
200	36.384068	2661.4662	2879.7706	9.1408841
220	37.923792	2691.1811	2918.7238	9.2215175
240	39.463296	2721.1203	2957.9	9.2993881
260	41.002629	2751.2905	2997.3063	9.3747209
280	42.541828	2781.6975	3036.9485	9.4477135
300	44.08092	2812.3459	3076.8315	9.5185409
320	45.619925	2843.2401	3116.9596	9.5873592
340	47.158859	2874.3837	3157.3369	9.6543081
360	48.697732	2905.7804	3197.9668	9.7195135
380	50.236555	2937.4335	3238.8528	9.7830891
400	51.775335	2969.3462	3279.9982	9.8451381
420	53.314078	3001.5214	3321.4059	9.9057544
440	54.852789	3033.9622	3363.0789	9.9650237
460	56.391472	3066.6711	3405.0199	10.023024
480	57.93013	3099.6508	3447.2316	10.079828
500	59.468767	3132.9036	3489.7162	10.1355
520	61.007386	3166.4317	3532.476	10.190102
540	62.545987	3200.237	3575.5129	10.243689
560	64.084573	3234.3212	3618.8286	10.296313
580	65.623146	3268.6856	3662.4245	10.348021
600	67.161707	3303.3315	3706.3018	10.398857

620	68.700257	3338.2598	3750.4613	10.44886	
640	70.238797	3373.471	3794.9038	10.49807	
660	71.777328	3408.9655	3839.6295	10.54652	
680	73.315852	3444.7435	3884.6386	10.594244	
700	74.854368	3480.805	3929.9312	10.64127	
720	76.392878	3517.1497	3975.5069	10.687628	
==========					
P=35.0	kPa				
Tsat=72.6806788087946degree	C				
T degree C	v m3/kg	u kJ/kg	h kJ/kg	s kJ/KgK	
72.68068	4.5252181	2472.2863	2630.669	7.714611	
100	4.8951171	2512.9799	2684.309	7.863915	
120	5.1634701	2542.4281	2723.1495	7.9653115	
140	5.4306128	2571.822	2761.8934	8.0614345	
160	5.6969366	2601.2568	2800.6496	8.1530407	
180	5.9626898	2630.7944	2839.4885	8.2406975	
200	6.2280346	2660.4764	2878.4576	8.3248486	
220	6.4930797	2690.3317	2917.5895	8.4058522	
240	6.7578996	2720.3809	2956.9074	8.4840048	
260	7.0225466	2750.6392	2996.4283	8.5595567	
280	7.2870581	2781.1177	3036.1647	8.632723	
300	7.5514615	2811.8253	3076.1265	8.7036903	
320	7.8157772	2842.7692	3116.3214	8.7726231	
340	8.0800207	2873.9551	3156.7558	8.839667	
360	8.3442041	2905.3883	3197.4354	8.9049521	
380	8.6083367	2937.0731	3238.3649	8.9685953	
400	8.8724262	2969.0137	3279.5486	9.0307022	
420	9.1364785	3001.2136	3320.9903	9.0913683	
440	9.4004987	3033.6763	3362.6938	9.1506809	
460	9.6644911	3066.405	3404.6622	9.2087193	
480	9.9284589	3099.4024	3446.8985	9.265556	
500	10.192405	3132.6713	3489.4055	9.3212578	
520	10.456333	3166.2139	3532.1856	9.3758858	
540	10.720243	3200.0325	3575.241	9.429496	
560	10.984138	3234.1288	3618.5736	9.4821403	
580	11.24802	3268.5043	3662.185	9.5338665	
600	11.51189	3303.1604	3706.0765	9.5847185	
620	11.775749	3338.0979	3750.2492	9.6347372	
640	12.039599	3373.3177	3794.7037	9.6839603	
660	12.30344	3408.8201	3839.4405	9.7324227	
680	12.567272	3444.6054	3884.4599	9.780157	
700	12.831098	3480.6736	3929.762	9.8271935	
720	13.094917	3517.0245	3975.3466	9.8735604	
740	13.35873	3553.6579	4021.2134	9.9192843	
760	13.622538	3590.5735	4067.3623	9.96439	
==========					
P=70.0	kPa				
Tsat=89.93151006644666degree	C				
T degree C	v m3/kg	u kJ/kg	h kJ/kg	s kJ/KgK	
89.93151	2.3648989	2493.8743	2659.4172	7.4789529	
110	2.5029615	2524.5982	2699.8055	7.5872365	
130	2.6389073	2554.7054	2739.4289	7.6880469	
150	2.7738106	2584.6016	2778.7683	7.7832856	
170	2.9080043	2614.436	2817.9963	7.8738664	
190	3.0417001	2644.3033	2857.2223	7.9604431	
210	3.1750369	2674.2668	2896.5194	8.043509	
230	3.3081083	2704.3698	2935.9373	8.1234499	
250	3.4409789	2734.6424	2975.5109	8.2005776	
270	3.5736944	2765.1064	3015.265	8.2751498	
290	3.7062876	2795.7778	3055.218	8.3473846	
310	3.8387825	2826.6688	3095.3835	8.4174692	
330	3.9711973	2857.7887	3135.7725	8.4855669	
350	4.1035459	2889.1453	3176.3935	8.5518214	
370	4.235839	2920.745	3217.2537	8.6163606	
390	4.3680852	2952.5932	3258.3592	8.6792987	
410	4.5002912	2984.6947	3299.7151	8.7407387	
430	4.6324626	3017.0538	3341.3262	8.8007739	
450	4.7646041	3049.6743	3383.1966	8.8594887	
470	4.8967193	3082.5596	3425.3299	8.9169604	
490	5.0288117	3115.7128	3467.7296	8.9732594	
510	5.1608837	3149.1367	3510.3986	9.0284503	
530	5.2929378	3182.8339	3553.3395	9.0825921	

550	5.424976	3216.8064	3596.5547	9.1357395	
570	5.557	3251.0561	3640.0461	9.1879425	
590	5.6890113	3285.5846	3683.8154	9.2392473	
610	5.8210111	3320.393	3727.8638	9.2896967	
630	5.9530008	3355.4824	3772.1925	9.33933	
650	6.0849812	3390.8534	3816.8021	9.3881838	
670	6.2169533	3426.5064	3861.6931	9.4362921	
690	6.3489179	3462.4415	3906.8658	9.4836864	
710	6.4808756	3498.6589	3952.3202	9.530396	
730	6.6128272	3535.1583	3998.0562	9.5764486	
750	6.7447731	3571.9395	4044.0736	9.6218697	
770	6.8767139	3609.0023	4090.3722	9.6666838	
==========					
P=100.0	kPa				
Tsat=99.60591861133764degree	C				
T degree C	v m3/kg	u kJ/kg	h kJ/kg	s kJ/KgK	
99.60592	1.6940225	2505.5474	2674.9496	7.3588066	
120	1.7932375	2537.2838	2716.6075	7.4676262	
140	1.889133	2567.7862	2756.6995	7.5670986	
160	1.9841386	2598.0068	2796.4207	7.6609887	
180	2.0785331	2628.1168	2835.9701	7.7502507	
200	2.1724948	2658.2256	2875.4751	7.8355603	
220	2.266142	2688.4064	2915.0206	7.9174209	
240	2.3595547	2718.7094	2954.6648	7.9962225	
260	2.4527887	2749.1693	2994.4481	8.0722765	
280	2.5458835	2779.8112	3034.3996	8.1458389	
300	2.6388678	2810.6536	3074.5404	8.2171244	
320	2.7317629	2841.7103	3114.8866	8.2863168	
340	2.8245848	2872.9921	3155.4506	8.3535755	
360	2.9173458	2904.5076	3196.2422	8.4190405	
380	3.0100556	2936.2641	3237.2696	8.4828361	
400	3.1027219	2968.2674	3278.5396	8.5450731	
420	3.195351	3000.5229	3320.058	8.6058515	
440	3.2879477	3033.0351	3361.8299	8.6652614	
460	3.3805164	3065.808	3403.8597	8.7233847	
480	3.4730606	3098.8453	3446.1514	8.7802961	
500	3.5655832	3132.1502	3488.7086	8.8360636	
520	3.6580867	3165.7256	3531.5343	8.8907498	
540	3.7505734	3199.5739	3574.6313	8.9444118	
560	3.8430449	3233.6974	3618.0018	8.9971023	
580	3.935503	3268.0977	3661.648	9.0488697	
600	4.0279491	3302.7766	3705.5715	9.0997587	
620	4.1203844	3337.7351	3749.7735	9.1498107	
640	4.21281	3372.9741	3794.255	9.1990637	
660	4.3052268	3408.4942	3839.0168	9.2475531	
680	4.3976356	3444.2958	3884.0593	9.2953119	
700	4.4900374	3480.379	3929.3827	9.3423705	
720	4.5824326	3516.7438	3974.987	9.3887576	
740	4.6748219	3553.39	4020.8722	9.4344997	
760	4.7672059	3590.3175	4067.0381	9.479622	
780	4.859585	3627.5261	4113.4846	9.5241482	
==========					
P=150.0	kPa				
Tsat=111.35004948446073degree	C				
T degree C	v m3/kg	u kJ/kg	h kJ/kg	s kJ/KgK	
111.35	1.1593574	2519.2096	2693.1133	7.2229418	
140	1.2533096	2564.5709	2752.5674	7.3721155	
160	1.3176295	2595.4362	2793.0807	7.4678802	
180	1.3813061	2626.011	2833.207	7.5584458	
200	1.4445311	2656.4638	2873.1435	7.6446882	
220	1.50743	2686.9052	2913.0197	7.7272339	
240	1.5700873	2717.4098	2952.9229	7.8065507	
260	1.6325614	2748.0292	2992.9134	7.8830011	
280	1.6948934	2778.7998	3033.0338	7.9568747	
300	1.7571131	2809.7478	3073.3147	8.0284092	
320	1.8192423	2840.8926	3113.7789	8.0978041	
340	1.8812976	2872.249	3154.4436	8.1652299	
360	1.9432914	2903.8286	3195.3223	8.2308345	
380	2.0052338	2935.6405	3236.4256	8.2947481	
400	2.0671323	2967.6925	3277.7623	8.3570859	
420	2.1289934	2999.9909	3319.3399	8.417951	
440	2.1908221	3032.5413	3361.1646	8.477436	

460	2.2526227	3065.3484	3403.2418	8.5356249	
480	2.3143987	3098.4165	3445.5763	8.5925938	
500	2.3761531	3131.7492	3488.1721	8.6484121	
520	2.4378883	3165.3497	3531.033	8.7031431	
540	2.4996066	3199.221	3574.162	8.756845	
560	2.5613099	3233.3653	3617.5618	8.809571	
580	2.6229997	3267.7849	3661.2348	8.8613702	
600	2.6846774	3302.4812	3705.1828	8.9122878	
620	2.7463443	3337.4558	3749.4075	8.9623653	
640	2.8080015	3372.7096	3793.9098	9.0116414	
660	2.86965	3408.2433	3838.6908	9.0601516	
680	2.9312905	3444.0575	3883.7511	9.1079292	
700	2.9929238	3480.1523	3929.0909	9.1550049	
720	3.0545507	3516.5278	3974.7104	9.2014074	
740	3.1161716	3553.1839	4020.6096	9.2471636	
760	3.1777872	3590.1205	4066.7886	9.2922987	
780	3.2393979	3627.3376	4113.2473	9.3368365	
800	3.3010042	3664.8351	4159.9857	9.3807996	
==========					
P=300.0	kPa				
Tsat=133.52535794654545degree	C				
T degree C	v m3/kg	u kJ/kg	h kJ/kg	s kJ/KgK	
133.5254	0.6057855	2543.156	2724.8917	6.9915659	
160	0.650828	2587.3472	2782.5956	7.1290736	
180	0.6838917	2619.4513	2824.6188	7.2239257	
200	0.7164446	2651.0187	2865.952	7.3131873	
220	0.7486362	2682.2947	2906.8855	7.3979236	
240	0.7805645	2713.4391	2947.6085	7.4788711	
260	0.812296	2744.5598	2988.2486	7.5565643	
280	0.8438767	2775.7315	3028.8945	7.6314061	
300	0.8753394	2807.0064	3069.6083	7.7037097	
320	0.906708	2838.4225	3110.435	7.7737266	
340	0.9380003	2870.0076	3151.4077	7.8416633	
360	0.9692295	2901.7825	3192.5513	7.9076935	
380	1.0004061	2933.7633	3233.8851	7.9719656	
400	1.0315382	2965.9628	3275.4242	8.0346086	
420	1.0626323	2998.3912	3317.1809	8.0957359	
440	1.0936937	3031.057	3359.1652	8.155448	
460	1.1247267	3063.9673	3401.3854	8.2138347	
480	1.155735	3097.1281	3443.8486	8.2709769	
500	1.1867214	3130.5445	3486.5609	8.3269479	
520	1.2176886	3164.221	3529.5275	8.381814	
540	1.2486389	3198.1612	3572.7529	8.4356358	
560	1.279574	3232.3685	3616.2407	8.4884687	
580	1.3104956	3266.8455	3659.9942	8.5403635	
600	1.3414051	3301.5946	3704.0161	8.5913666	
620	1.3723038	3336.6175	3748.3087	8.6415211	
640	1.4031927	3371.9158	3792.8736	8.6908665	
660	1.4340728	3407.4905	3837.7123	8.7394392	
680	1.4649451	3443.3423	3882.8258	8.7872733	
700	1.4958101	3479.4719	3928.2149	8.8344002	
720	1.5266686	3515.8795	3973.8801	8.8808491	
740	1.5575212	3552.5653	4019.8217	8.9266475	
760	1.5883685	3589.5294	4066.0399	8.971821	
780	1.6192108	3626.7718	4112.535	9.0163938	
800	1.6500487	3664.2928	4159.3074	9.0603888	
820	1.6808892	3702.1243	4206.3911	9.1038557	
==========					
P=500.0	kPa				
Tsat=151.83624387684512degree	C				
T degree C	v m3/kg	u kJ/kg	h kJ/kg	s kJ/KgK	
151.8362	0.3748044	2560.7054	2748.1076	6.8205808	
180	0.4046552	2610.1224	2812.45	6.9672413	
200	0.4250337	2643.3794	2855.8962	7.0610711	
220	0.4450013	2675.8944	2898.3951	7.1490511	
240	0.464676	2707.9739	2940.3119	7.2323738	
260	0.4841352	2739.8172	2981.8848	7.3118515	
280	0.5034318	2771.5596	3023.2755	7.3880656	
300	0.5226028	2803.2948	3064.5962	7.4614478	
320	0.5416747	2835.0892	3105.9265	7.5323287	
340	0.5606669	2866.9903	3147.3238	7.6009697	
360	0.5795939	2899.0334	3188.8304	7.6675826	

380	0.5984669	2931.2449	3230.4783	7.7323433	
400	0.6172943	2963.6449	3272.292	7.7954007	
420	0.636083	2996.2495	3314.2909	7.8568827	
440	0.6548384	3029.0712	3356.4904	7.916901	
460	0.6735652	3062.1206	3398.9032	7.975554	
480	0.6922669	3095.4061	3441.5395	8.0329292	
500	0.7109466	3128.9349	3484.4082	8.0891051	
520	0.7296071	3162.7131	3527.5166	8.1441524	
540	0.7482503	3196.7458	3570.871	8.1981349	
560	0.7668784	3231.0374	3614.4766	8.2511109	
580	0.7854929	3265.5914	3658.3379	8.3031335	
600	0.8040953	3300.411	3702.4586	8.3542512	
620	0.8226869	3335.4986	3746.842	8.4045085	
640	0.8412686	3370.8563	3791.4906	8.4539465	
660	0.8598415	3406.4856	3836.4064	8.5026028	
680	0.8784065	3442.3878	3881.5911	8.5505124	
700	0.8969643	3478.5639	3927.046	8.5977076	
720	0.9155156	3515.0143	3972.7721	8.6442186	
740	0.9340609	3551.7398	4018.7702	8.6900733	
760	0.9526009	3588.7404	4065.0409	8.735298	
780	0.9711359	3626.0167	4111.5847	8.7799175	
800	0.9896666	3663.569	4158.4023	8.8239551	
820	1.0081996	3701.4229	4205.5227	8.8674548	
840	1.026723	3739.5373	4252.8988	8.9104017	
==========					
P=700.0	kPa				
Tsat=164.95275255074188degree	C				
T degree C	v m3/kg	u kJ/kg	h kJ/kg	s kJ/KgK	
164.9528	0.272764	2571.8143	2762.7491	6.7069837	
190	0.292441	2617.8935	2822.6022	6.8399011	
210	0.3074312	2652.3714	2867.5732	6.9349733	
230	0.3220347	2685.8309	2911.2552	7.0235698	
250	0.3363636	2718.6692	2954.1237	7.1071245	
270	0.3504906	2751.14	2996.4835	7.186588	
290	0.3644655	2783.4148	3038.5407	7.2626296	
310	0.3783227	2815.6122	3080.4381	7.3357378	
330	0.3920872	2847.8158	3122.2769	7.4062812	
350	0.405777	2880.0858	3164.1297	7.4745457	
370	0.4194056	2912.4662	3206.0501	7.5407601	
390	0.4329834	2944.9899	3248.0783	7.6051115	
410	0.4465184	2977.682	3290.2449	7.6677563	
430	0.4600171	3010.562	3332.5739	7.7288277	
450	0.4734844	3043.6455	3375.0846	7.7884406	
470	0.4869246	3076.9451	3417.7923	7.8466959	
490	0.5003411	3110.4709	3460.7096	7.9036825	
510	0.5137369	3144.2314	3503.8473	7.9594796	
530	0.5271144	3178.2339	3547.214	8.0141585	
550	0.5404757	3212.4843	3590.8172	8.0677832	
570	0.5538225	3246.9876	3634.6633	8.120412	
590	0.5671565	3281.748	3678.7575	8.1720977	
610	0.5804789	3316.769	3723.1042	8.2228888	
630	0.593791	3352.0536	3767.7073	8.2728294	
650	0.6070938	3387.604	3812.5697	8.3219601	
670	0.6203882	3423.4223	3857.694	8.3703185	
690	0.6336751	3459.5098	3903.0823	8.417939	
710	0.646955	3495.8677	3948.7362	8.4648537	
730	0.6602288	3532.4969	3994.657	8.5110923	
750	0.6734969	3569.398	4040.8458	8.5566826	
770	0.6867598	3606.5717	4087.3036	8.6016508	
790	0.7000181	3644.0186	4134.0313	8.6460211	
810	0.7132783	3681.7491	4181.0439	8.6898245	
830	0.726529	3719.7586	4228.3289	8.7330811	
850	0.7397761	3758.031	4275.8743	8.7757944	
==========					
P=1000.0	kPa				
Tsat=179.88563239146663degree	C				
T degree C	v m3/kg	u kJ/kg	h kJ/kg	s kJ/KgK	
179.8856	0.1943489	2582.7707	2777.1195	6.584979	
200	0.2060036	2622.2639	2828.2675	6.6954882	
220	0.2169658	2658.5844	2875.5502	6.7933822	
240	0.2275508	2693.429	2920.9798	6.8836942	
260	0.2378711	2727.3579	2965.229	6.9682924	

280	0.2479979	2760.713	3008.711	7.0483601	
300	0.2579792	2793.724	3051.7032	7.1247125	
320	0.2678484	2826.5488	3094.3972	7.1979335	
340	0.2776292	2859.2986	3136.9279	7.2684547	
360	0.2873392	2892.0527	3179.3919	7.3366049	
380	0.2969911	2924.8691	3221.8602	7.4026417	
400	0.3065949	2957.7905	3264.3855	7.4667725	
420	0.3161582	2990.8497	3307.0079	7.5291675	
440	0.3256869	3024.0717	3349.7586	7.58997	
460	0.335186	3057.4763	3392.6623	7.6493022	
480	0.3446593	3091.0793	3435.7386	7.7072696	
500	0.3541103	3124.8934	3479.0037	7.763965	
520	0.3635415	3158.9293	3522.4708	7.8194704	
540	0.3729553	3193.1956	3566.151	7.8738587	
560	0.3823538	3227.6997	3610.0535	7.9271956	
580	0.3917385	3262.4477	3654.1862	7.9795401	
600	0.401111	3297.4447	3698.5556	8.030946	
620	0.4104725	3332.6948	3743.1673	8.0814619	
640	0.4198241	3368.2018	3788.0259	8.1311325	
660	0.4291669	3403.9684	3833.1353	8.1799985	
680	0.4385017	3439.9971	3878.4987	8.2280976	
700	0.4478292	3476.2896	3924.1188	8.2754644	
720	0.4571502	3512.8476	3969.9978	8.3221309	
740	0.4664653	3549.6723	4016.1376	8.3681268	
760	0.4757749	3586.7648	4062.5397	8.4134801	
780	0.4850796	3624.126	4109.2057	8.4582166	
800	0.4943799	3661.7569	4156.1368	8.502361	
820	0.5036824	3699.6675	4203.3499	8.5459434	
840	0.5129754	3737.8482	4250.8236	8.5889789	
860	0.522265	3776.2901	4298.5551	8.6314776	
==========					
P=1500.0	kPa				
Tsat=198.29524288241413degree	C				
T degree C	v m3/kg	u kJ/kg	h kJ/kg	s kJ/KgK	
198.2952	0.1317023	2593.457	2791.0105	6.4430547	
220	0.1406305	2639.2437	2850.1894	6.5658407	
240	0.148295	2677.558	2900.0006	6.6648718	
260	0.1556369	2713.9969	2947.4523	6.7555981	
280	0.1627515	2749.2434	2993.3707	6.8401557	
300	0.1696993	2783.7171	3038.266	6.9198901	
320	0.1765211	2817.6995	3082.4811	6.9957214	
340	0.1832454	2851.3852	3126.2534	7.0683024	
360	0.1898928	2884.9109	3169.75	7.1381105	
380	0.196478	2918.3745	3213.0914	7.2055057	
400	0.2030122	2951.8474	3256.3657	7.2707663	
420	0.2095039	2985.3827	3299.6386	7.3341139	
440	0.2159598	3019.0205	3342.9601	7.3957285	
460	0.222385	3052.7917	3386.3691	7.4557597	
480	0.2287838	3086.7205	3429.8962	7.5143338	
500	0.2351597	3120.8262	3473.5657	7.5715594	
520	0.2415154	3155.1245	3517.3977	7.6275308	
540	0.2478535	3189.6281	3561.4085	7.6823308	
560	0.254176	3224.3476	3605.6117	7.7360331	
580	0.2604846	3259.2917	3650.0187	7.788703	
600	0.2667809	3294.4678	3694.6391	7.8403997	
620	0.273066	3329.8819	3739.481	7.8911764	
640	0.2793413	3365.5392	3784.5512	7.9410812	
660	0.2856076	3401.444	3829.8554	7.9901584	
680	0.2918659	3437.5997	3875.3985	8.0384481	
700	0.2981169	3474.0094	3921.1847	8.0859873	
720	0.3043613	3510.6754	3967.2173	8.13281	
740	0.3105997	3547.5998	4013.4994	8.1789479	
760	0.3168327	3584.7845	4060.0335	8.2244302	
780	0.3230608	3622.2309	4106.822	8.2692843	
800	0.3292843	3659.9406	4153.8671	8.3135358	
820	0.33551	3697.9093	4201.1743	8.3572023	
840	0.3417263	3736.1568	4248.7462	8.4003268	
860	0.3479391	3774.6617	4296.5704	8.4429081	
880	0.3541487	3813.4252	4344.6483	8.4849662	
==========					
P=2000.0	kPa				
Tsat=212.38453531849052degree	C				

T degree C	v m3/kg	u kJ/kg	h kJ/kg	s kJ/KgK
212.3845	0.0995805	2599.2231	2798.3841	6.3391644
240	0.1084883	2660.2318	2877.2084	6.4971911
260	0.1144003	2699.6741	2928.4746	6.5952172
280	0.1200458	2737.114	2977.2056	6.684958
300	0.1255012	2773.2495	3024.2519	6.7685153
320	0.1308158	2808.5242	3070.1557	6.8472447
340	0.1360232	2843.238	3115.2845	6.9220762
360	0.1411472	2877.5992	3159.8935	6.9936705
380	0.1462046	2911.7549	3204.1641	7.0625111
400	0.151208	2945.811	3248.2271	7.1289617
420	0.1561669	2979.8451	3292.179	7.1933036
440	0.1610885	3013.9151	3336.0921	7.2557599
460	0.1659784	3048.0649	3380.0217	7.3165111
480	0.1708411	3082.3284	3424.0107	7.3757072
500	0.1756804	3116.7324	3468.0932	7.4334741
520	0.1804992	3151.2982	3512.2965	7.4899198
540	0.1853	3186.0429	3556.6429	7.5451378
560	0.190085	3220.9807	3601.1507	7.5992101
580	0.1948559	3256.1232	3645.8351	7.6522091
600	0.1996144	3291.4802	3690.7089	7.7041994
620	0.2043616	3327.0597	3735.7829	7.755239
640	0.2090989	3362.8685	3781.0663	7.8053799
660	0.2138272	3398.9123	3826.5666	7.8546695
680	0.2185473	3435.1958	3872.2905	7.9031509
700	0.2232602	3471.7232	3918.2436	7.9508634
720	0.2279664	3508.4977	3964.4306	7.9978432
740	0.2326667	3545.5223	4010.8556	8.0441236
760	0.2373614	3582.7994	4057.5223	8.0897355
780	0.2420513	3620.3314	4104.4339	8.1347076
800	0.2467366	3658.1203	4151.5935	8.1790669
820	0.2514238	3696.1484	4198.996	8.2228187
840	0.2561017	3734.463	4246.6665	8.2660326
860	0.2607762	3773.0315	4294.5839	8.3086968
880	0.2654475	3811.8549	4342.7499	8.350832
900	0.2701158	3850.9342	4391.1657	8.3924575
==========				
P=3000.0	kPa			
Tsat=233.85844500625223degree	C			
T degree C	v m3/kg	u kJ/kg	h kJ/kg	s kJ/KgK
233.8584	0.0666641	2603.2725	2803.2647	6.1857878
260	0.0728884	2667.753	2886.4182	6.3458487
280	0.077156	2710.6912	2942.1593	6.4485109
300	0.0811753	2750.8234	2994.3493	6.5412107
320	0.0850197	2789.1241	3044.1832	6.6266849
340	0.0887354	2826.1935	3092.3996	6.7066393
360	0.0923533	2862.4323	3139.4921	6.7822215
380	0.0958952	2898.1165	3185.8021	6.8542349
400	0.0993766	2933.4411	3231.571	6.9232593
420	0.102809	2968.5458	3276.9729	6.9897247
440	0.106201	3003.5329	3322.1358	7.0539591
460	0.109559	3038.4784	3367.1555	7.1162183
480	0.1128883	3073.4398	3412.1048	7.176707
500	0.116193	3108.4614	3457.0405	7.2355922
520	0.1194764	3143.578	3502.0072	7.2930128
540	0.1227412	3178.817	3547.0406	7.3490865
560	0.1259896	3214.2008	3592.1697	7.4039138
580	0.1292237	3249.7474	3637.4185	7.4575824
600	0.132445	3285.4718	3682.8068	7.5101689
620	0.1356549	3321.3865	3728.3512	7.5617411
640	0.1388546	3357.5019	3774.0657	7.6123596
660	0.1420452	3393.8266	3819.9623	7.6620785
680	0.1452276	3430.3682	3866.051	7.7109468
700	0.1484026	3467.1328	3912.3407	7.7590088
720	0.1515709	3504.1259	3958.8387	7.8063051
740	0.1547332	3541.3521	4005.5516	7.8528725
760	0.1578899	3578.8155	4052.4852	7.8987453
780	0.1610417	3616.5196	4099.6446	7.9439549
800	0.1641889	3654.4678	4147.0344	7.9885308
820	0.1673376	3692.6184	4194.6313	8.0324569
840	0.1704773	3731.0687	4242.5006	8.0758511
860	0.1736135	3769.7651	4290.6056	8.1186823

880	0.1767465	3808.7095	4338.9489	8.1609726	
900	0.1798764	3847.9031	4387.5324	8.2027422	
920	0.1830036	3887.3466	4436.3574	8.2440099	
==========					
P=4000.0	kPa				
Tsat=250.35751912406556degree	C				
T degree C	v m3/kg	u kJ/kg	h kJ/kg	s kJ/KgK	
250.3575	0.0497766	2601.7909	2800.8973	6.0697092	
280	0.0554948	2680.9051	2902.8845	6.2594346	
300	0.058868	2726.1796	2961.6515	6.3638265	
320	0.0620211	2768.192	3016.2763	6.4575239	
340	0.0650195	2808.0637	3068.1417	6.543533	
360	0.067904	2846.4826	3118.0985	6.6237149	
380	0.0707017	2883.9054	3166.7123	6.6993125	
400	0.0734318	2920.6463	3214.3735	6.771192	
420	0.0761079	2956.927	3261.3587	6.8399762	
440	0.0787401	2992.9072	3307.8677	6.9061258	
460	0.081336	3028.7038	3354.0478	6.9699904	
480	0.0839015	3064.4038	3400.0096	7.0318418	
500	0.086441	3100.0734	3445.8374	7.0918964	
520	0.0889583	3135.7636	3491.5969	7.1503297	
540	0.0914564	3171.5142	3537.3398	7.207287	
560	0.0939376	3207.3572	3583.1077	7.2628906	
580	0.0964041	3243.3182	3628.9345	7.3172449	
600	0.0988574	3279.4181	3674.8479	7.3704399	
620	0.1012992	3315.6743	3720.8712	7.4225545	
640	0.1037306	3352.1013	3767.0238	7.4736581	
660	0.1061528	3388.711	3813.3222	7.5238124	
680	0.1085666	3425.5138	3859.7803	7.5730724	
700	0.110973	3462.5185	3906.4104	7.621488	
720	0.1133726	3499.7324	3953.2227	7.6691039	
740	0.115766	3537.1621	4000.2262	7.7159611	
760	0.1181539	3574.8133	4047.4291	7.762097	
780	0.1205368	3612.691	4094.8382	7.8075462	
800	0.1229151	3650.7998	4142.4601	7.8523404	
820	0.1252946	3689.0779	4190.2564	7.8964456	
840	0.1276653	3727.6654	4238.3265	7.9400218	
860	0.1300324	3766.4911	4286.6207	7.9830215	
880	0.1323962	3805.5575	4335.1425	8.0254681	
900	0.1347571	3844.8665	4383.8949	8.0673829	
920	0.1371151	3884.4192	4432.8797	8.1087856	
940	0.1394705	3924.2161	4482.0983	8.1496944	
==========					
P=5000.0	kPa				
Tsat=263.9428711863302degree	C				
T degree C	v m3/kg	u kJ/kg	h kJ/kg	s kJ/KgK	
263.9429	0.0394463	2596.9957	2794.2271	5.9737041	
290	0.0438562	2673.7214	2893.0023	6.1534813	
310	0.0467667	2722.7458	2956.5795	6.2644532	
330	0.0494477	2767.4723	3014.7108	6.3624871	
350	0.0519714	2809.437	3069.2942	6.4515291	
370	0.0543809	2849.5349	3121.4395	6.533903	
390	0.0567042	2888.3437	3171.8648	6.611118	
410	0.0589607	2926.2556	3221.0591	6.6842078	
430	0.0611641	2963.5453	3269.3658	6.7539068	
450	0.0633245	3000.4094	3317.032	6.8207519	
470	0.0654495	3036.9915	3364.239	6.8851464	
490	0.0675449	3073.398	3411.1225	6.9474007	
510	0.0696152	3109.7097	3457.7855	7.007759	
530	0.0716639	3145.9883	3504.3079	7.0664176	
550	0.0736941	3182.2823	3550.7526	7.1235376	
570	0.075708	3218.6298	3597.1698	7.1792532	
590	0.0777077	3255.0619	3643.6002	7.2336779	
610	0.0796948	3291.6036	3690.0773	7.2869094	
630	0.0816707	3328.2756	3736.629	7.3390321	
650	0.0836367	3365.0952	3783.2785	7.3901203	
670	0.0855937	3402.077	3830.0456	7.4402395	
690	0.0875428	3439.2333	3876.9472	7.489448	
710	0.0894847	3476.5745	3923.9978	7.5377981	
730	0.09142	3514.1097	3971.2099	7.5853372	
750	0.0933495	3551.8467	4018.5944	7.6321079	
770	0.0952737	3589.7927	4066.1611	7.6781495	

790	0.097193	3627.9537	4113.9186	7.7234979	
810	0.0991129	3666.2489	4161.8132	7.7681018	
830	0.101024	3704.8617	4209.9818	7.8121667	
850	0.1029315	3743.7026	4258.36	7.8556283	
870	0.1048355	3782.7753	4306.9529	7.8985122	
890	0.1067364	3822.0825	4355.7646	7.9408419	
910	0.1086344	3861.6263	4404.7982	7.9826392	
930	0.1105296	3901.408	4454.0561	8.0239238	
950	0.1124223	3941.4285	4503.5401	8.0647142	
==========					
P=6000.0	kPa				
Tsat=275.58641075604896degree	C				
T degree C	v m3/kg	u kJ/kg	h kJ/kg	s kJ/KgK	
275.5864	0.0324487	2589.8697	2784.5617	5.890068	
300	0.0361911	2668.3437	2885.4905	6.070211	
320	0.0387819	2720.8541	2953.5456	6.1869669	
340	0.0411378	2768.0653	3014.8922	6.2887105	
360	0.0433364	2811.9374	3071.9555	6.3803057	
380	0.045422	2853.5672	3126.0989	6.464507	
400	0.047423	2893.6453	3178.183	6.5430599	
420	0.0493586	2932.6344	3228.786	6.6171426	
440	0.0512425	2970.8561	3278.3113	6.687584	
460	0.0530847	3008.54	3327.0481	6.7549856	
480	0.0548925	3045.8531	3375.2082	6.8197963	
500	0.0566717	3082.9193	3422.9493	6.8823589	
520	0.0584266	3119.8322	3470.3915	6.9429416	
540	0.0601607	3156.6632	3517.6274	7.0017583	
560	0.0618769	3193.4683	3564.7298	7.0589835	
580	0.0635775	3230.2918	3611.7567	7.1147615	
600	0.0652644	3267.169	3658.7552	7.1692139	
620	0.0669391	3304.1289	3705.7638	7.2224445	
640	0.0686032	3341.1955	3752.8146	7.2745429	
660	0.0702577	3378.3884	3799.9345	7.3255872	
680	0.0719036	3415.7245	3847.1461	7.3756463	
700	0.0735419	3453.2178	3894.469	7.4247813	
720	0.0751732	3490.8807	3941.9199	7.4730468	
740	0.0767983	3528.7235	3989.5133	7.5204921	
760	0.0784177	3566.7556	4037.2619	7.5671615	
780	0.080032	3604.9849	4085.177	7.6130958	
800	0.0816416	3643.419	4133.2689	7.6583321	
820	0.0832519	3681.9661	4181.4774	7.7028088	
840	0.0848536	3720.8329	4229.9547	7.7467541	
860	0.0864518	3759.9213	4278.632	7.7900951	
880	0.0880467	3799.2352	4327.5151	7.8328576	
900	0.0896384	3838.7778	4376.6085	7.8750656	
920	0.0912274	3878.5512	4425.9156	7.9167408	
940	0.0928137	3918.5571	4475.4393	7.9579032	
960	0.0943976	3958.7964	4525.1817	7.9985713	
==========					
P=8000.0	kPa				
Tsat=295.0091212293113degree	C				
T degree C	v m3/kg	u kJ/kg	h kJ/kg	s kJ/KgK	
295.0091	0.0235275	2570.3908	2758.6111	5.7448488	
320	0.0268425	2663.6098	2878.35	5.9513568	
340	0.0289946	2721.9104	2953.8675	6.0766255	
360	0.0309152	2773.2505	3020.5718	6.1837075	
380	0.0326848	2820.3113	3081.7895	6.278917	
400	0.0343477	2864.5288	3139.3106	6.3656745	
420	0.0359315	2906.7763	3194.228	6.4460766	
440	0.0374543	2947.6276	3247.2618	6.5215101	
460	0.038929	2987.4777	3298.9096	6.5929391	
480	0.0403649	3026.6072	3349.5267	6.6610574	
500	0.0417691	3065.2199	3399.3726	6.7263791	
520	0.0431467	3103.4672	3448.6408	6.7892941	
540	0.0445019	3141.4628	3497.478	6.8501053	
560	0.0458379	3179.2942	3545.9977	6.9090529	
580	0.0471574	3217.0295	3594.289	6.9663308	
600	0.0484625	3254.7224	3642.4227	7.0220987	
620	0.049755	3292.4163	3690.4563	7.0764902	
640	0.0510363	3330.1464	3738.437	7.1296183	
660	0.0523078	3367.9418	3786.4038	7.1815803	
680	0.0535704	3405.8266	3834.3896	7.2324604	

700	0.0548251	3443.8215	3882.4224	7.2823326
720	0.0560728	3481.9438	3930.526	7.3312622
740	0.0573141	3520.2086	3978.721	7.3793073
760	0.0585496	3558.6289	4027.0255	7.4265202
780	0.0597799	3597.2165	4075.4554	7.4729479
800	0.0610054	3635.9816	4124.0248	7.5186335
820	0.0622309	3674.815	4172.6625	7.5634982
840	0.0634484	3713.9672	4221.5541	7.6078192
860	0.0646622	3753.3235	4270.6208	7.6515068
880	0.0658726	3792.8894	4319.8702	7.6945898
900	0.0670799	3832.6693	4369.3088	7.7370947
920	0.0682844	3872.6666	4418.9418	7.7790453
940	0.0694862	3912.8841	4468.7736	7.8204638
960	0.0706855	3953.3237	4518.8076	7.8613704
980	0.0718825	3993.9865	4569.0464	7.9017836
==========				
P=10000.0	kPa			
Tsat=310.99948799852825degree	C			
T degree C	v m3/kg	u kJ/kg	h kJ/kg	s kJ/KgK
310.9995	0.0180336	2545.1368	2725.4726	5.6158899
340	0.0214897	2667.1595	2882.0569	5.8780022
360	0.0233274	2729.3404	2962.614	6.0073433
380	0.0249522	2783.5874	3033.1096	6.1169928
400	0.0264393	2832.9822	3097.3753	6.2139289
420	0.0278294	2879.1566	3157.4506	6.3018862
440	0.0291475	2923.0917	3214.5666	6.383129
460	0.0304102	2965.4273	3269.5295	6.4591445
480	0.0316292	3006.6027	3322.8946	6.5309624
500	0.0328129	3046.9298	3375.0584	6.5993225
520	0.0339675	3086.6358	3426.3109	6.6647722
540	0.0350979	3125.8894	3476.8686	6.7277262
560	0.0362078	3164.8178	3526.8962	6.7885063
580	0.0373002	3203.5192	3576.5215	6.8473668
600	0.0383775	3242.0699	3625.8446	6.9045131
620	0.0394415	3280.5308	3674.9459	6.9601137
640	0.040494	3318.9506	3723.8902	7.0143092
660	0.0415362	3357.3693	3772.7312	7.0672183
680	0.0425693	3395.8199	3821.5132	7.1189428
700	0.0435944	3434.33	3870.2739	7.1695708
720	0.0446122	3472.9232	3919.045	7.2191795
740	0.0456235	3511.6194	3967.854	7.2678368
760	0.0466289	3550.4362	4016.7248	7.3156032
780	0.0476289	3589.3888	4065.6782	7.3625329
800	0.0486242	3628.4909	4114.7328	7.4086749
820	0.0496189	3667.6266	4163.8151	7.4539428
840	0.0506058	3707.0701	4213.128	7.4986457
860	0.051589	3746.6994	4262.5897	7.542685
880	0.0525689	3786.5215	4312.2102	7.5860927
900	0.0535456	3826.5423	4361.998	7.6288979
920	0.0545194	3866.7666	4411.9603	7.6711268
940	0.0554905	3907.1983	4462.103	7.7128038
960	0.0564591	3947.8403	4512.4309	7.7539506
980	0.0574253	3988.6948	4562.9481	7.7945878
1000	0.0583894	4029.7634	4613.6577	7.8347337
==========				
P=12000.0	kPa			
Tsat=324.67830430181743degree	C			
T degree C	v m3/kg	u kJ/kg	h kJ/kg	s kJ/KgK
324.6783	0.0142689	2514.3553	2685.5827	5.4941163
350	0.0172227	2641.3398	2848.0126	5.7607107
370	0.0189442	2711.8183	2939.1489	5.9047403
390	0.0204258	2771.3801	3016.4901	6.0232019
410	0.0217597	2824.5864	3085.7023	6.1260516
430	0.0229924	2873.6828	3149.5916	6.2182455
450	0.0241515	2919.957	3209.7747	6.3026517
470	0.0252545	2964.2229	3267.2775	6.3810962
490	0.0263138	3007.0284	3322.7937	6.4548176
510	0.0273378	3048.7581	3376.8121	6.524693
530	0.0283331	3089.6897	3429.6871	6.5913637
550	0.0293045	3130.0281	3481.682	6.6553115
570	0.0302557	3169.9276	3532.9961	6.7169064
590	0.0311897	3209.5061	3583.7831	6.7764389

610	0.032109	3248.8557	3634.1632	6.8341415	
630	0.0330153	3288.0489	3684.2324	6.8902035	
650	0.0339104	3327.144	3734.0682	6.9447818	
670	0.0347955	3366.1885	3783.734	6.998008	
690	0.0356717	3405.2218	3833.2827	7.0499942	
710	0.0365402	3444.2765	3882.7586	7.1008371	
730	0.0374016	3483.3805	3932.1995	7.1506206	
750	0.0382567	3522.5577	3981.6376	7.1994187	
770	0.0391061	3561.8287	4031.1013	7.2472967	
790	0.0399503	3601.2117	4080.6152	7.294313	
810	0.0407932	3640.5914	4130.1092	7.3403812	
830	0.0416285	3680.2529	4179.7948	7.385834	
850	0.0424599	3720.0756	4229.5945	7.4305727	
870	0.0432877	3760.0693	4279.5221	7.4746346	
890	0.0441123	3800.2422	4329.5894	7.5180533	
910	0.0449337	3840.6013	4379.8061	7.5608592	
930	0.0457524	3881.152	4430.1806	7.6030798	
950	0.0465684	3921.899	4480.72	7.6447402	
970	0.047382	3962.8459	4531.4301	7.6858633	
990	0.0481934	4003.9955	4582.3158	7.7264702	
1010	0.0490026	4045.3498	4633.3809	7.7665801	
==========					
P=14000.0	kPa				
Tsat=336.6693686281236degree	C				
T degree C	v m3/kg	u kJ/kg	h kJ/kg	s kJ/KgK	
336.6694	0.0114889	2477.2494	2638.0934	5.3730467	
360	0.0142288	2617.188	2816.3911	5.6605416	
380	0.0158666	2696.1227	2918.2555	5.8190372	
400	0.017241	2760.8526	3002.2267	5.9457199	
420	0.0184603	2817.6924	3076.1371	6.0539463	
440	0.0195762	2869.5451	3143.6113	6.1499302	
460	0.0206177	2918.014	3206.662	6.2371366	
480	0.0216034	2964.0874	3266.5353	6.3177165	
500	0.0225457	3008.4193	3324.0588	6.393103	
520	0.0234533	3051.4635	3379.8091	6.4642984	
540	0.0243326	3093.5451	3434.2014	6.5320286	
560	0.0251885	3134.903	3487.5421	6.5968348	
580	0.0260247	3175.7162	3540.0624	6.65913	
600	0.0268442	3216.1214	3591.94	6.7192366	
620	0.0276492	3256.2246	3643.3138	6.7774111	
640	0.0284417	3296.1096	3694.294	6.8338612	
660	0.0292233	3335.8435	3744.9697	6.8887583	
680	0.0299952	3375.4808	3795.4139	6.9422454	
700	0.0307586	3415.0666	3845.6869	6.994444	
720	0.0315143	3454.6386	3895.8394	7.045458	
740	0.0322633	3494.2285	3945.9148	7.0953779	
760	0.0330061	3533.8639	3995.9499	7.1442825	
780	0.0337435	3573.5685	4045.9772	7.1922419	
800	0.0344758	3613.3633	4096.0249	7.2393181	
820	0.0352065	3653.1454	4146.0366	7.2854319	
840	0.0359302	3693.189	4196.2114	7.3309163	
860	0.03665	3733.3785	4246.4791	7.3756733	
880	0.0373664	3773.7251	4296.855	7.4197419	
900	0.0380796	3814.238	4347.3524	7.4631572	
920	0.0387898	3854.925	4397.9824	7.5059506	
940	0.0394973	3895.7925	4448.7545	7.5481506	
960	0.0402022	3936.8456	4499.6767	7.5897834	
980	0.0409048	3978.0886	4550.7558	7.6308726	
1000	0.0416052	4019.5249	4601.9975	7.6714399	
1020	0.0423035	4061.157	4653.4063	7.7115051	
==========					
P=16000.0	kPa				
Tsat=347.35653444596915degree	C				
T degree C	v m3/kg	u kJ/kg	h kJ/kg	s kJ/KgK	
347.3565	0.0093081	2431.8743	2580.8044	5.246271	
370	0.0120464	2595.5582	2788.3005	5.5755018	
390	0.0136131	2682.676	2900.4853	5.7473966	
410	0.0148991	2752.2309	2990.6164	5.8813573	
430	0.0160263	2812.4306	3068.8514	5.9942656	
450	0.0170494	2866.8213	3139.6122	6.0935148	
470	0.0179988	2917.3087	3205.2895	6.1831159	
490	0.0188931	2965.0463	3267.3365	6.2655132	

510	0.0197449	3010.787	3326.7056	6.3423124	
530	0.0205628	3055.0483	3384.0534	6.4146247	
550	0.0213532	3098.1984	3439.8496	6.483249	
570	0.0221208	3140.5066	3494.4395	6.548777	
590	0.0228693	3182.1739	3548.0826	6.6116583	
610	0.0236015	3223.3538	3600.9779	6.6722423	
630	0.0243197	3264.1651	3653.2809	6.730806	
650	0.0250259	3304.7013	3705.115	6.7875731	
670	0.0257214	3345.0377	3756.5795	6.8427272	
690	0.0264075	3385.2352	3807.7558	6.8964214	
710	0.0270854	3425.3442	3858.7113	6.948785	
730	0.027756	3465.4067	3909.5027	6.9999287	
750	0.02842	3505.4585	3960.1782	7.0499483	
770	0.0290781	3545.5303	4010.7792	7.0989273	
790	0.0297308	3585.6487	4061.3418	7.1469396	
810	0.0303812	3625.7229	4111.8223	7.1939163	
830	0.0310247	3666.0178	4162.413	7.2401974	
850	0.0316642	3706.4301	4213.0569	7.2856945	
870	0.0323	3746.9742	4263.7737	7.3304529	
890	0.0329324	3787.6622	4314.5804	7.3745129	
910	0.0335617	3828.5042	4365.4915	7.4179107	
930	0.0341881	3869.5089	4416.5192	7.4606789	
950	0.0348119	3910.6833	4467.6743	7.5028468	
970	0.0354333	3952.0334	4518.9656	7.5444413	
990	0.0360523	3993.5639	4570.4009	7.5854868	
1010	0.0366692	4035.2787	4621.9863	7.6260054	
1030	0.0372842	4077.1809	4673.7274	7.6660177	
==========					
P=18000.0	kPa				
Tsat=356.99181334434775degree	C				
T degree C	v m3/kg	u kJ/kg	h kJ/kg	s kJ/KgK	
356.9918	0.0074987	2374.5539	2509.5299	5.1055329	
380	0.0104189	2577.3531	2764.893	5.5048279	
400	0.0119147	2671.8413	2886.3054	5.6880676	
420	0.013122	2745.6914	2981.8866	5.8280526	
440	0.0141706	2808.8928	3063.9631	5.9448221	
460	0.0151165	2865.5598	3137.6561	6.0467557	
480	0.0159901	2917.8651	3205.6873	6.1383197	
500	0.0168102	2967.1088	3269.6925	6.222204	
520	0.017589	3014.1319	3330.7335	6.3001581	
540	0.018335	3059.5074	3389.5369	6.3733829	
560	0.0190544	3103.6409	3446.6193	6.4427364	
580	0.0197517	3146.828	3502.3591	6.5088512	
600	0.0204306	3189.2889	3557.04	6.5722066	
620	0.0210938	3231.1912	3610.8796	6.6331739	
640	0.0217435	3272.6648	3664.0475	6.6920469	
660	0.0223815	3313.8121	3716.6784	6.7490624	
680	0.0230092	3354.7152	3768.8809	6.8044143	
700	0.0236279	3395.4409	3820.7439	6.8582641	
720	0.0242387	3436.0443	3872.3412	6.9107478	
740	0.0248424	3476.5716	3923.7346	6.9619819	
760	0.0254397	3517.0619	3974.9769	7.0120665	
780	0.0260314	3557.549	4026.1135	7.0610895	
800	0.0266179	3598.0625	4077.184	7.109128	
820	0.0272017	3638.5384	4128.1695	7.1561318	
840	0.0277792	3679.2034	4179.2294	7.2024187	
860	0.0283528	3719.9713	4230.3224	7.2479106	
880	0.0289229	3760.8576	4281.4697	7.2926541	
900	0.0294897	3801.8754	4332.6897	7.3366907	
920	0.0300535	3843.0357	4383.9982	7.3800576	
940	0.0306145	3884.3479	4435.4085	7.4227882	
960	0.0311729	3925.8198	4486.9323	7.4649129	
980	0.031729	3967.4578	4538.5794	7.5064589	
1000	0.0322828	4009.2673	4590.358	7.5474513	
1020	0.0328346	4051.2525	4642.2753	7.5879129	
1040	0.0333845	4093.417	4694.3373	7.6278644	
==========					
P=20000.0	kPa				
Tsat=365.7459115457051degree	C				
T degree C	v m3/kg	u kJ/kg	h kJ/kg	s kJ/KgK	
365.7459	0.0058583	2294.2225	2411.3881	4.9299021	
390	0.0091898	2563.3784	2747.1737	5.4481798	

410	0.0106082	2663.8863	2876.0512	5.639803	
430	0.0117416	2741.3502	2976.1812	5.7843362	
450	0.0127202	2807.1304	3061.5343	5.9040655	
470	0.0135992	2865.7804	3137.765	6.0080716	
490	0.0144085	2919.6865	3207.8566	6.1011567	
510	0.0151662	2970.2697	3273.5934	6.1861964	
530	0.0158842	3018.4437	3336.1271	6.26505	
550	0.0165707	3064.828	3396.2412	6.3389865	
570	0.0172316	3109.8588	3454.4906	6.4089084	
590	0.0178714	3153.8532	3511.2804	6.4754793	
610	0.0184934	3197.047	3566.9144	6.5392007	
630	0.0191003	3239.6197	3621.6251	6.6004609	
650	0.0196942	3281.7106	3675.5941	6.6595667	
670	0.0202768	3323.4294	3728.9658	6.7167651	
690	0.0208496	3364.8642	3781.8566	6.7722584	
710	0.0214137	3406.0869	3834.3619	6.8262149	
730	0.0219702	3447.1569	3886.561	6.8787763	
750	0.0225198	3488.1239	3938.5205	6.9300634	
770	0.0230634	3529.0303	3990.2973	6.9801807	
790	0.0236014	3569.9127	4041.9404	7.0292191	
810	0.0241362	3610.7246	4093.4478	7.077145	
830	0.0246646	3651.6756	4144.968	7.1242764	
850	0.025189	3692.6965	4196.4759	7.1705498	
870	0.0257096	3733.8068	4247.9981	7.2160191	
890	0.0262267	3775.023	4299.5576	7.2607321	
910	0.0267407	3816.3592	4351.174	7.3047311	
930	0.0272518	3857.8272	4402.864	7.3480543	
950	0.0277603	3899.437	4454.642	7.3907359	
970	0.0282662	3941.1969	4506.5206	7.4328066	
990	0.0287698	3983.114	4558.5102	7.4742945	
1010	0.0292713	4025.194	4610.6199	7.515225	
1030	0.0297708	4067.4417	4662.8574	7.5556212	
1050	0.0302684	4109.861	4715.2293	7.5955045	
==========					
P=24000.0	kPa				
Tsat=376.96799547221303degree	C				
T degree C	v m3/kg	u kJ/kg	h kJ/kg	s kJ/KgK	
376.968	0.0031056	2013.0158	2087.55	4.412	
400	0.0067312	2475.8214	2637.3711	5.2366415	
420	0.0082046	2607.9538	2804.864	5.4821146	
440	0.0092993	2700.301	2923.4835	5.6509218	
460	0.0102146	2775.3776	3020.5284	5.7851779	
480	0.0110207	2840.5616	3105.0586	5.8989603	
500	0.0117526	2899.3718	3181.4347	5.9990653	
520	0.0124309	2953.8119	3252.1537	6.0893838	
540	0.0130686	3005.129	3318.7754	6.1723477	
560	0.0136745	3054.145	3382.3324	6.24957	
580	0.0142547	3101.4256	3443.5387	6.3221706	
600	0.0148139	3147.3741	3502.9068	6.3909581	
620	0.0153554	3192.2872	3560.8159	6.4565347	
640	0.0158819	3236.3882	3617.5541	6.519362	
660	0.0163956	3279.8496	3673.345	6.5798014	
680	0.0168983	3322.8067	3728.3655	6.6381419	
700	0.0173913	3365.3677	3782.7581	6.6946185	
720	0.0178757	3407.6208	3836.6387	6.7494253	
740	0.0183527	3449.6383	3890.1037	6.8027248	
760	0.018823	3491.4811	3943.2339	6.854655	
780	0.0192874	3533.2007	3996.098	6.9053342	
800	0.0197464	3574.8417	4048.7552	6.9548654	
820	0.0202015	3616.4265	4101.2629	7.0032663	
840	0.0206512	3658.0605	4153.6881	7.050791	
860	0.0210968	3699.7265	4206.0495	7.0974124	
880	0.0215388	3741.4479	4258.3788	7.14319	
900	0.0219774	3783.2443	4310.7031	7.1881761	
920	0.022413	3825.1326	4363.0457	7.2324172	
940	0.0228458	3867.1271	4415.4268	7.2759547	
960	0.023276	3909.2397	4467.8636	7.3188259	
980	0.0237038	3951.4808	4520.371	7.3610641	
1000	0.0241293	3993.8589	4572.9618	7.4026996	
1020	0.0245527	4036.3812	4625.6471	7.4437596	
1040	0.0249743	4079.0538	4678.4363	7.4842692	
1060	0.025394	4121.8816	4731.3375	7.5242512	

==========					
P=28000.0	kPa				
Tsat=377.0113725982492degree	C				
T degree C	v m3/kg	u kJ/kg	h kJ/kg	s kJ/KgK	
377.0114	0.0031056	2000.5935	2087.55	4.412	
400	0.0038545	2226.4898	2334.4162	4.7551712	
420	0.005904	2483.1491	2648.4624	5.2160133	
440	0.007123	2612.1585	2811.6013	5.4482589	
460	0.0080649	2705.9498	2931.7661	5.6145258	
480	0.008862	2782.9794	3031.1167	5.7482693	
500	0.0095676	2850.0526	3117.9447	5.8620801	
520	0.0102096	2910.602	3196.4714	5.9623745	
540	0.0108052	2966.6379	3269.1833	6.0529252	
560	0.0113653	3019.4271	3337.6545	6.1361202	
580	0.0118973	3069.8078	3402.9314	6.2135507	
600	0.0124065	3118.3574	3465.7395	6.2863249	
620	0.0128969	3165.4883	3526.6008	6.3552455	
640	0.0133714	3211.5049	3585.9038	6.4209134	
660	0.0138324	3256.6382	3643.9456	6.4837918	
680	0.0142818	3301.0676	3700.9585	6.5442452	
700	0.0147212	3344.9352	3757.1276	6.6025668	
720	0.0151517	3388.3554	3812.6027	6.6589957	
740	0.0155745	3431.4216	3867.5069	6.7137302	
760	0.0159904	3474.2117	3921.9428	6.7669368	
780	0.0164002	3516.7913	3975.9971	6.8187573	
800	0.0168046	3559.2166	4029.744	6.8693135	
820	0.0172043	3601.5739	4083.2947	6.9186749	
840	0.0175991	3643.876	4136.6511	6.9670437	
860	0.0179898	3686.1592	4189.8749	7.0144331	
880	0.0183769	3728.4524	4243.0055	7.0609118	
900	0.0187606	3770.7806	4296.0766	7.10654	
920	0.0191412	3813.1648	4349.1171	7.151371	
940	0.0195189	3855.6228	4402.1519	7.195452	
960	0.019894	3898.1698	4455.2025	7.238825	
980	0.0202667	3940.8189	4508.2873	7.2815278	
1000	0.0206372	3983.5811	4561.4225	7.3235942	
1020	0.0210056	4026.4656	4614.6219	7.3650549	
1040	0.021372	4069.4804	4667.8973	7.4059377	
1060	0.0217367	4112.6321	4721.2591	7.4462677	
==========					
P=32000.0	kPa				
Tsat=377.0168094973093degree	C				
T degree C	v m3/kg	u kJ/kg	h kJ/kg	s kJ/KgK	
377.0168	0.0031056	1988.1711	2087.55	4.412	
400	0.0023669	1981.0673	2056.8071	4.3255005	
420	0.0040531	2316.8055	2446.5042	4.8961454	
440	0.0054336	2507.2915	2681.1665	5.230333	
460	0.0064257	2627.8531	2833.4757	5.4411192	
480	0.007229	2720.0974	2951.4259	5.5999175	
500	0.0079216	2797.234	3050.7258	5.7300846	
520	0.0085405	2864.9674	3138.2629	5.8418922	
540	0.0091068	2926.394	3217.8117	5.9409605	
560	0.0096339	2983.3948	3291.6799	6.0307152	
580	0.0101306	3037.1766	3361.3557	6.1133651	
600	0.0106029	3088.5442	3427.8385	6.1903982	
620	0.0110553	3138.0559	3491.8266	6.2628605	
640	0.0114911	3186.1138	3553.8278	6.3315169	
660	0.0119127	3233.0177	3614.2238	6.3969461	
680	0.0123223	3278.9977	3673.3098	6.4595982	
700	0.0127214	3324.2343	3731.319	6.5198307	
720	0.0131115	3368.8721	3788.439	6.577933	
740	0.0134936	3413.0284	3844.8228	6.6341427	
760	0.0138687	3456.8	3900.5971	6.6886577	
780	0.0142375	3500.2683	3955.8683	6.7416449	
800	0.0146008	3543.5024	4010.7269	6.7932471	
820	0.0149589	3586.6506	4065.3345	6.8435838	
840	0.0153125	3629.6367	4119.6352	6.8928088	
860	0.0156619	3672.5499	4173.7319	6.9409754	
880	0.0160077	3715.4259	4227.672	6.9881623	
900	0.01635	3758.2946	4281.4961	7.034438	
920	0.0166893	3801.1818	4335.2392	7.0798629	
940	0.0170257	3844.1093	4388.9312	7.1244902	

960	0.0173595	3887.0956	4442.598	7.1683671
980	0.0176908	3930.1568	4496.2621	7.2115358
1000	0.0180199	3973.3063	4549.9426	7.254034
1020	0.0183469	4016.5557	4603.6565	7.2958958
1040	0.018672	4059.9149	4657.4182	7.3371517
1060	0.0189953	4103.3921	4711.2404	7.3778297

Thermodynamic properties of compressed water

Reference: [58] The International Association for the Properties of Water and Steam, Revised Relase on the IAPWS Formulation Program steam_IAPWS95 www.turhancoban.com

==========

P=20.0 kPa

Tsat=60.05864266005824degree C

T degree C	v m3/kg	u kJ/kg	h kJ/kg	s kJ/KgK
60.058643	0.0010171	251.3794	251.39974	0.8319525
60	0.0010171	251.1341	251.15444	0.8312162
50	0.0010121	209.32255	209.3428	0.7037904
40	0.0010079	167.53207	167.55223	0.5724247
30	0.0010044	125.73944	125.75953	0.4367881
20	0.0010018	83.916482	83.936519	0.2964995
10	0.0010003	42.019393	42.039399	0.1510834

==========

P=40.0 kPa

Tsat=75.85682151452102degree C

T degree C	v m3/kg	u kJ/kg	h kJ/kg	s kJ/KgK
75.856822	0.0010264	317.52543	317.56648	1.0259017
70	0.0010228	292.98422	293.02513	0.9549874
60	0.0010171	251.13055	251.17124	0.8312056
50	0.0010121	209.31956	209.36005	0.7037811
40	0.0010079	167.52964	167.56996	0.5724169
30	0.0010044	125.7376	125.77778	0.4367821
20	0.0010018	83.915269	83.955342	0.2964953
10	0.0010003	42.018895	42.058908	0.1510817

==========

P=60.0 kPa

Tsat=85.92577749050952degree C

T degree C	v m3/kg	u kJ/kg	h kJ/kg	s kJ/KgK
85.925777	0.0010331	359.77454	359.83652	1.1452439
80	0.001029	334.89697	334.95871	1.0753831
70	0.0010227	292.98012	293.04149	0.9549754
60	0.0010171	251.12701	251.18803	0.8311949
50	0.0010121	209.31657	209.3773	0.7037718
40	0.0010079	167.52721	167.58768	0.5724092
30	0.0010044	125.73576	125.79602	0.436776
20	0.0010018	83.914057	83.974166	0.2964912
10	0.0010003	42.018397	42.078416	0.1510799

==========

P=80.0 kPa

Tsat=93.48535412870564degree C

T degree C	v m3/kg	u kJ/kg	h kJ/kg	s kJ/KgK
93.485354	0.0010385	391.55568	391.63876	1.2328341
90	0.0010359	376.89317	376.97604	1.1926501
80	0.001029	334.89231	334.97463	1.0753699
70	0.0010227	292.97602	293.05784	0.9549635
60	0.0010171	251.12346	251.20483	0.8311843
50	0.0010121	209.31358	209.39455	0.7037626
40	0.0010078	167.52478	167.60541	0.5724014
30	0.0010044	125.73391	125.81426	0.4367699
20	0.0010018	83.912844	83.992989	0.2964871
10	0.0010003	42.017898	42.097923	0.1510782

==========

P=100.0 kPa

Tsat=99.60591861133764degree C

T degree C	v m3/kg	u kJ/kg	h kJ/kg	s kJ/KgK
99.605919	0.0010431	417.33217	417.43649	1.3025602
90	0.0010359	376.88792	376.99151	1.1926357
80	0.001029	334.88764	334.99054	1.0753567
70	0.0010227	292.97192	293.0742	0.9549515
60	0.0010171	251.11992	251.22162	0.8311737
50	0.0010121	209.31059	209.4118	0.7037533
40	0.0010078	167.52236	167.62314	0.5723937
30	0.0010044	125.73207	125.83251	0.4367638
20	0.0010018	83.911631	84.011811	0.2964829
10	0.0010003	42.0174	42.11743	0.1510764

==========

P=101.325 kPa

Tsat=99.97430000048058degree C

T degree C	v m3/kg	u kJ/kg	h kJ/kg	s kJ/KgK
99.9743	0.0010434	418.88499	418.99072	1.306724
90	0.0010359	376.88757	376.99254	1.1926347
80	0.001029	334.88733	334.9916	1.0753558
70	0.0010227	292.97165	293.07528	0.9549507
60	0.0010171	251.11968	251.22274	0.831173
50	0.0010121	209.31039	209.41294	0.7037527
40	0.0010078	167.52219	167.62431	0.5723932
30	0.0010044	125.73195	125.83371	0.4367634
20	0.0010018	83.911551	84.013058	0.2964826
10	0.0010003	42.017367	42.118722	0.1510763

==========

P=200.0 kPa

Tsat=120.21154593648885degree C

T degree C	v m3/kg	u kJ/kg	h kJ/kg	s kJ/KgK
120.21155	0.0010605	504.47174	504.68385	1.5300982
120	0.0010603	503.57344	503.7855	1.5278138
110	0.0010516	461.19424	461.40455	1.4186216
100	0.0010434	418.96451	419.1732	1.3069371
90	0.0010359	376.86171	377.06889	1.1925635
80	0.001029	334.86433	335.07013	1.0752907
70	0.0010227	292.95144	293.15597	0.9548918
60	0.001017	251.1022	251.30561	0.8311205
50	0.0010121	209.29564	209.49805	0.703707
40	0.0010078	167.51021	167.71177	0.5723549
30	0.0010043	125.72285	125.92372	0.4367334
20	0.0010018	83.905569	84.105919	0.2964622
10	0.0010003	42.014908	42.214958	0.1510676

==========

P=300.0 kPa

Tsat=133.52535794654545degree C

T degree C	v m3/kg	u kJ/kg	h kJ/kg	s kJ/KgK
133.52536	0.0010732	561.13346	561.45541	1.6717647
130	0.0010697	546.08704	546.40795	1.6346022
120	0.0010603	503.53769	503.85577	1.5277229
110	0.0010515	461.16185	461.4773	1.4185371
100	0.0010434	418.9353	419.2483	1.3068588
90	0.0010358	376.83551	377.14626	1.1924913
80	0.0010289	334.84104	335.14971	1.0752247
70	0.0010226	292.93095	293.23774	0.9548321
60	0.001017	251.08449	251.38958	0.8310673
50	0.001012	209.28069	209.58429	0.7036607
40	0.0010077	167.49807	167.8004	0.5723161
30	0.0010043	125.71364	126.01492	0.436703
20	0.0010017	83.899506	84.200018	0.2964415
10	0.0010002	42.012413	42.312474	0.1510587

==========

P=400.0 kPa

Tsat=143.6125329983828degree C

T degree C	v m3/kg	u kJ/kg	h kJ/kg	s kJ/KgK
143.61253	0.0010836	604.29005	604.72347	1.7765982
140	0.0010797	588.79331	589.2252	1.7392488
130	0.0010696	546.04773	546.47558	1.6345047
120	0.0010602	503.50197	503.92605	1.5276319
110	0.0010514	461.12948	461.55005	1.4184525
100	0.0010433	418.90609	419.32342	1.3067804
90	0.0010358	376.80933	377.22364	1.1924192
80	0.0010289	334.81775	335.2293	1.0751587
70	0.0010226	292.91048	293.31952	0.9547724
60	0.0010169	251.06678	251.47356	0.8310141
50	0.001012	209.26575	209.67053	0.7036145
40	0.0010077	167.48594	167.88902	0.5722773
30	0.0010042	125.70443	126.10612	0.4366725
20	0.0010017	83.893444	84.294108	0.2964208
10	0.0010002	42.009917	42.409979	0.1510498

==========

P=500.0 kPa

Tsat=151.83624387684512degree C

T degree C	v m3/kg	u kJ/kg	h kJ/kg	s kJ/KgK
151.83624	0.0010926	639.63906	640.18534	1.8605993
150	0.0010905	631.72106	632.2663	1.8419253
140	0.0010797	588.75015	589.28999	1.7391443
130	0.0010696	546.00843	546.54322	1.6344071
120	0.0010602	503.46625	503.99633	1.5275411
110	0.0010514	461.09712	461.62282	1.418368
100	0.0010433	418.8769	419.39853	1.3067022
90	0.0010357	376.78315	377.30102	1.192347
80	0.0010288	334.79447	335.30888	1.0750927
70	0.0010225	292.89002	293.40129	0.9547127

60	0.0010169	251.04909	251.55754	0.8309609
50	0.0010119	209.25081	209.75677	0.7035682
40	0.0010077	167.47381	167.97764	0.5722385
30	0.0010042	125.69522	126.19731	0.4366421
20	0.0010016	83.887382	84.38819	0.2964
10	0.0010001	42.007418	42.507472	0.1510409

==========

P=600.0 kPa

Tsat=158.8324239544848degree C

T degree C	v m3/kg	u kJ/kg	h kJ/kg	s kJ/KgK
158.83242	0.0011006	669.84084	670.50121	1.9310959
150	0.0010904	631.67375	632.328	1.8418134
140	0.0010796	588.70702	589.35479	1.7390398
130	0.0010695	545.96916	546.61087	1.6343096
120	0.0010601	503.43056	504.06662	1.5274502
110	0.0010513	461.06478	461.69558	1.4182835
100	0.0010432	418.84773	419.47365	1.3066239
90	0.0010357	376.75699	377.3784	1.1922749
80	0.0010288	334.7712	335.38847	1.0750267
70	0.0010225	292.86956	293.48306	0.954653
60	0.0010169	251.0314	251.64151	0.8309077
50	0.0010119	209.23588	209.84301	0.7035219
40	0.0010076	167.46168	168.06625	0.5721997
30	0.0010041	125.68601	126.2885	0.4366116
20	0.0010016	83.881321	84.482262	0.2963793
10	0.0010001	42.004917	42.604954	0.151032

==========

P=700.0 kPa

Tsat=164.95275255074188degree C

T degree C	v m3/kg	u kJ/kg	h kJ/kg	s kJ/KgK
164.95275	0.001108	696.36779	697.14336	1.9920831
160	0.0011019	674.85105	675.6224	1.9426804
150	0.0010904	631.62646	632.38971	1.8417016
140	0.0010795	588.66392	589.4196	1.7389354
130	0.0010695	545.9299	546.67852	1.6342122
120	0.00106	503.39488	504.13692	1.5273593
110	0.0010513	461.03245	461.76835	1.4181991
100	0.0010432	418.81856	419.54877	1.3065456
90	0.0010356	376.73083	377.45578	1.1922028
80	0.0010287	334.74794	335.46806	1.0749608
70	0.0010225	292.84912	293.56483	0.9545933
60	0.0010168	251.01372	251.72548	0.8308545
50	0.0010118	209.22096	209.92924	0.7034756
40	0.0010076	167.44956	168.15486	0.5721609
30	0.0010041	125.67681	126.37968	0.4365812
20	0.0010015	83.87526	84.576326	0.2963585
10	0.001	42.002415	42.702423	0.1510231

==========

P=800.0 kPa

Tsat=170.41351081360017degree C

T degree C	v m3/kg	u kJ/kg	h kJ/kg	s kJ/KgK
170.41351	0.0011148	720.12602	721.01785	2.0459892
170	0.0011143	718.31933	719.21073	2.0419132
160	0.0011019	674.79923	675.68072	1.9425607
150	0.0010903	631.5792	632.45143	1.8415898
140	0.0010795	588.62084	589.48442	1.738831
130	0.0010694	545.89067	546.74619	1.6341147
120	0.00106	503.35923	504.20722	1.5272685
110	0.0010512	461.00014	461.84113	1.4181146
100	0.0010431	418.78941	419.62389	1.3064674
90	0.0010356	376.70469	377.53316	1.1921307
80	0.0010287	334.7247	335.54764	1.0748949
70	0.0010224	292.82868	293.6466	0.9545337
60	0.0010168	250.99604	251.80945	0.8308014
50	0.0010118	209.20604	210.01546	0.7034294
40	0.0010075	167.43744	168.24346	0.5721221
30	0.0010041	125.66761	126.47085	0.4365507
20	0.0010015	83.8692	84.670381	0.2963377
10	1.00E-03	41.99991	42.799882	0.1510141

==========

P=900.0 kPa

Tsat=175.3578221170128degree C

T degree C	v m3/kg	u kJ/kg	h kJ/kg	s kJ/KgK
175.35782	0.0011212	741.71555	742.72462	2.0944048
170	0.0011142	718.26251	719.26527	2.0417848
160	0.0011018	674.74744	675.73905	1.942441
150	0.0010902	631.53197	632.51317	1.841478
140	0.0010794	588.57778	589.54925	1.7387266
130	0.0010693	545.85146	546.81386	1.6340174
120	0.0010599	503.32359	504.27752	1.5271778
110	0.0010512	460.96785	461.91391	1.4180302
100	0.001043	418.76028	419.69902	1.3063892

90	0.0010355	376.67856	377.61055	1.1920587
80	0.0010286	334.70146	335.62723	1.0748289
70	0.0010224	292.80825	293.72837	0.954474
60	0.0010167	250.97837	251.89341	0.8307482
50	0.0010117	209.19112	210.10169	0.7033831
40	0.0010075	167.42533	168.33206	0.5720833
30	0.001004	125.65841	126.56201	0.4365202
20	0.0010014	83.86314	84.764428	0.2963169
10	1.00E-03	41.997403	42.897328	0.1510051

==========

P=1000.0 kPa

Tsat=179.88563239146663degree C

T degree C	v m3/kg	u kJ/kg	h kJ/kg	s kJ/KgK
179.88563	0.0011272	761.55561	762.68284	2.1384314
170	0.0011141	718.20574	719.31984	2.0416565
160	0.0011017	674.69568	675.79739	1.9423213
150	0.0010902	631.48477	632.57492	1.8413663
140	0.0010794	588.53474	589.61409	1.7386223
130	0.0010693	545.81226	546.88154	1.63392
120	0.0010599	503.28796	504.34784	1.527087
110	0.0010511	460.93557	461.98669	1.4179458
100	0.001043	418.73115	419.77415	1.3063111
90	0.0010355	376.65245	377.68793	1.1919866
80	0.0010286	334.67823	335.70682	1.074763
70	0.0010223	292.78783	293.81014	0.9544144
60	0.0010167	250.96071	251.97738	0.8306951
50	0.0010117	209.17621	210.18791	0.7033368
40	0.0010074	167.41322	168.42065	0.5720444
30	0.001004	125.64921	126.65317	0.4364897
20	0.0010014	83.85708	84.858465	0.2962961
10	1.00E-03	41.994894	42.994764	0.1509961

==========

P=1500.0 kPa

Tsat=198.29524288241413degree C

T degree C	v m3/kg	u kJ/kg	h kJ/kg	s kJ/KgK
198.29524	0.0011539	842.98611	844.71691	2.3146816
190	0.0011412	805.96596	807.67779	2.2354182
180	0.001127	761.74816	763.43861	2.1388556
170	0.0011137	717.92238	719.59295	2.041016
160	0.0011013	674.43734	676.08936	1.9417238
150	0.0010898	631.24914	632.88386	1.8408085
140	0.001079	588.3199	589.93846	1.7381014
130	0.001069	545.61659	547.22006	1.6334337
120	0.0010596	503.11011	504.6995	1.5266337
110	0.0010509	460.7744	462.35068	1.4175243
100	0.0010427	418.58574	420.14985	1.3059205
90	0.0010352	376.52203	378.0749	1.1916266
80	0.0010284	334.56224	336.10477	1.0744337
70	0.0010221	292.68584	294.21897	0.9541163
60	0.0010164	250.8725	252.39717	0.8304295
50	0.0010115	209.10175	210.61897	0.7031055
40	0.0010072	167.35272	168.86355	0.5718504
30	0.0010037	125.60326	127.10887	0.4363372
20	0.0010012	83.826788	85.328522	0.2961918
10	1.00E-03	41.982319	43.481766	0.1509506

==========

P=2000.0 kPa

Tsat=212.38453531849052degree C

T degree C	v m3/kg	u kJ/kg	h kJ/kg	s kJ/KgK
212.38454	0.0011768	906.26835	908.62185	2.4470239
210	0.0011726	895.41525	897.76045	2.4245988
200	0.0011561	850.26036	852.57248	2.3300916
190	0.0011408	805.62499	807.90649	2.2346802
180	0.0011265	761.43809	763.69117	2.1381697
170	0.0011133	717.63992	719.86656	2.0403771
160	0.001101	674.17976	676.38174	1.9411277
150	0.0010895	631.01417	633.19313	1.8402518
140	0.0010787	588.10563	590.26308	1.7375814
130	0.0010687	545.42142	547.55879	1.6329483
120	0.0010593	502.93269	505.05132	1.5261812
110	0.0010506	460.61361	462.71479	1.4171035
100	0.0010425	418.44065	420.52563	1.3055305
90	0.001035	376.3919	378.4619	1.1912671
80	0.0010281	334.44649	336.50273	1.0741048
70	0.0010219	292.58407	294.62778	0.9538186
60	0.0010162	250.78447	252.81691	0.8301641
50	0.0010113	209.02743	211.04994	0.7028743
40	0.001007	167.29233	169.30632	0.5716563
30	0.0010035	125.55736	127.5644	0.4361846
20	0.0010009	83.796505	85.798361	0.2960871
10	9.99E-04	41.969692	43.968479	0.1509045

==========

P=2500.0 kPa

Tsat=223.95648747437707degree C

T degree C	v m3/kg	u kJ/kg	h kJ/kg	s kJ/KgK
223.95649	0.0011974	958.98957	961.98317	2.5544256
220	0.0011899	940.71556	943.69039	2.5174801
210	0.001172	895.00131	897.93142	2.4237395
200	0.0011556	849.88594	852.77482	2.3292979
190	0.0011403	805.28523	808.13596	2.2339444
180	0.0011261	761.12904	763.94434	2.1374856
170	0.0011129	717.35834	720.14066	2.0397397
160	0.0011006	673.92294	676.67452	1.9405329
150	0.0010891	630.77986	633.50272	1.8396963
140	0.0010784	587.89193	590.58796	1.7370625
130	0.0010684	545.22674	547.89772	1.6324638
120	0.001059	502.7557	505.40329	1.5257294
110	0.0010503	460.45319	463.07901	1.4166832
100	0.0010422	418.29589	420.90148	1.305141
90	0.0010348	376.26206	378.84895	1.1909081
80	0.0010279	334.33099	336.9007	1.0737763
70	0.0010216	292.4825	295.03657	0.9535211
60	0.001016	250.69661	253.23661	0.8298988
50	0.001011	208.95324	211.48082	0.7026432
40	0.0010068	167.23203	169.74897	0.5714622
30	0.0010033	125.51152	128.01976	0.4360317
20	0.0010007	83.766232	86.267981	0.2959821
10	9.99E-04	41.957015	44.454905	0.1508579

==========

P=3000.0 kPa

Tsat=233.85844500625223degree C

T degree C	v m3/kg	u kJ/kg	h kJ/kg	s kJ/KgK
233.85845	0.0012167	1004.7213	1008.3714	2.6456205
230	0.0012087	986.62121	990.24739	2.6097371
220	0.0011893	940.2582	943.8261	2.5165491
210	0.0011715	894.58912	898.10356	2.4228831
200	0.001155	849.51296	852.9781	2.3285066
190	0.0011398	804.94668	808.36618	2.2332107
180	0.0011257	760.82103	764.19812	2.1368033
170	0.0011125	717.07763	720.41526	2.0391039
160	0.0011003	673.66688	676.96769	1.9399395
150	0.0010888	630.5462	633.81263	1.8391419
140	0.0010781	587.6788	590.9131	1.7365445
130	0.0010681	545.03256	548.23685	1.6319801
120	0.0010588	502.57914	505.75542	1.5252784
110	0.0010501	460.29315	463.44334	1.4162636
100	0.001042	418.15146	421.27741	1.3047521
90	0.0010345	376.1325	379.23604	1.1905495
80	0.0010276	334.21573	337.29868	1.0734481
70	0.0010214	292.38115	295.44534	0.9532239
60	0.0010158	250.60892	253.65625	0.8296338
50	0.0010108	208.87919	211.91163	0.7024122
40	0.0010066	167.17184	170.1915	0.571268
30	0.0010031	125.46575	128.47496	0.4358787
20	0.0010005	83.735968	86.737384	0.2958767
10	9.99E-04	41.944287	44.941044	0.1508106

==========

P=3500.0 kPa

Tsat=242.56171159061591degree C

T degree C	v m3/kg	u kJ/kg	h kJ/kg	s kJ/KgK
242.56171	0.001235	1045.4529	1049.7753	2.7253906
240	0.0012292	1033.234	1037.5362	2.701599
230	0.001208	986.11375	990.34182	2.608724
220	0.0011887	939.80293	943.9633	2.5156218
210	0.0011709	894.17864	898.27686	2.4220298
200	0.0011545	849.14142	853.18231	2.3277179
190	0.0011394	804.60934	808.59714	2.2324791
180	0.0011253	760.51403	764.45251	2.1361228
170	0.0011122	716.7978	720.69035	2.0384696
160	0.0010999	673.41156	677.26125	1.9393474
150	0.0010885	630.31318	634.12285	1.8385887
140	0.0010778	587.46622	591.23848	1.7360275
130	0.0010678	544.83886	548.57618	1.6314973
120	0.0010585	502.40301	506.1077	1.5248281
110	0.0010498	460.13349	463.80778	1.4158447
100	0.0010417	418.00735	421.65342	1.3043637
90	0.0010343	376.00322	379.62317	1.1901913
80	0.0010274	334.10072	337.69667	1.0731202
70	0.0010212	292.28	295.85409	0.952927
60	0.0010156	250.5214	254.07583	0.8293689
50	0.0010106	208.80528	212.34234	0.7021812
40	0.0010063	167.11174	170.63391	0.5710738
30	0.0010028	125.42002	128.93	0.4357255
20	0.0010002	83.705714	87.20657	0.295771

10	9.99E-04	41.931509	45.426897	0.1507628

==========

P=4000.0 kPa

Tsat=250.35751912406556degree C

T degree C	v m3/kg	u kJ/kg	h kJ/kg	s kJ/KgK
250.35752	0.0012526	1082.4157	1087.426	2.7966534
250	0.0012517	1080.6793	1085.6861	2.7933287
240	0.0012284	1032.668	1037.5817	2.7004903
230	0.0012073	985.60884	990.43812	2.6077152
220	0.0011881	939.34972	944.10195	2.5146981
210	0.0011704	893.76986	898.45132	2.4211794
200	0.001154	848.77128	853.38744	2.3269316
190	0.0011389	804.27319	808.82885	2.2317497
180	0.0011249	760.20804	764.70749	2.1354441
170	0.0011118	716.51883	720.96592	2.0378368
160	0.0010996	673.15699	677.55521	1.9387566
150	0.0010881	630.08081	634.43339	1.8380366
140	0.0010775	587.25421	591.56412	1.7355116
130	0.0010675	544.64566	548.9157	1.6310153
120	0.0010582	502.2273	506.46013	1.5243786
110	0.0010495	459.9742	464.17234	1.4154264
100	0.0010415	417.86358	422.0295	1.3039759
90	0.001034	375.87423	380.01035	1.1898336
80	0.0010272	333.98595	338.09468	1.0727928
70	0.0010209	292.17905	296.26282	0.9526303
60	0.0010153	250.43406	254.49537	0.8291042
50	0.0010104	208.7315	212.77298	0.7019504
40	0.0010061	167.05174	171.0762	0.5708796
30	0.0010026	125.37436	129.38487	0.4355721
20	0.001	83.67547	87.67554	0.2956649
10	9.98E-04	41.91868	45.912464	0.1507143

==========

P=4500.0 kPa

Tsat=257.4393713108474degree C

T degree C	v m3/kg	u kJ/kg	h kJ/kg	s kJ/KgK
257.43937	0.0012697	1116.4295	1122.143	2.8613276
250	0.0012508	1080.044	1085.6725	2.7921068
240	0.0012276	1032.1052	1037.6295	2.6993869
230	0.0012066	985.10645	990.53626	2.6067109
220	0.0011874	938.89856	944.24204	2.5137779
210	0.0011698	893.36277	898.62692	2.420332
200	0.0011535	848.40256	853.59349	2.3261479
190	0.0011385	803.93822	809.0613	2.2310222
180	0.0011244	759.90306	764.96307	2.1347672
170	0.0011114	716.24072	721.24197	2.0372055
160	0.0010992	672.90315	677.84956	1.9381671
150	0.0010878	629.84908	634.74424	1.8374856
140	0.0010772	587.04276	591.89001	1.7349966
130	0.0010672	544.45294	549.25542	1.6305342
120	0.0010579	502.05202	506.8127	1.5239297
110	0.0010493	459.81528	464.537	1.4150087
100	0.0010412	417.72012	422.40565	1.3035886
90	0.0010338	375.74552	380.39757	1.1894764
80	0.0010269	333.87142	338.49269	1.0724657
70	0.0010207	292.07831	296.67153	0.9523339
60	0.0010151	250.34688	254.91486	0.8288397
50	0.0010102	208.65785	213.20353	0.7017196
40	0.0010059	166.99184	171.51836	0.5706853
30	0.0010024	125.32875	129.83958	0.4354185
20	1.00E-03	83.645236	88.144294	0.2955585
10	9.98E-04	41.905802	46.397746	0.1506653

==========

P=5000.0 kPa

Tsat=263.9428711863302degree C

T degree C	v m3/kg	u kJ/kg	h kJ/kg	s kJ/KgK
263.94287	0.0012864	1148.07	1154.502	2.920746
260	0.0012755	1128.3967	1134.7741	2.8838798
250	0.0012499	1079.4126	1085.662	2.7908918
240	0.0012268	1031.5455	1037.6797	2.6982889
230	0.0012059	984.60655	990.63621	2.6057108
220	0.0011868	938.44943	944.38356	2.5128613
210	0.0011693	892.95735	898.80366	2.4194874
200	0.001153	848.03523	853.80044	2.3253665
190	0.001138	803.60442	809.29447	2.2302969
180	0.001124	759.59908	765.21923	2.134092
170	0.001111	715.96346	721.51849	2.0365758
160	0.0010988	672.65006	678.14429	1.9375788
150	0.0010875	629.61799	635.0554	1.8369358
140	0.0010769	586.83186	592.21614	1.7344825
130	0.0010669	544.2607	549.59533	1.6300539
120	0.0010577	501.87717	507.16543	1.5234816
110	0.001049	459.65672	464.90177	1.4145916

100	0.001041	417.57699	422.78188	1.3032018
90	0.0010335	375.61709	380.78482	1.1891195
80	0.0010267	333.75714	338.89071	1.0721389
70	0.0010205	291.97778	297.08022	0.9520378
60	0.0010149	250.25987	255.33429	0.8285754
50	0.0010099	208.58434	213.63399	0.7014889
40	0.0010057	166.93203	171.96041	0.570491
30	0.0010022	125.2832	130.29413	0.4352648
20	1.00E-03	83.615011	88.612832	0.2954517
10	9.98E-04	41.892873	46.882745	0.1506157

==========

P=6000.0 kPa

Tsat=275.58641075604896degree C

T degree C	v m3/kg	u kJ/kg	h kJ/kg	s kJ/KgK
275.58641	0.0013193	1205.8154	1213.7311	3.0274397
270	0.0013018	1177.1116	1184.9222	2.9746711
260	0.0012734	1126.9739	1134.6142	2.8811895
250	0.0012481	1078.1616	1085.6501	2.7884816
240	0.0012253	1030.4352	1037.7869	2.6961086
230	0.0012046	983.61411	990.84147	2.6037234
220	0.0011856	937.55717	944.6708	2.5110384
210	0.0011682	892.15146	899.16047	2.417807
200	0.0011521	847.3047	854.21702	2.323811
190	0.0011371	802.94032	809.76297	2.2288523
180	0.0011232	758.99407	765.73331	2.1327469
170	0.0011102	715.41149	722.07295	2.0353208
160	0.0010981	672.14605	678.73489	1.9364063
150	0.0010868	629.1577	635.67864	1.8358395
140	0.0010762	586.4117	592.86914	1.7334574
130	0.0010663	543.87767	550.27574	1.6290959
120	0.0010571	501.52871	507.87133	1.5225876
110	0.0010485	459.34072	465.63164	1.4137594
100	0.0010405	417.29168	423.53455	1.3024298
90	0.0010331	375.36106	381.55945	1.1884072
80	0.0010262	333.52928	339.68678	1.0714865
70	0.00102	291.77732	297.89753	0.9514464
60	0.0010144	250.08637	256.173	0.8280472
50	0.0010095	208.43772	214.49467	0.7010277
40	0.0010052	166.81272	172.84414	0.5701023
30	0.0010017	125.19227	131.20273	0.4349569
20	9.99E-04	83.554589	89.549264	0.2952371
10	9.98E-04	41.866868	47.851893	0.1505147

==========

P=7000.0 kPa

Tsat=285.8300228057516degree C

T degree C	v m3/kg	u kJ/kg	h kJ/kg	s kJ/KgK
285.83002	0.0013519	1257.9742	1267.4372	3.1219943
280	0.0013311	1227.026	1236.3439	3.0660783
270	0.0012993	1175.5001	1184.5954	2.9716749
260	0.0012713	1125.5709	1134.4701	2.8785328
250	0.0012463	1076.9258	1085.65	2.7860974
240	0.0012237	1029.337	1037.9032	2.693949
230	0.0012032	982.63131	991.05374	2.6017525
220	0.0011844	936.67278	944.96356	2.5092291
210	0.0011671	891.35208	899.52167	2.4161379
200	0.0011511	846.57961	854.63712	2.3222651
190	0.0011362	802.28081	810.23431	2.2274158
180	0.0011224	758.39298	766.24967	2.1314087
170	0.0011095	714.86287	722.62928	2.0340717
160	0.0010974	671.64493	679.32699	1.9352388
150	0.0010862	628.69991	636.30309	1.8347475
140	0.0010756	585.99373	593.52311	1.732436
130	0.0010658	543.49654	550.95691	1.6281412
120	0.0010566	501.18192	508.57781	1.5216965
110	0.001048	459.02618	466.36193	1.4129296
100	0.00104	417.00764	424.2875	1.30166
90	0.0010326	375.10613	382.33424	1.1876967
80	0.0010258	333.30238	340.48289	1.0708354
70	0.0010196	291.57767	298.71476	0.950856
60	0.001014	249.91353	257.01151	0.8275198
50	0.0010091	208.29163	215.35501	0.7005668
40	0.0010048	166.69378	173.72739	0.5697134
30	0.0010013	125.10157	132.11069	0.4346482
20	9.99E-04	83.494207	90.48484	0.2950211
10	9.97E-04	41.840666	48.819913	0.1504114

==========

P=8000.0 kPa

Tsat=295.0091212293113degree C

T degree C	v m3/kg	u kJ/kg	h kJ/kg	s kJ/KgK
295.00912	0.0013847	1306.0025	1317.0798	3.207651
290	0.0013643	1278.4114	1289.3258	3.1585863
280	0.0013282	1225.1813	1235.8072	3.0627042

270	0.0012969	1173.9141	1184.2896	2.968722
260	0.0012693	1124.187	1134.3414	2.8759087
250	0.0012446	1075.7048	1085.6614	2.7837385
240	0.0012222	1028.2505	1038.0282	2.6918094
230	0.0012019	981.65794	991.27284	2.599798
220	0.0011832	935.7961	945.26172	2.5074332
210	0.001166	890.5591	899.88717	2.41448
200	0.0011501	845.85987	855.06067	2.3207285
190	0.0011353	801.62582	810.70843	2.2259872
180	0.0011216	757.79573	766.7683	2.1300772
170	0.0011087	714.31755	723.18743	2.0328284
160	0.0010967	671.14667	679.92058	1.9340764
150	0.0010855	628.24461	636.92874	1.83366
140	0.001075	585.57792	594.17804	1.7314185
130	0.0010652	543.11729	551.63882	1.6271898
120	0.001056	500.83678	509.28486	1.5208082
110	0.0010474	458.71307	467.09265	1.4121022
100	0.0010395	416.72487	425.04074	1.3008922
90	0.0010321	374.8523	383.10918	1.1869879
80	0.0010253	333.07643	341.27902	1.0701858
70	0.0010191	291.37883	299.5319	0.9502667
60	0.0010136	249.74135	257.8498	0.8269931
50	0.0010086	208.14606	216.21501	0.7001062
40	0.0010044	166.57523	174.61015	0.5693244
30	0.0010009	125.01108	133.01799	0.4343389
20	9.98E-04	83.433862	91.419563	0.2948038
10	9.97E-04	41.814269	49.786813	0.1503058

==========

P=9000.0 kPa

Tsat=303.34680497481475degree C

T degree C	v m3/kg	u kJ/kg	h kJ/kg	s kJ/KgK
303.3468	0.0014181	1350.8876	1363.6507	3.2865721
300	0.0014024	1331.6479	1344.2693	3.2528552
290	0.0013608	1276.2703	1288.5174	3.1547314
280	0.0013254	1223.3706	1235.2993	3.0593873
270	0.0012946	1172.3528	1184.0041	2.9658107
260	0.0012673	1122.8217	1134.2274	2.873316
250	0.0012429	1074.4982	1085.6839	2.7814041
240	0.0012207	1027.1754	1038.1619	2.6896894
230	0.0012005	980.6938	991.49862	2.5978594
220	0.001182	934.92698	945.56516	2.5056504
210	0.0011649	889.77239	900.25689	2.4128329
200	0.0011491	845.1454	855.48761	2.3192011
190	0.0011344	800.97529	811.1853	2.2245665
180	0.0011208	757.20229	767.28915	2.1287524
170	0.001108	713.7755	723.74739	2.0315909
160	0.001096	670.65123	680.51563	1.932919
150	0.0010849	627.79175	637.55557	1.8325768
140	0.0010744	585.16424	594.8339	1.7304047
130	0.0010646	542.73992	552.32148	1.6262416
120	0.0010555	500.49328	509.99248	1.5199227
110	0.0010469	458.40141	467.82377	1.4112773
100	0.001039	416.44335	425.79424	1.3001265
90	0.0010316	374.59956	383.88427	1.1862808
80	0.0010249	332.85141	342.07519	1.0695375
70	0.0010187	291.18078	300.34895	0.9496785
60	0.0010131	249.56984	258.68789	0.826467
50	0.0010082	208.00102	217.07467	0.6996459
40	0.0010039	166.45706	175.49244	0.5689353
30	0.0010004	124.92082	133.92466	0.4340288
20	9.98E-04	83.373554	92.353437	0.2945851
10	9.96E-04	41.787679	50.752597	0.1501979

==========

P=10000.0 kPa

Tsat=310.99948799852825degree C

T degree C	v m3/kg	u kJ/kg	h kJ/kg	s kJ/KgK
310.99949	0.0014526	1393.3413	1407.8675	3.3602907
310	0.0014471	1387.2969	1401.7679	3.34984
300	0.001398	1329.1163	1343.0966	3.2483661
290	0.0013574	1274.176	1287.7499	3.1509551
280	0.0013226	1221.5924	1234.8188	3.0561251
270	0.0012923	1170.8152	1183.7381	2.9629395
260	0.0012653	1121.4744	1134.1278	2.870754
250	0.0012412	1073.3056	1085.7172	2.7790937
240	0.0012192	1026.1114	1038.3038	2.6875886
230	0.0011992	979.73869	991.73093	2.5959363
220	0.0011808	934.06529	945.87376	2.5038805
210	0.0011639	888.99185	900.63074	2.4111967
200	0.0011482	844.43612	855.91788	2.3176828
190	0.0011336	800.32915	811.66486	2.2231534
180	0.00112	756.6126	767.81218	2.1274343
170	0.0011072	713.23667	724.30912	2.030359

160	0.0010954	670.15858	681.11212	1.9317665
150	0.0010842	627.34131	638.18355	1.8314978
140	0.0010738	584.75267	595.49069	1.7293946
130	0.001064	542.36439	553.00486	1.6252967
120	0.0010549	500.15139	510.70065	1.5190401
110	0.0010464	458.09116	468.5553	1.4104548
100	0.0010385	416.16307	426.54802	1.2993628
90	0.0010312	374.34791	384.6595	1.1855754
80	0.0010244	332.62732	342.87138	1.0688907
70	0.0010182	290.98352	301.16591	0.9490913
60	0.0010127	249.39898	259.52576	0.8259416
50	0.0010078	207.85649	217.934	0.699186
40	0.0010035	166.33926	176.37425	0.568546
30	1.00E-03	124.83078	134.83067	0.4337181
20	9.97E-04	83.313285	93.286466	0.294365
10	9.96E-04	41.760896	51.717272	0.1500878

==========

P=15000.0 kPa

Tsat=342.1578712489836degree C

T degree C	v m3/kg	u kJ/kg	h kJ/kg	s kJ/KgK
342.15787	0.001657	1585.2973	1610.1518	3.684446
340	0.0016311	1567.8072	1592.2733	3.6553393
330	0.0015394	1495.5489	1518.6394	3.5342853
320	0.0014733	1431.7469	1453.846	3.4259748
310	0.0014212	1372.8944	1394.2132	3.3245917
300	0.0013783	1317.3894	1338.0633	3.2274766
290	0.0013416	1264.328	1284.452	3.1331187
280	0.0013096	1213.1442	1232.7889	3.0405595
270	0.0012813	1163.4558	1182.6759	2.9491389
260	0.001256	1114.9894	1133.829	2.8583713
250	0.001233	1067.5405	1086.0356	2.7678794
240	0.0012121	1020.9503	1039.1314	2.6773567
230	0.0011929	975.09233	992.98526	2.586544
220	0.0011751	929.86344	947.49064	2.4952158
210	0.0011587	885.17813	902.55926	2.4031702
200	0.0011435	840.96466	858.11709	2.3102224
190	0.0011293	797.16218	814.10148	2.2162001
180	0.001116	753.71871	770.45892	2.1209394
170	0.0011036	710.58956	727.14346	2.0242828
160	0.001092	667.73606	684.11542	1.9260757
150	0.001081	625.12461	641.34032	1.8261655
140	0.0010708	582.72585	598.78811	1.7243986
130	0.0010612	540.5139	556.43235	1.6206194
120	0.0010523	498.46579	514.2496	1.5146678
110	0.0010439	456.56079	472.21882	1.4063776
100	0.0010361	414.77993	430.3208	1.2955746
90	0.0010288	373.1055	388.53769	1.1820734
80	0.0010221	331.52061	346.8526	1.0656764
70	0.001016	290.00887	305.24925	0.9461704
60	0.0010105	248.55433	263.71192	0.8233249
50	0.0010056	207.14152	222.22555	0.6968904
40	0.0010014	165.75583	180.77615	0.5665973
30	9.98E-04	124.38383	139.3512	0.4321546
20	9.95E-04	83.012501	97.939065	0.2932446
10	9.93E-04	41.624167	56.524211	0.1495033

==========

P=20000.0 kPa

Tsat=365.7459115457051degree C

T degree C	v m3/kg	u kJ/kg	h kJ/kg	s kJ/KgK
365.74591	0.0020386	1786.328	1827.1009	4.0153651
360	0.0018247	1703.6393	1740.1337	3.8787141
350	0.0016649	1612.6537	1645.9511	3.7288486
340	0.0015693	1540.1296	1571.5159	3.6084554
330	0.0014998	1475.7909	1505.7863	3.5003862
320	0.0014449	1416.4033	1445.3021	3.3992749
310	0.0013997	1360.4107	1388.4038	3.3025388
300	0.0013611	1306.9178	1334.1395	3.2086837
290	0.0013275	1255.3556	1281.906	3.1167499
280	0.0012978	1205.3365	1231.2933	3.0260719
270	0.0012713	1156.5819	1182.0072	2.9361593
260	0.0012472	1108.8828	1133.8276	2.8466314
250	0.0012254	1062.0764	1086.5836	2.7571793
240	0.0012053	1016.0329	1040.1388	2.6675432
230	0.0011868	970.64612	994.38233	2.577497
220	0.0011697	925.8279	949.22185	2.4868395
210	0.0011538	881.50402	904.58002	2.3953869
200	0.001139	837.61143	860.39115	2.3029688
190	0.0011251	794.09613	816.59905	2.2094237
180	0.0011122	750.91156	773.15539	2.1145972
170	0.0011	708.01744	730.01837	2.0183386
160	0.0010886	665.37872	687.15171	1.9204999
150	0.0010779	622.9648	644.52378	1.8209333

140	0.0010679	580.74884	602.10687	1.7194899
130	0.0010585	538.70714	559.87661	1.6160181
120	0.0010496	496.8186	517.81133	1.5103616
110	0.0010414	455.06415	475.89157	1.4023575
100	0.0010337	413.42635	434.09961	1.2918348
90	0.0010265	371.88885	392.41895	1.178612
80	0.0010199	330.43612	350.83406	1.0624948
70	0.0010138	289.05313	309.33004	0.943274
60	0.0010084	247.72535	267.89267	0.8207242
50	0.0010035	206.43897	226.50866	0.6946017
40	9.99E-04	165.18143	185.16623	0.5646449
30	9.96E-04	123.94217	143.85598	0.4305745
20	9.93E-04	82.712644	102.57109	0.2920915
10	9.91E-04	41.482888	61.304283	0.1488644

==========

P=25000.0 kPa

Tsat=376.9938798223608degree C

T degree C	v m3/kg	u kJ/kg	h kJ/kg	s kJ/KgK
376.99388	0.0031056	2009.9103	2087.55	4.412
370	0.0018503	1743.6741	1789.9314	3.942354
360	0.0016969	1656.2114	1698.635	3.7993363
350	0.0015988	1583.8948	1623.8646	3.6803314
340	0.0015264	1519.318	1557.4773	3.5729461
330	0.001469	1459.5601	1496.2842	3.4723316
320	0.0014215	1403.1789	1438.7159	3.3760926
310	0.001381	1349.3169	1383.8429	3.2827985
300	0.0013459	1297.4165	1331.0633	3.1915103
290	0.0013148	1247.0923	1279.9624	3.1015692
280	0.001287	1198.0654	1230.2407	3.0124872
270	0.0012619	1150.1252	1181.6731	2.9238849
260	0.0012391	1103.1074	1134.0841	2.8354543
250	0.0012181	1056.88	1087.3336	2.7469364
240	0.0011989	1011.3349	1041.3066	2.6581063
230	0.001181	966.38189	995.90782	2.568764
220	0.0011645	921.94486	951.05678	2.4787275
210	0.001149	877.95887	906.68505	2.3878282
200	0.0011346	834.36813	862.73396	2.2959072
190	0.0011211	791.12443	819.15289	2.2028129
180	0.0011085	748.18591	775.89796	2.1083982
170	0.0010966	705.5161	732.93102	2.0125191
160	0.0010854	663.08312	690.21878	1.9150328
150	0.0010749	620.85906	647.73215	1.8157962
140	0.0010651	578.81932	605.44557	1.7146645
130	0.0010558	536.94217	563.33651	1.6114896
120	0.0010471	495.2082	521.38494	1.5061186
110	0.0010389	453.59989	479.57287	1.3983919
100	0.0010313	412.10115	437.88391	1.2881417
90	0.0010242	370.69697	396.30287	1.1751894
80	0.0010177	329.37304	354.81545	1.0593443
70	0.0010117	288.11561	313.40807	0.9404012
60	0.0010063	246.91148	272.06791	0.8181388
50	0.0010014	205.7484	230.78332	0.6923194
40	9.97E-04	164.61574	189.54461	0.5626887
30	9.94E-04	123.50563	148.34528	0.4289784
20	9.91E-04	82.413707	107.183	0.2909068
10	9.89E-04	41.33726	66.058192	0.1481727

==========

P=30000.0 kPa

Tsat=377.0148624313849degree C

T degree C	v m3/kg	u kJ/kg	h kJ/kg	s kJ/KgK
377.01486	0.0031056	1994.3823	2087.55	4.412
370	0.001727	1698.3796	1750.1911	3.8667152
360	0.0016277	1626.7368	1675.5674	3.7497953
350	0.0015529	1562.2103	1608.7975	3.6435174
340	0.0014932	1502.2714	1547.0671	3.5436611
330	0.0014436	1445.6239	1488.9326	3.4480736
320	0.0014014	1391.4737	1433.515	3.3554286
310	0.0013646	1339.2886	1380.2277	3.2648295
300	0.0013322	1288.6948	1328.6604	3.1756376
290	0.0013032	1239.4186	1278.5141	3.0873762
280	0.001277	1191.2521	1229.5622	2.9996729
270	0.0012532	1144.0316	1181.6273	2.9122245
260	0.0012314	1097.6251	1134.5666	2.8247754
250	0.0012113	1051.9235	1088.2629	2.7371033
240	0.0011927	1006.8357	1042.6181	2.6490107
230	0.0011755	962.28404	997.54951	2.560318
220	0.0011595	918.20234	952.98621	2.470859
210	0.0011445	874.53335	908.86734	2.3804776
200	0.0011304	831.2274	865.14017	2.2890249
190	0.0011173	788.24123	821.75884	2.1963571
180	0.0011049	745.53704	778.68338	2.1023341
170	0.0010932	703.0817	735.87884	2.0068173

160	0.0010823	660.84613	693.31461	1.9096688	
150	0.001072	618.8048	650.96385	1.8107495	
140	0.0010623	576.93514	608.80295	1.7099183	
130	0.0010531	535.21718	566.81105	1.6070305	
120	0.0010446	493.63308	524.96964	1.5019361	
110	0.0010365	452.16671	483.26208	1.3944787	
100	0.001029	410.80328	441.67322	1.2844931	
90	0.001022	369.52895	400.18906	1.1718041	
80	0.0010155	328.3306	358.7965	1.0562238	
70	0.0010096	287.19566	317.48317	0.9375509	
60	0.0010042	246.1122	276.23756	0.8155681	
50	9.99E-04	205.0694	235.04956	0.6900431	
40	9.95E-04	164.05846	193.91143	0.5607286	
30	9.92E-04	123.07404	152.81937	0.4273665	
20	9.89E-04	82.115683	111.77523	0.2896913	
10	9.87E-04	41.187479	70.786626	0.1474302	

Thermodynamic properties of ice-vapor and ice-liquid interface

// IAPWS R10-06(2009)
// The International Association for the Properties of water and Steam,
// Doorwerth, The netherlands, September 2009
// Revised release on the Equation of State 2006 for H2O Ice Ih
// Dr. Daniel G. Friend Thermophysical Properties Division
// National institute of Standarts and Technology 325 broadway Boulder Colarado 80305, USA

// Programmed By Dr. Mustafa Turhan ÇOBAN
// Ege university, School of Engineering, Department of Mechanical Engineeri
program: ice_IAPWSR10_06 and steam_IAPWS95

Saturated solid(Water ice Ih) vapor_interface

Temp.	Temp.	Pressure	Spec.	Spec.	Int. En.	Int. En.	Enthalpy	Enthalpy	Entropy	Entropy
degree K	degre	kPa	m^3/kg	m^3/kg	kJ/kg	kJ/kg	kJ/kg	kJ/kg	kJ/kgK	kJ/kgK
			Sat. Solid	Sat. Vapor	Sat. Solid	Sat. Vapor	Sat. Solid	Sat. Vapor	Sat. Solid	Sat. Vapor
273.15	0	0.611154	0.00109	206.132	-333.466	2374.90	-333.465	2500.89	-1.22077	9.1557596
272.15	-1	0.562693	0.00109	220.927	-335.559	2373.52	-335.558	2499.06	-1.22845	9.1826804
271.15	-2	0.517759	0.00109	236.923	-337.645	2372.15	-337.644	2497.22	-1.23613	9.2098536
270.15	-3	0.47612	0.00109	254.229	-339.723	2370.77	-339.722	2495.39	-1.2438	9.2372822
269.15	-4	0.437557	0.00109	272.964	-341.794	2369.39	-341.793	2493.55	-1.25148	9.2649694
268.15	-5	0.401864	0.00109	293.258	-343.857	2368.01	-343.857	2491.71	-1.25917	9.2929182
267.15	-6	0.368847	0.00109	315.256	-345.914	2366.63	-345.913	2489.87	-1.26685	9.3211319
266.15	-7	0.338325	0.00109	339.116	-347.962	2365.25	-347.962	2488.03	-1.27453	9.3496137
265.15	-8	0.310127	0.00108	365.012	-350.004	2363.87	-350.003	2486.2	-1.28222	9.3783668
264.15	-9	0.284092	0.00108	393.137	-352.038	2362.49	-352.038	2484.36	-1.2899	9.4073946
263.15	-10	0.260069	0.00108	423.704	-354.065	2361.11	-354.064	2482.52	-1.29759	9.4367005
262.15	-11	0.237917	0.00108	456.946	-356.084	2359.73	-356.084	2480.67	-1.30528	9.4662879
261.15	-12	0.217503	0.00108	493.124	-358.096	2358.35	-358.096	2478.83	-1.31297	9.4961602
260.15	-13	0.198704	0.00108	532.522	-360.101	2356.96	-360.101	2476.99	-1.32066	9.5263210
259.15	-14	0.181403	0.00108	575.46	-362.098	2355.58	-362.098	2475.15	-1.32835	9.5567738
258.15	-15	0.165492	0.00108	622.286	-364.089	2354.20	-364.088	2473.31	-1.33605	9.5875224
257.15	-16	0.150868	0.00108	673.392	-366.071	2352.82	-366.071	2471.46	-1.34374	9.6185703
256.15	-17	0.137438	0.00108	729.207	-368.047	2351.43	-368.047	2469.62	-1.35144	9.6499214
255.15	-18	0.125111	0.00108	790.212	-370.015	2350.05	-370.015	2467.78	-1.35914	9.6815795
254.15	-19	0.113806	0.00108	856.938	-371.976	2348.66	-371.976	2465.93	-1.36684	9.7135484
253.15	-20	0.103445	0.00108	929.977	-373.929	2347.28	-373.929	2464.09	-1.37454	9.7458320
252.15	-21	0.093956	0.00108	1009.98	-375.875	2345.9	-375.875	2462.24	-1.38224	9.7784343
251.15	-22	0.085272	0.00108	1097.70	-377.814	2344.51	-377.814	2460.40	-1.38995	9.8113594
250.15	-23	0.077331	0.00108	1193.94	-379.746	2343.13	-379.746	2458.55	-1.39765	9.8446115
249.15	-24	0.070074	0.00108	1299.61	-381.67	2341.74	-381.67	2456.71	-1.40536	9.8781946
248.15	-25	0.063448	0.00108	1415.73	-383.587	2340.35	-383.587	2454.86	-1.41307	9.9121130
247.15	-26	0.057403	0.00108	1543.44	-385.497	2338.97	-385.497	2453.02	-1.42078	9.9463710
246.15	-27	0.051891	0.00108	1684.01	-387.399	2337.58	-387.399	2451.17	-1.4285	9.9809731
245.15	-28	0.046869	0.00108	1838.86	-389.294	2336.20	-389.294	2449.32	-1.43621	10.015923
244.15	-29	0.042299	0.00108	2009.60	-391.182	2334.81	-391.182	2447.48	-1.44393	10.051227
243.15	-30	0.038142	0.00108	2198.00	-393.063	2333.42	-393.063	2445.63	-1.45165	10.086888
242.15	-31	0.034364	0.00108	2406.07	-394.936	2332.04	-394.936	2443.78	-1.45937	10.122911
241.15	-32	0.030933	0.00108	2636.08	-396.802	2330.65	-396.802	2441.93	-1.46709	10.159302
240.15	-33	0.027821	0.00108	2890.56	-398.661	2329.26	-398.661	2440.08	-1.47481	10.196064
239.15	-34	0.024999	0.00108	3172.36	-400.513	2327.87	-400.513	2438.24	-1.48254	10.233203
238.15	-35	0.022444	0.00108	3484.70	-402.357	2326.49	-402.357	2436.39	-1.49027	10.270724
237.15	-36	0.020131	0.00108	3831.21	-404.194	2325.10	-404.194	2434.54	-1.498	10.308632
236.15	-37	0.01804	0.00108	4215.98	-406.024	2323.71	-406.024	2432.69	-1.50573	10.346932
235.15	-38	0.016152	0.00108	4643.65	-407.846	2322.32	-407.846	2430.84	-1.51346	10.385629
234.15	-39	0.014447	0.00108	5119.44	-409.662	2320.93	-409.662	2428.99	-1.5212	10.424728
233.15	-40	0.01291	0.00108	5649.29	-411.47	2319.55	-411.47	2427.14	-1.52894	10.464236
232.15	-41	0.011525	0.00108	6239.92	-413.271	2318.16	-413.271	2425.29	-1.53668	10.504157
231.15	-42	0.010279	0.00108	6898.95	-415.065	2316.77	-415.065	2423.44	-1.54442	10.544497
230.15	-43	0.009158	0.00108	7635.04	-416.851	2315.38	-416.851	2421.59	-1.55217	10.585262
229.15	-44	0.008151	0.00108	8458.04	-418.63	2313.99	-418.63	2419.74	-1.55992	10.626458
228.15	-45	0.007248	0.00108	9379.16	-420.402	2312.60	-420.402	2417.89	-1.56767	10.668090
227.15	-46	0.006438	0.00108	10411.1	-422.167	2311.21	-422.167	2416.04	-1.57542	10.710165

226.15	-47	0.005712	0.00108	11568.7	-423.925	2309.82	-423.925	2414.19	-1.58317	10.752688
225.15	-48	0.005063	0.00108	12868.3	-425.676	2308.43	-425.676	2412.34	-1.59093	10.795667
224.15	-49	0.004483	0.00108	14329.1	-427.419	2307.05	-427.419	2410.49	-1.59869	10.839106
223.15	-50	0.003965	0.00108	15972.9	-429.155	2305.66	-429.155	2408.64	-1.60645	10.883014

Saturated_solid (Water ice Ih) liquid_interface

Temp.	Temp.	Pressure	Spec.	Spec.	Int. En.	Int. En.	Enthalpy	Enthalpy	Entropy	Entropy
degree K	degre	kPa	m^3/kg	m^3/kg	kJ/kg	kJ/kg	kJ/kg	kJ/kg	kJ/kgK	kJ/kgK
			Sat. Solid	Sat. Vapor	Sat. Solid	Sat. Vapor	Sat. Solid	Sat. Vapor	Sat. Solid	Sat. Vapor
273.15	0	136.2137	0.00109	0.001	-333.472	-0.0422	-333.324	-0.04159	-1.22079	-1.54E-04
272.15	-1	13386.45	0.00108	0.001	-336.176	-4.26393	-321.598	-4.26336	-1.23076	-0.0156386
271.15	-2	26054.64	0.00108	0.001	-338.806	-8.48941	-310.479	-8.48888	-1.24057	-0.0311935
270.15	-3	38185.96	0.00108	0.001	-341.371	-12.7189	-299.919	-12.7185	-1.25024	-0.0468208
269.15	-4	49822.73	0.00108	0.00100	-343.876	-16.9529	-289.873	-16.9524	-1.2598	-0.0625223
268.15	-5	61004.63	0.00108	0.00100	-346.328	-21.1915	-280.297	-21.1911	-1.26924	-0.0783
267.15	-6	71768.85	0.00108	0.00100	-348.729	-25.4353	-271.153	-25.4349	-1.27857	-0.0941557
266.15	-7	82150.26	0.00108	0.00100	-351.086	-29.6846	-262.404	-29.6842	-1.28782	-0.1100916
265.15	-8	92181.54	0.00107	0.00100	-353.401	-33.94	-254.015	-33.9396	-1.29698	-0.1261103
264.15	-9	101893.3	0.00107	0.00100	-355.678	-38.2019	-245.954	-38.2015	-1.30606	-0.1422142
263.15	-10	111314.3	0.00107	0.00100	-357.92	-42.4709	-238.19	-42.4706	-1.31507	-0.1584063
262.15	-11	120471.4	0.00107	0.00100	-360.129	-46.7477	-230.695	-46.7474	-1.32401	-0.1746896
261.15	-12	129390	0.00107	0.00100	-362.306	-51.033	-223.443	-51.0327	-1.33289	-0.1910675
260.15	-13	138093.7	0.00107	0.00100	-364.455	-55.3275	-216.408	-55.3273	-1.34172	-0.2075439
259.15	-14	146604.6	0.00107	0.00100	-366.577	-59.6323	-209.568	-59.6321	-1.3505	-0.224123
258.15	-15	154943.8	0.00107	0.00100	-368.674	-63.9482	-202.9	-63.948	-1.35923	-0.2408095
257.15	-16	163130.8	0.00106	0.00100	-370.746	-68.2765	-196.384	-68.2763	-1.36792	-0.2576086
256.15	-17	171183.9	0.00106	0.00100	-372.794	-72.6184	-190.001	-72.6183	-1.37658	-0.2745263
255.15	-18	179120.5	0.00106	0.00100	-374.821	-76.9754	-183.733	-76.9753	-1.38519	-0.2915692
254.15	-19	186956.8	0.00106	0.00100	-376.826	-81.3492	-177.563	-81.3491	-1.39378	-0.3087448
253.15	-20	194708.1	0.00106	0.00100	-378.811	-85.7416	-171.475	-85.7415	-1.40233	-0.3260618
252.15	-21	202388.8	0.00106	0.00100	-380.775	-90.1549	-165.454	-90.1548	-1.41086	-0.3435299

APPENDIX III

Thermodynamic properties of air as perfect gas

Proporties of air as an ideal gas

Program: Air_PG

Reference Pressure: 101.325 kPa

Reference Temperature: 273.15 K

T deg K	h kJ/kg	u kJ/kg	s0 kJ/kg	Pr	Vr
200	-73.5749	-52.5773	-0.31352	0.962994	0.760335
210	-63.5097	-45.3826	-0.26441	0.968699	0.793651
220	-53.4485	-38.1918	-0.21761	0.974167	0.826776
230	-43.3901	-31.004	-0.1729	0.97942	0.859721
240	-33.3338	-23.8181	-0.1301	0.984475	0.892495
250	-23.2785	-16.6333	-0.08905	0.989347	0.925103
260	-13.2234	-9.4487	-0.04961	0.994051	0.957555
270	-3.16777	-2.26356	-0.01166	0.998598	0.989856
280	6.889204	4.922917	0.02491	1.003	1.022011
290	16.94825	12.11147	0.060209	1.007268	1.054027
300	27.01008	19.30281	0.09432	1.011409	1.085909
310	37.07556	26.4978	0.127324	1.015431	1.117661
320	47.14549	33.69724	0.159295	1.019343	1.149286
330	57.22068	40.90194	0.190298	1.023151	1.18079
340	67.30198	48.11275	0.220394	1.026862	1.212176
350	77.39021	55.33048	0.249637	1.03048	1.243448
360	87.48621	62.55599	0.278078	1.03401	1.274607
370	97.59081	69.7901	0.305764	1.037459	1.305658
380	107.7049	77.03366	0.332736	1.04083	1.336603
390	117.8292	84.28752	0.359034	1.044128	1.367445
400	127.9647	91.5525	0.384695	1.047355	1.398186
410	138.1121	98.82945	0.409752	1.050516	1.428828
420	148.2723	106.1192	0.434235	1.053614	1.459374
430	158.4461	113.4225	0.458175	1.056652	1.489825
440	168.6342	120.7401	0.481597	1.059633	1.520184
450	178.8375	128.0729	0.504526	1.062559	1.550452
460	189.0566	135.4215	0.526987	1.065433	1.580631
470	199.2922	142.7866	0.549	1.068258	1.610722
480	209.5451	150.169	0.570585	1.071035	1.640728
490	219.8158	157.5692	0.591763	1.073766	1.670649
500	230.105	164.9879	0.612549	1.076454	1.700487
510	240.4132	172.4256	0.632962	1.0791	1.730244
520	250.7411	179.883	0.653017	1.081706	1.75992
530	261.0891	187.3605	0.672728	1.084273	1.789517
540	271.4576	194.8586	0.692109	1.086804	1.819036

550	281.8473	202.3777	0.711173	1.089299	1.848479
560	292.2584	209.9184	0.729932	1.091759	1.877846
570	302.6914	217.4809	0.748398	1.094186	1.907139
580	313.1466	225.0656	0.766582	1.096582	1.936358
590	323.6243	232.6728	0.784493	1.098947	1.965505
600	334.1248	240.3028	0.802141	1.101282	1.99458
610	344.6484	247.9559	0.819536	1.103588	2.023586
620	355.1952	255.6322	0.836685	1.105867	2.052521
630	365.7655	263.332	0.853598	1.108119	2.081388
640	376.3594	271.0554	0.870282	1.110344	2.110188
650	386.9771	278.8026	0.886743	1.112545	2.13892
660	397.6186	286.5737	0.90299	1.114721	2.167587
670	408.2841	294.3687	0.919029	1.116873	2.196189
680	418.9737	302.1877	0.934866	1.119003	2.224726
690	429.6872	310.0308	0.950506	1.12111	2.2532
700	440.4249	317.898	0.965956	1.123195	2.281612
710	451.1866	325.7892	0.981221	1.125259	2.309961
720	461.9724	333.7045	0.996306	1.127302	2.33825
730	472.7821	341.6437	1.011217	1.129326	2.366478
740	483.6158	349.6069	1.025956	1.13133	2.394646
750	494.4734	357.594	1.040531	1.133314	2.422756
760	505.3547	365.6048	1.054943	1.135281	2.450808
770	516.2597	373.6393	1.069198	1.137229	2.478801
780	527.1883	381.6974	1.0833	1.139159	2.506739
790	538.1402	389.7789	1.097251	1.141072	2.53462
800	549.1155	397.8836	1.111057	1.142968	2.562445
810	560.1138	406.0115	1.12472	1.144848	2.590216
820	571.1352	414.1624	1.138243	1.146712	2.617932
830	582.1793	422.336	1.15163	1.148559	2.645595
840	593.246	430.5323	1.164884	1.150392	2.673205
850	604.3352	438.7509	1.178007	1.152209	2.700763
860	615.4466	446.9918	1.191003	1.154011	2.728269
870	626.5801	455.2548	1.203874	1.155799	2.755724
880	637.7353	463.5396	1.216623	1.157573	2.783128
890	648.9122	471.846	1.229252	1.159332	2.810482
900	660.1105	480.1738	1.241765	1.161078	2.837787
910	671.33	488.5228	1.254162	1.162811	2.865043
920	682.5705	496.8928	1.266447	1.16453	2.892251
930	693.8317	505.2835	1.278621	1.166236	2.91941
940	705.1134	513.6947	1.290687	1.16793	2.946523
950	716.4154	522.1262	1.302647	1.169611	2.973588
960	727.7375	530.5778	1.314503	1.17128	3.000607
970	739.0794	539.0493	1.326256	1.172937	3.027581
980	750.441	547.5403	1.337909	1.174582	3.054509
990	761.8218	556.0507	1.349463	1.176216	3.081393
1000	773.2219	564.5802	1.360921	1.177838	3.108232
1010	784.6408	573.1286	1.372283	1.179448	3.135027
1020	796.0784	581.6957	1.383551	1.181048	3.161779
1030	807.5344	590.2813	1.394728	1.182637	3.188488
1040	819.0087	598.885	1.405814	1.184214	3.215154
1050	830.5009	607.5068	1.416812	1.185782	3.241778
1060	842.0109	616.1463	1.427722	1.187339	3.268361
1070	853.5384	624.8033	1.438546	1.188886	3.294902
1080	865.0831	633.4776	1.449285	1.190422	3.321403
1090	876.645	642.1689	1.459941	1.191949	3.347863
1100	888.2237	650.8771	1.470516	1.193466	3.374283
1110	899.819	659.602	1.481009	1.194973	3.400664
1120	911.4308	668.3432	1.491423	1.196471	3.427006
1130	923.0587	677.1006	1.501759	1.197959	3.453308
1140	934.7025	685.874	1.512018	1.199438	3.479573
1150	946.3622	694.6631	1.522201	1.200908	3.505799
1160	958.0374	703.4678	1.53231	1.202369	3.531987
1170	969.7279	712.2879	1.542345	1.203821	3.558139
1180	981.4335	721.123	1.552307	1.205264	3.584253
1190	993.1541	729.9731	1.562198	1.206699	3.610331
1200	1004.889	738.8379	1.572018	1.208125	3.636372
1210	1016.639	747.7172	1.581769	1.209542	3.662377
1220	1028.403	756.6108	1.591452	1.210952	3.688347
1230	1040.182	765.5186	1.601067	1.212353	3.714282
1240	1051.974	774.4403	1.610615	1.213746	3.740182
1250	1063.78	783.3758	1.620098	1.215131	3.766047
1260	1075.599	792.3248	1.629516	1.216508	3.791878
1270	1087.432	801.2872	1.63887	1.217878	3.817675
1280	1099.278	810.2628	1.648161	1.219239	3.843438
1290	1111.137	819.2514	1.65739	1.220593	3.869167
1300	1123.009	828.2528	1.666557	1.22194	3.894864
1310	1134.894	837.2669	1.675664	1.223279	3.920528
1320	1146.791	846.2935	1.684712	1.224611	3.94616
1330	1158.7	855.3324	1.6937	1.225935	3.971759
1340	1170.622	864.3835	1.70263	1.227253	3.997326
1350	1182.555	873.4466	1.711502	1.228563	4.022862
1360	1194.501	882.5214	1.720318	1.229866	4.048366

1370	1206.458	891.608	1.729078	1.231163	4.073839
1380	1218.426	900.7061	1.737782	1.232452	4.099281
1390	1230.406	909.8155	1.746432	1.233735	4.124693
1400	1242.397	918.9361	1.755028	1.235011	4.150074
1410	1254.4	928.0678	1.763571	1.236281	4.175426
1420	1266.413	937.2104	1.77206	1.237544	4.200747
1430	1278.437	946.3638	1.780498	1.2388	4.226039
1440	1290.471	955.5277	1.788885	1.240051	4.251301
1450	1302.516	964.7022	1.79722	1.241294	4.276535
1460	1314.571	973.887	1.805506	1.242532	4.301739
1470	1326.637	983.082	1.813742	1.243763	4.326915
1480	1338.712	992.2871	1.821929	1.244989	4.352063
1490	1350.798	1001.502	1.830067	1.246208	4.377182
1500	1362.893	1010.727	1.838157	1.247421	4.402273
1510	1374.998	1019.961	1.846201	1.248628	4.427337
1520	1387.113	1029.206	1.854197	1.24983	4.452373
1530	1399.237	1038.459	1.862147	1.251025	4.477381
1540	1411.37	1047.722	1.870052	1.252215	4.502363
1550	1423.513	1056.994	1.877911	1.2534	4.527318
1560	1435.664	1066.275	1.885726	1.254578	4.552246
1570	1447.825	1075.565	1.893496	1.255751	4.577147
1580	1459.994	1084.864	1.901223	1.256919	4.602022
1590	1472.172	1094.172	1.908906	1.258081	4.626871
1600	1484.359	1103.488	1.916547	1.259237	4.651694
1610	1496.554	1112.813	1.924145	1.260389	4.676492
1620	1508.758	1122.146	1.931701	1.261535	4.701264
1630	1520.97	1131.488	1.939217	1.262675	4.72601
1640	1533.19	1140.837	1.946691	1.263811	4.750732
1650	1545.419	1150.195	1.954124	1.264941	4.775428
1660	1557.655	1159.561	1.961518	1.266067	4.8001
1670	1569.899	1168.935	1.968872	1.267187	4.824747
1680	1582.152	1178.316	1.976187	1.268302	4.84937
1690	1594.411	1187.706	1.983463	1.269413	4.873968
1700	1606.679	1197.103	1.9907	1.270518	4.898542
1710	1618.954	1206.507	1.997899	1.271619	4.923092
1720	1631.236	1215.919	2.005061	1.272715	4.947619
1730	1643.526	1225.338	2.012186	1.273806	4.972122
1740	1655.823	1234.765	2.019273	1.274892	4.996601
1750	1668.127	1244.199	2.026325	1.275974	5.021057
1760	1680.439	1253.64	2.03334	1.277051	5.04549
1770	1692.757	1263.088	2.040319	1.278123	5.0699
1780	1705.082	1272.542	2.047263	1.279191	5.094288
1790	1717.415	1282.004	2.054172	1.280254	5.118652
1800	1729.754	1291.473	2.061046	1.281313	5.142994
1810	1742.099	1300.948	2.067885	1.282368	5.167314
1820	1754.452	1310.43	2.074691	1.283418	5.191611
1830	1766.81	1319.918	2.081463	1.284463	5.215887
1840	1779.176	1329.413	2.088202	1.285505	5.24014
1850	1791.548	1338.914	2.094907	1.286542	5.264372
1860	1803.926	1348.422	2.10158	1.287575	5.288582
1870	1816.31	1357.936	2.108221	1.288604	5.31277
1880	1828.701	1367.456	2.114829	1.289628	5.336937
1890	1841.097	1376.982	2.121406	1.290649	5.361083
1900	1853.5	1386.514	2.127951	1.291665	5.385208
1910	1865.909	1396.053	2.134464	1.292677	5.409312
1920	1878.324	1405.597	2.140947	1.293686	5.433394
1930	1890.745	1415.147	2.1474	1.29469	5.457457
1940	1903.171	1424.703	2.153822	1.29569	5.481498
1950	1915.603	1434.265	2.160213	1.296687	5.505519
1960	1928.041	1443.832	2.166576	1.297679	5.52952
1970	1940.485	1453.405	2.172908	1.298668	5.5535
1980	1952.934	1462.984	2.179212	1.299653	5.577461
1990	1965.388	1472.568	2.185486	1.300634	5.601401
2000	1977.849	1482.158	2.191732	1.301612	5.625322

Thermodynamic properties of air as real gas

Program: air_NIST

air as a real gas				
P=100.0 kPa	P0	101.325 kPa	T0	273.15 K
T deg K	h kJ/kg	u kJ/kg	s kJ/kgK	s0 kJ/kgK
200	-73.59237832	142.415138	-0.3136	-0.3174
210	-63.52486901	149.594829	-0.26448	-0.26828
220	-53.46126823	156.772445	-0.21766	-0.22146
230	-43.40052664	163.948753	-0.17294	-0.17674
240	-33.34171366	171.124456	-0.13013	-0.13392
250	-23.28397423	178.300223	-0.08907	-0.09287
260	-13.2264986	185.476713	-0.04963	-0.05342
270	-3.168501263	192.654585	-0.01167	-0.01546

280	6.890793522	199.83451	0.024916	0.021126
290	16.95216152	207.017175	0.060223	0.056434
300	27.0163848	214.203286	0.094342	0.090554
310	37.08425536	221.393572	0.127354	0.123567
320	47.1565768	228.588779	0.159332	0.155546
330	57.2341645	235.789677	0.190343	0.186556
340	67.3178446	242.997048	0.220446	0.21666
350	77.40845221	250.211692	0.249696	0.24591
360	87.50682889	257.434419	0.278144	0.274359
370	97.61381978	264.666045	0.305836	0.302051
380	107.7302704	271.907391	0.332814	0.32903
390	117.8570232	279.159277	0.359119	0.355335
400	127.9949142	286.422521	0.384786	0.381002
410	138.1447697	293.697929	0.409849	0.406065
420	148.3074029	300.9863	0.434338	0.430555
430	158.4836105	308.288416	0.458283	0.4545
440	168.6741704	315.605043	0.481711	0.477928
450	178.8798383	322.936924	0.504646	0.500863
460	189.1013458	330.28478	0.527111	0.523329
470	199.339398	337.649307	0.549129	0.545347
480	209.5946715	345.031174	0.57072	0.566938
490	219.867813	352.431018	0.591903	0.58812
500	230.1594382	359.849449	0.612694	0.608912
510	240.4701301	367.287042	0.633112	0.62933
520	250.8004389	374.744341	0.653172	0.64939
530	261.1508809	382.221859	0.672887	0.669106
540	271.5219386	389.720071	0.692273	0.688491
550	281.9140605	397.239421	0.711342	0.70756
560	292.327661	404.78032	0.730105	0.726324
570	302.7631209	412.343145	0.748575	0.744794
580	313.2207875	419.928238	0.766763	0.762982
590	323.7009758	427.535912	0.784678	0.780897
600	334.2039683	435.166445	0.802331	0.79855
610	344.7300167	442.820087	0.81973	0.815949
620	355.279342	450.497055	0.836883	0.833102
630	365.852136	458.197539	0.8538	0.850019
640	376.448562	465.921699	0.870488	0.866707
650	387.0687562	473.66967	0.886953	0.883173
660	397.7128282	481.44156	0.903204	0.899423
670	408.3808627	489.237449	0.919247	0.915466
680	419.0729205	497.057399	0.935087	0.931306
690	429.7890395	504.901444	0.950731	0.946951
700	440.5292359	512.769599	0.966185	0.962404
710	451.2935054	520.661859	0.981453	0.977673
720	462.0818242	528.578198	0.996542	0.992762
730	472.8941503	536.518573	1.011456	1.007676
740	483.7304245	544.482924	1.026199	1.022419
750	494.5905715	552.471173	1.040777	1.036997
760	505.4745008	560.48323	1.055193	1.051413
770	516.3821077	568.518988	1.069451	1.065671
780	527.3132747	576.578329	1.083556	1.079776
790	538.2678718	584.661121	1.097511	1.093731
800	549.2457579	592.767224	1.11132	1.10754
810	560.2467814	600.896484	1.124986	1.121206
820	571.2707812	609.04874	1.138513	1.134733
830	582.3175873	617.223821	1.151903	1.148123
840	593.3870219	625.421547	1.16516	1.16138
850	604.4788996	633.641734	1.178286	1.174506
860	615.5930287	641.884189	1.191286	1.187506
870	626.7292114	650.148712	1.20416	1.20038
880	637.8872447	658.435102	1.216912	1.213132
890	649.0669207	666.743149	1.229545	1.225765
900	660.2680273	675.07264	1.24206	1.23828
910	671.4903491	683.42336	1.25446	1.25068
920	682.7336672	691.795089	1.266748	1.262968
930	693.9977602	700.187606	1.278926	1.275146
940	705.2824042	708.600686	1.290995	1.287215
950	716.5873738	717.034103	1.302958	1.299178
960	727.912442	725.487629	1.314817	1.311037
970	739.2573806	733.961037	1.326573	1.322793
980	750.6219608	742.454097	1.338229	1.33445
990	762.0059533	750.96658	1.349787	1.346007
1000	773.4091287	759.498254	1.361247	1.357468
1010	784.8312576	768.048892	1.372613	1.368833
1020	796.2721112	776.618264	1.383885	1.380105
1030	807.7314612	785.20614	1.395065	1.391285
1040	819.2090799	793.812294	1.406154	1.402374
1050	830.704741	802.436499	1.417155	1.413375
1060	842.2182192	811.078528	1.428068	1.424288
1070	853.7492907	819.738159	1.438895	1.435116
1080	865.2977331	828.415168	1.449638	1.445859
1090	876.8633258	837.109334	1.460298	1.456518

1100	888.4458499	845.820439	1.470876	1.467096
1110	900.0450885	854.548265	1.481373	1.477593
1120	911.6608267	863.292597	1.49179	1.488011
1130	923.2928518	872.053222	1.50213	1.49835
1140	934.9409532	880.82993	1.512393	1.508613
1150	946.6049227	889.622512	1.52258	1.5188
1160	958.2845543	898.430761	1.532692	1.528912
1170	969.9796445	907.254475	1.542731	1.538951
1180	981.6899924	916.093452	1.552697	1.548917
1190	993.4153994	924.947494	1.562592	1.558812
1200	1005.15567	933.816403	1.572416	1.568637
1210	1016.91061	942.699988	1.582172	1.578392
1220	1028.680029	951.598056	1.591858	1.588079
1230	1040.463739	960.51042	1.601478	1.597698
1240	1052.261554	969.436894	1.611031	1.607251
1250	1064.073292	978.377295	1.620518	1.616739
1260	1075.898773	987.331443	1.629941	1.626161
1270	1087.737818	996.29916	1.6393	1.63552
1280	1099.590254	1005.28027	1.648596	1.644816
1290	1111.455908	1014.2746	1.65783	1.65405
1300	1123.33461	1023.28199	1.667003	1.663223
1310	1135.226194	1032.30226	1.676115	1.672336
1320	1147.130496	1041.33525	1.685168	1.681388
1330	1159.047353	1050.38081	1.694162	1.690382
1340	1170.976608	1059.43876	1.703098	1.699318
1350	1182.918103	1068.50895	1.711976	1.708197
1360	1194.871685	1077.59124	1.720798	1.717018
1370	1206.837202	1086.68547	1.729564	1.725784
1380	1218.814506	1095.79148	1.738275	1.734495
1390	1230.80345	1104.90914	1.746931	1.743152
1400	1242.803891	1114.0383	1.755533	1.751754
1410	1254.815688	1123.17881	1.764083	1.760303
1420	1266.8387	1132.33055	1.77258	1.7688
1430	1278.872793	1141.49336	1.781025	1.777245
1440	1290.91783	1150.66713	1.789418	1.785639
1450	1302.973681	1159.85171	1.797762	1.793982
1460	1315.040216	1169.04698	1.806055	1.802275
1470	1327.117307	1178.25281	1.814299	1.810519
1480	1339.20483	1187.46907	1.822494	1.818714
1490	1351.302661	1196.69564	1.83064	1.826861
1500	1363.410679	1205.9324	1.838739	1.83496
1510	1375.528765	1215.17923	1.846791	1.843012
1520	1387.656804	1224.43601	1.854797	1.851017
1530	1399.79468	1233.70264	1.862756	1.858976
1540	1411.94228	1242.97899	1.87067	1.86689
1550	1424.099495	1252.26496	1.878538	1.874759
1560	1436.266215	1261.56043	1.886363	1.882583
1570	1448.442334	1270.86531	1.894143	1.890364
1580	1460.627747	1280.17948	1.90188	1.8981
1590	1472.82235	1289.50284	1.909574	1.905794
1600	1485.026043	1298.8353	1.917225	1.913445
1610	1497.238726	1308.17675	1.924834	1.921055
1620	1509.460302	1317.52709	1.932402	1.928622
1630	1521.690673	1326.88623	1.939928	1.936149
1640	1533.929748	1336.25407	1.947414	1.943634
1650	1546.177431	1345.63052	1.954859	1.95108
1660	1558.433633	1355.0155	1.962265	1.958485
1670	1570.698265	1364.4089	1.969631	1.965851
1680	1582.971237	1373.81065	1.976958	1.973179
1690	1595.252464	1383.22066	1.984247	1.980467
1700	1607.541862	1392.63883	1.991497	1.987718
1710	1619.839346	1402.06509	1.99871	1.99493
1720	1632.144834	1411.49936	2.005885	2.002105
1730	1644.458247	1420.94156	2.013023	2.009244
1740	1656.779505	1430.3916	2.020125	2.016345
1750	1669.108531	1439.84941	2.02719	2.023411
1760	1681.445247	1449.31491	2.034219	2.03044
1770	1693.789579	1458.78803	2.041213	2.037434
1780	1706.141453	1468.2687	2.048172	2.044393
1790	1718.500796	1477.75683	2.055096	2.051317
1800	1730.867537	1487.25236	2.061986	2.058206
1810	1743.241605	1496.75522	2.068841	2.065062
1820	1755.622932	1506.26534	2.075663	2.071884
1830	1768.01145	1515.78265	2.082451	2.078672
1840	1780.407092	1525.30709	2.089206	2.085427
1850	1792.809792	1534.83859	2.095929	2.092149
1860	1805.219486	1544.37708	2.102619	2.098839
1870	1817.636111	1553.9225	2.109276	2.105497
1880	1830.059603	1563.47479	2.115902	2.112123
1890	1842.489902	1573.03389	2.122497	2.118717
1900	1854.926946	1582.59974	2.12906	2.12528
1910	1867.370678	1592.17227	2.135592	2.131812

1920	1879.821037	1601.75143	2.142093	2.138314
1930	1892.277967	1611.33717	2.148564	2.144785
1940	1904.74141	1620.92941	2.155005	2.151226
1950	1917.211312	1630.52812	2.161417	2.157637
1960	1929.687617	1640.13323	2.167799	2.164019
1970	1942.17027	1649.74469	2.174151	2.170372
1980	1954.65922	1659.36245	2.180475	2.176695
1990	1967.154413	1668.98645	2.186769	2.18299
2000	1979.655799	1678.61665	2.193036	2.189256
		air as a real gas		
P=200.0 kPa	P0	101.325 kPa	T0	273.15 K
T deg K	h kJ/kg	u kJ/kg	s kJ/kgK	s0 kJ/kgK
200	-74.08190018	142.060783	-0.51438	-0.3174
210	-63.97322387	149.264895	-0.46506	-0.26828
220	-53.87332887	156.463931	-0.41808	-0.22146
230	-43.78034116	163.659185	-0.37321	-0.17674
240	-33.69268321	170.851773	-0.33028	-0.13392
250	-23.60898517	178.042689	-0.28912	-0.09287
260	-13.52802241	185.232852	-0.24958	-0.05342
270	-3.448671306	192.42313	-0.21154	-0.01546
280	6.630122378	199.614366	-0.17488	0.021126
290	16.70936592	206.807387	-0.13951	0.056434
300	26.79003549	214.003019	-0.10534	0.090554
310	36.87308713	221.202087	-0.07228	0.123567
320	46.95946404	228.405424	-0.04025	0.155546
330	57.05010121	235.613868	-0.0092	0.186556
340	67.14592787	242.828264	0.020936	0.21666
350	77.24786848	250.049466	0.050219	0.24591
360	87.35684248	257.278326	0.078696	0.274359
370	97.47376315	264.515704	0.106416	0.302051
380	107.5995359	271.762454	0.133419	0.32903
390	117.7350563	279.019426	0.159747	0.355335
400	127.8812072	286.287466	0.185435	0.381002
410	138.0388567	293.567406	0.210516	0.406065
420	148.2088552	300.860063	0.235024	0.430555
430	158.392033	308.166241	0.258985	0.4545
440	168.5891978	315.486722	0.282428	0.477928
450	178.8011326	322.822265	0.305377	0.500863
460	189.0285933	330.173605	0.327856	0.523329
470	199.2723072	337.54145	0.349886	0.545347
480	209.532971	344.92648	0.371488	0.566938
490	219.8112499	352.329345	0.392681	0.58812
500	230.107776	359.750661	0.413483	0.608912
510	240.4231478	367.191014	0.43391	0.62933
520	250.7579294	374.650957	0.453978	0.64939
530	261.11265	382.131006	0.473702	0.669106
540	271.487804	389.631647	0.493095	0.688491
550	281.8838506	397.153329	0.512171	0.70756
560	292.3012143	404.696467	0.530941	0.726324
570	302.7402852	412.261444	0.549418	0.744794
580	313.2014192	419.848609	0.567612	0.762982
590	323.6849392	427.458276	0.585533	0.780897
600	334.1911352	435.09073	0.60319	0.79855
610	344.7202656	442.746223	0.620594	0.815949
620	355.272558	450.424978	0.637753	0.833102
630	365.84821	458.127185	0.654674	0.850019
640	376.4473906	465.85301	0.671366	0.866707
650	387.0702409	473.60259	0.687836	0.883173
660	397.7168757	481.376034	0.704091	0.899423
670	408.3873841	489.173428	0.720137	0.915466
680	419.0818311	496.994833	0.735981	0.931306
690	429.8002587	504.840287	0.751628	0.946951
700	440.5426867	512.709808	0.767085	0.962404
710	451.3091144	520.603391	0.782357	0.977673
720	462.0995215	528.521013	0.797449	0.992762
730	472.9138689	536.462633	0.812365	1.007676
740	483.7521004	544.428191	0.827111	1.022419
750	494.6141435	552.417612	0.841691	1.036997
760	505.4999103	560.430807	0.85611	1.051413
770	516.4092986	568.467672	0.87037	1.065671
780	527.3421932	576.528088	0.884478	1.079776
790	538.2984664	584.611926	0.898435	1.093731
800	549.277979	592.719045	0.912246	1.10754
810	560.2805817	600.849295	0.925914	1.121206
820	571.306115	609.002513	0.939442	1.134733
830	582.3544109	617.178531	0.952834	1.148123
840	593.4252931	625.377171	0.966093	1.16138
850	604.518578	633.598247	0.979221	1.174506
860	615.6340753	641.841568	0.992222	1.187506

870	626.7715888	650.106937	1.005098	1.20038
880	637.9309167	658.39415	1.017851	1.213132
890	649.1118526	666.703	1.030485	1.225765
900	660.3141856	675.033276	1.043002	1.23828
910	671.5377014	683.384761	1.055404	1.25068
920	682.7821824	691.757237	1.067693	1.262968
930	694.0474081	700.150483	1.079871	1.275146
940	705.333156	708.564274	1.091942	1.287215
950	716.6392013	716.998387	1.103906	1.299178
960	727.9653182	725.452593	1.115766	1.311037
970	739.3112793	733.926665	1.127523	1.322793
980	750.6768567	742.420375	1.139181	1.33445
990	762.061822	750.933492	1.150739	1.346007
1000	773.4659465	759.465788	1.1622	1.357468
1010	784.8890017	768.017033	1.173567	1.368833
1020	796.3307594	776.586999	1.18484	1.380105
1030	807.790992	785.175457	1.19602	1.391285
1040	819.2694726	793.782181	1.207111	1.402374
1050	830.7659755	802.406942	1.218112	1.413375
1060	842.280276	811.049518	1.229026	1.424288
1070	853.8121508	819.709683	1.239854	1.435116
1080	865.3613782	828.387215	1.250598	1.445859
1090	876.9277382	837.081895	1.261258	1.456518
1100	888.5110123	845.793502	1.271837	1.467096
1110	900.1109842	854.52182	1.282334	1.477593
1120	911.7274395	863.266636	1.292753	1.488011
1130	923.3601659	872.027734	1.303093	1.49835
1140	935.0089532	880.804906	1.313356	1.508613
1150	946.6735938	889.597944	1.323544	1.5188
1160	958.353882	898.40664	1.333657	1.528912
1170	970.0496149	907.230792	1.343696	1.538951
1180	981.7605917	916.070199	1.353663	1.548917
1190	993.4866144	924.924663	1.363558	1.558812
1200	1005.227487	933.793986	1.373383	1.568637
1210	1016.983018	942.677978	1.383139	1.578392
1220	1028.753015	951.576445	1.392826	1.588079
1230	1040.537291	960.489201	1.402446	1.597698
1240	1052.335661	969.41606	1.412	1.607251
1250	1064.147942	978.356839	1.421487	1.616739
1260	1075.973955	987.311359	1.430911	1.626161
1270	1087.813522	996.279441	1.44027	1.63552
1280	1099.666469	1005.26091	1.449566	1.644816
1290	1111.532624	1014.2556	1.458801	1.65405
1300	1123.411818	1023.26333	1.467974	1.663223
1310	1135.303883	1032.28394	1.477087	1.672336
1320	1147.208657	1041.31727	1.48614	1.681388
1330	1159.125978	1050.36315	1.495134	1.690382
1340	1171.055686	1059.42142	1.50407	1.699318
1350	1182.997627	1068.49194	1.512949	1.708197
1360	1194.951646	1077.57454	1.521771	1.717018
1370	1206.917592	1086.66907	1.530538	1.725784
1380	1218.895316	1095.77539	1.539249	1.734495
1390	1230.884674	1104.89334	1.547905	1.743152
1400	1242.88552	1114.0228	1.556508	1.751754
1410	1254.897714	1123.1636	1.565058	1.760303
1420	1266.921116	1132.31562	1.573555	1.7688
1430	1278.955592	1141.47872	1.582	1.777245
1440	1291.001005	1150.65276	1.590394	1.785639
1450	1303.057225	1159.83761	1.598738	1.793982
1460	1315.124123	1169.03314	1.607031	1.802275
1470	1327.20157	1178.23923	1.615275	1.810519
1480	1339.289442	1187.45575	1.62347	1.818714
1490	1351.387616	1196.68258	1.631617	1.826861
1500	1363.495971	1205.91959	1.639716	1.83496
1510	1375.614389	1215.16667	1.647769	1.843012
1520	1387.742753	1224.4237	1.655774	1.851017
1530	1399.880949	1233.69056	1.663734	1.858976
1540	1412.028863	1242.96715	1.671648	1.86689
1550	1424.186387	1252.25335	1.679517	1.874759
1560	1436.35341	1261.54906	1.687341	1.882583
1570	1448.529827	1270.85416	1.695121	1.890364
1580	1460.715533	1280.16855	1.702858	1.8981
1590	1472.910424	1289.49214	1.710552	1.905794
1600	1485.1144	1298.82481	1.718204	1.913445
1610	1497.327361	1308.16647	1.725813	1.921055
1620	1509.54921	1317.51702	1.733381	1.928622
1630	1521.779851	1326.87637	1.740907	1.936149
1640	1534.01919	1336.24442	1.748393	1.943634
1650	1546.267134	1345.62107	1.755839	1.95108
1660	1558.523592	1355.00625	1.763245	1.958485
1670	1570.788474	1364.39985	1.770611	1.965851
1680	1583.061694	1373.80179	1.777938	1.973179

1690	1595.343165	1383.21199	1.785227	1.980467
1700	1607.632801	1392.63035	1.792478	1.987718
1710	1619.930521	1402.0568	1.79969	1.99493
1720	1632.236241	1411.49126	1.806866	2.002105
1730	1644.549882	1420.93363	1.814004	2.009244
1740	1656.871364	1430.38385	1.821106	2.016345
1750	1669.20061	1439.84184	1.828171	2.023411
1760	1681.537543	1449.30752	1.835201	2.03044
1770	1693.882088	1458.78081	1.842195	2.037434
1780	1706.234172	1468.26164	1.849154	2.044393
1790	1718.593722	1477.74994	1.856078	2.051317
1800	1730.960666	1487.24564	1.862968	2.058206
1810	1743.334935	1496.74866	1.869823	2.065062
1820	1755.716459	1506.25895	1.876645	2.071884
1830	1768.105171	1515.77642	1.883433	2.078672
1840	1780.501004	1525.30101	1.890189	2.085427
1850	1792.903892	1534.83266	1.896911	2.092149
1860	1805.313771	1544.37131	1.903601	2.098839
1870	1817.730578	1553.91688	1.910259	2.105497
1880	1830.15425	1563.46933	1.916885	2.112123
1890	1842.584725	1573.02857	1.923479	2.118717
1900	1855.021944	1582.59457	1.930042	2.12528
1910	1867.465846	1592.16724	1.936575	2.131812
1920	1879.916374	1601.74655	1.943076	2.138314
1930	1892.37347	1611.33242	1.949548	2.144785
1940	1904.837077	1620.92481	1.955989	2.151226
1950	1917.30714	1630.52366	1.9624	2.157637
1960	1929.783603	1640.1289	1.968782	2.164019
1970	1942.266413	1649.7405	1.975134	2.170372
1980	1954.755517	1659.35839	1.981458	2.176695
1990	1967.250861	1668.98252	1.987753	2.18299
2000	1979.752396	1678.61285	1.994019	2.189256
		air as a real gas		
P=300.0 kPa	P0	101.325 kPa	T0	273.15 K
T deg K	h kJ/kg	u kJ/kg	s kJ/kgK	s0 kJ/kgK
200	-74.57247859	141.705078	-0.63258	-0.3174
210	-64.42220042	148.933949	-0.58305	-0.26828
220	-54.28569516	156.154656	-0.5359	-0.22146
230	-44.16023054	163.369049	-0.49089	-0.17674
240	-34.04355876	170.578671	-0.44783	-0.13392
250	-23.93377875	177.784852	-0.40656	-0.09287
260	-13.8292393	184.988778	-0.36693	-0.05342
270	-3.728470112	192.191533	-0.32881	-0.01546
280	6.369867769	199.394134	-0.29208	0.021126
290	16.46701777	206.597556	-0.25665	0.056434
300	26.56415364	213.802742	-0.22242	0.090554
310	36.66239807	221.01062	-0.18931	0.123567
320	46.76283585	228.222107	-0.15724	0.155546
330	56.8665231	235.438115	-0.12615	0.186556
340	66.97449331	242.65955	-0.09598	0.21666
350	77.08776119	249.887319	-0.06666	0.24591
360	87.20732475	257.122322	-0.03815	0.274359
370	97.33416597	264.365456	-0.01041	0.302051
380	107.4692507	271.617614	0.016623	0.32903
390	117.6135276	278.879676	0.042974	0.355335
400	127.7679269	286.152514	0.068682	0.381002
410	137.9333586	293.436986	0.093783	0.406065
420	148.1107106	300.733931	0.118308	0.430555
430	158.3008466	308.04417	0.142286	0.4545
440	168.5046047	315.368504	0.165744	0.477928
450	178.7227948	322.707708	0.188707	0.500863
460	188.9561973	330.06253	0.211199	0.523329
470	199.2055618	337.433692	0.233241	0.545347
480	209.4716051	344.821884	0.254855	0.566938
490	219.7550107	352.227767	0.276058	0.58812
500	230.0564276	359.651967	0.29687	0.608912
510	240.3764694	367.095079	0.317306	0.62933
520	250.7157143	374.557662	0.337383	0.64939
530	261.0747043	382.040242	0.357115	0.669106
540	271.4539456	389.543309	0.376516	0.688491
550	281.8539083	397.06732	0.395599	0.70756
560	292.275027	404.612695	0.414376	0.726324
570	302.7177009	412.179823	0.432859	0.744794
580	313.1822946	419.769057	0.451059	0.762982
590	323.669139	427.380716	0.468985	0.780897
600	334.1785313	435.015089	0.486649	0.79855
610	344.710737	442.672431	0.504058	0.815949
620	355.2659898	450.352969	0.521221	0.833102
630	365.8444935	458.056899	0.538147	0.850019
640	376.4464225	465.784387	0.554843	0.866707

650	387.0719232	473.535573	0.571317	0.883173
660	397.7211151	481.31057	0.587576	0.899423
670	408.3940919	489.109466	0.603625	0.915466
680	419.0909229	496.932325	0.619473	0.931306
690	429.811654	504.779188	0.635124	0.946951
700	440.5563088	512.650072	0.650584	0.962404
710	451.3248902	520.544977	0.665859	0.977673
720	462.1173809	528.463881	0.680953	0.992762
730	472.9337453	536.406743	0.695873	1.007676
740	483.7739299	544.373507	0.710621	1.022419
750	494.6378651	552.3641	0.725204	1.036997
760	505.5254654	560.378432	0.739625	1.051413
770	516.4366314	568.416401	0.753888	1.065671
780	527.37125	576.477891	0.767997	1.079776
790	538.3291958	584.562774	0.781957	1.093731
800	549.3103317	592.670909	0.79577	1.10754
810	560.3145102	600.802146	0.80944	1.121206
820	571.341574	608.956326	0.82297	1.134733
830	582.3913566	617.13328	0.836364	1.148123
840	593.4636835	625.332832	0.849624	1.16138
850	604.5583728	633.554796	0.862754	1.174506
860	615.6752356	641.798983	0.875756	1.187506
870	626.8140772	650.065196	0.888634	1.20038
880	637.9746972	658.353232	0.901389	1.213132
890	649.1568904	666.662885	0.914024	1.225765
900	660.3604475	674.993944	0.926542	1.23828
910	671.585155	683.346193	0.938945	1.25068
920	682.8307967	691.719415	0.951236	1.262968
930	694.097153	700.113389	0.963416	1.275146
940	705.3840025	708.527892	0.975487	1.287215
950	716.6911216	716.962699	0.987453	1.299178
960	728.0182852	725.417585	0.999314	1.311037
970	739.3652669	733.892321	1.011072	1.322793
980	750.7318397	742.386679	1.02273	1.33445
990	762.117776	750.90043	1.03429	1.346007
1000	773.5228479	759.433346	1.045752	1.357468
1010	784.9468276	767.985199	1.057119	1.368833
1020	796.3894878	776.555758	1.068393	1.380105
1030	807.8506014	785.144797	1.079575	1.391285
1040	819.3299424	793.752089	1.090666	1.402374
1050	830.8272855	802.377408	1.101668	1.413375
1060	842.3424068	811.020529	1.112583	1.424288
1070	853.8750836	819.681228	1.123412	1.435116
1080	865.4250947	828.359283	1.134156	1.445859
1090	876.9922206	837.054475	1.144817	1.456518
1100	888.5762434	845.766584	1.155396	1.467096
1110	900.1769474	854.495395	1.165895	1.477593
1120	911.7941185	863.240693	1.176314	1.488011
1130	923.4275449	872.002265	1.186655	1.49835
1140	935.0770171	880.779901	1.196919	1.508613
1150	946.7423277	889.573393	1.207107	1.5188
1160	958.4232715	898.382535	1.21722	1.528912
1170	970.1196459	907.207125	1.22726	1.538951
1180	981.8312506	916.046962	1.237227	1.548917
1190	993.557888	924.901847	1.247123	1.558812
1200	1005.299363	933.771585	1.256949	1.568637
1210	1017.055482	942.655982	1.266705	1.578392
1220	1028.826056	951.554849	1.276393	1.588079
1230	1040.610898	960.467996	1.286013	1.597698
1240	1052.409821	969.39524	1.295567	1.607251
1250	1064.222645	978.336397	1.305055	1.616739
1260	1076.04919	987.291287	1.314479	1.626161
1270	1087.889278	996.259734	1.323839	1.63552
1280	1099.742735	1005.24156	1.333135	1.644816
1290	1111.60939	1014.2366	1.34237	1.65405
1300	1123.489074	1023.24468	1.351544	1.663223
1310	1135.381621	1032.26563	1.360657	1.672336
1320	1147.286866	1041.29929	1.36971	1.681388
1330	1159.204649	1050.3455	1.378705	1.690382
1340	1171.134811	1059.4041	1.387641	1.699318
1350	1183.077197	1068.47493	1.39652	1.708197
1360	1195.031652	1077.55785	1.405343	1.717018
1370	1206.998026	1086.65269	1.41411	1.725784
1380	1218.976171	1095.75931	1.422821	1.734495
1390	1230.96594	1104.87756	1.431478	1.743152
1400	1242.96719	1114.0073	1.440081	1.751754
1410	1254.979781	1123.1484	1.448631	1.760303
1420	1267.003574	1132.3007	1.457128	1.7688
1430	1279.038431	1141.46408	1.465574	1.777245
1440	1291.08422	1150.63839	1.473968	1.785639
1450	1303.140809	1159.82352	1.482312	1.793982
1460	1315.208068	1169.01932	1.490606	1.802275

1470	1327.285871	1178.22567	1.49885	1.810519
1480	1339.374092	1187.44245	1.507045	1.818714
1490	1351.472609	1196.66953	1.515192	1.826861
1500	1363.581301	1205.90679	1.523292	1.83496
1510	1375.70005	1215.15411	1.531344	1.843012
1520	1387.828738	1224.41139	1.53935	1.851017
1530	1399.967253	1233.67849	1.54731	1.858976
1540	1412.115482	1242.95532	1.555224	1.86689
1550	1424.273313	1252.24175	1.563093	1.874759
1560	1436.44064	1261.53768	1.570918	1.882583
1570	1448.617354	1270.84301	1.578698	1.890364
1580	1460.803352	1280.15763	1.586436	1.8981
1590	1472.998531	1289.48143	1.59413	1.905794
1600	1485.20279	1298.81432	1.601781	1.913445
1610	1497.416029	1308.1562	1.609391	1.921055
1620	1509.638152	1317.50696	1.616959	1.928622
1630	1521.869061	1326.86652	1.624485	1.936149
1640	1534.108664	1336.23477	1.631971	1.943634
1650	1546.356867	1345.61163	1.639417	1.95108
1660	1558.61358	1354.997	1.646823	1.958485
1670	1570.878714	1364.3908	1.65419	1.965851
1680	1583.152181	1373.79294	1.661517	1.973179
1690	1595.433895	1383.20332	1.668806	1.980467
1700	1607.72377	1392.62188	1.676057	1.987718
1710	1620.021725	1402.04851	1.683269	1.99493
1720	1632.327676	1411.48315	1.690445	2.002105
1730	1644.641545	1420.92571	1.697583	2.009244
1740	1656.963251	1430.37611	1.704685	2.016345
1750	1669.292717	1439.83427	1.711751	2.023411
1760	1681.629867	1449.30012	1.718781	2.03044
1770	1693.974625	1458.77359	1.725775	2.037434
1780	1706.326919	1468.25459	1.732734	2.044393
1790	1718.686675	1477.74306	1.739658	2.051317
1800	1731.053823	1487.23892	1.746548	2.058206
1810	1743.428291	1496.74211	1.753404	2.065062
1820	1755.810013	1506.25255	1.760226	2.071884
1830	1768.198918	1515.77019	1.767014	2.078672
1840	1780.594942	1525.29494	1.773769	2.085427
1850	1792.998018	1534.82675	1.780492	2.092149
1860	1805.408082	1544.36554	1.787182	2.098839
1870	1817.825071	1553.91127	1.79384	2.105497
1880	1830.248921	1563.46386	1.800466	2.112123
1890	1842.679573	1573.02326	1.807061	2.118717
1900	1855.116966	1582.5894	1.813624	2.12528
1910	1867.561039	1592.16222	1.820156	2.131812
1920	1880.011736	1601.74167	1.826658	2.138314
1930	1892.468997	1611.32768	1.833129	2.144785
1940	1904.932768	1620.92021	1.83957	2.151226
1950	1917.402991	1630.51919	1.845982	2.157637
1960	1929.879613	1640.12458	1.852364	2.164019
1970	1942.362579	1649.73631	1.858716	2.170372
1980	1954.851836	1659.35433	1.86504	2.176695
1990	1967.347332	1668.9786	1.871335	2.18299
2000	1979.849015	1678.60905	1.877602	2.189256
air as a real gas				
P=400.0 kPa	P0	101.325 kPa	T0	273.15 K
T deg K	h kJ/kg	u kJ/kg	s kJ/kgK	s0 kJ/kgK
200	-75.06412025	141.348019	-0.71696	-0.3174
210	-64.87180138	148.601989	-0.66723	-0.26828
220	-54.69836805	155.844624	-0.6199	-0.22146
230	-44.54019506	163.078351	-0.57475	-0.17674
240	-34.39434051	170.305155	-0.53156	-0.13392
250	-24.25835539	177.526716	-0.49019	-0.09287
260	-14.13015	184.744494	-0.45046	-0.05342
270	-4.007898782	191.959795	-0.41226	-0.01546
280	6.110028255	199.173818	-0.37547	0.021126
290	16.22511533	206.387683	-0.33997	0.056434
300	26.33873726	213.602458	-0.30568	0.090554
310	36.45218598	220.819173	-0.27252	0.123567
320	46.56668988	228.038832	-0.24041	0.155546
330	56.68342769	235.26242	-0.20928	0.186556
340	66.80353835	242.490907	-0.17907	0.21666
350	76.92812773	249.725253	-0.14972	0.24591
360	87.05827307	256.966405	-0.12118	0.274359
370	97.19502561	264.215303	-0.09341	0.302051
380	107.339412	271.472872	-0.06635	0.32903
390	117.4924344	278.740027	-0.03998	0.355335
400	127.6550707	286.017665	-0.01425	0.381002
410	137.8282729	293.30667	0.010868	0.406065
420	148.0129665	300.607902	0.035411	0.430555

430	158.2100491	307.922203	0.059405	0.4545
440	168.4203886	315.25039	0.082878	0.477928
450	178.6448224	322.593254	0.105855	0.500863
460	188.8841555	329.951557	0.12836	0.523329
470	199.1391595	337.326033	0.150415	0.545347
480	209.4105716	344.717386	0.172039	0.566938
490	219.6990936	352.126285	0.193254	0.58812
500	230.005391	359.553367	0.214075	0.608912
510	240.330093	366.999235	0.234521	0.62933
520	250.6737917	374.464457	0.254606	0.64939
530	261.0370419	381.949565	0.274346	0.669106
540	271.4203617	389.455056	0.293755	0.688491
550	281.824232	396.981394	0.312845	0.70756
560	292.2490973	404.529005	0.331629	0.726324
570	302.6953664	412.098281	0.350118	0.744794
580	313.1634122	419.689582	0.368324	0.762982
590	323.6535736	427.303231	0.386256	0.780897
600	334.1661553	434.93952	0.403925	0.79855
610	344.7014293	442.59871	0.421339	0.815949
620	355.2596361	450.281031	0.438507	0.833102
630	365.8409852	457.98668	0.455437	0.850019
640	376.4456566	465.715829	0.472138	0.866707
650	387.0738018	473.46862	0.488616	0.883173
660	397.7255452	481.245169	0.504878	0.899423
670	408.400985	489.045566	0.520932	0.915466
680	419.1001948	496.869877	0.536783	0.931306
690	429.8232244	504.718145	0.552437	0.946951
700	440.5701012	512.590392	0.567901	0.962404
710	451.3408315	520.486617	0.583178	0.977673
720	462.1354015	528.4068	0.598276	0.992762
730	472.9537786	536.350905	0.613198	1.007676
740	483.7959122	544.318874	0.627949	1.022419
750	494.6617354	552.310636	0.642535	1.036997
760	505.5511655	560.326104	0.656958	1.051413
770	516.4641054	568.365177	0.671223	1.065671
780	527.4004444	576.42774	0.685335	1.079776
790	538.3600593	584.513665	0.699296	1.093731
800	549.3428151	592.622814	0.713111	1.10754
810	560.3485663	600.755039	0.726783	1.121206
820	571.3771574	608.910179	0.740315	1.134733
830	582.4284237	617.088069	0.753711	1.148123
840	593.5021925	625.288531	0.766973	1.16138
850	604.5982833	633.511383	0.780105	1.174506
860	615.7165089	641.756435	0.793109	1.187506
870	626.856676	650.02349	0.805987	1.20038
880	638.0185856	658.312349	0.818744	1.213132
890	649.2020337	666.622803	0.831381	1.225765
900	660.4068124	674.954644	0.8439	1.23828
910	671.6327094	683.307656	0.856305	1.25068
920	682.8795094	691.681623	0.868596	1.262968
930	694.1469943	700.076325	0.880777	1.275146
940	705.4349434	708.491539	0.89285	1.287215
950	716.7431342	716.92704	0.904817	1.299178
960	728.0713425	725.382604	0.916679	1.311037
970	739.4193429	733.858003	0.928438	1.322793
980	750.7869093	742.353009	0.940098	1.33445
990	762.1738148	750.867394	0.951658	1.346007
1000	773.5798323	759.40093	0.963121	1.357468
1010	785.004735	767.953388	0.97449	1.368833
1020	796.4482959	776.524541	0.985764	1.380105
1030	807.9102891	785.114161	0.996947	1.391285
1040	819.3904888	793.722021	1.008039	1.402374
1050	830.8886707	802.347896	1.019042	1.413375
1060	842.4046114	810.991561	1.029957	1.424288
1070	853.9380887	819.652793	1.040787	1.435116
1080	865.4888821	828.331371	1.051532	1.445859
1090	877.0567726	837.027075	1.062194	1.456518
1100	888.6415429	845.739686	1.072773	1.467096
1110	900.2429776	854.468989	1.083273	1.477593
1120	911.8608634	863.214769	1.093692	1.488011
1130	923.4949888	871.976813	1.104034	1.49835
1140	935.1451446	880.754912	1.114298	1.508613
1150	946.811124	889.548859	1.124487	1.5188
1160	958.4927223	898.358448	1.134601	1.528912
1170	970.1897372	907.183475	1.144641	1.538951
1180	981.9019689	916.023741	1.154609	1.548917
1190	993.6292199	924.879047	1.164506	1.558812
1200	1005.371296	933.749198	1.174332	1.568637
1210	1017.128003	942.634002	1.184088	1.578392
1220	1028.899154	951.533266	1.193777	1.588079
1230	1040.684559	960.446805	1.203397	1.597698
1240	1052.484036	969.374433	1.212952	1.607251

1250	1064.297401	978.315968	1.22244	1.616739
1260	1076.124476	987.271229	1.231864	1.626161
1270	1087.965084	996.24004	1.241225	1.63552
1280	1099.819051	1005.22223	1.250522	1.644816
1290	1111.686206	1014.21762	1.259757	1.65405
1300	1123.56638	1023.22604	1.268931	1.663223
1310	1135.459407	1032.24733	1.278044	1.672336
1320	1147.365123	1041.28133	1.287098	1.681388
1330	1159.283367	1050.32787	1.296093	1.690382
1340	1171.213982	1059.38679	1.30503	1.699318
1350	1183.156812	1068.45794	1.313909	1.708197
1360	1195.111703	1077.54116	1.322732	1.717018
1370	1207.078504	1086.63631	1.331499	1.725784
1380	1219.057068	1095.74323	1.340211	1.734495
1390	1231.047249	1104.86179	1.348868	1.743152
1400	1243.048903	1113.99182	1.357472	1.751754
1410	1255.06189	1123.1332	1.366022	1.760303
1420	1267.086072	1132.28579	1.374519	1.7688
1430	1279.121311	1141.44945	1.382965	1.777245
1440	1291.167475	1150.62404	1.39136	1.785639
1450	1303.224432	1159.80943	1.399704	1.793982
1460	1315.292053	1169.0055	1.407998	1.802275
1470	1327.370211	1178.21211	1.416242	1.810519
1480	1339.45878	1187.42915	1.424438	1.818714
1490	1351.557639	1196.65648	1.432585	1.826861
1500	1363.666668	1205.894	1.440685	1.83496
1510	1375.785747	1215.14157	1.448738	1.843012
1520	1387.91476	1224.39909	1.456744	1.851017
1530	1400.053594	1233.66643	1.464703	1.858976
1540	1412.202135	1242.94349	1.472618	1.86689
1550	1424.360275	1252.23016	1.480487	1.874759
1560	1436.527903	1261.52632	1.488312	1.882583
1570	1448.704915	1270.83188	1.496093	1.890364
1580	1460.891206	1280.14672	1.50383	1.8981
1590	1473.086672	1289.47074	1.511525	1.905794
1600	1485.291213	1298.80385	1.519176	1.913445
1610	1497.50473	1308.14593	1.526786	1.921055
1620	1509.727125	1317.49691	1.534354	1.928622
1630	1521.958302	1326.85667	1.541881	1.936149
1640	1534.198169	1336.22513	1.549367	1.943634
1650	1546.446632	1345.60219	1.556813	1.95108
1660	1558.7036	1354.98776	1.564219	1.958485
1670	1570.968985	1364.38176	1.571586	1.965851
1680	1583.242698	1373.78409	1.578913	1.973179
1690	1595.524654	1383.19467	1.586202	1.980467
1700	1607.814769	1392.61341	1.593453	1.987718
1710	1620.112958	1402.04023	1.600666	1.99493
1720	1632.41914	1411.47505	1.607842	2.002105
1730	1644.733236	1420.91779	1.61498	2.009244
1740	1657.055165	1430.36837	1.622082	2.016345
1750	1669.384851	1439.82671	1.629148	2.023411
1760	1681.722218	1449.29274	1.636178	2.03044
1770	1694.067189	1458.76637	1.643172	2.037434
1780	1706.419693	1468.24754	1.650132	2.044393
1790	1718.779655	1477.73618	1.657056	2.051317
1800	1731.147006	1487.23221	1.663946	2.058206
1810	1743.521674	1496.73556	1.670802	2.065062
1820	1755.903592	1506.24617	1.677624	2.071884
1830	1768.292691	1515.76396	1.684412	2.078672
1840	1780.688905	1525.28887	1.691168	2.085427
1850	1793.092169	1534.82083	1.69789	2.092149
1860	1805.502417	1544.35978	1.70458	2.098839
1870	1817.919588	1553.90566	1.711238	2.105497
1880	1830.343618	1563.4584	1.717865	2.112123
1890	1842.774446	1573.01795	1.724459	2.118717
1900	1855.212011	1582.58423	1.731023	2.12528
1910	1867.656256	1592.1572	1.737555	2.131812
1920	1880.10712	1601.73679	1.744057	2.138314
1930	1892.564548	1611.32295	1.750528	2.144785
1940	1905.028481	1620.91561	1.75697	2.151226
1950	1917.498865	1630.51473	1.763381	2.157637
1960	1929.975645	1640.12025	1.769763	2.164019
1970	1942.458767	1649.73212	1.776116	2.170372
1980	1954.948177	1659.35027	1.78244	2.176695
1990	1967.443825	1668.97467	1.788735	2.18299
2000	1979.945657	1678.60525	1.795001	2.189256
air as a real gas				
P=500.0 kPa	P0	101.325 kPa	T0	273.15 K
T deg K	h kJ/kg	u kJ/kg	s kJ/kgK	s0 kJ/kgK
200	-75.55683132	140.989605	-0.78281	-0.3174

210	-65.32202902	148.269017	-0.73288	-0.26828
220	-55.11134806	155.533837	-0.68538	-0.22146
230	-44.92023465	162.787092	-0.64008	-0.17674
240	-34.74502837	170.031228	-0.59677	-0.13392
250	-24.58271527	177.268283	-0.55529	-0.09287
260	-14.43075507	184.500003	-0.51547	-0.05342
270	-4.286958256	191.727921	-0.47719	-0.01546
280	5.850602533	198.953419	-0.44032	0.021126
290	15.98365699	206.177771	-0.40476	0.056434
300	26.11378446	213.402169	-0.37042	0.090554
310	36.24244876	220.627748	-0.3372	0.123567
320	46.37102384	227.8556	-0.30505	0.155546
330	56.5008126	235.086784	-0.27388	0.186556
340	66.63306053	242.322336	-0.24363	0.21666
350	76.76896556	249.563268	-0.21425	0.24591
360	86.90968486	256.810578	-0.18568	0.274359
370	97.05633948	264.065244	-0.15788	0.302051
380	107.2100172	271.328229	-0.1308	0.32903
390	117.3717743	278.600479	-0.10441	0.355335
400	127.5426361	285.88292	-0.07866	0.381002
410	137.7235971	293.176458	-0.05352	0.406065
420	147.9156207	300.481978	-0.02896	0.430555
430	158.1196379	307.80034	-0.00495	0.4545
440	168.3365474	315.132379	0.018543	0.477928
450	178.5672134	322.478901	0.041534	0.500863
460	188.8124658	329.840684	0.064052	0.523329
470	199.0730983	337.218474	0.086119	0.545347
480	209.3498685	344.612986	0.107755	0.566938
490	219.6434963	352.024899	0.128979	0.58812
500	229.9546643	359.454861	0.149811	0.608912
510	240.2840168	366.903483	0.170265	0.62933
520	250.6321598	374.371341	0.190359	0.64939
530	260.9996611	381.858975	0.210108	0.669106
540	271.3870505	389.366889	0.229524	0.688491
550	281.7948199	396.895552	0.248621	0.70756
560	292.2234239	404.445396	0.267412	0.726324
570	302.6732802	412.016819	0.285908	0.744794
580	313.1447706	419.610184	0.304119	0.762982
590	323.6382416	427.225821	0.322057	0.780897
600	334.1540056	434.864025	0.339731	0.79855
610	344.6923414	442.525061	0.35715	0.815949
620	355.2534956	450.209161	0.374323	0.833102
630	365.8376838	457.916529	0.391258	0.850019
640	376.4450915	465.647337	0.407963	0.866707
650	387.0758755	473.40173	0.424445	0.883173
660	397.7301648	481.179829	0.440712	0.899423
670	408.4080623	488.981725	0.456769	0.915466
680	419.1096457	496.807487	0.472623	0.931306
690	429.8349689	504.65716	0.488281	0.946951
700	440.5840629	512.530767	0.503747	0.962404
710	451.3569376	520.428311	0.519028	0.977673
720	462.1535824	528.349773	0.534129	0.992762
730	472.9739678	536.295117	0.549054	1.007676
740	483.8180463	544.26429	0.563808	1.022419
750	494.6857536	552.25722	0.578395	1.036997
760	505.5770096	560.273823	0.592821	1.051413
770	516.4917198	568.313998	0.607089	1.065671
780	527.4297756	576.377632	0.621203	1.079776
790	538.3910562	584.464599	0.635166	1.093731
800	549.3754286	592.574762	0.648983	1.10754
810	560.3827492	600.707972	0.662657	1.121206
820	571.4128645	608.864072	0.676191	1.134733
830	582.4656116	617.042896	0.689588	1.148123
840	593.5408193	625.244268	0.702852	1.16138
850	604.6383089	633.468006	0.715985	1.174506
860	615.7578946	641.713922	0.728991	1.187506
870	626.8993846	649.98182	0.741871	1.20038
880	638.0625812	658.271499	0.754629	1.213132
890	649.2472819	666.582754	0.767268	1.225765
900	660.4532797	674.915376	0.779788	1.23828
910	671.6803639	683.269151	0.792194	1.25068
920	682.9283202	691.643863	0.804487	1.262968
930	694.1969314	700.039291	0.816669	1.275146
940	705.485978	708.455215	0.828743	1.287215
950	716.7952385	716.89141	0.840711	1.299178
960	728.1244896	725.347651	0.852574	1.311037
970	739.4735069	733.823712	0.864335	1.322793
980	750.842065	742.319365	0.875995	1.33445
990	762.2299379	750.834383	0.887556	1.346007
1000	773.6368994	759.368538	0.899021	1.357468
1010	785.0627233	767.921602	0.91039	1.368833
1020	796.5071835	776.493347	0.921665	1.380105

1030	807.9700545	785.083547	0.932849	1.391285
1040	819.4511115	793.691975	0.943941	1.402374
1050	830.9501307	802.318405	0.954945	1.413375
1060	842.4668893	810.962615	0.965862	1.424288
1070	854.0011658	819.62438	0.976692	1.435116
1080	865.5527402	828.30348	0.987438	1.445859
1090	877.121394	836.999695	0.9981	1.456518
1100	888.7069104	845.712808	1.008681	1.467096
1110	900.3090747	854.442602	1.019181	1.477593
1120	911.9276739	863.188863	1.029601	1.488011
1130	923.562497	871.951379	1.039943	1.49835
1140	935.2133354	880.729942	1.050208	1.508613
1150	946.8799825	889.524342	1.060397	1.5188
1160	958.5622342	898.334376	1.070512	1.528912
1170	970.2598885	907.159841	1.080553	1.538951
1180	981.9727461	916.000536	1.090521	1.548917
1190	993.7006098	924.856263	1.100418	1.558812
1200	1005.443285	933.726827	1.110245	1.568637
1210	1017.20058	942.612036	1.120002	1.578392
1220	1028.972306	951.511699	1.129691	1.588079
1230	1040.758275	960.425629	1.139312	1.597698
1240	1052.558304	969.353641	1.148867	1.607251
1250	1064.37221	978.295552	1.158356	1.616739
1260	1076.199815	987.251184	1.16778	1.626161
1270	1088.040942	996.220359	1.177141	1.63552
1280	1099.895418	1005.2029	1.186439	1.644816
1290	1111.763071	1014.19864	1.195674	1.65405
1300	1123.643734	1023.20741	1.204848	1.663223
1310	1135.53724	1032.22905	1.213962	1.672336
1320	1147.443426	1041.26337	1.223016	1.681388
1330	1159.362132	1050.31024	1.232012	1.690382
1340	1171.293199	1059.36949	1.240949	1.699318
1350	1183.236472	1068.44095	1.249829	1.708197
1360	1195.191798	1077.52449	1.258652	1.717018
1370	1207.159026	1086.61995	1.267419	1.725784
1380	1219.138009	1095.72717	1.276131	1.734495
1390	1231.128601	1104.84602	1.284789	1.743152
1400	1243.130658	1113.97635	1.293392	1.751754
1410	1255.144041	1123.11802	1.301943	1.760303
1420	1267.168611	1132.27089	1.310441	1.7688
1430	1279.204232	1141.43482	1.318887	1.777245
1440	1291.25077	1150.60969	1.327282	1.785639
1450	1303.308095	1159.79535	1.335626	1.793982
1460	1315.376076	1168.99169	1.34392	1.802275
1470	1327.454588	1178.19856	1.352165	1.810519
1480	1339.543506	1187.41586	1.360361	1.818714
1490	1351.642707	1196.64345	1.368508	1.826861
1500	1363.752071	1205.88121	1.376608	1.83496
1510	1375.87148	1215.12903	1.384661	1.843012
1520	1388.000817	1224.38679	1.392667	1.851017
1530	1400.139969	1233.65437	1.400627	1.858976
1540	1412.288824	1242.93167	1.408542	1.86689
1550	1424.447271	1252.21857	1.416411	1.874759
1560	1436.615201	1261.51496	1.424236	1.882583
1570	1448.79251	1270.82074	1.432018	1.890364
1580	1460.979092	1280.13581	1.439755	1.8981
1590	1473.174845	1289.46005	1.44745	1.905794
1600	1485.379668	1298.79337	1.455102	1.913445
1610	1497.593462	1308.13567	1.462711	1.921055
1620	1509.81613	1317.48686	1.47028	1.928622
1630	1522.047575	1326.84683	1.477807	1.936149
1640	1534.287705	1336.21549	1.485293	1.943634
1650	1546.536427	1345.59275	1.492739	1.95108
1660	1558.79365	1354.97853	1.500145	1.958485
1670	1571.059285	1364.37272	1.507512	1.965851
1680	1583.333245	1373.77524	1.51484	1.973179
1690	1595.615443	1383.18601	1.522129	1.980467
1700	1607.905796	1392.60494	1.52938	1.987718
1710	1620.20422	1402.03195	1.536593	1.99493
1720	1632.510633	1411.46696	1.543769	2.002105
1730	1644.824955	1420.90988	1.550908	2.009244
1740	1657.147108	1430.36063	1.55801	2.016345
1750	1669.477014	1439.81915	1.565076	2.023411
1760	1681.814596	1449.28535	1.572106	2.03044
1770	1694.159781	1458.75916	1.5791	2.037434
1780	1706.512493	1468.2405	1.586059	2.044393
1790	1718.872662	1477.72931	1.592984	2.051317
1800	1731.240215	1487.2255	1.599874	2.058206
1810	1743.615083	1496.72901	1.60673	2.065062
1820	1755.997197	1506.23978	1.613552	2.071884
1830	1768.386489	1515.75773	1.62034	2.078672
1840	1780.782894	1525.2828	1.627096	2.085427

1850	1793.186345	1534.81492	1.633819	2.092149
1860	1805.596778	1544.35402	1.640509	2.098839
1870	1818.01413	1553.90005	1.647167	2.105497
1880	1830.438338	1563.45294	1.653793	2.112123
1890	1842.869342	1573.01264	1.660388	2.118717
1900	1855.307081	1582.57907	1.666952	2.12528
1910	1867.751496	1592.15218	1.673484	2.131812
1920	1880.202529	1601.73191	1.679986	2.138314
1930	1892.660121	1611.31821	1.686457	2.144785
1940	1905.124218	1620.91102	1.692899	2.151226
1950	1917.594762	1630.51027	1.69931	2.157637
1960	1930.0717	1640.11593	1.705693	2.164019
1970	1942.554978	1649.72793	1.712045	2.170372
1980	1955.044542	1659.34622	1.718369	2.176695
1990	1967.54034	1668.97075	1.724664	2.18299
2000	1980.042321	1678.60146	1.730931	2.189256
		air as a real gas		
P=600.0 kPa	P0	101.325 kPa	T0	273.15 K
T deg K	h kJ/kg	u kJ/kg	s kJ/kgK	s0 kJ/kgK
200	-76.05061737	140.629833	-0.83696	-0.3174
210	-65.77288511	147.935034	-0.78681	-0.26828
220	-55.52463533	155.222298	-0.73914	-0.22146
230	-45.30034893	162.495276	-0.69369	-0.17674
240	-35.09562202	169.756893	-0.65025	-0.13392
250	-24.90685837	177.009558	-0.60866	-0.09287
260	-14.73105489	184.255308	-0.56875	-0.05342
270	-4.565649319	191.495912	-0.53039	-0.01546
280	5.591589434	198.732941	-0.49345	0.021126
290	15.74264123	205.967822	-0.45783	0.056434
300	25.88929345	213.201877	-0.42343	0.090554
310	36.0331844	220.436346	-0.39017	0.123567
320	46.17583556	227.672411	-0.35796	0.155546
330	56.31867549	234.911209	-0.32675	0.186556
340	66.46305742	242.153837	-0.29647	0.21666
350	76.6102722	249.401366	-0.26705	0.24591
360	86.76155762	256.65484	-0.23846	0.274359
370	96.91810505	263.91528	-0.21063	0.302051
380	107.0810639	271.183685	-0.18353	0.32903
390	117.2515446	278.461033	-0.15711	0.355335
400	127.4306206	285.748278	-0.13134	0.381002
410	137.6193288	293.046351	-0.10618	0.406065
420	147.8186706	300.356159	-0.0816	0.430555
430	158.0296109	307.678582	-0.05757	0.4545
440	168.2530786	315.014472	-0.03407	0.477928
450	178.4899655	322.364652	-0.01107	0.500863
460	188.7411259	329.729913	0.011465	0.523329
470	199.0073761	337.111014	0.033544	0.545347
480	209.2894937	344.508683	0.055191	0.566938
490	219.5882171	351.923609	0.076426	0.58812
500	229.9042456	359.356449	0.097267	0.608912
510	240.2382388	366.807823	0.117731	0.62933
520	250.5908167	374.278316	0.137834	0.64939
530	260.9625601	381.768474	0.15759	0.669106
540	271.3540105	389.278808	0.177014	0.688491
550	281.7656706	396.809793	0.196118	0.70756
560	292.198005	404.361868	0.214916	0.726324
570	302.6514407	411.935435	0.233418	0.744794
580	313.1263681	419.530863	0.251636	0.762982
590	323.6231417	427.148485	0.269579	0.780897
600	334.1420809	434.788602	0.287258	0.79855
610	344.6834717	442.451482	0.304683	0.815949
620	355.2475669	450.137361	0.32186	0.833102
630	365.8345881	457.846444	0.3388	0.850019
640	376.4447262	465.57891	0.355509	0.866707
650	387.0781431	473.334905	0.371995	0.883173
660	397.7349729	481.114551	0.388266	0.899423
670	408.4153227	488.917945	0.404326	0.915466
680	419.1192746	496.745156	0.420184	0.931306
690	429.8468864	504.596232	0.435845	0.946951
700	440.5981929	512.471198	0.451315	0.962404
710	451.3732073	520.370058	0.466599	0.977673
720	462.1719225	528.292797	0.481702	0.992762
730	472.9943121	536.23938	0.49663	1.007676
740	483.8403313	544.209755	0.511387	1.022419
750	494.7099187	552.203853	0.525977	1.036997
760	505.6029969	560.221589	0.540405	1.051413
770	516.5194736	568.262865	0.554675	1.065671
780	527.4592428	576.327569	0.568791	1.079776
790	538.4221856	584.415577	0.582757	1.093731
800	549.4081713	592.526751	0.596576	1.10754

810	560.4170583	600.660946	0.610252	1.121206
820	571.4486947	608.818005	0.623787	1.134733
830	582.5029195	616.997762	0.637187	1.148123
840	593.5795633	625.200043	0.650452	1.16138
850	604.6784489	633.424666	0.663587	1.174506
860	615.7993921	641.671445	0.676594	1.187506
870	626.9422023	649.940183	0.689476	1.20038
880	638.1066834	658.230683	0.702236	1.213132
890	649.2926342	666.542738	0.714875	1.225765
900	660.499849	674.876141	0.727397	1.23828
910	671.7281181	683.230677	0.739804	1.25068
920	682.9772283	691.606132	0.752099	1.262968
930	694.2469638	700.002287	0.764282	1.275146
940	705.5371059	708.418919	0.776357	1.287215
950	716.8474341	716.855807	0.788326	1.299178
960	728.1777261	725.312725	0.80019	1.311037
970	739.5277583	733.789447	0.811952	1.322793
980	750.8973063	742.285747	0.823613	1.33445
990	762.2861449	750.801398	0.835176	1.346007
1000	773.6940487	759.336171	0.846641	1.357468
1010	785.1207921	767.88984	0.858011	1.368833
1020	796.5661499	776.462177	0.869287	1.380105
1030	808.0298973	785.052956	0.880471	1.391285
1040	819.5118101	793.661951	0.891565	1.402374
1050	831.0116652	802.288937	0.90257	1.413375
1060	842.5292403	810.933689	0.913487	1.424288
1070	854.0643145	819.595987	0.924318	1.435116
1080	865.6166685	828.275609	0.935065	1.445859
1090	877.1860843	836.972335	0.945728	1.456518
1100	888.7723456	845.685948	0.956309	1.467096
1110	900.3752382	854.416233	0.966809	1.477593
1120	911.9945496	863.162975	0.97723	1.488011
1130	923.6300693	871.925964	0.987573	1.49835
1140	935.2815891	880.704989	0.997839	1.508613
1150	946.9489029	889.499843	1.008029	1.5188
1160	958.6318069	898.310322	1.018144	1.528912
1170	970.3300996	907.136223	1.028185	1.538951
1180	982.043582	915.977346	1.038154	1.548917
1190	993.7720574	924.833494	1.048052	1.558812
1200	1005.515332	933.704471	1.057879	1.568637
1210	1017.273213	942.590085	1.067636	1.578392
1220	1029.045513	951.490146	1.077325	1.588079
1230	1040.832045	960.404466	1.086947	1.597698
1240	1052.632624	969.332862	1.096502	1.607251
1250	1064.44707	978.27515	1.105992	1.616739
1260	1076.275204	987.231152	1.115417	1.626161
1270	1088.11685	996.200691	1.124778	1.63552
1280	1099.971834	1005.18359	1.134076	1.644816
1290	1111.839986	1014.17968	1.143312	1.65405
1300	1123.721137	1023.1888	1.152487	1.663223
1310	1135.615122	1032.21077	1.161601	1.672336
1320	1147.521777	1041.24543	1.170655	1.681388
1330	1159.440943	1050.29263	1.179651	1.690382
1340	1171.372462	1059.3522	1.188588	1.699318
1350	1183.316177	1068.42398	1.197468	1.708197
1360	1195.271938	1077.50783	1.206292	1.717018
1370	1207.239592	1086.60359	1.21506	1.725784
1380	1219.218993	1095.71112	1.223772	1.734495
1390	1231.209995	1104.83027	1.23243	1.743152
1400	1243.212455	1113.96089	1.241034	1.751754
1410	1255.226233	1123.10284	1.249584	1.760303
1420	1267.251191	1132.256	1.258083	1.7688
1430	1279.287192	1141.42021	1.266529	1.777245
1440	1291.334105	1150.59535	1.274924	1.785639
1450	1303.391796	1159.78129	1.283268	1.793982
1460	1315.460138	1168.97789	1.291563	1.802275
1470	1327.539004	1178.18502	1.299808	1.810519
1480	1339.628269	1187.40258	1.308004	1.818714
1490	1351.727812	1196.63042	1.316152	1.826861
1500	1363.837511	1205.86843	1.324252	1.83496
1510	1375.957249	1215.1165	1.332305	1.843012
1520	1388.08691	1224.3745	1.340311	1.851017
1530	1400.22638	1233.64233	1.348272	1.858976
1540	1412.375547	1242.91986	1.356187	1.86689
1550	1424.534301	1252.20699	1.364056	1.874759
1560	1436.702533	1261.50361	1.371882	1.882583
1570	1448.880139	1270.80962	1.379663	1.890364
1580	1461.067012	1280.1249	1.387401	1.8981
1590	1473.263051	1289.44937	1.395095	1.905794
1600	1485.468156	1298.78291	1.402747	1.913445
1610	1497.682227	1308.12542	1.410357	1.921055
1620	1509.905166	1317.47682	1.417926	1.928622

1630	1522.136879	1326.83699	1.425453	1.936149
1640	1534.377272	1336.20586	1.43294	1.943634
1650	1546.626253	1345.58333	1.440386	1.95108
1660	1558.88373	1354.9693	1.447792	1.958485
1670	1571.149616	1364.36368	1.455159	1.965851
1680	1583.423821	1373.7664	1.462487	1.973179
1690	1595.706262	1383.17736	1.469776	1.980467
1700	1607.996853	1392.59648	1.477027	1.987718
1710	1620.29551	1402.02368	1.484241	1.99493
1720	1632.602154	1411.45886	1.491417	2.002105
1730	1644.916703	1420.90197	1.498555	2.009244
1740	1657.239079	1430.3529	1.505658	2.016345
1750	1669.569204	1439.8116	1.512724	2.023411
1760	1681.907002	1449.27797	1.519754	2.03044
1770	1694.252399	1458.75195	1.526748	2.037434
1780	1706.60532	1468.23346	1.533708	2.044393
1790	1718.965695	1477.72243	1.540632	2.051317
1800	1731.33345	1487.21879	1.547522	2.058206
1810	1743.708517	1496.72247	1.554378	2.065062
1820	1756.090828	1506.2334	1.561201	2.071884
1830	1768.480313	1515.75151	1.567989	2.078672
1840	1780.876908	1525.27673	1.574745	2.085427
1850	1793.280546	1534.80901	1.581468	2.092149
1860	1805.691163	1544.34827	1.588158	2.098839
1870	1818.108696	1553.89445	1.594817	2.105497
1880	1830.533083	1563.44749	1.601443	2.112123
1890	1842.964263	1573.00733	1.608038	2.118717
1900	1855.402175	1582.57391	1.614601	2.12528
1910	1867.84676	1592.14716	1.621134	2.131812
1920	1880.297961	1601.72704	1.627636	2.138314
1930	1892.755719	1611.31348	1.634107	2.144785
1940	1905.219978	1620.90642	1.640549	2.151226
1950	1917.690682	1630.50582	1.646961	2.157637
1960	1930.167778	1640.11161	1.653343	2.164019
1970	1942.651211	1649.72374	1.659696	2.170372
1980	1955.140928	1659.34217	1.66602	2.176695
1990	1967.636877	1668.96683	1.672315	2.18299
2000	1980.139007	1678.59767	1.678582	2.189256
		air as a real gas		
P=700.0 kPa	P0	101.325 kPa	T0	273.15 K
T deg K	h kJ/kg	u kJ/kg	s kJ/kgK	s0 kJ/kgK
200	-76.54548338	140.268699	-0.88302	-0.3174
210	-66.22437092	147.600041	-0.83266	-0.26828
220	-55.93822957	154.910009	-0.78481	-0.22146
230	-45.68053718	162.202907	-0.73921	-0.17674
240	-35.44612084	169.482155	-0.69565	-0.13392
250	-25.23078442	176.750544	-0.65395	-0.09287
260	-15.03104962	184.010414	-0.61395	-0.05342
270	-4.843972595	191.263772	-0.5755	-0.01546
280	5.332987928	198.512385	-0.53849	0.021126
290	15.50206668	205.757839	-0.5028	0.056434
300	25.66526256	213.001583	-0.46835	0.090554
310	35.82439098	220.244969	-0.43504	0.123567
320	45.98112294	227.489268	-0.40279	0.155546
330	56.13701414	234.735694	-0.37154	0.186556
340	66.29352668	241.985412	-0.34122	0.21666
350	76.45204524	249.239547	-0.31177	0.24591
360	86.6138889	256.499193	-0.28315	0.274359
370	96.78031986	263.765412	-0.25529	0.302051
380	106.9525496	271.039241	-0.22816	0.32903
390	117.1317431	278.321689	-0.20172	0.355335
400	127.3190218	285.613739	-0.17593	0.381002
410	137.5154657	292.916348	-0.15075	0.406065
420	147.7221139	300.230444	-0.12616	0.430555
430	157.9399656	307.556928	-0.10211	0.4545
440	168.16998	314.896668	-0.0786	0.477928
450	178.4130764	322.250504	-0.05558	0.500863
460	188.6701338	329.619242	-0.03303	0.523329
470	198.9419908	337.003654	-0.01094	0.545347
480	209.2294451	344.404477	0.010716	0.566938
490	219.5332539	351.822414	0.031962	0.58812
500	229.8541329	359.258131	0.052813	0.608912
510	240.1927572	366.712255	0.073286	0.62933
520	250.5497608	374.18538	0.093397	0.64939
530	260.9257373	381.678059	0.113162	0.669106
540	271.3212399	389.190812	0.132593	0.688491
550	281.7367823	396.724117	0.151704	0.70756
560	292.1728391	404.278421	0.170509	0.726324
570	302.6298465	411.854131	0.189017	0.744794
580	313.1082034	419.451619	0.207241	0.762982

590	323.6082723	427.071225	0.22519	0.780897
600	334.13038	434.713253	0.242874	0.79855
610	344.674819	442.377974	0.260304	0.815949
620	355.2418489	450.065629	0.277486	0.833102
630	365.8316968	457.776427	0.29443	0.850019
640	376.4445593	465.510548	0.311144	0.866707
650	387.0806035	473.268142	0.327634	0.883173
660	397.7399682	481.049335	0.343908	0.899423
670	408.4227651	488.854224	0.359973	0.915466
680	419.1290804	496.682882	0.375834	0.931306
690	429.8589759	504.53536	0.391498	0.946951
700	440.6124901	512.411684	0.406971	0.962404
710	451.3896398	520.31186	0.422258	0.977673
720	462.190421	528.235874	0.437364	0.992762
730	473.0148105	536.183695	0.452295	1.007676
740	483.8627664	544.15527	0.467054	1.022419
750	494.7342301	552.150533	0.481647	1.036997
760	505.6291266	560.169401	0.496077	1.051413
770	516.5473662	568.211777	0.51035	1.065671
780	527.4888452	576.27755	0.524468	1.079776
790	538.4534469	584.366597	0.538436	1.093731
800	549.4410426	592.478783	0.552257	1.10754
810	560.4514927	600.613961	0.565935	1.121206
820	571.4846472	608.771977	0.579472	1.134733
830	582.5403468	616.952666	0.592873	1.148123
840	593.6184239	625.155855	0.606141	1.16138
850	604.7187027	633.381363	0.619277	1.174506
860	615.8410006	641.629003	0.632286	1.187506
870	626.9851286	649.898582	0.645169	1.20038
880	638.1508918	658.189901	0.65793	1.213132
890	649.3380903	666.502755	0.670571	1.225765
900	660.5465196	674.836937	0.683095	1.23828
910	671.7759713	683.192235	0.695503	1.25068
920	683.0262334	691.568433	0.707799	1.262968
930	694.297091	699.965312	0.719983	1.275146
940	705.5883266	708.382653	0.73206	1.287215
950	716.8997205	716.820232	0.74403	1.299178
960	728.2310514	725.277826	0.755895	1.311037
970	739.5820968	733.75521	0.767658	1.322793
980	750.9526329	742.252156	0.77932	1.33445
990	762.3424354	750.768438	0.790883	1.346007
1000	773.7512797	759.303829	0.802349	1.357468
1010	785.1789411	767.858102	0.81372	1.368833
1020	796.6251949	776.431031	0.824998	1.380105
1030	808.0898171	785.022388	0.836183	1.391285
1040	819.5725842	793.63195	0.847277	1.402374
1050	831.0732736	802.25949	0.858283	1.413375
1060	842.5916638	810.904786	0.869201	1.424288
1070	854.1275345	819.567615	0.880033	1.435116
1080	865.6806667	828.247758	0.89078	1.445859
1090	877.2508432	836.944995	0.901443	1.456518
1100	888.8378482	845.659108	0.912025	1.467096
1110	900.4414678	854.389883	0.922526	1.477593
1120	912.0614902	863.137106	0.932948	1.488011
1130	923.6977053	871.900566	0.943291	1.49835
1140	935.3499054	880.680053	0.953558	1.508613
1150	947.0178848	889.475361	0.963748	1.5188
1160	958.70144	898.286284	0.973864	1.528912
1170	970.4003701	907.112622	0.983906	1.538951
1180	982.1144763	915.954173	0.993875	1.548917
1190	993.8435625	924.810741	1.003773	1.558812
1200	1005.587435	933.68213	1.013601	1.568637
1210	1017.345902	942.568149	1.023359	1.578392
1220	1029.118775	951.468607	1.033049	1.588079
1230	1040.905868	960.383318	1.042671	1.597698
1240	1052.706998	969.312097	1.052226	1.607251
1250	1064.521983	978.254762	1.061716	1.616739
1260	1076.350645	987.211133	1.071142	1.626161
1270	1088.192809	996.181035	1.080503	1.63552
1280	1100.0483	1005.16429	1.089802	1.644816
1290	1111.916949	1014.16074	1.099038	1.65405
1300	1123.798587	1023.1702	1.108213	1.663223
1310	1135.69305	1032.19251	1.117328	1.672336
1320	1147.600174	1041.2275	1.126382	1.681388
1330	1159.5198	1050.27503	1.135378	1.690382
1340	1171.451769	1059.33492	1.144316	1.699318
1350	1183.395927	1068.40702	1.153197	1.708197
1360	1195.352121	1077.49118	1.162021	1.717018
1370	1207.320201	1086.58725	1.170788	1.725784
1380	1219.300019	1095.69508	1.179501	1.734495
1390	1231.291431	1104.81452	1.188159	1.743152
1400	1243.294293	1113.94543	1.196763	1.751754

1410	1255.308466	1123.08768	1.205314	1.760303
1420	1267.333811	1132.24112	1.213813	1.7688
1430	1279.370193	1141.40561	1.222259	1.777245
1440	1291.417478	1150.58102	1.230655	1.785639
1450	1303.475536	1159.76723	1.239	1.793982
1460	1315.544238	1168.96409	1.247294	1.802275
1470	1327.623458	1178.17149	1.255539	1.810519
1480	1339.71307	1187.3893	1.263736	1.818714
1490	1351.812953	1196.6174	1.271884	1.826861
1500	1363.922988	1205.85566	1.279984	1.83496
1510	1376.043055	1215.10398	1.288038	1.843012
1520	1388.173039	1224.36222	1.296044	1.851017
1530	1400.312826	1233.63028	1.304005	1.858976
1540	1412.462305	1242.90805	1.31192	1.86689
1550	1424.621365	1252.19542	1.31979	1.874759
1560	1436.789899	1261.49227	1.327615	1.882583
1570	1448.9678	1270.7985	1.335397	1.890364
1580	1461.154965	1280.11401	1.343134	1.8981
1590	1473.35129	1289.43869	1.350829	1.905794
1600	1485.556676	1298.77245	1.358482	1.913445
1610	1497.771023	1308.11517	1.366092	1.921055
1620	1509.994234	1317.46678	1.37366	1.928622
1630	1522.226215	1326.82716	1.381188	1.936149
1640	1534.46687	1336.19624	1.388674	1.943634
1650	1546.716109	1345.5739	1.396121	1.95108
1660	1558.973841	1354.96007	1.403527	1.958485
1670	1571.239976	1364.35466	1.410894	1.965851
1680	1583.514427	1373.75757	1.418222	1.973179
1690	1595.79711	1383.16872	1.425512	1.980467
1700	1608.087938	1392.58803	1.432763	1.987718
1710	1620.38683	1402.01541	1.439977	1.99493
1720	1632.693703	1411.45078	1.447153	2.002105
1730	1645.008479	1420.89406	1.454292	2.009244
1740	1657.331077	1430.34518	1.461394	2.016345
1750	1669.661421	1439.80405	1.46846	2.023411
1760	1681.999435	1449.27059	1.47549	2.03044
1770	1694.345044	1458.74475	1.482485	2.037434
1780	1706.698174	1468.22643	1.489445	2.044393
1790	1719.058754	1477.71557	1.496369	2.051317
1800	1731.426712	1487.21209	1.503259	2.058206
1810	1743.801978	1496.71593	1.510116	2.065062
1820	1756.184484	1506.22702	1.516938	2.071884
1830	1768.574162	1515.74529	1.523727	2.078672
1840	1780.970947	1525.27067	1.530483	2.085427
1850	1793.374771	1534.8031	1.537206	2.092149
1860	1805.785573	1544.34251	1.543896	2.098839
1870	1818.203287	1553.88885	1.550554	2.105497
1880	1830.627852	1563.44204	1.557181	2.112123
1890	1843.059207	1573.00203	1.563776	2.118717
1900	1855.497292	1582.56875	1.570339	2.12528
1910	1867.942048	1592.14215	1.576872	2.131812
1920	1880.393416	1601.72217	1.583374	2.138314
1930	1892.851339	1611.30875	1.589846	2.144785
1940	1905.31576	1620.90183	1.596287	2.151226
1950	1917.786625	1630.50137	1.602699	2.157637
1960	1930.263879	1640.10729	1.609081	2.164019
1970	1942.747467	1649.71956	1.615434	2.170372
1980	1955.237337	1659.33812	1.621758	2.176695
1990	1967.733436	1668.96291	1.628054	2.18299
2000	1980.235714	1678.59388	1.63432	2.189256
		air as a real gas		
P=800.0 kPa	P0	101.325 kPa	T0	273.15 K
T deg K	h kJ/kg	u kJ/kg	s kJ/kgK	s0 kJ/kgK
200	-77.0414337	139.906203	-0.92316	-0.3174
210	-66.67648717	147.26404	-0.87259	-0.26828
220	-56.35213007	154.596974	-0.82456	-0.22146
230	-46.06079833	161.909988	-0.77881	-0.17674
240	-35.79652393	169.207016	-0.73513	-0.13392
250	-25.55449294	176.491244	-0.69332	-0.09287
260	-15.33073927	183.765322	-0.65322	-0.05342
270	-5.121928548	191.031505	-0.61469	-0.01546
280	5.074797122	198.291756	-0.57761	0.021126
290	15.26193208	205.547823	-0.54186	0.056434
300	25.44169022	212.80129	-0.50735	0.090554
310	35.6160667	220.053618	-0.47399	0.123567
320	45.78688397	227.306172	-0.4417	0.155546
330	55.95582639	234.560243	-0.41041	0.186556
340	66.12446606	241.817062	-0.38005	0.21666
350	76.29428235	249.077813	-0.35057	0.24591
360	86.46667632	256.343638	-0.32191	0.274359

370	96.64298151	263.615641	-0.29403	0.302051
380	106.8244718	270.894898	-0.26688	0.32903
390	117.0123671	278.182448	-0.24042	0.355335
400	127.2078373	285.479305	-0.2146	0.381002
410	137.4120053	292.78645	-0.18941	0.406065
420	147.6259484	300.104835	-0.16479	0.430555
430	157.8506998	307.435378	-0.14073	0.4545
440	168.0872495	314.778968	-0.1172	0.477928
450	178.3365441	322.136459	-0.09417	0.500863
460	188.5994874	329.508673	-0.07161	0.523329
470	198.8769403	336.896393	-0.04951	0.545347
480	209.1697209	344.30037	-0.02784	0.566938
490	219.4786047	351.721316	-0.00658	0.58812
500	229.8043244	359.159906	0.014279	0.608912
510	240.1475702	366.616778	0.034761	0.62933
520	250.5089902	374.092533	0.054881	0.64939
530	260.8891908	381.587732	0.074653	0.669106
540	271.2887371	389.102901	0.094092	0.688491
550	281.7081535	396.638525	0.113211	0.70756
560	292.1479246	404.195055	0.132022	0.726324
570	302.608496	411.772905	0.150536	0.744794
580	313.0902751	419.372452	0.168766	0.762982
590	323.5936321	426.99404	0.186721	0.780897
600	334.1189013	434.637976	0.204411	0.79855
610	344.6663821	442.304537	0.221845	0.815949
620	355.2363401	449.993967	0.239032	0.833102
630	365.8290087	457.706477	0.255981	0.850019
640	376.4445898	465.442251	0.272699	0.866707
650	387.0832556	473.201444	0.289193	0.883173
660	397.7451497	480.984181	0.305471	0.899423
670	408.4303884	488.790564	0.321539	0.915466
680	419.1390621	496.620668	0.337404	0.931306
690	429.8712364	504.474545	0.353072	0.946951
700	440.6269536	512.352225	0.368548	0.962404
710	451.4062341	520.253715	0.383838	0.977673
720	462.209077	528.179004	0.398947	0.992762
730	473.0354621	536.12806	0.41388	1.007676
740	483.8853507	544.100834	0.428642	1.022419
750	494.7586867	552.097261	0.443237	1.036997
760	505.6553979	560.11726	0.45767	1.051413
770	516.5753968	568.160735	0.471945	1.065671
780	527.5185821	576.227576	0.486065	1.079776
790	538.4848392	584.317661	0.500035	1.093731
800	549.4740417	592.430856	0.513858	1.10754
810	560.4860518	600.567017	0.527538	1.121206
820	571.5207213	608.725989	0.541077	1.134733
830	582.5778929	616.907609	0.55448	1.148123
840	593.6574003	625.111705	0.567749	1.16138
850	604.7590697	633.338096	0.580887	1.174506
860	615.8827197	641.586597	0.593898	1.187506
870	627.0281628	649.857015	0.606783	1.20038
880	638.1952056	658.149152	0.619545	1.213132
890	649.3836494	666.462805	0.632188	1.225765
900	660.593291	674.797766	0.644712	1.23828
910	671.8239231	683.153823	0.657122	1.25068
920	683.0753349	691.530763	0.669419	1.262968
930	694.3473125	699.928367	0.681605	1.275146
940	705.6396395	708.346415	0.693682	1.287215
950	716.9520971	716.784686	0.705653	1.299178
960	728.2844652	725.242956	0.71752	1.311037
970	739.6365218	733.720999	0.729284	1.322793
980	751.0080442	742.21859	0.740947	1.33445
990	762.3988089	750.735503	0.752511	1.346007
1000	773.8085921	759.271512	0.763978	1.357468
1010	785.2371697	767.826389	0.77535	1.368833
1020	796.684318	776.399908	0.786628	1.380105
1030	808.1498134	784.991843	0.797814	1.391285
1040	819.6334333	793.601971	0.808909	1.402374
1050	831.1349557	802.230065	0.819916	1.413375
1060	842.6541595	810.875903	0.830834	1.424288
1070	854.1908253	819.539264	0.841667	1.435116
1080	865.7447344	828.219928	0.852415	1.445859
1090	877.3156703	836.917674	0.863079	1.456518
1100	888.9034177	845.632288	0.873662	1.467096
1110	900.5077632	854.363552	0.884164	1.477593
1120	912.1284954	863.111256	0.894586	1.488011
1130	923.7654048	871.875186	0.90493	1.49835
1140	935.418284	880.655135	0.915197	1.508613
1150	947.0869278	889.450895	0.925388	1.5188
1160	958.7711333	898.262263	0.935504	1.528912
1170	970.4706997	907.089036	0.945547	1.538951
1180	982.1854288	915.931015	0.955517	1.548917

1190	993.9151246	924.788003	0.965415	1.558812
1200	1005.659594	933.659804	0.975243	1.568637
1210	1017.418646	942.546227	0.985002	1.578392
1220	1029.192091	951.447083	0.994692	1.588079
1230	1040.979746	960.362183	1.004315	1.597698
1240	1052.781425	969.291345	1.013871	1.607251
1250	1064.596948	978.234386	1.023361	1.616739
1260	1076.426137	987.191128	1.032787	1.626161
1270	1088.268817	996.161393	1.042149	1.63552
1280	1100.124815	1005.14501	1.051447	1.644816
1290	1111.993961	1014.1418	1.060684	1.65405
1300	1123.876086	1023.15161	1.06986	1.663223
1310	1135.771026	1032.17426	1.078975	1.672336
1320	1147.678618	1041.20958	1.08803	1.681388
1330	1159.598703	1050.25744	1.097026	1.690382
1340	1171.531122	1059.31765	1.105964	1.699318
1350	1183.475721	1068.39007	1.114845	1.708197
1360	1195.432348	1077.47454	1.123669	1.717018
1370	1207.400853	1086.57091	1.132437	1.725784
1380	1219.381089	1095.67904	1.14115	1.734495
1390	1231.372909	1104.79878	1.149809	1.743152
1400	1243.376173	1113.92999	1.158413	1.751754
1410	1255.39074	1123.07252	1.166965	1.760303
1420	1267.416472	1132.22624	1.175463	1.7688
1430	1279.453233	1141.39101	1.18391	1.777245
1440	1291.500891	1150.5667	1.192306	1.785639
1450	1303.559316	1159.75317	1.200651	1.793982
1460	1315.628377	1168.95031	1.208946	1.802275
1470	1327.707949	1178.15797	1.217191	1.810519
1480	1339.797908	1187.37604	1.225388	1.818714
1490	1351.898132	1196.60439	1.233536	1.826861
1500	1364.0085	1205.8429	1.241637	1.83496
1510	1376.128896	1215.09146	1.24969	1.843012
1520	1388.259203	1224.34995	1.257697	1.851017
1530	1400.399307	1233.61825	1.265658	1.858976
1540	1412.549098	1242.89625	1.273573	1.86689
1550	1424.708464	1252.18385	1.281443	1.874759
1560	1436.877299	1261.48093	1.289269	1.882583
1570	1449.055496	1270.78739	1.29705	1.890364
1580	1461.242951	1280.10312	1.304788	1.8981
1590	1473.439562	1289.42802	1.312483	1.905794
1600	1485.645228	1298.76199	1.320136	1.913445
1610	1497.859851	1308.10493	1.327746	1.921055
1620	1510.083334	1317.45675	1.335315	1.928622
1630	1522.315581	1326.81734	1.342843	1.936149
1640	1534.556499	1336.18662	1.350329	1.943634
1650	1546.805996	1345.56448	1.357776	1.95108
1660	1559.063981	1354.95085	1.365183	1.958485
1670	1571.330366	1364.34563	1.37255	1.965851
1680	1583.605063	1373.74874	1.379878	1.973179
1690	1595.887986	1383.16008	1.387168	1.980467
1700	1608.179052	1392.57957	1.394419	1.987718
1710	1620.478178	1402.00714	1.401633	1.99493
1720	1632.785281	1411.4427	1.408809	2.002105
1730	1645.100282	1420.88616	1.415948	2.009244
1740	1657.423103	1430.33745	1.42305	2.016345
1750	1669.753666	1439.7965	1.430117	2.023411
1760	1682.091895	1449.26322	1.437147	2.03044
1770	1694.437716	1458.73755	1.444142	2.037434
1780	1706.791055	1468.2194	1.451101	2.044393
1790	1719.15184	1477.7087	1.458026	2.051317
1800	1731.519999	1487.20539	1.464917	2.058206
1810	1743.895464	1496.7094	1.471773	2.065062
1820	1756.278166	1506.22065	1.478595	2.071884
1830	1768.668037	1515.73908	1.485384	2.078672
1840	1781.065011	1525.26461	1.49214	2.085427
1850	1793.469022	1534.7972	1.498863	2.092149
1860	1805.880007	1544.33677	1.505554	2.098839
1870	1818.297902	1553.88325	1.512212	2.105497
1880	1830.722645	1563.43659	1.518839	2.112123
1890	1843.154176	1572.99672	1.525434	2.118717
1900	1855.592434	1582.56359	1.531997	2.12528
1910	1868.037359	1592.13714	1.53853	2.131812
1920	1880.488894	1601.7173	1.545032	2.138314
1930	1892.946982	1611.30402	1.551504	2.144785
1940	1905.411566	1620.89724	1.557946	2.151226
1950	1917.882591	1630.49691	1.564358	2.157637
1960	1930.360002	1640.10298	1.57074	2.164019
1970	1942.843745	1649.71538	1.577093	2.170372
1980	1955.333767	1659.33407	1.583417	2.176695
1990	1967.830017	1668.95899	1.589712	2.18299
2000	1980.332443	1678.59009	1.595979	2.189256

air as a real gas				
P=900.0 kPa	P0	101.325 kPa	T0	273.15 K
T deg K	h kJ/kg	u kJ/kg	s kJ/kgK	s0 kJ/kgK
200	-77.53847202	139.542341	-0.95879	-0.3174
210	-67.12923407	146.92703	-0.90801	-0.26828
220	-56.76633569	154.283195	-0.8598	-0.22146
230	-46.44113096	161.616523	-0.8139	-0.17674
240	-36.14683013	168.93148	-0.77009	-0.13392
250	-25.8779832	176.231662	-0.72817	-0.09287
260	-15.63012363	183.520036	-0.68797	-0.05342
270	-5.399517481	190.799112	-0.64936	-0.01546
280	4.817016263	198.071054	-0.61221	0.021126
290	15.02223628	205.337777	-0.5764	0.056434
300	25.21857498	212.601	-0.54183	0.090554
310	35.40820982	219.862296	-0.50842	0.123567
320	45.59311673	227.123124	-0.47608	0.155546
330	55.77511015	234.384855	-0.44475	0.186556
340	65.95587335	241.648788	-0.41436	0.21666
350	76.13698125	248.916164	-0.38484	0.24591
360	86.31991754	256.188174	-0.35616	0.274359
370	96.50608765	263.465967	-0.32825	0.302051
380	106.6968283	270.750654	-0.30107	0.32903
390	116.8934145	278.04331	-0.27459	0.355335
400	127.0970649	285.344976	-0.24875	0.381002
410	137.3089453	292.656658	-0.22354	0.406065
420	147.5301717	299.97933	-0.19891	0.430555
430	157.7618114	307.313933	-0.17483	0.4545
440	168.0048849	314.661372	-0.15128	0.477928
450	178.2603664	322.022517	-0.12824	0.500863
460	188.5291845	329.398204	-0.10567	0.523329
470	198.8122226	336.789232	-0.08355	0.545347
480	209.1103191	344.19636	-0.06187	0.566938
490	219.4242678	351.620312	-0.0406	0.58812
500	229.7548183	359.061775	-0.01973	0.608912
510	240.1026761	366.521393	7.58E-04	0.62933
520	250.4685033	373.999776	0.020887	0.64939
530	260.8529191	381.497493	0.040667	0.669106
540	271.2565004	389.015075	0.060114	0.688491
550	281.6797826	396.553015	0.079239	0.70756
560	292.1232602	404.11177	0.098057	0.726324
570	302.5873879	411.691758	0.116578	0.744794
580	313.0725816	419.293362	0.134813	0.762982
590	323.5792197	426.916929	0.152774	0.780897
600	334.1076436	434.562772	0.170469	0.79855
610	344.6581595	442.231171	0.187908	0.815949
620	355.2310395	449.922373	0.2051	0.833102
630	365.8265226	457.636594	0.222053	0.850019
640	376.4448163	465.37402	0.238775	0.866707
650	387.0860981	473.134808	0.255274	0.883173
660	397.7505163	480.919088	0.271556	0.899423
670	408.4381916	488.726963	0.287627	0.915466
680	419.1492185	496.558511	0.303496	0.931306
690	429.8836669	504.413787	0.319167	0.946951
700	440.6415825	512.292821	0.334646	0.962404
710	451.4229892	520.195623	0.349939	0.977673
720	462.2278895	528.122185	0.365051	0.992762
730	473.0562662	536.072475	0.379987	1.007676
740	483.9080834	544.046448	0.394752	1.022419
750	494.7832879	552.044038	0.409349	1.036997
760	505.6818099	560.065166	0.423785	1.051413
770	516.6035645	568.109738	0.438062	1.065671
780	527.5484526	576.177645	0.452184	1.079776
790	538.5163619	584.268767	0.466156	1.093731
800	549.5071679	592.382971	0.479981	1.10754
810	560.5207348	600.520114	0.493663	1.121206
820	571.5569164	608.680041	0.507204	1.134733
830	582.615557	616.862591	0.520609	1.148123
840	593.696492	625.067592	0.53388	1.16138
850	604.7995492	633.294866	0.547019	1.174506
860	615.9245487	641.544227	0.560031	1.187506
870	627.0713045	649.815483	0.572918	1.20038
880	638.2396244	658.108438	0.585682	1.213132
890	649.4293111	666.422888	0.598326	1.225765
900	660.6401627	674.758627	0.610852	1.23828
910	671.8719729	683.115443	0.623263	1.25068
920	683.1245323	691.493124	0.635561	1.262968
930	694.3976278	699.891452	0.647748	1.275146
940	705.6910441	708.310207	0.659827	1.287215
950	717.0045636	716.749168	0.671799	1.299178
960	728.3379668	725.208112	0.683666	1.311037

970	739.6910328	733.686815	0.695431	1.322793
980	751.0635398	742.185051	0.707095	1.33445
990	762.455265	750.702594	0.718661	1.346007
1000	773.8659853	759.239219	0.730129	1.357468
1010	785.2954776	767.794699	0.741501	1.368833
1020	796.7435188	776.368808	0.75278	1.380105
1030	808.209886	784.961321	0.763967	1.391285
1040	819.6943572	793.572014	0.775063	1.402374
1050	831.196711	802.200662	0.78607	1.413375
1060	842.7167271	810.847042	0.79699	1.424288
1070	854.2541865	819.510934	0.807823	1.435116
1080	865.8088713	828.192117	0.818572	1.445859
1090	877.3805653	836.890373	0.829237	1.456518
1100	888.9690538	845.605486	0.83982	1.467096
1110	900.574124	854.33724	0.850323	1.477593
1120	912.1955649	863.085423	0.860745	1.488011
1130	923.8331674	871.849824	0.87109	1.49835
1140	935.4867247	880.630234	0.881358	1.508613
1150	947.1560318	889.426447	0.891549	1.5188
1160	958.8408864	898.238259	0.901666	1.528912
1170	970.5410881	907.065467	0.911709	1.538951
1180	982.256439	915.907873	0.92168	1.548917
1190	993.9867437	924.76528	0.931579	1.558812
1200	1005.731809	933.637493	0.941407	1.568637
1210	1017.491444	942.52432	0.951166	1.578392
1220	1029.265462	951.425573	0.960857	1.588079
1230	1041.053676	960.341063	0.97048	1.597698
1240	1052.855903	969.270607	0.980037	1.607251
1250	1064.671964	978.214024	0.989527	1.616739
1260	1076.50168	987.171135	0.998954	1.626161
1270	1088.344876	996.141763	1.008316	1.63552
1280	1100.20138	1005.12573	1.017615	1.644816
1290	1112.071021	1014.12288	1.026852	1.65405
1300	1123.953632	1023.13303	1.036028	1.663223
1310	1135.849049	1032.15602	1.045143	1.672336
1320	1147.757108	1041.19168	1.054199	1.681388
1330	1159.677651	1050.23986	1.063196	1.690382
1340	1171.61052	1059.30039	1.072134	1.699318
1350	1183.55556	1068.37312	1.081015	1.708197
1360	1195.512619	1077.45791	1.08984	1.717018
1370	1207.481549	1086.55459	1.098608	1.725784
1380	1219.4622	1095.66302	1.107321	1.734495
1390	1231.454429	1104.78306	1.11598	1.743152
1400	1243.458094	1113.91456	1.124585	1.751754
1410	1255.473054	1123.05738	1.133136	1.760303
1420	1267.499172	1132.21138	1.141635	1.7688
1430	1279.536313	1141.37643	1.150083	1.777245
1440	1291.584344	1150.55239	1.158478	1.785639
1450	1303.643133	1159.73913	1.166824	1.793982
1460	1315.712553	1168.93653	1.175119	1.802275
1470	1327.792478	1178.14445	1.183365	1.810519
1480	1339.882783	1187.36278	1.191561	1.818714
1490	1351.983347	1196.59138	1.19971	1.826861
1500	1364.094049	1205.83015	1.207811	1.83496
1510	1376.214773	1215.07895	1.215864	1.843012
1520	1388.345402	1224.33768	1.223871	1.851017
1530	1400.485823	1233.60622	1.231832	1.858976
1540	1412.635925	1242.88446	1.239748	1.86689
1550	1424.795597	1252.17229	1.247618	1.874759
1560	1436.964732	1261.4696	1.255444	1.882583
1570	1449.143224	1270.77628	1.263226	1.890364
1580	1461.330969	1280.09223	1.270964	1.8981
1590	1473.527866	1289.41735	1.278659	1.905794
1600	1485.733812	1298.75154	1.286312	1.913445
1610	1497.948711	1308.0947	1.293923	1.921055
1620	1510.172465	1317.44672	1.301491	1.928622
1630	1522.404979	1326.80752	1.309019	1.936149
1640	1534.646159	1336.177	1.316506	1.943634
1650	1546.895913	1345.55507	1.323953	1.95108
1660	1559.154152	1354.94164	1.33136	1.958485
1670	1571.420786	1364.33661	1.338727	1.965851
1680	1583.695728	1373.73991	1.346055	1.973179
1690	1595.978892	1383.15144	1.353345	1.980467
1700	1608.270195	1392.57113	1.360597	1.987718
1710	1620.569554	1401.99888	1.36781	1.99493
1720	1632.876886	1411.43462	1.374987	2.002105
1730	1645.192114	1420.87826	1.382126	2.009244
1740	1657.515157	1430.32974	1.389229	2.016345
1750	1669.845938	1439.78896	1.396295	2.023411
1760	1682.184382	1449.25585	1.403325	2.03044
1770	1694.530415	1458.73035	1.41032	2.037434
1780	1706.883962	1468.21237	1.41728	2.044393

1790	1719.244951	1477.70184	1.424205	2.051317
1800	1731.613313	1487.1987	1.431095	2.058206
1810	1743.988976	1496.70287	1.437952	2.065062
1820	1756.371874	1506.21428	1.444774	2.071884
1830	1768.761937	1515.73286	1.451563	2.078672
1840	1781.1591	1525.25856	1.458319	2.085427
1850	1793.563298	1534.7913	1.465043	2.092149
1860	1805.974466	1544.33102	1.471733	2.098839
1870	1818.392542	1553.87765	1.478392	2.105497
1880	1830.817463	1563.43114	1.485018	2.112123
1890	1843.249168	1572.99143	1.491613	2.118717
1900	1855.687598	1582.55844	1.498177	2.12528
1910	1868.132694	1592.13213	1.50471	2.131812
1920	1880.584396	1601.71243	1.511212	2.138314
1930	1893.042649	1611.2993	1.517684	2.144785
1940	1905.507395	1620.89266	1.524126	2.151226
1950	1917.978579	1630.49247	1.530538	2.157637
1960	1930.456147	1640.09867	1.53692	2.164019
1970	1942.940045	1649.7112	1.543273	2.170372
1980	1955.43022	1659.33002	1.549598	2.176695
1990	1967.92662	1668.95508	1.555893	2.18299
2000	1980.429194	1678.58631	1.56216	2.189256
		air as a real gas		
P=1000.0 kPa	P0	101.325 kPa	T0	273.15 K
T deg K	h kJ/kg	u kJ/kg	s kJ/kgK	s0 kJ/kgK
200	-78.03660134	139.177111	-0.99086	-0.3174
210	-67.58261126	146.589014	-0.93986	-0.26828
220	-57.18084483	153.968676	-0.89147	-0.22146
230	-46.82153331	161.322516	-0.84542	-0.17674
240	-36.49703798	168.655552	-0.80148	-0.13392
250	-26.20125427	175.971802	-0.75945	-0.09287
260	-15.92920232	183.274561	-0.71916	-0.05342
270	-5.676739537	190.566596	-0.68047	-0.01546
280	4.559644732	197.850284	-0.64324	0.021126
290	14.78297826	205.127704	-0.60736	0.056434
300	24.99591546	212.400715	-0.57274	0.090554
310	35.20081873	219.671004	-0.53928	0.123567
320	45.39981937	226.940126	-0.5069	0.155546
330	55.59486342	234.209533	-0.47553	0.186556
340	65.78774644	241.480591	-0.4451	0.21666
350	75.98013974	248.754601	-0.41555	0.24591
360	86.17361032	256.032803	-0.38684	0.274359
370	96.36963597	263.316391	-0.3589	0.302051
380	106.5696166	270.606513	-0.3317	0.32903
390	116.7748828	277.904276	-0.30519	0.355335
400	126.9867021	285.210751	-0.27934	0.381002
410	137.2062836	292.526971	-0.2541	0.406065
420	147.4347817	299.853931	-0.22945	0.430555
430	157.6732981	307.192593	-0.20536	0.4545
440	167.922884	314.543879	-0.1818	0.477928
450	178.1845412	321.908678	-0.15874	0.500863
460	188.4592233	329.287837	-0.13615	0.523329
470	198.7478358	336.682169	-0.11403	0.545347
480	209.0512378	344.092447	-0.09234	0.566938
490	219.3702413	351.519405	-0.07106	0.58812
500	229.7056127	358.963737	-0.05018	0.608912
510	240.058073	366.426099	-0.02968	0.62933
520	250.4282982	373.907108	-0.00954	0.64939
530	260.8169205	381.40734	0.010248	0.669106
540	271.2245284	388.927334	0.029702	0.688491
550	281.6516681	396.467589	0.048835	0.70756
560	292.0988441	404.028566	0.067659	0.726324
570	302.5665205	411.61069	0.086186	0.744794
580	313.0551217	419.214348	0.104428	0.762982
590	323.5650338	426.839893	0.122394	0.780897
600	334.0966056	434.48764	0.140094	0.79855
610	344.6501501	442.157875	0.157538	0.815949
620	355.2259456	449.850848	0.174735	0.833102
630	365.8242372	457.566777	0.191693	0.850019
640	376.4452379	465.305853	0.208419	0.866707
650	387.0891301	473.068235	0.224921	0.883173
660	397.7560668	480.854057	0.241207	0.899423
670	408.4461735	488.663422	0.257283	0.915466
680	419.1595488	496.496413	0.273155	0.931306
690	429.8962664	504.353085	0.288829	0.946951
700	440.6563758	512.233471	0.304311	0.962404
710	451.4399042	520.137585	0.319607	0.977673
720	462.2468576	528.065418	0.334722	0.992762
730	473.0772217	536.016941	0.349661	1.007676
740	483.9309636	543.99211	0.364428	1.022419

750	494.8080327	551.990862	0.379028	1.036997
760	505.7083618	560.013118	0.393466	1.051413
770	516.6318685	568.058786	0.407745	1.065671
780	527.5784561	576.127759	0.42187	1.079776
790	538.5480142	584.219917	0.435844	1.093731
800	549.5404205	592.335128	0.449671	1.10754
810	560.5555412	600.473251	0.463355	1.121206
820	571.5932318	608.634132	0.476898	1.134733
830	582.6533385	616.817611	0.490304	1.148123
840	593.7356984	625.023517	0.503577	1.16138
850	604.8401406	633.251672	0.516718	1.174506
860	615.966487	641.501892	0.529732	1.187506
870	627.1145529	649.773986	0.54262	1.20038
880	638.2841475	658.067757	0.555385	1.213132
890	649.4750748	666.383003	0.56803	1.225765
900	660.687134	674.719519	0.580558	1.23828
910	671.9201203	683.077094	0.59297	1.25068
920	683.173825	691.455515	0.605269	1.262968
930	694.4480364	699.854566	0.617458	1.275146
940	705.7425401	708.274027	0.629538	1.287215
950	717.0571194	716.713677	0.641511	1.299178
960	728.3915559	725.173296	0.653379	1.311037
970	739.7456295	733.652657	0.665145	1.322793
980	751.1191192	742.151537	0.676811	1.33445
990	762.5118032	750.66971	0.688377	1.346007
1000	773.9234591	759.206951	0.699846	1.357468
1010	785.3538644	767.763033	0.711219	1.368833
1020	796.8027968	776.337732	0.722499	1.380105
1030	808.2700343	784.930822	0.733687	1.391285
1040	819.7553553	793.542079	0.744784	1.402374
1050	831.2585392	802.17128	0.755792	1.413375
1060	842.7793662	810.818202	0.766712	1.424288
1070	854.3176179	819.482624	0.777546	1.435116
1080	865.8730771	828.164327	0.788296	1.445859
1090	877.4455279	836.863092	0.798961	1.456518
1100	889.0347563	845.578704	0.809545	1.467096
1110	900.6405499	854.310947	0.820048	1.477593
1120	912.2626983	863.059609	0.830472	1.488011
1130	923.9009928	871.82448	0.840817	1.49835
1140	935.5552269	880.605351	0.851085	1.508613
1150	947.2251964	889.402016	0.861277	1.5188
1160	958.9106991	898.214271	0.871395	1.528912
1170	970.6115351	907.041914	0.881438	1.538951
1180	982.3275068	915.884748	0.891409	1.548917
1190	994.0584193	924.742573	0.901309	1.558812
1200	1005.80408	933.615197	0.911138	1.568637
1210	1017.564298	942.502428	0.920898	1.578392
1220	1029.338886	951.404077	0.930589	1.588079
1230	1041.127659	960.319957	0.940212	1.597698
1240	1052.930434	969.249883	0.949769	1.607251
1250	1064.747031	978.193676	0.95926	1.616739
1260	1076.577273	987.151156	0.968687	1.626161
1270	1088.420984	996.122146	0.97805	1.63552
1280	1100.277993	1005.10647	0.987349	1.644816
1290	1112.148129	1014.10397	0.996587	1.65405
1300	1124.031226	1023.11446	1.005763	1.663223
1310	1135.927118	1032.13779	1.014879	1.672336
1320	1147.835644	1041.17378	1.023935	1.681388
1330	1159.756644	1050.22229	1.032932	1.690382
1340	1171.689962	1059.28314	1.04187	1.699318
1350	1183.635442	1068.35619	1.050752	1.708197
1360	1195.592934	1077.44129	1.059577	1.717018
1370	1207.562287	1086.53828	1.068345	1.725784
1380	1219.543354	1095.64701	1.077059	1.734495
1390	1231.535991	1104.76734	1.085718	1.743152
1400	1243.540056	1113.89913	1.094323	1.751754
1410	1255.555409	1123.04224	1.102875	1.760303
1420	1267.581913	1132.19652	1.111374	1.7688
1430	1279.619432	1141.36185	1.119822	1.777245
1440	1291.667835	1150.53808	1.128218	1.785639
1450	1303.726989	1159.7251	1.136563	1.793982
1460	1315.796768	1168.92276	1.144859	1.802275
1470	1327.877044	1178.13094	1.153105	1.810519
1480	1339.967695	1187.34953	1.161302	1.818714
1490	1352.068598	1196.57839	1.16945	1.826861
1500	1364.179634	1205.8174	1.177552	1.83496
1510	1376.300685	1215.06645	1.185605	1.843012
1520	1388.431636	1224.32542	1.193613	1.851017
1530	1400.572374	1233.5942	1.201574	1.858976
1540	1412.722786	1242.87268	1.209489	1.86689
1550	1424.882763	1252.16074	1.21736	1.874759
1560	1437.052198	1261.45827	1.225186	1.882583

1570	1449.230985	1270.76518	1.232968	1.890364
1580	1461.419021	1280.08136	1.240706	1.8981
1590	1473.616202	1289.4067	1.248402	1.905794
1600	1485.822429	1298.7411	1.256055	1.913445
1610	1498.037603	1308.08447	1.263665	1.921055
1620	1510.261627	1317.4367	1.271235	1.928622
1630	1522.494407	1326.79771	1.278762	1.936149
1640	1534.735849	1336.16739	1.28625	1.943634
1650	1546.98586	1345.54566	1.293696	1.95108
1660	1559.244352	1354.93243	1.301103	1.958485
1670	1571.511235	1364.3276	1.308471	1.965851
1680	1583.786421	1373.73109	1.315799	1.973179
1690	1596.069827	1383.14281	1.323089	1.980467
1700	1608.361367	1392.56269	1.330341	1.987718
1710	1620.660958	1401.99062	1.337555	1.99493
1720	1632.96852	1411.42654	1.344731	2.002105
1730	1645.283973	1420.87037	1.351871	2.009244
1740	1657.607237	1430.32202	1.358973	2.016345
1750	1669.938237	1439.78142	1.36604	2.023411
1760	1682.276896	1449.24849	1.37307	2.03044
1770	1694.62314	1458.72316	1.380065	2.037434
1780	1706.976895	1468.20535	1.387025	2.044393
1790	1719.338089	1477.69499	1.39395	2.051317
1800	1731.706652	1487.19201	1.400841	2.058206
1810	1744.082514	1496.69634	1.407697	2.065062
1820	1756.465606	1506.20791	1.41452	2.071884
1830	1768.855862	1515.72666	1.421309	2.078672
1840	1781.253214	1525.25251	1.428065	2.085427
1850	1793.657598	1534.7854	1.434789	2.092149
1860	1806.068949	1544.32528	1.441479	2.098839
1870	1818.487205	1553.87206	1.448138	2.105497
1880	1830.912304	1563.4257	1.454765	2.112123
1890	1843.344185	1572.98613	1.46136	2.118717
1900	1855.782787	1582.55329	1.467924	2.12528
1910	1868.228052	1592.12713	1.474457	2.131812
1920	1880.679921	1601.70757	1.480959	2.138314
1930	1893.138338	1611.29457	1.487431	2.144785
1940	1905.603246	1620.88807	1.493873	2.151226
1950	1918.07459	1630.48802	1.500285	2.157637
1960	1930.552315	1640.09436	1.506667	2.164019
1970	1943.036367	1649.70703	1.513021	2.170372
1980	1955.526695	1659.32598	1.519345	2.176695
1990	1968.023245	1668.95116	1.52564	2.18299
2000	1980.525967	1678.58253	1.531907	2.189256