Summer Fellowship Report

On

Development of Thermodynamic Models and Functions in OpenModelica

Submitted by

Gaurav Tiwari

Under the guidance of

Prof.Kannan M. Moudgalya
Chemical Engineering Department
IIT Bombay

July 7, 2019

Acknowledgment

I wish to express my deepest gratitude to my internship guide Dr. Kannan M. Moudgalya, Professor, Department of Chemical Engineering, IIT Bombay for his continuous support and supervision throughout the internship. I would also like to express my profound and heart felt thanks to our mentors Priyam Nayak, Department of Chemical Engineering, IIT Bombay, A. S. Rahul, Department of Chemical Engineering IIT Bombay, our college professor Dr. P. R. Naren, SCBT, SASTRA Deemed University, and the FOSSEE team for their timely help and guidance.

Contents

Chapter 1

Introduction

"OpenModelica" is a free and open-source modelling environment that uses "Modelica" modelling language. It follows equation oriented approach. OpenModelica can be used for modelling, simulation, optimization and analysis of complex steady state and dynamic systems. Modelica modelling language allows users to express a system in the form of equations. OpenModelica complies expressions, equations, functions and algorithms into efficient C code. The generated C code is combined with a library of utility functions, a run-time library, and a numerical Differential-Algebric Equation (DAE) solver. OpenModelica Connection Editor, called as OMEdit is the integrated Graphical User Interface (GUI) in OpenModelica for graphical modelling and editing. OMEdit consists of several libraries for various domains like Electrical, Magnetic, Math, Thermal, etc. It provides various user friendly features like representation of a model in the form of block diagrams. OMEdit can be used for creating custom models and for editing or drawing connections between the model interfaces. It also allows users to plot graphs between parameters of the model simulated.

Chapter 2

Background

OMEdit, the integrated Graphical User Interface (GUI) of OpenModelica consists of libraries across various domains such as Electrical, Thermal, Math, Mechanics, etc. that can be used to develop models of a physical system or process. However, OMEdit does not contain libraries for modelling of chemical process systems. This limits the use of OpenModelica for modelling chemical process systems. Hence, it becomes necessary to build models or packages that aids in modelling of chemical process systems. An essential requirement is the availability of thermodynamic packages to estimate fluid properties. The main aim of this work is to develop standard thermodynamic packages in OpenModelica. In this work, a thermodynamic package, namely "Extended UNIQUAC developed in OpenModelica. Inorder to develop thermodynamic packages in OpenModelica, properties such as critical temperature, critical pressure, acentric factor, molecular weight, etc. of the components that constitutes the system are required. For now, we are not using the electrolytes present in DWSIM but relying only on the current database of OpenModelica. The developed thermodynamic packages can be used to calculate fugacity coefficient and activity coefficient of solutions. This package was designed for the use of electrolytic solutions which involve water as well as some limited number of ionic compounds.

Chapter 3

Entropy Function

3.1 Description

We can think of Entropy as a measure of randomness or disorder in an isolated system. Entropy can be measured in two possible ways i.e. absolute entropy and relative entropy. Absolute entropy can be measured using statistical method, whereas the relative entropy is more related with the heat involved in the process, thus making it a thermodynamic process. Entropy is a state function just like Enthalpy,Internal Energy, thus independent of path. Entropy of a system is maximum when it reaches equilibrium, thus the change in entropy becomes zero. Under isothermal conditions, the total entropy of a system in reversible process is zero while in an irreversible process it is never zero,this is not the case with change in Internal Energy which remains zero for both processes.

3.2 Equations

Entropy equation for an ideal gas can be derived using the first law of thermodynamics

$$dU = dq + dw$$

at constant volume, W = 0
thus

$$dU_v = dq_v = nc_v dT$$

Assuming a monoatomic gas, where only translational energies are present ,

$$c_v = \frac{3}{2}R$$

here, R can be substituted with equation of state for ideal gases and thus

$$dU_v = \frac{3}{2}pV\frac{dT}{T}$$

Integrating the equation gives,

$$\Delta U_v = \frac{3}{2} pV \int_{T_i}^{T_f} \frac{dT}{T} = \frac{3}{2} pV \ln \frac{T_f}{T_i}$$

Employing equation of state in the above equation,

$$\Delta U_v = nc_v T ln \frac{T_f}{T_i}$$

This equation contains the expression of entropy for constant volume.

$$\Delta S_v = nc_v ln \frac{T_f}{T_i}$$

Internal Energy then,

$$\Delta U_v = \Delta q_v = T \Delta S_v$$

At constant temperature the heat term is zero,

$$dU_T = dw = -pdV$$

Substituting equation of state and integrating the above equation we get,

$$\Delta U_T = -nRT \int_{V_i}^{V_f} \frac{dV}{V} = -nRT ln \frac{V_f}{V_i}$$

Hence, the expression of entropy for constant temperature

$$\Delta S_T = nR ln \frac{V_f}{V_i}$$

Thus, internal energy becomes,

$$\Delta U_T = \Delta w = -T \Delta S_T$$

$$\Delta U_T = \Delta q_v + \Delta w = \Delta U_v + \Delta U_T = T \Delta S_v - T \Delta S_T = T(\Delta S_V - \Delta S_T)$$

$$\Delta Q = \Delta q_v - \Delta w = T \Delta S_v - (-T \Delta S_T) = T(\Delta S_v + \Delta S_T)$$

Summing the two parts,

$$\Delta S = \Delta S_v + \Delta S_T$$

Heat equation then,

$$\Delta Q = T \Delta S$$

Finally,

$$\Delta S = \frac{\Delta Q}{T}$$

3.3 Development

The equation is basically obtained from Maxwell's relations

$$Sm(T,P) = Sm(T_0,P_0) + \int_{T_0}^{T} \frac{Cpm}{T} dT - Rln(P/P_0)$$

$$Cp_m = Cp_0 + Cp_1 * T + Cp_2 * T^2 + Cp_3 * T^3$$

The routine of this equation was basically referred from the DWSIM source code, which is available on GitHub as well as on Visual Studio. The code previously developed on OpenModelica had two separate equations for entropy. The equations involved are liquid phase entropy and gas phase entropy out of which the liquid phase entropy had a correction term missing. After searching through the code of entropy function in visual basic we across a term which involved liquid density as well as temperature and pressure. This term was added to the previous equation of liquid phase entropy equation and there was a significant improvement in the accuracy of the values obtained.

3.4 Table

Compound	**OpenModelica (kJ/kg K)**	**DWSIM (kJ/kg K)**	**Error Percentage**
Benzene - Toluene	-1.32887	-1.3284	**0.04%**
Acetone - 1-Propanol	-2.10373	-2.1037	**0.00%**
N-Octane - N-Nonane	-1.14877	-1.14862	**0.01%**

Chapter 4

Extended UNIQUAC

4.1 Description

Extended UNIQUAC (Universal QuasiChemical) is a thermodynamic (activity coefficient) model, specially designed for electrolyte solutions (both aqueous solutions and mixed solvents). The model has proven itself applicable for calculations of vapor-liquid-liquid-solid equilibria and of thermal properties in aqueous solutions containing electrolytes and non-electrolytes. The extended UNIQUAC model consists of three terms: a combinatorial or entropic term, a residual or enthalpic term and an electrostatic term. The combinatorial and the residual terms are identical to the terms used in the traditional UNIQUAC equation (ref 3 and 4). The electrostatic term corresponds to the extended Debye-Hückel law. The combinatorial, entropic term is independent of temperature and only depends on the relative sizes of the species.

4.2 Equations

As said earlier, the extended UNIQUAC model consists of three terms: a combinatorial or entropic term, a residual or enthalpic term and an electrostatic term. They are:

$$G^{ex} = G^{ex}_{combinatorial} + G^{ex}_{residual} + G^{ex}_{ExtendedDebye-Huckel}$$

$$\frac{G^{ex}_{combinatorial}}{RT} = \sum x_i ln \frac{\phi_i}{x_i} - \frac{z}{2} \sum q_i x_i ln \frac{\phi_i}{\theta_i}$$

$$\phi_i = \frac{x_i r_i}{\sum_j x_j r_j}; \theta_i = \frac{x_i q_i}{\sum_j x_j q_j}$$

$$\frac{G^{ex}_{residual}}{RT} = -\sum_i (x_i q_i ln(\sum_j \theta_j \psi_{ji}))$$

$$\psi_{ji} = \exp^{-(u_{ji}-u_{ii})/T}$$

$$u_{ji} = u_{ij} = u_{ij}^0 + u_{ij}^T(T - 298.15)$$

$$ln\gamma_i^C = ln\frac{\phi_i}{x_i} + 1 - \frac{\phi_i}{x_i} - \frac{z}{2}q[ln\frac{\phi_i}{\theta_i} + 1 - \frac{\phi_i}{\theta_i}]$$

$$ln\gamma_i^R = q_i[1 - ln(\sum_j \theta_j\psi_{ji}) - \sum_j \frac{\theta_i\phi_{ij}}{\sum_k \theta_k\psi_{kj}}]$$

$$ln\gamma_i^{C\infty} = ln\frac{r_i}{r_w} + 1 - \frac{r_i}{r_w} - \frac{z}{2}q_i[ln\frac{r_iq_w}{r_wq_i} + 1 - \frac{r_iq_w}{r_wq_i}]$$

$$ln\gamma_i^{R\infty} = q_i[1 - ln\psi_{wi} - \psi_{iw}]$$

$$ln\gamma_w^{DH} = \frac{2}{3}M_wAI^{\frac{3}{2}}\sigma(bI^{\frac{1}{2}})$$

$$\sigma(x) = \frac{3}{x^3}1 + x - \frac{1}{1+x} - 2ln(1+x)$$

$$ln\gamma_i^{*DH} = -Z_i^2\frac{A\sqrt{I}}{1+b\sqrt{I}}$$

$$A = [1.131 + 1.335 * 10^{-03}(T - 273.15) + 1.164 * 10^{-05}(T - 273.15)^2](kg/kmol)$$

$$ln\gamma_w = ln\gamma_w^C + ln\gamma_w^R + ln\gamma_w^{DH}$$

$$ln\gamma_i^* = ln\frac{\gamma_i^C}{\gamma_i^{C\infty}} + ln\frac{\gamma_i^R}{\gamma_i^{R\infty}} + ln\gamma_i^{*DH}$$

4.3 Development

This model is quite similar to the UNIQUAC model when we consider the residual and combinatorial terms in the model, and in addition to that we have used an electrostatic term which is also called as Debye-Huckel Law term. This term is used for calling out the interaction parameters of the ionic elements which will be used during the simulation for instance a two component system containing $water - Na^+ or Na^+ - I^-$ for which interaction parameters are completely different from that of other compounds(organic mostly) and thus the need of Debye-Huckel Law arises.
If at all the compounds contain organic/water elements then the system would basically call out the UNIQUAC interaction parameters. Thus, in OpenModelica we will use the UNIQUAC interaction parameters only as for now since we don't have the ionic element database and their corresponding parameters.Here, the mixture will comprise of water and organic compound because of which the interaction parameters of the mixture will become zero, this will cause residual term in the activity

coefficient to become zero, as of now we are not using ionic compounds thus the Debye-Huckel term will also cease to exist. Hence, we will be left with only combinatorial term which will give us the activity coefficient and hence the fugacity coefficient.

4.4 Table

	Activity Coefficient		**Fugacity Coefficient**		
	OpenModelica	**DWSIM**	**OpenModelica**	**DWSIM**	**Error %age**
Methanol	1.04062	1.0694536	1.71366	1.90565	11.20350595
Ethanol	1.07305	1.119691256	1.03971	1.18511	13.9846688
Water	1.25837	1.2065084	0.539727	0.556549	3.116760881

Chapter 5

Conclusion

Entropy function contains liquid and gas entropy equations, out of which the liquid equation was missing the liquid density correction factor which was forced upon in DWSIM, we added that term to the liquid entropy formula and figured out the output values in OpenModelica as well as DWSIM and compared them.

Extended UNIQUAC model is used for electrolytic compounds to obtain activity coefficients and fugacity coefficients respectively. If we use compounds other than ions, provided they contain water we can still use the model. But there will be a difference between the Extended UNIQUAC and UNIQUAC results. This difference is due to the reduction of the residual term in Extended model when we use organic compounds with water and ions are not present.We have implemented the same in the OpenModelica.

There was an erroneous routine followed in the calculation of the residual term in UNIQUAC thermodynamic model which was giving inaccurate values of fugacity coefficients that was also solved.

Reference

- B. Sander; P. Rasmussen and Aa. Fredenslund, "Calculation of Solid-Liquid Equilibria in Aqueous Solutions of Nitrate Salts Using an Extended UNIQUAC Equation". Chemical Engineering Science, 41(1986)1197-1202
- Thomsen, K., Aqueous electrolytes: model parameters and process simulation, Ph.D. Thesis, Department of Chemical Engineering, Technical University of Denmark, 1997.
- GitHub - Source Code DWSIM

Chapter 6

OpenModelica Code

```
function SId_FuncENTROPY

  import Modelica.Constants.*;

  input Real AS;
  input Real VapCp[6];
  input Real HOV[6];
  input Real Tb;
  input Real Tc;
  input Real T;
  input Real P;
  input Real x;
  input Real y;
  input Real Phase_Density;
  output Real Sliq, Svap;
protected
  parameter Real Tref = 298.15, Pref = 101325;
  Real Entr, Cp[n - 1];
  parameter Integer n = 10;

algorithm
  Entr := 0;
  for i in 1:n - 1 loop
    Cp[i] := Simulator.Files.Thermodynamic_Functions.VapCpId(VapCp, 298.15
        + i * (T - 298.15) / n) / (298.15 + i * (T - 298.15) / n);
  end for;
  if T >= Tref then
    Entr := (T - 298.15) * (Simulator.Files.Thermodynamic_Functions.VapCpId
        (VapCp, T) / (2 * T) + sum(Cp[:]) +
        Simulator.Files.Thermodynamic_Functions.VapCpId(VapCp, 298.15) / (2
         * 298.15)) / n;
  else
    Entr := -(T - 298.15) * (
        Simulator.Files.Thermodynamic_Functions.VapCpId(VapCp, T) / (2 * T)
         + sum(Cp[:]) + Simulator.Files.Thermodynamic_Functions.VapCpId(
        VapCp, 298.15) / (2 * 298.15)) / n;
  end if;
  if x > 0 and y > 0 then
    Sliq := Entr - R * log(P / Pref) - R * log(x) - HV(HOV, Tc, T) / T + P
        /1000/Phase_Density/T;
    Svap := Entr - R * log(P / Pref) - R * log(y);
  elseif x <= 0 and y <= 0 then
    Sliq := 0;
    Svap := 0;
  elseif x == 0 then
    Sliq := 0;
```

```
39      Svap := Entr - R * log(P / Pref) - R * log(y);
40    elseif y == 0 then
41      Sliq := Entr - R * log(P / Pref) - R * log(x) - HV(HOV, Tc, T) / T + P
            /1000/Phase_Density/T;
42      Svap := 0;
43    else
44      Sliq := 0;
45      Svap := 0;
46    end if;
47
48  end SId_FuncENTROPY;
```

```
1   model EXT_UNIQUAC
2   //Libraries
3       import Simulator.Files.*;
4       //Parameter Section
5       //Binary Interaction Parameters
6       //Function :BIP_UNIQUAC is used to obtain the interaction parameters
7       parameter Real a[NOC, NOC] = Thermodynamic_Functions.BIP_UNIQUAC(NOC,
            comp.name);
8       //Uniquac Parameters R and Q called from Chemsep Database
9       parameter Real R[NOC] = comp.UniquacR;
10      parameter Real Q[NOC] = comp.UniquacQ;
11      parameter Integer Z = 10 "Compresseblity-Factor";
12      //Variable Section
13      Real tow[NOC, NOC] "Energy interaction parameter";
14      //Intermediate variables to calculate the combinatorial and residual
            part of activity coefficient at the input conditions
15      Real r(each start=2, min=0,max=1), q(each start=2);
16      Real teta[NOC];
17      Real S[NOC](each start = 1);
18      Real sum[NOC];
19      //Activity Coefficients
20      Real gammac[NOC](each start = 1.2) "Combinatorial Part of activity
            coefficent at input conditions";
21      Real gammar[NOC](each start = 1.2) "Residual part of activity
            coefficient at input conditions";
22      Real gamma_new[NOC](each start = 1.2);
23      Real gamma[NOC](each start = 1.2) "Activity coefficient with Poynting
            correction";
24      //Fugacity coefficient
25      Real phil[NOC](each start = 0.5) "Fugacity coefficient at the input
            conditions";
26      //Dew Point Calculation Variables
27     // Real dewLiqMolFrac[NOC](each start=0.5, each min=0, each max=1);
28      //Intermediate variables to calculate the combinatorial and residual
            part of activity coefficient at dew point
29     // Real r_dew(start=2), q_dew(start=2);
30     // Real teta_dew[NOC](each start=2);
31    //  Real S_dew[NOC](each start = 1);
32     // Real sum_dew[NOC](each start=2);
33      //Activity Coefficients
34     // Real gammac_dew[NOC](each start = 5) "Combinatorial Part of activity
           coefficent at dew point";
35     // Real gammar_dew[NOC](each start = 2.5) "Residual part of activity
           coefficient at dew point";
36    //  Real gammaDew_old[NOC](each start = 2.2) "Combinatorial Part of
          activity coefficent(without correction)";
37      Real gammaDew[NOC](each start = 2.2) "Activity coefficent at dew point"
            ;
38      //Fugacity coefficient
39      Real vapfugcoeff_dew[NOC] "Vapour Fugacity coefficient at dew point";
40    //  Real phil_dew[NOC](each start = 0.5);
```

```
41      // Real PCF_dew[NOC] "Poynting Correction Factor";
42       //Bubble Point Calculation Variables
43       //Intermediate variables to calculate the combinatorial and residual
             part of activity coefficient at bubble point
44      // Real r_bubl(start=2), q_bubl(start=2);
45      // Real teta_bubl[NOC];
46      // Real S_bubl[NOC];
47      // Real sum_bubl[NOC];
48       //Activity Coefficients
49      // Real gammac_bubl[NOC](each start = 2) "Combinatorial Part of activity
             coefficent at bubble point";
50      // Real gammar_bubl[NOC](each start = 1) "Residual part of activity
             coefficent at bubble point";
51      // Real gammaBubl_old[NOC](each start = 1) "Combinatorial Part of
             activity coefficent(without correction)";
52      Real gammaBubl[NOC](each start = 1) "Activity coefficent at bubble point
            ";
53       //Fugacity coefficient
54       Real liqfugcoeff_bubl[NOC];
55      //Real phil_bubl[NOC](each start = 0.5) "Liquid Phase Fugacity
             coefficient";
56      // Real PCF_bubl[NOC] "Poynting Correction Factor";
57       //Phase Envelope
58       Real Psat[NOC](each unit = "Pa") "Saturated Vapour Pressure at the
             input temperature";
59       Real PCF[NOC] "Poynting correction factor";
60       Real K[NOC](each start = 0.7) "Distribution Coefficient";
61       //Residual Energy Parameters
62       Real resMolSpHeat[3], resMolEnth[3], resMolEntr[3];
63       //Transport Properties at the input conditions
64       Real Density[NOC](each unit = "kmol/m^3");
65       Real A[NOC],B[NOC],D[NOC],E[NOC],Ff[NOC];
66       Real C[NOC];
67     //   Real A_bubl[NOC], B_bubl[NOC], C_bubl[NOC], D_bubl[NOC], E_bubl[NOC],
           F_bubl[NOC];
68     //   Real A_dew[NOC], B_dew[NOC], C_dew[NOC], D_dew[NOC], E_dew[NOC], F_dew[NOC
           ];
69       //
             ==========================================================================

70       //Equation Section
71    equation
72  //Fugacity coefficients set to 1 since the model type is Activity
       Coefficient
73       for i in 1:NOC loop
74             gammaBubl[i] = 1;
75             gammaDew[i] = 1;
76         liqfugcoeff_bubl[i] = 1;
77         vapfugcoeff_dew[i] = 1;
78       end for;
79
80
81
82  //Calculation of Intermediate parameters to evaluate combinatorial and
       residual part of the activity coefficient
83  //Note : compMolFrac is the referenced from "Material Stream" model
84
85       r = sum(compMolFrac[2, :] .* R[:]);
86       q = sum(compMolFrac[2, :] .* Q[:]);
87
88       //Calculation of Energy interaction parameter at the input tempetraure
89  //Function :Tow_UNIQUAC is used to instantiated
90       tow = Simulator.Files.Thermodynamic_Functions.Tow_UNIQUAC(NOC, a, T);
91  //Calculation of Combinatorial and Residual Activity coefficient
```

```
92
93      for i in 1:NOC loop
94      if(q>0) then
95      teta[i] = compMolFrac[2, i] * Q[i] * (1 / q);
96      elseif(q<0) then
97      teta[i]=0;
98      else
99      teta[i]=0;
100     end if;
101     end for;
102
103     for i in 1:NOC loop
104       if  (teta[i]==0) then
105       S[i]=1;
106       else
107       S[i] = sum(teta[:] .* tow[i, :]);
108       end if;
109
110       if(S[i]==1) then
111       sum[i]=sum(teta[:] .* tow[i,:]);
112       else
113       sum[i] = sum(teta[:] .* tow[i, :] ./ S[:]);
114       end if;
115     end for;
116
117     for i in 1:NOC loop
118
119       if(S[i]==1) then
120       C[i] = 0;
121       elseif(S[i]>0 and S[i]<>1) then
122       C[i] = log(S[i]);
123       else
124       C[i]=0;
125       end if;
126
127      (gammar[i]) = exp(Q[i] * (1 - C[i] - sum[i]));
128     end for;
129 // //=============================================================================
130 //      equation
131
132      for i in 1:NOC loop
133      if(r>0) then
134      D[i] = R[i]/r;
135      elseif(r<=0) then
136      D[i] =0;
137      else
138      D[i]=0;
139      end if;
140
141       if(q>0) then
142       E[i] = Q[i]/q;
143       elseif(q<=0) then
144       E[i] = 0;
145       else
146       E[i] = 0;
147       end if;
148
149       if(E[i]==0 or D[i]==0) then
150       Ff[i]=0;
151       else
152       Ff[i] = D[i]/E[i];
153       end if;
154
155
```

```
156         if(D[i]>0) then
157         A[i] =log(D[i]);
158         elseif(D[i]==1) then
159         A[i]=0;
160         else
161         A[i]=0;
162         end if;
163
164         if(Ff[i]>0.5) then
165         B[i] =log(Ff[i]);
166         elseif(Ff[i]==1) then
167         B[i]=0;
168         else
169         B[i]=0;
170         end if;
171
172         log(gammac[i])=1-D[i] + A[i] + (-Z / 2 * Q[i] * (1 - Ff[i] + B[i]));
173
174       (gamma[i]) = (gammac[i]) ;// (gammar[i]);
175     end for;
176 //  ==========================================================================

177 //Excess Energy parameters are set to 0 since the calculation mode is Ideal
178     resMolSpHeat[:] = zeros(3);
179     resMolEnth[:] = zeros(3);
180     resMolEntr[:] = zeros(3);
181 //Calculation of Saturated vapour pressure and Density at the given input
        condition
182     for i in 1:NOC loop
183       Psat[i] = Simulator.Files.Thermodynamic_Functions.Psat(comp[i].VP, T)
            ;
184       Density[i] = Simulator.Files.Thermodynamic_Functions.Dens(comp[i].
            LiqDen, comp[i].Tc, T, P) * 1E-3;
185     end for;
186 //Calculation of Poynting correction Factor at input conditions,Bubble
        Point and Dew Point
187 //Function :Poynting_CF is called from the Simulator Package
188     PCF[:] = Thermodynamic_Functions.PoyntingCF(NOC, comp[:].Pc, comp[:].
            Tc, comp[:].Racketparam, comp[:].AF, comp[:].MW, T, P, gamma[:],
            Psat[:], Density[:]);
189     phil[:] = gamma[:] .* Psat[:] ./ P .* PCF[:];
190     phil[:] = gamma_new[:] .* Psat[:] ./ P;
191       K[:] = gamma_new[:] .* Psat[:] ./ P;
192 end EXT_UNIQUAC;
```

```
1  model Mat_Stream
2
3    //1 -  Mixture, 2 - Liquid phase, 3 - Gas Phase
4    extends Modelica.Icons.SourcesPackage;
5    import Simulator.Files.*;
6    parameter Integer NOC;
7    parameter Simulator.Files.Chemsep_Database.General_Properties comp[NOC];
8    Real P(min = 0, start = 101325) "Pressure", T(start = 273) "Temperature";
9    Real Pbubl(min = 0, start = sum(comp[:].Pc) / NOC) "Bubble point pressure
         ", Pdew(min = 0, start = sum(comp[:].Pc) / NOC) "dew point pressure";
10   Real liqPhasMolFrac(start = 0.5, min = 0, max = 1) "Liquid Phase mole
         fraction", vapPhasMolFrac(start = 0.5, min = 0, max = 1) "Vapor Phase
          mole fraction", liqPhasMasFrac(start = 0.5, min = 0, max = 1) "
         Liquid Phase mass fraction", vapPhasMasFrac(start = 0.5, min = 0, max
          = 1) "Vapor Phase Mass fraction";
11   Real totMolFlo[3](each min = 0, each start = 100) "Total molar flow",
         totMasFlo[3](each min = 0, each start = 100) "Total Mass Flow", MW
```

```
        [3](each start = 0, each min = 0) "Average Molecular weight of Phases
        ";
12    Real compMolFrac[3, NOC](each min = 0, each max = 1, each start = 1 / (
        NOC + 1)) "Component mole fraction", compMasFrac[3, NOC](each start =
         1 / (NOC + 1), each min = 0, each max = 1) "Component Mass fraction"
        , compMolFlo[3, NOC](each start = 100, each min = 0) "Component Molar
         flow", compMasFlo[3, NOC](each min = 0, each start = 100) "Component
         Mass Fraction";
13    Real phasMolSpHeat[3] "phase Molar Specific Heat", compMolSpHeat[3, NOC]
        "Component Molar Specific Heat";
14    Real phasMolEnth[3] "Phase Molar Enthalpy", compMolEnth[3, NOC] "
        Component Molar Enthalpy";
15    Real phasMolEntr[3] "Phase Molar Entropy", compMolEntr[3, NOC] "Component
         Molar Entropy";
16    Real Liquid_Phase_Density;
17    Real LiqDens[NOC];
18    Simulator.Files.Connection.matConn inlet(connNOC = NOC) annotation(
19      Placement(visible = true, transformation(origin = {-100, 0}, extent =
          {{-10, -10}, {10, 10}}, rotation = 0), iconTransformation(origin =
          {-100, 0}, extent = {{-10, -10}, {10, 10}}, rotation = 0)));
20    Simulator.Files.Connection.matConn outlet(connNOC = NOC) annotation(
21      Placement(visible = true, transformation(origin = {100, 0}, extent =
          {{-10, -10}, {10, 10}}, rotation = 0), iconTransformation(origin =
          {100, 0}, extent = {{-10, -10}, {10, 10}}, rotation = 0)));
22  equation
23  //Connector equations
24    inlet.P = P;
25    inlet.T = T;
26    inlet.mixMolFlo = totMolFlo[1];
27    inlet.mixMolEnth = phasMolEnth[1];
28    inlet.mixMolEntr = phasMolEntr[1];
29    inlet.mixMolFrac = compMolFrac;
30    inlet.vapPhasMolFrac = vapPhasMolFrac;
31    outlet.P = P;
32    outlet.T = T;
33    outlet.mixMolFlo = totMolFlo[1];
34    outlet.mixMolEnth = phasMolEnth[1];
35    outlet.mixMolEntr = phasMolEntr[1];
36    outlet.mixMolFrac = compMolFrac;
37    outlet.vapPhasMolFrac = vapPhasMolFrac;
38  //
      ================================================================================

39  //Mole Balance
40    totMolFlo[1] = totMolFlo[2] + totMolFlo[3];
41    compMolFrac[1, :] .* totMolFlo[1] = compMolFrac[2, :] .* totMolFlo[2] +
        compMolFrac[3, :] .* totMolFlo[3];
42  //component molar and mass flows
43    for i in 1:NOC loop
44      compMolFlo[:, i] = compMolFrac[:, i] .* totMolFlo[:];
45    end for;
46    if P >= Pbubl then
47  //below bubble point region
48      compMasFrac[3, :] = zeros(NOC);
49      compMasFlo[1, :] = compMasFrac[1, :] .* totMasFlo[1];
50      compMasFrac[2, :] = compMasFrac[1, :];
51    elseif P >= Pdew then
52      for i in 1:NOC loop
53        compMasFlo[:, i] = compMasFrac[:, i] .* totMasFlo[:];
54      end for;
55    else
56  //above dew point region
57      compMasFrac[2, :] = zeros(NOC);
58      compMasFlo[1, :] = compMasFrac[1, :] .* totMasFlo[1];
```

```
59      compMasFrac[3, :] = compMasFrac[1, :];
60    end if;
61  //phase molar and mass fractions
62    liqPhasMolFrac = totMolFlo[2] / totMolFlo[1];
63    vapPhasMolFrac = totMolFlo[3] / totMolFlo[1];
64    liqPhasMasFrac = totMasFlo[2] / totMasFlo[1];
65    vapPhasMasFrac = totMasFlo[3] / totMasFlo[1];
66  //Conversion between mole and mass flow
67    for i in 1:NOC loop
68      compMasFlo[:, i] = compMolFlo[:, i] * comp[i].MW;
69    end for;
70    totMasFlo[:] = totMolFlo[:] .* MW[:];
71  //Energy Balance
72    for i in 1:NOC loop
73  //Specific Heat and Enthalpy calculation
74      compMolSpHeat[2, i] = Thermodynamic_Functions.LiqCpId(comp[i].LiqCp, T)
            ;
75      compMolSpHeat[3, i] = Thermodynamic_Functions.VapCpId(comp[i].VapCp, T)
            ;
76      compMolEnth[2, i] = Thermodynamic_Functions.HLiqId(comp[i].SH, comp[i].
            VapCp, comp[i].HOV, comp[i].Tc, T);
77      compMolEnth[3, i] = Thermodynamic_Functions.HVapId(comp[i].SH, comp[i].
            VapCp, comp[i].HOV, comp[i].Tc, T);
78      (compMolEntr[2, i], compMolEntr[3, i]) = Thermodynamic_Functions.SId(
            comp[i].AS, comp[i].VapCp, comp[i].HOV, comp[i].Tb, comp[i].Tc, T,
            P, compMolFrac[2, i], compMolFrac[3, i], Liquid_Phase_Density);
79    end for;
80    for i in 2:3 loop
81      phasMolSpHeat[i] = sum(compMolFrac[i, :] .* compMolSpHeat[i, :]) +
            resMolSpHeat[i];
82      phasMolEnth[i] = sum(compMolFrac[i, :] .* compMolEnth[i, :]) +
            resMolEnth[i];
83      phasMolEntr[i] = sum(compMolFrac[i, :] .* compMolEntr[i, :]) +
            resMolEntr[i];
84    end for;
85    phasMolSpHeat[1] = liqPhasMolFrac * phasMolSpHeat[2] + vapPhasMolFrac *
          phasMolSpHeat[3];
86    compMolSpHeat[1, :] = compMolFrac[1, :] .* phasMolSpHeat[1];
87    phasMolEnth[1] = liqPhasMolFrac * phasMolEnth[2] + vapPhasMolFrac *
          phasMolEnth[3];
88    compMolEnth[1, :] = compMolFrac[1, :] .* phasMolEnth[1];
89    phasMolEntr[1] = liqPhasMolFrac * phasMolEntr[2] + vapPhasMolFrac *
          phasMolEntr[3];
90    compMolEntr[1, :] = compMolFrac[1, :] * phasMolEntr[1];
91  //Bubble point calculation
92    Pbubl = sum(gammaBubl[:] .* compMolFrac[1, :] .* exp(comp[:].VP[2] + comp
          [:].VP[3] / T + comp[:].VP[4] * log(T) + comp[:].VP[5] .* T .^ comp
          [:].VP[6]) ./ liqfugcoeff_bubl[:]);
93  //Dew point calculation
94    Pdew = 1 / sum(compMolFrac[1, :] ./ (gammaDew[:] .* exp(comp[:].VP[2] +
          comp[:].VP[3] / T + comp[:].VP[4] * log(T) + comp[:].VP[5] .* T .^
          comp[:].VP[6])) .* vapfugcoeff_dew[:]);
95    if P >= Pbubl then
96  //below bubble point region
97      compMolFrac[3, :] = zeros(NOC);
98      sum(compMolFrac[2, :]) = 1;
99    elseif P >= Pdew then
100 //VLE region
101     for i in 1:NOC loop
102       compMolFrac[3, i] = K[i] * compMolFrac[2, i];
103     end for;
104     sum(compMolFrac[3, :]) = 1;
105 //sum y = 1
106   else
```

```
107 //above dew point region
108     compMolFrac[2, :] = zeros(NOC);
109     sum(compMolFrac[3, :]) = 1;
110   end if;
111 algorithm
112   for i in 1:NOC loop
113     MW[:] := MW[:] + comp[i].MW * compMolFrac[:, i];
114   end for;
115    LiqDens[:] := Thermodynamic_Functions.Density_Racket(NOC, T, P, comp[:].
         Pc, comp[:].Tc, comp[:].Racketparam, comp[:].AF, comp[:].MW, Psat
         [:]);
116
117   Liquid_Phase_Density := 1 / sum(compMasFrac[2, :] ./ LiqDens[:]) / MW[2];
118
119 end Mat_Stream;
```

```
1  model UNIQUAC_Correction
2   //Libraries
3     import Simulator.Files.*;
4     //Parameter Section
5     //Binary Interaction Parameters
6     //Function :BIP_UNIQUAC is used to obtain the interaction parameters
7     parameter Real a[NOC, NOC] = Thermodynamic_Functions.BIP_UNIQUAC(NOC,
          comp.name);
8     //Uniquac Parameters R and Q called from Chemsep Database
9     parameter Real R[NOC] = comp.UniquacR;
10    parameter Real Q[NOC] = comp.UniquacQ;
11    parameter Integer Z = 10 "Compresseblity-Factor";
12    //Variable Section
13    Real tow[NOC, NOC] "Energy interaction parameter";
14    //Intermediate variables to calculate the combinatorial and residual
          part of activity coefficient at the input conditions
15    Real r(each start=2, min=0,max=1), q(each start=2);
16    Real teta[NOC];
17    Real S[NOC](each start = 1);
18    Real sum[NOC];
19    //Activity Coefficients
20    Real gammac[NOC](each start = 1.2) "Combinatorial Part of activity
          coefficent at input conditions";
21    Real gammar[NOC](each start = 1.2) "Residual part of activity
          coefficient at input conditions";
22    Real gamma_new[NOC](each start = 1.2);
23    Real gamma[NOC](each start = 1.2) "Activity coefficient with Poynting
          correction";
24    //Fugacity coefficient
25    Real phil[NOC](each start = 0.5) "Fugacity coefficient at the input
          conditions";
26    //Dew Point Calculation Variables
27    Real dewLiqMolFrac[NOC](each start=0.5, each min=0, each max=1);
28    //Intermediate variables to calculate the combinatorial and residual
          part of activity coefficient at dew point
29    Real r_dew(start=2), q_dew(start=2);
30    Real teta_dew[NOC](each start=2);
31    Real S_dew[NOC](each start = 1);
32    Real sum_dew[NOC](each start=2);
33    //Activity Coefficients
34    Real gammac_dew[NOC](each start = 5) "Combinatorial Part of activity
          coefficent at dew point";
35    Real gammar_dew[NOC](each start = 2.5) "Residual part of activity
          coefficient at dew point";
36    Real gammaDew_old[NOC](each start = 2.2) "Combinatorial Part of
          activity coefficent(without correction)";
37    Real gammaDew[NOC](each start = 2.2) "Activity coefficent at dew point"
```

```
        ;
38      //Fugacity coefficient
39      Real vapfugcoeff_dew[NOC] "Vapour Fugacity coefficient at dew point";
40      Real phil_dew[NOC](each start = 0.5);
41      Real PCF_dew[NOC] "Poynting Correction Factor";
42      //Bubble Point Calculation Variables
43      //Intermediate variables to calculate the combinatorial and residual
            part of activity coefficient at bubble point
44      Real r_bubl(start=2), q_bubl(start=2);
45      Real teta_bubl[NOC];
46      Real S_bubl[NOC];
47      Real sum_bubl[NOC];
48      //Activity Coefficients
49      Real gammac_bubl[NOC](each start = 2) "Combinatorial Part of activity
            coefficent at bubble point";
50      Real gammar_bubl[NOC](each start = 1) "Residual part of activity
            coefficent at bubble point";
51      Real gammaBubl_old[NOC](each start = 1) "Combinatorial Part of activity
             coefficent(without correction)";
52      Real gammaBubl[NOC](each start = 1) "Activity coefficent at bubble
            point";
53      //Fugacity coefficient
54      Real liqfugcoeff_bubl[NOC];
55      Real phil_bubl[NOC](each start = 0.5) "Liquid Phase Fugacity
            coefficient";
56      Real PCF_bubl[NOC] "Poynting Correction Factor";
57      //Phase Envelope
58      Real Psat[NOC](each unit = "Pa") "Saturated Vapour Pressure at the
            input temperature";
59      Real PCF[NOC] "Poynting correction factor";
60      Real K[NOC](each start = 0.7) "Distribution Coefficient";
61      //Residual Energy Parameters
62      Real resMolSpHeat[3], resMolEnth[3], resMolEntr[3];
63      //Transport Properties at the input conditions
64      Real Density[NOC](each unit = "kmol/m^3");
65      Real A[NOC],B[NOC],D[NOC],E[NOC],Ff[NOC];
66      Real C[NOC];
67      Real A_bubl[NOC],B_bubl[NOC],C_bubl[NOC],D_bubl[NOC],E_bubl[NOC],F_bubl
            [NOC];
68      Real A_dew[NOC],B_dew[NOC],C_dew[NOC],D_dew[NOC],E_dew[NOC],F_dew[NOC];
69      //
            ================================================================

70      //Equation Section
71    equation
72  //Fugacity coefficients set to 1 since the model type is Activity
        Coefficient
73      for i in 1:NOC loop
74        liqfugcoeff_bubl[i] = 1;
75        vapfugcoeff_dew[i] = 1;
76      end for;
77  //Calculation of Intermediate parameters to evaluate combinatorial and
        residual part of the activity coefficient
78  //Note : compMolFrac is the referenced from "Material Stream" model
79
80      r = sum(compMolFrac[2, :] .* R[:]);
81      q = sum(compMolFrac[2, :] .* Q[:]);
82
83      //Calculation of Energy interaction parameter at the input tempetraure
84  //Function :Tow_UNIQUAC is used to instantiated
85      tow = Simulator.Files.Thermodynamic_Functions.Tow_UNIQUAC(NOC, a, T);
86  //Calculation of Combinatorial and Residual Activity coefficient
87
88      for i in 1:NOC loop
```

```
89      if(q>0) then
90      teta[i] = compMolFrac[2, i] * Q[i] * (1 / q);
91      elseif(q<0) then
92      teta[i]=0;
93      else
94      teta[i]=0;
95      end if;
96      end for;
97
98      for i in 1:NOC loop
99        if  (teta[i]==0) then
100       S[i]=1;
101       else
102       S[i] = sum(teta[:] .* tow[i, :]);
103       end if;
104
105       if(S[i]==1) then
106       sum[i]=0;
107       else
108       sum[i] = sum(teta[:] .* tow[i, :] ./ S[:]);
109       end if;
110     end for;
111
112     for i in 1:NOC loop
113
114       if(S[i]==1) then
115       C[i] = 0;
116       elseif(S[i]>0) then
117       C[i] = log(S[i]);
118       else
119       C[i]=0;
120       end if;
121
122      (gammar[i]) = exp(Q[i] * (1 - C[i] - sum[i]));
123     end for;
124 // //=============================================================================
125 //      equation
126
127      for i in 1:NOC loop
128      if(r>0) then
129      D[i] = R[i]/r;
130      elseif(r<=0) then
131      D[i] =0;
132      else
133      D[i]=0;
134      end if;
135
136       if(q>0) then
137       E[i] = Q[i]/q;
138       elseif(q<=0) then
139       E[i] = 0;
140       else
141       E[i] = 0;
142       end if;
143
144       if(E[i]==0 or D[i]==0) then
145       Ff[i]=0;
146       else
147       Ff[i] = D[i]/E[i];
148       end if;
149
150
151       if(D[i]>0) then
152       A[i] =log(D[i]);
```

```
153         elseif(D[i]==1) then
154         A[i]=0;
155         else
156         A[i]=0;
157         end if;
158
159         if(Ff[i]>1) then
160         B[i] =log(Ff[i]);
161         elseif(Ff[i]==1) then
162         B[i]=0;
163         else
164         B[i]=0;
165         end if;
166
167         log(gammac[i])=1-D[i] + A[i] + (-Z / 2 * Q[i] * (1 - Ff[i] + B[i]));
168
169       (gamma[i]) = (gammac[i]) * (gammar[i]);
170     end for;
171 //  ================================================================================

172 //Excess Energy parameters are set to 0 since the calculation mode is Ideal
173     resMolSpHeat[:] = zeros(3);
174     resMolEnth[:] = zeros(3);
175     resMolEntr[:] = zeros(3);
176 //Calculation of Saturated vapour pressure and Density at the given input
        condition
177     for i in 1:NOC loop
178       Psat[i] = Simulator.Files.Thermodynamic_Functions.Psat(comp[i].VP, T)
            ;
179       Density[i] = Simulator.Files.Thermodynamic_Functions.Dens(comp[i].
            LiqDen, comp[i].Tc, T, P) * 1E-3;
180     end for;
181 //Calculation of Poynting correction Factor at input conditions,Bubble
        Point and Dew Point
182 //Function :Poynting_CF is called from the Simulator Package
183     PCF[:] = Thermodynamic_Functions.PoyntingCF(NOC, comp[:].Pc, comp[:].
            Tc, comp[:].Racketparam, comp[:].AF, comp[:].MW, T, P, gamma[:],
            Psat[:], Density[:]);
184     PCF_bubl[:] = Thermodynamic_Functions.PoyntingCF(NOC, comp[:].Pc, comp
            [:].Tc, comp[:].Racketparam, comp[:].AF, comp[:].MW, T, Pbubl,
            gamma[:], Psat[:], Density[:]);
185     PCF_dew[:] = Thermodynamic_Functions.PoyntingCF(NOC, comp[:].Pc, comp
            [:].Tc, comp[:].Racketparam, comp[:].AF, comp[:].MW, T, Pdew, gamma
            [:], Psat[:], Density[:]);
186 //Calculation of Fugacity coefficient with Poynting correction
187     phil[:] = gamma[:] .* Psat[:] ./ P .* PCF[:];
188     phil[:] = gamma_new[:] .* Psat[:] ./ P;
189 //Calculation of Distribution coefficient
190     K[:] = gamma_new[:] .* Psat[:] ./ P;
191 //Binary Phase Envelope
192 //The same calculation routine is followed at the DewPoint
193 //Dew Point
194     r_dew = sum(dewLiqMolFrac[:] .* R[:]);
195     q_dew = sum(dewLiqMolFrac[:] .* Q[:]);
196     for i in 1:NOC loop
197      if(q_dew==0 or compMolFrac[1,i]==0) then
198      dewLiqMolFrac[i]=0;
199      else
200       dewLiqMolFrac[i] = compMolFrac[1, i] * Pdew / (gammaDew[i] * Psat[i])
            ;
201      end if;
202      if(q_dew==0 or dewLiqMolFrac[i]==0) then
203      teta_dew[i]=0;
```

```
204         else
205          teta_dew[i] = dewLiqMolFrac[i] * Q[i] * (1 / q_dew);
206         end if;
207          if(teta_dew[i]==0) then
208          S_dew[i] =1;
209          else
210          S_dew[i] = sum(teta_dew[:] .* tow[i, :]);
211          end if;
212          end for;
213   //
      ==========================================================================

214
215          for i in 1:NOC loop
216          if(S_dew[i]==1) then
217          sum_dew[i]=0;
218          else
219          sum_dew[i] = sum(teta_dew[:] .* tow[i, :] ./ (S_dew[:]));
220          end if;
221
222
223          if(S_dew[i]==1) then
224          C_dew[i]=0;
225          elseif(S_dew[i]>0) then
226          C_dew[i] =log(S_dew[i]);
227          else
228          C_dew[i]=0;
229          end if;
230
231          (gammar_dew[i]) = exp(Q[i] * (1 - C_dew[i] - sum_dew[i]));
232          end for;
233   //
      ==========================================================================

234
235         for i in 1:NOC loop
236         if(r_dew==0) then
237         D_dew[i] =0;
238         else
239         D_dew[i] = R[i]/r_dew;
240         end if;
241
242          if(q_dew==0) then
243          E_dew[i] = 0;
244          else
245          E_dew[i] = Q[i]/q_dew;
246          end if;
247
248          if(E_dew[i]==0) then
249          F_dew[i]=0;
250          else
251          F_dew[i] = D_dew[i]/E_dew[i];
252          end if;
253
254
255          if(D_dew[i]>0) then
256          A_dew[i] =log(D_dew[i]);
257          elseif(D_dew[i]==1) then
258          A_dew[i]=0;
259          else
260          A_dew[i]=0;
261          end if;
262
263          if(F_dew[i]>0) then
```

```
264         B_dew[i] =log(F_dew[i]);
265         elseif(F_dew[i]==1) then
266         B_dew[i]=0;
267         else
268         B_dew[i]=0;
269         end if;
270
271         log(gammac_dew[i])=1-D_dew[i] + A_dew[i] + (-Z / 2 * Q[i] * (1 -
                F_dew[i] + B_dew[i]));
272
273         (gammaDew_old[i]) = (gammac_dew[i]) * (gammar_dew[i]);
274       end for;
275
276       for i in 1:NOC loop
277       if(Pdew==0) then
278       phil_dew[i]=1;
279       gammaDew[i]=1;
280
281       else
282
283       phil_dew[i] = gammaDew_old[i] .* Psat[i] ./ Pdew .* PCF_dew[i];
284       phil_dew[i] = gammaDew[i] .* Psat[i] ./ Pdew;
285       end if;
286       end for;
287   //The same calculation routine is followed at the Bubble Point
288   //Bubble Point
289       r_bubl = sum(compMolFrac[1, :] .* R[:]);
290       q_bubl = sum(compMolFrac[1, :] .* Q[:]);
291       for i in 1:NOC loop
292       if(compMolFrac[1,i]==0) then
293       teta_bubl[i]=0;
294       else
295         teta_bubl[i] = compMolFrac[1, i] * Q[i] * (1 / q_bubl);
296        end if;
297
298        if(teta_bubl[i]==0) then
299         S_bubl[i] =1;
300         else
301         S_bubl[i] = sum(teta_bubl[:] .* tow[i, :]);
302        end if;
303
304         if(S_bubl[i]==1) then
305         sum_bubl[i]=0;
306         else
307         sum_bubl[i] = sum(teta_bubl[:] .* tow[i, :] ./ S_bubl[:]);
308         end if;
309
310
311        if(S_bubl[i]==1) then
312         C_bubl[i] =0 ;
313         elseif(S_bubl[i]>0) then
314         C_bubl[i]=log(S_bubl[i]);
315         else
316         C_bubl[i]=0;
317         end if;
318         log(gammar_bubl[i]) = Q[i] * (1 - C_bubl[i] - sum_bubl[i]);
319   //    ==========================================================================================

320
321        if(r_bubl==0) then
322        D_bubl[i] =0;
323        else
324        D_bubl[i] = R[i]/r_bubl;
```

```
325       end if;
326
327        if(q_bubl==0) then
328        E_bubl[i] = 0;
329        else
330        E_bubl[i] = Q[i]/q_bubl;
331        end if;
332
333        if(E_bubl[i]==0) then
334        F_bubl[i]=0;
335        else
336        F_bubl[i] = D_bubl[i]/E_bubl[i];
337        end if;
338
339
340        if(D_bubl[i]>0) then
341        A_bubl[i] =log(D_bubl[i]);
342        elseif(D_bubl[i]==1) then
343        A_bubl[i]=0;
344        else
345        A_bubl[i]=0;
346        end if;
347
348        if(F_bubl[i]>0) then
349        B_bubl[i] =log(F_bubl[i]);
350        elseif(F_bubl[i]==1) then
351        B_bubl[i]=0;
352        else
353        B_bubl[i]=0;
354        end if;
355
356        log(gammac_bubl[i])=1-D_bubl[i] + A_bubl[i] + (-Z / 2 * Q[i] * (1 -
               F_bubl[i] + B_bubl[i]));
357
358       (gammaBubl_old[i]) = (gammac_bubl[i]) * (gammar_bubl[i]);
359      end for;
360
361     for i in 1:NOC loop
362     if(Pbubl==0) then
363      phil_bubl[i]=1;
364      gammaBubl[i]=1;
365     else
366      phil_bubl[i] = gammaBubl_old[i] .* Psat[i] ./ Pbubl .* PCF_bubl[i];
367      phil_bubl[i] = gammaBubl[i] .* Psat[i] ./ Pbubl;
368     end if;
369     end for;
370 end UNIQUAC_Correction;
```