Summer Fellowship Report

On

Extending Capabilities of OMChemSim Chemical Library in OpenModelica

Submitted by

AVSS.Praneeth

Under the guidance of

Prof.Kannan M. Moudgalya
Chemical Engineering Department
IIT Bombay

June 30, 2020

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 have reaped benefits from his wisdom, guidance and patience all along.

I would also like to thank my mentors Mr. Priyam Nayak and Rahul AS for providing valuable insight and expertise, as well as assisting me in overcoming the several difficulties I faced during the course of this project. Their advice on the modeling and simulation of unit operations significantly improved the accuracy in results. I would never be able to finish the project without their support

I would also like to show my gratitude to my fellow interns and peers, who were constantly there to clarify my doubts and recommend improvements to the project. Their help and support was of immense value to me.

Contents

Contents **3**

1 Unit Conversions **4**

1.1 Introduction: 4

1.2 How to use: 4

1.3 Syntax to use: 5

2 Pump **7**

2.1 Introduction 7

2.2 Calculations 7

2.3 Algorithm for calculation type -Curves: 9

2.4 Results: 11

3 Barycentric interpolation: **12**

3.1 Introduction: 12

3.2 Calculations and algorithm: 12

4 Valve **14**

4.1 Introduction: 14

4.2 Modelling: 14

4.3 Results: 17

5 Fittings: **18**

5.1 Introduction: 18

5.2 Calculations: 18

5.3 Results: 20

6 Integration of UNIFAC: 21

6.1 Introduction: 21

6.2 Implementation: 21

7 Reduction of Thermodynamic Model: 22

7.1 Introduction: 22

7.2 For Peng Robinson: 22

7.3 For UNIFAC and UNIQUAC: 24

8 OpenModelica Code 26

8.1 Unit conversions 26

8.2 Centrifugal pump 26

8.3 Barycentric Interpolation 30

8.4 Valve 32

8.5 Fittings 34

8.6 Reduction of Thermodynamics 38

8.6.1 Peng Robinson 38

8.6.2 UNIQUAC 41

8.6.3 UNIFAC 43

Chapter 1

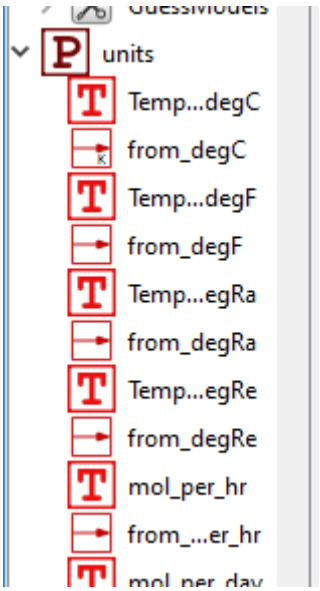
Unit Conversions

1.1 Introduction:

For a simulator all the calculations and correlations are coded in SI units so it's important for a simulator to provide a conversion package so that user can have a flexibility to convert a given parameter unit which is non-SI to a SI unit.

1.2 How to use:

There is a package called Units can be used as unit converters where few ,which are frequently used Non SI units are defined and with unit conversions from that units are defined .As shown in the following figure there the Type is used to define the Non SI units and function which is used to convert



For the following Parameters Unit conversions are provided

- Molar Flow Rate
- Mass Flow Rate
- Volume Flow Rate
- Temperature
- Pressure
- Molar Enthalpy
- Specific Enthalpy
- Molar Entropy
- Specific Entropy
- Density

1.3 Syntax to use:

```
  model Unitconversion
  import SI = Modelica.SIunits;
  import units.*; //importing units package
 Real a= from_atm(1);// to convert 1atm to 101325Pa
end Unitconversion;
```

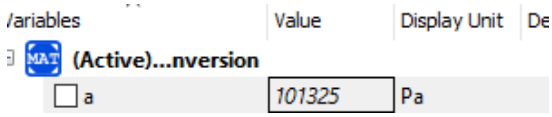
To use the conversion, one has to import the package and call the function with the Non-SI value in it as shown below

To display a Non-SI unit value at output panel:

To get the required unit at the output use the command displayUnit as shown below

```
  model Unitconversion
  import SI = Modelica.SIunits;
  import units.*; //importing units package
 parameter SI.Pressure a(displayUnit="bar")= from_atm(1);// to convert 1atm to 101325Pa
end Unitconversion;
```

at results panel unit conversion is as shown


/ariables
Value
Display Unit
De
(Active)...nversion
a
101325
Pa


(Active)...nversion
a
1.01325
bar

Chapter 2

Pump

2.1 Introduction

In simulations, a centrifugal pump used as a pressure changing device specially to increase the energy of outlet fluid. measured in terms of pressure increase.

2.2 Calculations

There are different methods/modes in which one can specify the specifications of a centrifugal pump.

The following are the modes included in OpenModelica.

1.Outlet pressure

2.Pressure increase

3.Power required

4.using curves

Note: all calculations are based on two main energies, energy supplied fluid and energy supplied to pump

Energy supplied to fluid is the energy required to increase the fluid

Energy supplied to pump is the energy required to overcome loses also so that required energy is supplied to the fluid

Therefore

Efficiency =energy required to fluid /energy supplied to pump;

Energy required to fluid is change in enthalpy of the fluid

Considering pump is operated adiabatically (Qdot =0), applying open system energy balance

$$H2 = H1 + \Delta P/\rho$$

1.Outletpressure:

When outlet pressure is provided the calculations in the code are done as followed

$$\Delta P = P2 - P1$$

$$H2 = H1 + \Delta P/\rho$$

$$\Delta H = H2 - H1$$

$$\text{power requried}\,(Q) = \frac{w\,(\Delta H)}{\eta}$$

2.Pressure increase:

When pressure increase is provided the calculations in the code are done as followed

$$P2 = P1 + \Delta P$$

$$H2 = H1 + \Delta P/\rho$$

$$\Delta H = H2 - H1$$

$$\text{power requried}\,(Q) = \frac{w\,(\Delta H)}{\eta}$$

3.Power supplied:

When Power supplied(Q)is provided the calculations in the code are done as followed

$$\Delta H = \eta/w$$

$$\Delta H * \rho = \Delta P$$

$$P2 = P1 + \Delta P$$

4.curves:

This method involves finding an operating point for give flowrate when the data points are provided. The following data points can be provided

1)Head vs Flowrate

2)Power vs Flowrate

3) efficiency vs Flowrate

4)NPSHR vs Flowrate

5)System head vs Flowrate

Note: Any or all of the data can be provided but providing head is mandatory as all other parameters can be calculated using and are dependent on head

They following is the algorithm used to evaluate the unknows parameters given the data

2.3 Algorithm for calculation type -Curves:

Mode: From curves

1) Read input parameters Pressure (P_i), Temperature (I_t), Mass flow (W)
2) Find density value(rho) from Thermodynamic functions
3) Calculate volumetric flowrate (Q=W/rho)

Finding Operating points:

Head:

Plot Head vs Q

For given flowrate interpolate the head (curve head=head)

NPSH:

If NPSH is enabled

Plot NPSH vs Q

Interpolate NPSH

else

Npshr=0

Efficiency:

If Efficiency(n) is enabled

Plot n vs Q

Interpolate n for given Q

else

n=given value

System Head:

If system head(Hsys) is enabled

Plot Hsys vs Q

Interpolate Hsys for given Q

else

Hsys=Head

$P_2=P_i+ Hsys * 9.81* rho$

$\Delta P=P_2-P_i$

Power:

If Power (Pow) is enabled

Plot Power vs Q

Interpolate Pow for given Q

Else:

$Pow=W* 9.81* Hsys/eff$

$H_2=H_i+power * eff /W$

Run PH-flash to get T_2

$\Delta T=T_2-T_i$

NPSH=(Pi - Pbub) / (rho * 9.81)

Note:1. The interpolation used here is barycentric interpolation.

2. For all the above modes the final temperature is calculated by using HV flash.

2.4 Results:

Mode: Curves

Inlet stream: Water

Input Molar Flow Rate:1.65914E+07

Thermodynamics: Raoults law

variable	DWSIM	Openmodelica	Percentage of error
Pin(Pa)	101325	101325	0
Tin(K)	298.15	298.15	0
Hin(KJ/Kmol)	2440.95	2441.087982	0.005652799
delP(Pa)	123043	125730	0.02183789407
Head(m)	12.6017	12.8641	0.0208230584
Efficiency(%)	50.2223	50.4861	0.525264673
Power(W)	1.54554	1.5673	0.0052527
Pout(Pa)	224368	227055	0.01197586109
Tout(K)	298.15	298.15	0
Hout(KJ/Kmol)	2440.95	2441.087982	0.005652799
NPSH(m)	10.041	10.0424	0.013942834

Chapter 3

Barycentric interpolation:

3.1 Introduction:

In the mathematical field of numerical analysis, **interpolation** is a type of estimation, a method of constructing new data points within the range of a discrete set of known data points.

In engineering and science, one often has a number of data points, obtained by sampling or experimentation, which represent the values of a function for a limited number of values of the independent variable. It is often required to **interpolate**, i.e., estimate the value of that function for an intermediate value of the independent variable.

This interpolation especially in Chemical engineering is useful in many ways like in Pump Model when the data points are giving to obtain the operating point and even in correlations can be used to estimate for intermediate values.

3.2 Calculations and algorithm:

In barycentric interpolation our estimate polynomial is given by

$$f(t) = \frac{\sum_{i=1}^{n} \frac{W_i}{t - x_i\, i}}{\sum_{i=1}^{n} \frac{W_i}{t - x_i}}$$

Where

t= variable (value we want)

W_i=weights of each point

X_i=X coordinates of the given data

Y_i=Y coordinates of the given data

N=number of points

Calculating weights:

For calculation of weight here floaterhormannrationalinterpolant method is used as follows

$$w_k = \sum_{i \in JK} (-1)^i \prod_{j=i, j \neq k}^{i+d} \frac{1}{x_k - x_j}$$

$$JK = \{i \in I : k - d \leq i \leq k\}$$

$$I = \{0, 1 \ldots, n - d\}$$

Chapter 4

Valve

4.1 Introduction:

Introduction: valve in simulator is used for pressure drop purpose. A valve is a device that regulates, directs or controls the flow of a fluid by opening, closing, or partially obstructing various passageways. Valves have many uses, including controlling water for irrigation, industrial uses for controlling processes. Valves are found in almost every industrial process, small valves fitted to washing machines and dishwashers including water and sewage processing, mining, power generation, processing of oil, gas and petroleum, food manufacturing, chemical and plastic manufacturing and many other fields

4.2 Modelling:

In the current valve model that was build one can calculate pressure drop or outlet pressure by giving the following as the input

1.Outlet pressure

2.Pressure drop

3.Kv liquid

4. Kv gas

The following equation are being used in the calculation of the unknown or desired parameters

1.When outlet Pressure or Pressure drop is given as input

$$\Delta P = P1 - P2$$

$$H2 = H1 + \Delta H$$

$$F1 = F2$$

After getting outlet pressure and enthalpy, Running PH flash will give final temperature

$$\Delta T = T2 - T1$$

2.Kv is specified:

The procedure specified in the IEC 60534 standard is applied to determine the valve flow coefficient KV. The relevant device-specific data can be found in the associated data sheets. The equations below are given to allow a preliminary, simplified calculation of the valve flow coefficient to be performed. Please note that these equations do not take account of the influence of the connection fittings and the flow limitation in case of critical flow velocities.

Note: The following will work only if ($\Delta P < P1/2$

If $K_v$liq is specified then

where

K_v= Valve flow coefficient

W=Mass flow rate in Kg/hr

ρ = Density in kg/m³

Δ P= Pressure difference in Bar

If K_v gas is specified then

where

K_v= Valve flow coefficient

W=Mass flow rate in Kg/hr

ρ_g = Density of gas in kg/m³ at $^{\circ}Candat1atm$

Δ P= Pressure difference in Bar

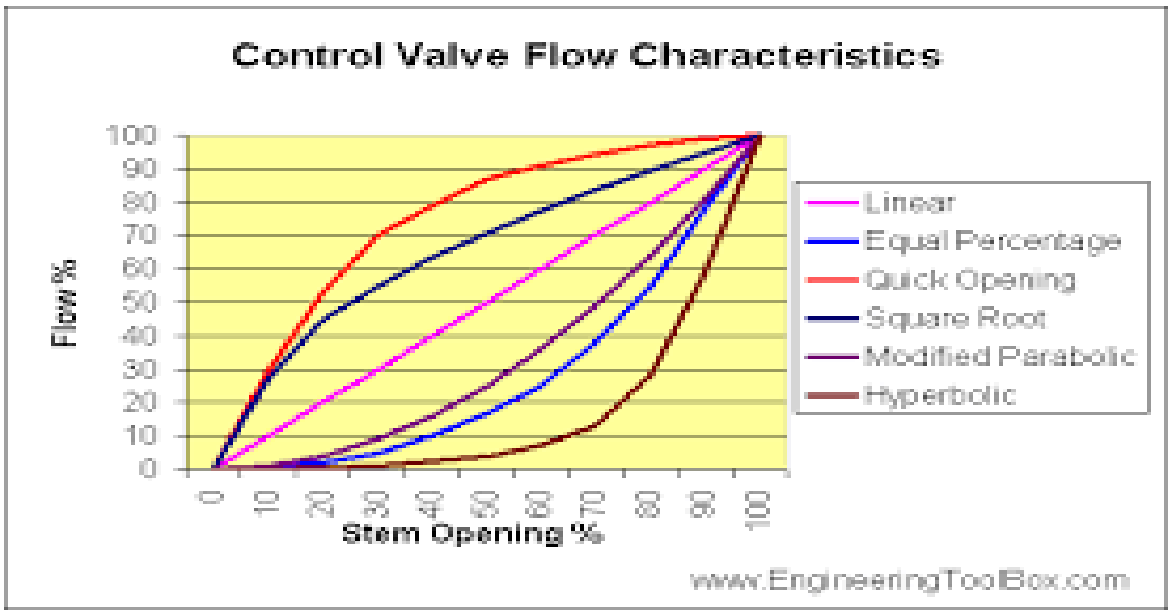
P_2= Outlet Pressure

And in addition to above case if one specifies Valve opening percentage and expression for K_v/K_{vmax} as a function of opening of valve percentage then Kv is updated as follows

$$\frac{K_v}{K_{vmax}} = x\,(OP\%)^y$$

The above expression is the equation which represents the general form

Where OP% is opening percentage of valve


Control Valve Flow Characteristics
Flow %
100
90
80
70
60
50
40
30
20
10
0
0
10
20
30
40
50
60
70
80
90
100
Stem Opening %
Linear
Equal Percentage
Quick Opening
Square Root
Modified Parabolic
Hyperbolic
www.EngineeringToolBox.com

4.3 Results:

Mode-Gas service

Inlet stream-Nitrogen

Thermodynamics: Raoults law

Variable	DWSIM	OpenModelica	Percentage of Error
Pin(Pa)	101325	101325	0
Tin(K)	298.15	298.15	0
deltaP(Pa)	25079.3	25077.6	-0.006778499
Pout(Pa)	76245.7	76247.4	0.002229634
Tout(K)	298.15	298.15	0
Kv	245	245	0
Fin(Mol/s)	35.6972	35.6972	0

Mode-liquid service

Inlet stream-Water

Thermodynamics: Raoults law

Variable	DWSIM	OpenModelica	Percentage of Error
Pin(Pa)	101325	101325	0
Tin(K)	298.15	298.15	0
deltaP(Pa)	25424.6	25424.5	-0.00039332
Pout(Pa)	75900.4	75900.5	0
Tout(K)	298.156	298.15	-0.002012369
Kv	7.152777	7.152777	1061033.319
Fin(Mol/s)	55.0584	55.0584	0

Chapter 5

Fittings:

5.1 Introduction:

A **fitting** or **adapter** is used in pipe systems to connect straight sections of pipe or tube, adapt to different sizes or shapes, and for other purposes such as regulating (or measuring) fluid flow. These fittings are used in plumbing to manipulate the conveyance of water, gas, or liquid waste in domestic or commercial environments, within a system of pipes or tubes.

5.2 Calculations:

There are many Fittings included in the model designed as mentioned below

Depending on the type of the fitting the following formulas are used to calculate the pressure drop across the fitting's

$$\Delta P_f = Constant1 * \left(0.0101^{-0.2232}\right) * \left(\frac{Qliq*\rho liq}{Q}\right) * \left(\frac{Qvap*\rho vap}{Q}\right) * \frac{(Vliq+Vgas)^2}{2}$$

Or

$$\Delta P_f = Constant1 * \left(\frac{Qliq*\rho liq}{Q}\right) * \left(\frac{Qvap*\rho vap}{Q}\right) * \frac{(Vliq+Vgas)^2}{2}$$

Where,

Constant1 depends on the type of fitting

Q= total volumetric flow rate of fluid flowing

Qliq= Volumetric flow rate of liquid phase flowing

Qvap=Volumetric flow rate of vapour phase flowing

ρvap =density of vapour phase at given conditions

ρliq = density of liquid phase at given conditions

Vliq= velocity of liquid phase

Vgas= velocity of gas phase

The following fittings are included

- Elbow 45^0
- Elbow 90^0
- Elbow 180^0
- Angle Valve
- Butterfly Valve
- Ball Valve
- Gate Valve
- Globe Valve
- Lift-Check Valve
- Poppet Disk Valve
- Check Valve(Swing)
- Check Valve (Ball)
- Tee (Branch Blanked)
- Tee (Elbow)
- Quick Reduction(d/D=1/2)
- Quick Reduction (d/D=1/4)
- Quick Reduction (d/D=3/4)
- Border Inlet
- Normal Inlet
- Quick Expansion(d/D=1/2)
- Quick Expansion(d/D=1/4)
- Quick Expansion(d/D=3/4)
- Bend 90^0
- Normal Reduction (2:1)
- Normal Reduction (4:3)
- Border Exit
- Normal Exit
- Fixed Delta P

5.3 Results:

Type: Butterfly valve

Molar flow 50kg/s

Thermodynamics: Raoults law

Variable	DWSIM	OpenModelica	Error
Pin(Pa)	101325	101325	0
Tin(K)	298.15	298.15	0
Xin of N2	0.6	0.6	0
Xin of H20	0.4	0.4	0
Pout(Pa)	58234.4	57137.3	-1.883938016
Pdel(Pa)	43090.6	44187.7	2.546030921
Tout(K)	298.15	292.787	-1.798759014
Inside diameter(m)	0.06	0.06	0

Type: gatevalve

Molar flow rate :50kg/s

Thermodynamics: Raoults law

Variable	DWSIM	OpenModelica	Error
Pin(Pa)	101325	101325	0
Tin(K)	298.15	298.15	0
Xin of N2	0.6	0.6	0
Xin of H20	0.4	0.4	0
Pout(Pa)	92706.9	92487.5	-0.236659839
Pdel(Pa)	8618.13	8837.55	2.546027967
Tout(K)	298.15	297.331	-0.274693946
Inside diameter(m)	0.06	0.06	0

Chapter 6

Integration of UNIFAC:

6.1 Introduction:

In previous version of the UNIFAC there is manual input for the unifac subgroup's Q and R value and interaction parameter. In this all the Parameters for all the compounds are updated with the data base which is available so that the automating input is possible .Here the discussion is not about unifac model but how unifac is integrated and Condensed for fast compilation and to decrease the load on the model

6.2 Implementation:

1) Integrating Interaction Parameters to Chemsepdatabase :

There are 2 groups Main group and subgroup and these variables are initialized in General Properties present in Chemsepdatabase with variable names

UNIFAC_SubGroup[5,2]; (of size 5 X 2)

UNIFAC_MainGroup[5,2];(of size 4 X 2)

And all the Constants for given compound is given in respective compound likewise for all compounds

Chapter 7

Reduction of Thermodynamic Model:

7.1 Introduction:

In all thermodynamics model, gamma at given conditions, Bubble point and dew point needs to be calculated. For calculation of these gamma values 3 different but same routine is followed with only few changes which takes a considerable amount of time so to decrease the load in the code and to increase the ease of execution of the code the follows changes have been done to the code for each thermodynamic model.

7.2 For Peng Robinson:

Working Before Optimisation:

In Peng Robinson previously was straight forward means first Phi liquid and Phi gas is calculated and then K_c (distribution coefficient) is calculated which is further used in calculation of material stream.

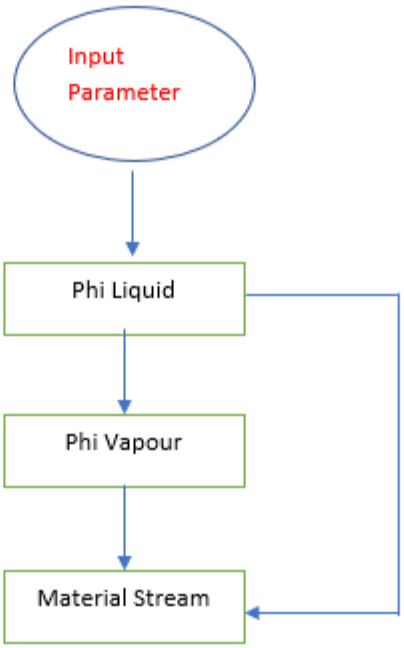
Working After Optimisation:

After optimisation, all the parameters which is used in the calculation of the material stream are brought to main model and Phi calculations which follow same routine is generalised and called twice in main model and the generalised routine is there in secondary model.

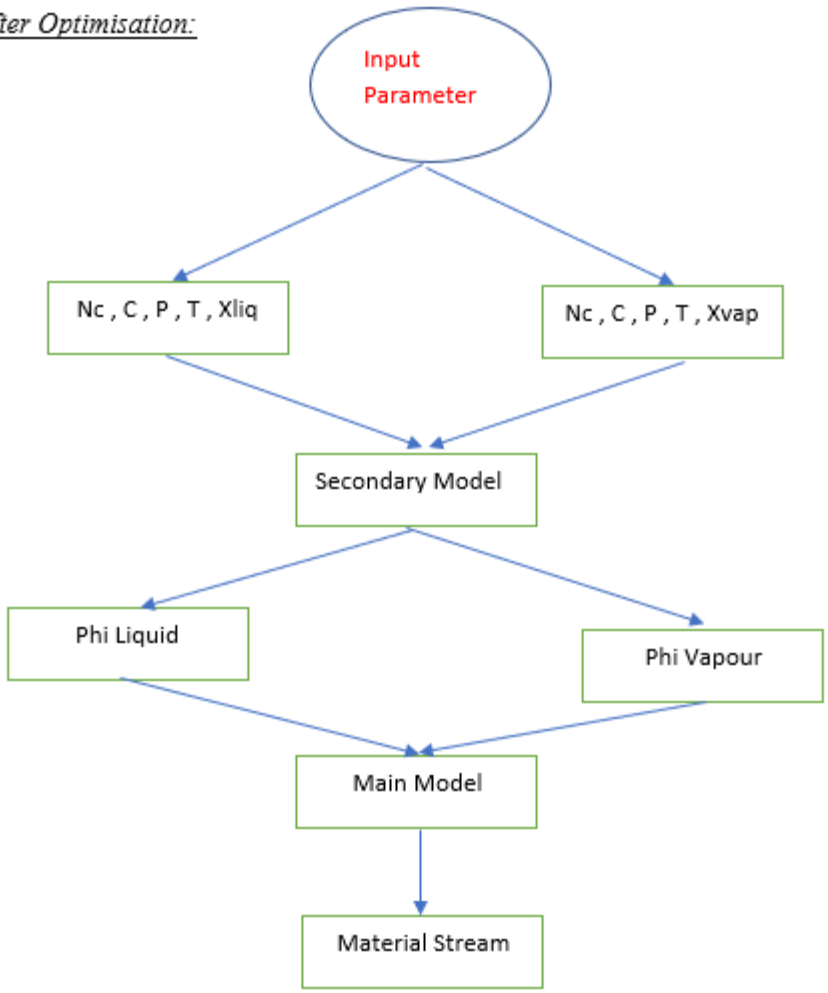
Algorithm is as shown below.

Flowsheet of Algorithm:

Before Optimisation:



After Optimisation:



7.3 For UNIFAC and UNIQUAC:

Working Before Optimisation:

In both UNIFAC and UNIQUAC previously it was a straight forward calculations, means first Gma at given conditions, Gma at dew and bubble point are calculated. liquid and then K_c (distribution coefficient) is calculated which is further used in calculation of material stream.

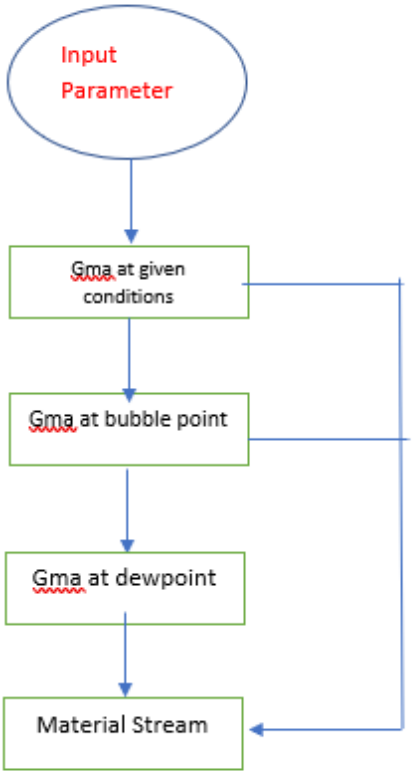
Working After Optimisation:

After optimisation, all the parameters which is used in the calculation of the material stream are brought to main model and Gma calculations which follow same routine is generalised and called thrice in main model and the generalised routine is there in secondary model.

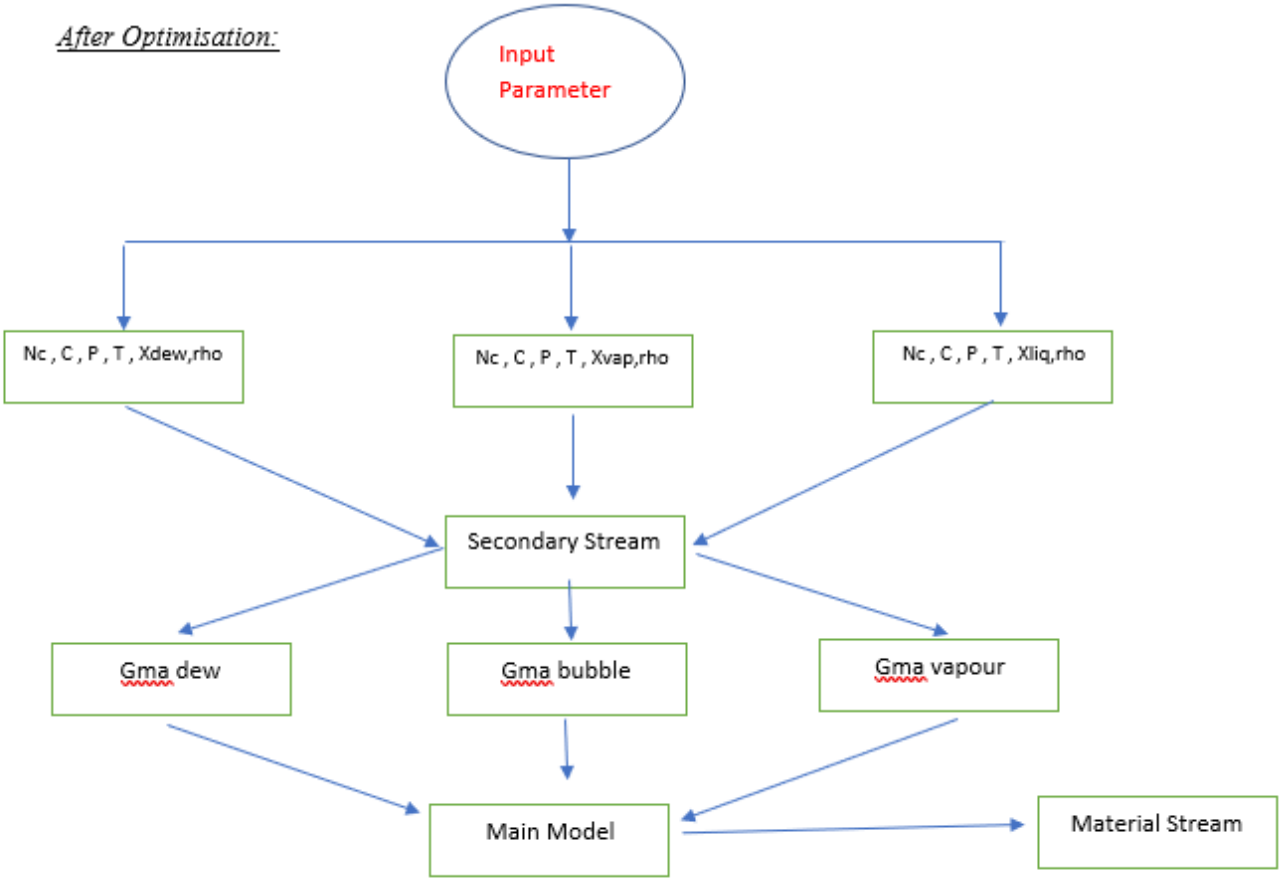
Algorithm is as shown below.

Flow sheet of Algorithm :

Before Optimisation:



After Optimisation:



Chapter 8

OpenModelica Code

8.1 Unit conversions

```
package UnitConversions
 type Temperature_degC = Real(final quantity = "ThermodynamicTemperature", final
     unit = "degC");

  function from_degC "Convert from degCelsius to Kelvin"
    extends Modelica.SIunits.Icons.Conversion;
    import Modelica.SIunits.Conversions.*;
    import SI = Modelica.SIunits;
    input units.Temperature_degC Celsius "Celsius value";
    output SI.Temperature Kelvin "Kelvin value";
  algorithm
    Kelvin := Celsius + 273.15;

  end from_degC;
end UnitConversions;
```

Note:only conversion of one unit is shown as every other conversion follows same routine

8.2 Centrifugal pump

```
within Simulator.UnitOperations;

model CentrifugalPumpnew "Model of a centrifugal pump to provide energy to liquid
    stream in form of pressure"
  //=============================================================================
  //Header files and Parameters
  extends Simulator.Files.Icons.CentrifugalPump;
  parameter Simulator.Files.ChemsepDatabase.GeneralProperties C[Nc] "Component
      instances array" annotation(
    Dialog(tab = "Pump Specifications", group = "Component Parameters"));
  parameter Integer Nc "Number of components" annotation(
    Dialog(tab = "Pump Specifications", group = "Component Parameters"));
  Real Eff(unit = "-") "Efficiency" annotation(
    Dialog(tab = "Pump Specifications", group = "Calculation Parameters"));
  //=============================================================================
  //Model Variables
  Real Pin(unit = "Pa", min = 0, start = Pg) "Inlet stream pressure";
  Real Tin(unit = "K", min = 0, start = Tg) "Inlet stream temperature";
  Real Hin(unit = "kJ/kmol",start=Htotg) "Inlet stream molar enthalpy";
  Real Fin(unit = "mol/s", min = 0, start = Fg) "Inlet stream molar flow";
```

```
19    Real xin_c[Nc](each unit = "-", each min = 0, each max = 1, start=xg) "Inlet
          stream components molar fraction";
20    Real Tdel(unit = "K") "Temperature increase";
21    Real Pdel(unit = "K") "Pressure increase";
22    Real Q(unit = "W") "Power required";
23    Real rho_c[Nc](each unit = "kmol/m3", each min = 0) "Component molar density";
24    Real rho(unit = "mol/m3", min = 0) "Density";
25    Real Pvap(unit = "Pa", min = 0, start = Pg) "Vapor pressure of mixture at Outlet
          temperature";
26    Real NPSH(unit = "m") "Net Positive Suction Head";
27    Real Pout(unit = "Pa", min = 0, start = Pg) "Outlet stream pressure";
28    Real Tout(unit = "K", min = 0, start = Tg) "Outlet stream temperature";
29    Real Hout(unit = "kJ/kmol",start=Htotg) "Outlet stream molar enthalpy";
30    Real Fout(unit = "mol/s", min = 0, start = Fg) "Outlet stream molar flow";
31    Real xout_c[Nc](each unit = "-", each min = 0, each max = 1, start=xg) "Outlet
          stream molar fraction";
32
33   //--------------------------------
34
35    Real qin(unit="m3/s")"input volumetric flowrate";
36    Real head(unit="m");
37    Real syshead(unit="m");
38    Real Power(unit"w");
39
40    parameter Integer nop=1"no of points for calmode=Curves";
41    parameter Real xhead[nop]=zeros(nop) ;
42    parameter Real yhead[nop] = zeros(nop);
43    parameter Real xsys[nop]= zeros(nop) ;
44    parameter Real ysys[nop]= zeros(nop) ;
45    parameter Real xpower[nop]= zeros(nop) ;
46    parameter Real ypower[nop]= zeros(nop)  ;
47    parameter Real xeff[nop] = zeros(nop) ;
48    parameter Real yeff[nop] = zeros(nop) ;
49    parameter Real xnpshr[nop]= zeros(nop);
50    parameter Real ynpshr[nop]= zeros(nop);
51
52    Real effin=0;
53    Real MW[Nc]"molecular weight of each component";
54    Real TMW(unit="g/mol")"total molecular weight of mixture";
55    Real mdot(unit="kg/s")"massflowrate ";
56    Real NPSHR(unit="m");
57
58    Integer deg=3;
59    //---interpolation varibles
60    parameter String CalcMode ;
61   //Real index[5];
62   Integer flag[5];
63   Real sort[5];
64  Real range"range of input molarflowrate for finding opeation point";
65    //============================================================================
66    //Instantiation of Connectors
67    Simulator.Files.Interfaces.matConn In(Nc = Nc) annotation(
68      Placement(visible = true, transformation(origin = {-100, -2}, extent = {{-10,
          -10}, {10, 10}}, rotation = 0), iconTransformation(origin = {-100, 16},
          extent = {{-10, -10}, {10, 10}}, rotation = 0)));
69    Simulator.Files.Interfaces.matConn Out(Nc = Nc) annotation(
70      Placement(visible = true, transformation(origin = {102, 0}, extent = {{-10,
          -10}, {10, 10}}, rotation = 0), iconTransformation(origin = {100, 100},
          extent = {{-10, -10}, {10, 10}}, rotation = 0)));
71    Simulator.Files.Interfaces.enConn En annotation(
72      Placement(visible = true, transformation(origin = {2, -100}, extent = {{-10,
          -10}, {10, 10}}, rotation = 0), iconTransformation(origin = {0, -70},
          extent = {{-10, -10}, {10, 10}}, rotation = 0)));
73
74    extends GuessModels.InitialGuess;
75  equation
76  //============================================================================
```

```
77  //Connector equation
78    In.P = Pin;
79    In.T = Tin;
80    In.F = Fin;
81    In.H = Hin;
82    In.x_pc[1, :] = xin_c[:];
83
84    Out.P = Pout;
85    Out.T = Tout;
86    Out.F = Fout;
87    Out.H = Hout;
88    Out.x_pc[1, :] = xout_c[:];
89    En.Q = Q;
90  //================================================================================
91  //Pump equations
92    Fin = Fout;
93    xin_c = xout_c;
94
95  //================================================================================
96  //Calculation of Density
97    for i in 1:Nc loop
98      rho_c[i] = Simulator.Files.ThermodynamicFunctions.Dens(C[i].LiqDen, C[i].Tc,
          Tin, Pin);
99    end for;
100   rho = (1 / sum(xin_c ./ rho_c));
101   qin=Fin/rho;
102 //================================================================================
103 for j in 1:Nc loop
104    MW[j]=C[j].MW*xin_c[j];
105   end for;
106
107 TMW=sum(MW);
108 mdot=TMW*Fin*0.001;
109
110
111
112
113
114
115 //////////////////////////////////////////////////////////
116
117
118 if CalcMode =="Normal" then
119
120   Pin + Pdel = Pout;
121   Tin + Tdel = Tout;
122   Hout = Hin + Pdel / rho;
123   Q = Fin * (Hout - Hin) / Eff;
124   head=Hout-Hin;
125   syshead=head/Eff;
126   Power=Q;
127
128 NPSHR=0;
129
130 end if;
131
132
133 //////////////////////////////////////////////////////////
134
135 if CalcMode =="Curves" then
136
137
138
139
140
141
142
```

```
143 head= Simulator.Files.Interpolation.interpolant(nop,xhead,yhead,qin,deg);
144 if xeff[nop]<>0 then
145
146
147   Eff= Simulator.Files.Interpolation.interpolant(nop,xeff,yeff,qin,deg);
148    else
149      effin=Eff ;
150
151   end if;
152
153 if  xsys[nop]<>0 then
154
155
156       syshead=Simulator.Files.Interpolation.interpolant(nop,xsys,ysys,qin,deg);
157
158     else
159        syshead= head/(Eff/100);
160
161    end if;
162
163
164
165   Pout=Pin + head * 9.81*rho*TMW*0.001;
166   Pdel=Pout-Pin;
167
168
169 if xpower[nop]<>0 then
170
171
172
173     Power= Simulator.Files.Interpolation.interpolant(nop,xpower,ypower,qin,deg);
174   else
175     Power=1000*TMW*Fin*9.81*syshead ;
176
177   end if;
178
179  Q=Power;
180  Tdel = Tin-Tout;
181  Hout = Hin +(Power*Eff)/Fin ;
182
183
184
185 if xnpshr[nop]<>0 then
186
187    NPSHR =Simulator.Files.Interpolation. interpolant(nop,xnpshr,ynpshr,qin,deg);
188   else
189    NPSHR=0;
190
191   end if;
192
193
194 end if;
195
196
197
198
199
200
201
202 //=============================================================================
203 //Vapor Pressure of mixture at Outlet Temperature
204   Pvap = sum(xin_c .* exp(C[:].VP[2] + C[:].VP[3] / Tout + C[:].VP[4] * log(Tout)
        + C[:].VP[5] .* Tout .^ C[:].VP[6]));
205
206
207
208
```

```
209
210
211  NPSH = (Pin - Pvap) / (rho*9.81*TMW*0.001);
212
213
214 algorithm
215
216  (sort,flag):= Modelica.Math.Vectors.sort({xeff[nop],xsys[nop],xhead[nop],xpower[
       nop],xnpshr[nop]});
217    for i in 1:5 loop
218   if sort[i]<>0 then
219    range:=sort[i];
220    break;
221
222    end if;
223  end for;
224
225
226 end CentrifugalPumpnew;
```

8.3 Barycentric Interpolation

```
1  within Simulator.Files;
2
3  package Interpolation
4    function floaterhormannrationalinterpolant
5
6      input Integer N; //no of points
7      input Real x[N];
8      input Real y[N];
9      input Integer d;//degree of polynomial
10
11     output Real w[N];//weights
12
13   protected
14     Real s;
15     Real threshold = 1.0E-300;
16
17     Real flag;
18     Real v;
19
20     Real s0;
21     Integer prem[N];
22     Real X[N];
23     Integer flag1;
24     Real wtemp[N];
25   algorithm
26     X := x;
27     s0 := 1;
28     for i in 1:N loop
29       s0 := -1 * s0;
30     end for;
31     for i in 1:N loop
32       prem[i] := i;
33     end for;
34     for i in 1:N loop
35       for j in i + 1:N loop
36         if X[j] < X[i] then
37           flag := X[i];
38           X[i] := X[j];
39           X[j] := flag;
40           flag1 := prem[i];
41           prem[i] := prem[j];
42           prem[j] := flag1;
```

```
43          end if;
44        end for;
45      end for;
46      for k in 1:N loop
47        s := 0;
48        for i in max(k - d, 1):min(k, N - d) loop
49          v := 1;
50          for j in i:i + d loop
51            if j <> k then
52              v := v / abs(X[k] - X[j]);
53            end if;
54          end for;
55          s := s + v;
56        end for;
57        w[k] := s0 * s;
58        s0 := -1 * s0;
59      end for;
60      for i in 1:N loop
61        wtemp[i] := w[i];
62      end for;
63      for i in 1:N loop
64        w[prem[i]] := wtemp[i];
65      end for;
66  // ////////////////////////////////////////////
67    end floaterhormannrationalinterpolant;
68
69    function interpolant
70    input Integer N;
71      input Real x[N];
72      input Real y[N];
73
74      input Real t;
75      input Integer d;
76      output Real result;
77    protected
78      Integer j;
79      Real s;
80      Real w[N];
81      Real threshold = 1.0E-300;
82      Real s1;
83      Real s2;
84      Real v;
85
86    algorithm
87      w := floaterhormannrationalinterpolant(N, x, y, d);
88
89      j := 0;
90      s := t - x[1];
91      for i in 1:N loop
92        if abs(t - x[i]) < abs(s) then
93          s := t - x[i];
94          j := i;
95        end if;
96      end for;
97      if s == 0 then
98        result := y[j];
99      end if;
100     if abs(s) > threshold then
101       j := -1;
102       s := 1;
103     end if;
104     s1 := 0;
105     s2 := 0;
106     for i in 1:N loop
107       if i <> j then
108         v := s * w[i] / (t - x[i]);
109         s1 := s1 + v * y[i];
```

```
110           s2 := s2 + v;
111         end if;
112       end for;
113       result := s1 / s2;
114     end interpolant;
115   end Interpolation;
```

8.4 Valve

```
1
2
3   model Valvenew"Model of a valve to regulate the pressure of a material stream"
4     extends Simulator.Files.Icons.Valve;
5       parameter Simulator.Files.ChemsepDatabase.GeneralProperties C[Nc] "Component
            instances array" annotation(
6       Dialog(tab = "Valve Specifications", group = "Component Parameters"));
7       parameter Integer Nc = 3 "Number of components" annotation(
8       Dialog(tab = "Valve Specifications", group = "Component Parameters"));
9       parameter Real x=1;
10      parameter Real y=1;
11      parameter Real OP=100;
12    //
          ================================================================================

13    Real Fin(unit = "mol/s", min = 0, start = Fg) "Inlet stream molar flow rate";
14    Real Pin(unit = "Pa", min = 0, start = Pg) "Inlet stream pressure";
15    Real Tin(unit = "K", min = 0, start = Tg) "Inlet stream emperature";
16    Real Hin(unit = "kJ/kmol",start=Htotg) "Inlet stream molar enthalpy";
17    Real Sin(unit = "kJ/[kmol.K]") "Inlet stream molar entropy";
18    Real xvapin(unit = "-", min = 0, max = 1, start = xvapg) "Inlet stream vapor
          phase mole fraction";
19
20    Real Tdel(unit = "K") "Temperature increase";
21    Real Pdel(unit = "Pa") "Pressure drop";
22
23    Real Fout(unit = "mol/s", min = 0, start = Fg) "outlet stream molar flow rate";
24    Real Pout(unit = "Pa", min = 0, start = Pg) "Outlet stream pressure";
25    Real Tout(unit = "K", min = 0, start = Tg) "Outlet stream temperature";
26    Real Hout(unit = "kJ/kmol",start=Htotg) "Outlet stream molar enthalpy";
27    Real Sout(unit = "kJ/[kmol.K]")  "Outlet stream molar entropy";
28    Real x_c[Nc](each unit = "-", each min = 0, each max = 1,  start = xg) "
          Component mole fraction";
29    Real xvapout(unit = "-", min = 0, max = 1, start = xvapg) "Outlet stream vapor
          phase mole fraction";
30    //
31
32  Real mdot(unit="kg/s")"massflowrate";
33
34
35  String Calmode"caluclation mode like kvliq,kvgas";
36  //
37  Real Kvc"coefficient of Valve sizing";
38  Real Kvmax"maximum coefficient of Valve sizing";
39
40  //
41
42   //////////////////
43   Real MW[Nc] (unit="g/mol")" molecular weight of each component";
44  Real TMW(unit="g/mol")"total molecular weight of mixture";
45  Real rho(unit="mol/m3")"density of each compound";
46  Real rho_c[Nc](unit="mol/m3")"total density of mixture";
47  Real rhovap0(unit="kg/m3")"density of vapor at 273.15K ";
48
49  Real rhovap0_c[Nc](unit="kg/m3")"density of each component vapor at 273.15K ";
```

```
Real TMWliq,TMWvap;
Real xliq_c[Nc];
Real xvap_c[Nc];
Real MWliq[Nc];
Real MWvap[Nc];
 //
    ============================================================================================

  //
      ============================================================================================

  Simulator.Files.Interfaces.matConn In(Nc = Nc) annotation(
    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)));
  Simulator.Files.Interfaces.matConn Out(Nc = Nc) annotation(
    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)));
  //
      ============================================================================================

  extends Simulator.GuessModels.InitialGuess;
equation
//connector equations
  In.P = Pin;
  In.T = Tin;
  In.F = Fin;
  In.H = Hin;
  In.S = Sin;
  In.x_pc[1, :] = x_c[:];
  In.xvap = xvapin;
  Out.P = Pout;
  Out.T = Tout;
  Out.F = Fout;
  Out.H = Hout;
  Out.S = Sout;
  Out.x_pc[1, :] = x_c[:];
  Out.xvap = xvapout;
  In.x_pc[2,:]= xliq_c[:];
  In.x_pc[3,:]= xvap_c[:];
//
    ============================================================================================

  Fin = Fout;
//material balance
  Hin = Hout;
//energy balance
 // Pin - Pdel = Pout;
//pressure calculation
  Tin + Tdel = Tout;
//temperature calculation


  Kvc=Kvmax*0.01* x * OP^(y);


//////////////////////////////////////////////////
for j in 1:Nc loop
   MW[j]=C[j].MW*x_c[j];
  end for;
  TMW=sum(MW);
mdot=TMW*Fin*0.001;
for j in 1:Nc loop
    MWliq[j]=C[j].MW*xliq_c[j];
  end for;
TMWliq=sum(MWliq);
```

```
105
106 for j in 1:Nc loop
107    MWvap[j]=C[j].MW*xvap_c[j];
108   end for;
109 TMWvap=sum(MWvap);
110 //////////////////////////////////////////////
111
112 for i in 1:Nc loop
113     rho_c[i] = Simulator.Files.ThermodynamicFunctions.Dens(C[i].LiqDen, C[i].Tc,
          Tin, Pin);
114   end for;
115   rho = ((TMWliq/ sum(xliq_c./ rho_c)))*0.001;
116     for i in 1:Nc loop
117       rhovap0_c[i] = 101325 / (1 * 8.314 * 273.15) * C[i].MW * 1E-3;
118     end for;
119     rhovap0 = TMWvap/ sum(MWvap[ :] ./ rhovap0_c[:]) ;
120
121
122 if Calmode=="Kvliq" then
123    Pin=((Pout/100000) +( 1/(1000*rho) *(mdot*3600/Kvc)^2))*100000;
124
125
126 elseif Calmode=="Kvgas" then
127         Pin=((Pout/100000)+(Tin/(rhovap0*(Pout/100000)))*(((mdot*3600/(519*Kvc))
              ^2)))*100000 ;
128
129 else
130 Pin=Pout;
131 end if;
132 Pdel=Pin-Pout;
133
134
135
136
137     end Valvenew;
```

8.5 Fittings

```
1 within Simulator.UnitOperations;
2
3 model fittings
4
5     parameter Simulator.Files.ChemsepDatabase.GeneralProperties C[Nc] "Component
          instances array" annotation(
6     Dialog(tab = "Valve Specifications", group = "Component Parameters"));
7     parameter Integer Nc = 3 "Number of components" annotation(
8     Dialog(tab = "Valve Specifications", group = "Component Parameters"));
9   //
        ================================================================================

10   Real Fin(unit = "mol/s", min = 0, start = Fg) "Inlet stream molar flow rate";
11   Real Pin(unit = "Pa", min = 0, start = Pg) "Inlet stream pressure";
12   Real Tin(unit = "K", min = 0, start = Tg) "Inlet stream emperature";
13   Real Hin(unit = "kJ/kmol",start=Htotg) "Inlet stream molar enthalpy";
14   Real Sin(unit = "kJ/[kmol.K]") "Inlet stream molar entropy";
15   Real xvapin(unit = "-", min = 0, max = 1, start = xvapg) "Inlet stream vapor
       phase mole fraction";
16
17   Real Tdel(unit = "K") "Temperature increase";
18   Real Pdel(unit = "Pa") "Pressure drop";
19
20   Real Fout(unit = "mol/s", min = 0, start = Fg) "outlet stream molar flow rate";
21   Real Pout(unit = "Pa", min = 0, start = Pg) "Outlet stream pressure";
22   Real Tout(unit = "K", min = 0, start = Tg) "Outlet stream temperature";
```

```
23    Real Hout(unit = "kJ/kmol",start=Htotg) "Outlet stream molar enthalpy";
24    Real Sout(unit = "kJ/[kmol.K]")  "Outlet stream molar entropy";
25    Real x_c[Nc](each unit = "-", each min = 0, each max = 1,  start = xg) "
          Component mole fraction";
26   Real x_cl[Nc](each unit = "-", each min = 0, each max = 1,  start = xg) "
          Component mole fraction";
27    Real x_cg[Nc](each unit = "-", each min = 0, each max = 1,  start = xg) "
          Component mole fraction";
28    Real xvapout(unit = "-", min = 0, max = 1, start = xvapg) "Outlet stream vapor
          phase mole fraction";
29    //
          ============================================================================

30
31  Real resv[2](each unit = "-");
32  Real dph (unit="Pa");
33  Real dpf;
34  Real Qvin(unit = "m3/s");
35  Real Qlin(unit="m3/s");
36  Real rho_l(unit="kg/m3");
37  Real rho_v(unit="kg/m3");
38  Real liqvel(unit="m/s");
39  Real gasvel(unit ="m/s");
40  Real ID(unit="m");
41  String name;
42  Real rho_cl[Nc](each unit ="kg/m3");
43  Real MWl[Nc],TMWl;
44  Real mdotl(unit="kg/s");
45  Real rho_cg[Nc](each unit ="kg/m3");
46  Real MWg[Nc],TMWg;
47  Real mdotg(unit="kg/s");
48    //
          ============================================================================

49    Simulator.Files.Interfaces.matConn In(Nc = Nc) annotation(
50      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)));
51    Simulator.Files.Interfaces.matConn Out(Nc = Nc) annotation(
52      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)));
53    //
          ============================================================================

54    extends GuessModels.InitialGuess;
55  equation
56  //connector equations
57    In.P = Pin;
58    In.T = Tin;
59    In.F = Fin;
60    In.H = Hin;
61    In.S = Sin;
62    In.x_pc[1, :] = x_c[:];
63    In.xvap = xvapin;
64    In.x_pc[2,:]= x_cl[:];
65    In.x_pc[3,:]= x_cg[:];
66  //  In.Fm_pc[2,:] = fmpc[:];
67    Out.P = Pout;
68    Out.T = Tout;
69    Out.F = Fout;
70    Out.H = Hout;
71    Out.S = Sout;
72    Out.x_pc[1, :] = x_c[:];
73    Out.xvap = xvapout;
74  //
        ============================================================================
```

```
75    Fin = Fout;
76  //material balance
77    Hin = Hout;
78  //energy balance
79   // Pin - Pdel = Pout;
80  //pressure calculation
81    Tin + Tdel = Tout;
82  //temperature calculation
83  //////////////////////////////////////
84  for j in 1:Nc loop
85     MWl[j]=C[j].MW*x_cl[j];
86    end for;
87
88  TMWl=sum(MWl);
89
90
91
92  for i in 1:Nc loop
93       rho_cl[i] = Simulator.Files.ThermodynamicFunctions.Dens(C[i].LiqDen, C[i].Tc,
             Tin, Pin);
94    end for;
95    rho_l = (1 / sum(x_cl./ rho_cl))*0.001*TMWl;
96    mdotl=TMWl*Fin*0.001*(1-xvapin);
97  //////////////////////////////////////
98  for j in 1:Nc loop
99     MWg[j]=C[j].MW*x_cg[j];
100   end for;
101
102 TMWg=sum(MWg);
103
104
105 /*for i in 1:Nc loop
106      rho_cg[i] = Simulator.Files.ThermodynamicFunctions.Dens(C[i].LiqDen, C[i].Tc,
             Tin, Pin);
107   end for;
108   rho_v = (1 / sum(x_cg./ rho_cg))*0.001*TMWl;
109   mdotg=TMWg*Fin*0.001*(xvapin);*/
110 //////////////////
111  for  i in 1:Nc loop
112       rho_cg[i] = Pin/ (1 * 8.314 * Tin) * C[i].MW * 1E-3;
113     end for;
114     rho_v = TMWg/ sum(MWg./ rho_cg) ;
115     mdotg=TMWg*Fin*0.001*(xvapin);
116 ////////////////////////////////
117 Qlin=mdotl/rho_l;
118 Qvin=mdotg/rho_v;
119 liqvel=Qlin/((22/7)*(ID^2)*0.25);
120 gasvel=Qvin/((22/7)*(ID^2)*0.25);
121 algorithm
122 if name=="elbow90dg" then
123 resv[1]:=30;
124 resv[2]:=1;
125 end if ;
126 if name=="elbow45dg" then
127 resv[1]:=16;
128 resv[2]:=1;
129 end if ;
130
131 if name=="elbow180dg" then
132 resv[1]:=50;
133 resv[2]:=1;
134 end if ;
135 if name=="anglevalve" then
136 resv[1]:=55;
137 resv[2]:=1;
138 end if ;
```

```
139 if name=="butterflyvalve" then //2 to 14 inches
140 resv[1]:=40;
141 resv[2]:=1;
142 end if ;
143 if name=="ballvalve" then
144 resv[1]:=3;
145 resv[2]:=1;
146 end if ;
147
148 if name=="gatevalve" then //open
149 resv[1]:=8;
150 resv[2]:=1;
151 end if ;
152 if name=="globevalve" then
153 resv[1]:=340;
154 resv[2]:=1;
155 end if ;
156 if name=="liftcheckvalve" then
157 resv[1]:=600;
158 resv[2]:=1;
159 end if ;
160 if name=="poppetdiscvalve" then
161 resv[1]:=420;
162 resv[2]:=1;
163 end if ;
164 if name=="swingcheckvalve" then
165 resv[1]:=100;
166 resv[2]:=1;
167 end if ;
168
169 if name=="ballcheckvalve" then
170 resv[1]:=400;
171 resv[2]:=1;
172 end if ;
173
174
175
176 if name=="tee" then//branched blanked
177 resv[1]:=20;
178 resv[2]:=1;
179 end if ;
180 if name=="tee" then//elbow
181 resv[1]:=60;
182 resv[2]:=1;
183 end if ;
184
185 if name=="quickreduction(d/D=1/2)" then
186 resv[1]:=9.6;
187 resv[2]:=0;
188 end if ;
189
190 if name=="quickreduction(d/D=1/4)" then
191 resv[1]:=96;
192 resv[2]:=0;
193 end if ;
194 if name=="quickreduction(d/D=3/4)" then
195 resv[1]:=11;
196 resv[2]:=0;
197 end if ;
198 if name=="borderinlet" then
199 resv[1]:=0.25;
200 resv[2]:=0;
201 end if ;
202 if name=="normalinlet" then
203 resv[1]:=0.78;
204 resv[2]:=0;
205 end if ;
```

```
206 if name=="quickexpansion(d/D=1/2)" then
207 resv[1]:=9;
208 resv[2]:=0;
209 end if ;
210 if name=="quickexpansion(d/D=1/4)" then
211 resv[1]:=225;
212 resv[2]:=0;
213 end if ;
214 if name=="quickexpansion(d/D=3/4)" then
215 resv[1]:=0.6;
216 resv[2]:=0;
217 end if ;
218 if name=="bend90dg" then
219 resv[1]:=60;
220 resv[2]:=1;
221 end if ;
222 if name=="normalreduction(2:1)" then
223 resv[1]:=5.67;
224 resv[2]:=0;
225 end if ;
226 if name=="normalreduction(4:3)" then
227 resv[1]:=0.65;
228 resv[2]:=0;
229 end if ;
230 if name=="borderexit" then
231 resv[1]:=1;
232 resv[2]:=0;
233 end if ;
234 if name=="normalexit" then
235 resv[1]:=1;
236 resv[2]:=0;
237 end if ;
238
239
240
241
242
243
244
245
246
247
248
249 if resv[2]==1 then
250   dph:=0;
251 dpf:= resv[1]*(0.0101*(ID)^(-0.2232))*(((Qlin*rho_l)/(Qlin+Qvin)) + (Qvin*rho_v)/(
      Qlin+Qvin)) *( (liqvel+gasvel)^2)*0.5;
252 else
253 dph:=0;
254 dpf:=resv[1]*(((Qlin*rho_l)/(Qlin+Qvin)) + (Qvin*rho_v)/(Qlin+Qvin)) *( (liqvel+
      gasvel)^2)*0.5;
255
256
257 end if;
258
259      Pdel:=dpf;
260      Pout:=Pin-dpf;
261      end fittings;
```

8.6 Reduction of Thermodynamics

8.6.1 Peng Robinson

```
package pgnew
  model PengRobinson
  pgnew.base1 philiq(Nc = Nc, C = C, P = P, T = T, x = x_pc[2, :], mode = "liq");
    pgnew.base1 phivap(Nc = Nc, C = C, P = P, T = T, x = x_pc[3, :], mode = "vap")
        ;
    Real Cpres_p[3] "residual specific heat", Hres_p[3] "residual enthalpy",
        Sres_p[3] "residual Entropy";
    Real philiqbubl_c[Nc], phivapdew_c[Nc], Pvap_c[Nc];
    Real philiq_c[Nc], phivap_c[Nc];
    Real gmabubl_c[Nc] "Activity coefficent at bubble point";
    Real gmadew_c[Nc] "Activity coefficent at dew point";
    Real K_c[Nc];
    Real gma[Nc];
  equation
    Cpres_p[:] = zeros(3);
    Hres_p[:] = zeros(3);
    Sres_p[:] = zeros(3);
    for i in 1:Nc loop
      philiqbubl_c[i] = 1;
      phivapdew_c[i] = 1;
      gmadew_c[i] = 1;
      gmabubl_c[i] = 1;
      gma[i] = 1;
    end for;
    for i in 1:Nc loop
      Pvap_c[i] = Simulator.Files.ThermodynamicFunctions.Psat(C[i].VP, T);
    end for;
    philiq_c = philiq.phi_c;
    phivap_c = phivap.phi_c;
//xliqdew_c i snot there
    for i in 1:Nc loop
      if philiq.phi_c[i] == 0 or phivap.phi_c[i] == 0 then
        K_c[i] = 0;
      else
        K_c[i] = philiq.phi_c[i] / phivap.phi_c[i];
//  K_c[i]=1;
      end if;
    end for;
  end PengRobinson;



  model base1
  parameter Integer Nc;
    parameter Real R = 8.314 "Ideal Gas Constant";
    parameter Simulator.Files.ChemsepDatabase.GeneralProperties C[Nc];
    parameter Real kij_c[Nc, Nc](each start = 1) =
        Simulator.Files.ThermodynamicFunctions.BIPPR(Nc, C.name);
    Real aM, bM;
    Real x[Nc];
    Real aij[Nc, Nc];
    Real b_c[Nc];
    Real A, B;
    Real P, T;
    Real Cc[4];
    Real Z_R[3, 2];
    Real Zs[3];
    Real Zss;
    Real sumx[Nc];
    Real E, F, G, H_c[Nc], I_c[Nc];
    // Real b[Nc] ;
    Real phi_c[Nc];
    String mode;
    Real Tr_c[Nc];
    Real m_c[Nc], q_c[Nc], a_c[Nc];
    extends Simulator.GuessModels.InitialGuess;
    //equation
```

```
65    equation
66      Tr_c = T ./ C.Tc;
67      b_c = 0.0778 * R * C.Tc ./ C.Pc;
68      m_c = 0.37464 .+ 1.54226 * C.AF .- 0.26992 * C.AF .^ 2;
69      q_c = 0.45724 * R ^ 2 * C.Tc .^ 2 ./ C.Pc;
70      a_c = q_c .* (1 .+ m_c .* (1 .- sqrt(Tr_c))) .^ 2;
71      aij = {{(1 - kij_c[i, j]) * sqrt(a_c[i] * a_c[j]) for i in 1:Nc} for j in 1:Nc
            };
72      aM = sum({{x[i] * x[j] * aij[i, j] for i in 1:Nc} for j in 1:Nc});
73      bM = sum(b_c .* x[:]);
74      A = aM * P / (R * T) ^ 2;
75      B = bM * P / (R * T);
76      Cc[1] = 1;
77      Cc[2] = B - 1;
78      Cc[3] = A - 3 * B ^ 2 - 2 * B;
79      Cc[4] = B ^ 3 + B ^ 2 - A * B;
80      Z_R = Modelica.Math.Vectors.Utilities.roots(Cc);
81      Zs = {Z_R[i, 1] for i in 1:3};
82      if mode == "vap" then
83        Zss = max({Zs});
84      else
85        Zss = min({Zs});
86      end if;
87      sumx = {sum({x[j] * aij[i, j] for j in 1:Nc}) for i in 1:Nc};
88  //
89      if Zss + 2.4142135 * A <= 0 and mode == "vap" then
90        E = 1;
91      elseif Zss + 2.4142135 * B <= 0 and mode == "liq" then
92        E = 1;
93      else
94        E = Zss + 2.4142135 * B;
95      end if;
96  // change
97  //
98      if Zss - 0.414213 * B <= 0 then
99        F = 1;
100     else
101       F = Zss - 0.414213 * B;
102     end if;
103 //
104     if Zss - B <= 0 then
105       G = 0;
106     else
107       G = log(Zss - B);
108     end if;
109 //
110     for i in 1:Nc loop
111       if bM == 0 then
112         H_c[i] = 0;
113       else
114         H_c[i] = b_c[i] / bM;
115       end if;
116     end for;
117     for i in 1:Nc loop
118       if aM == 0 then
119         I_c[i] = 0;
120       else
121         I_c[i] = sumx[i] / aM;
122       end if;
123     end for;
124     phi_c = exp(A / (B * sqrt(8)) * log(E / F) .* (H_c .- 2 * I_c) .+ (Zss - 1) *
          H_c .- G);
125   end base1;
126 end pgnew;
```

8.6.2 UNIQUAC

```
package UNIQUACnew
  model UNIQUAC
  UNIQUACnew.Qnew gma(Nc = Nc, C = C, x =  x_pc[2, :], P = P     , T = T, Pvap_c =
      Pvap_c,rho_c=rho_c);
  UNIQUACnew.Qnew gmabubl(Nc = Nc, C = C, x = x_pc[1, :], P = Pbubl, T = T, Pvap_c
       = Pvap_c,rho_c=rho_c);
  UNIQUACnew.Qnew gmadew(Nc = Nc, C = C, x = xliqdew_c, P = Pdew, T = T, Pvap_c =
      Pvap_c,rho_c=rho_c);

    Real rho_c[Nc];
    Real Cpres_p[3] "residual specific heat", Hres_p[3] "residual enthalpy",
        Sres_p[3] "residual Entropy", K_c[Nc];
    Real philiqbubl_c[Nc], phivapdew_c[Nc], Pvap_c[Nc];
    // Real K_c[Nc](each start = 0.7) "Distribution Coefficient";
    Real xliqdew_c[Nc](each start = 0.5, each min = 0, each max = 1);
    Real gmanew_c[Nc](each start = 1.2);
    Real gmabubl_c[Nc](each start = 1) "Activity coefficent at bubble point";
    Real gmadew_c[Nc](each start = 2.2) "Activity coefficent at dew point";
  equation
    Cpres_p = zeros(3);

    Hres_p = zeros(3);

    Sres_p = zeros(3);

    for i in 1:Nc loop
      rho_c[i] = Simulator.Files.ThermodynamicFunctions.Dens(C[i].LiqDen, C[i].Tc,
          T, P) * 1E-3;

    end for;

    for i in 1:Nc loop
      philiqbubl_c[i] = 1;
      phivapdew_c[i] = 1;
    end for;
    for i in 1:Nc loop
      Pvap_c[i] = Simulator.Files.ThermodynamicFunctions.Psat(C[i].VP, T);

    end for;
//  ///////////////
    for i in 1:Nc loop
      if gmadew.q== 0 or  x_pc[1, i] == 0 then
        xliqdew_c[i] = 0;
      else
        xliqdew_c[i] = x_pc[1, i] * Pdew/ (gmadew_c[i] * Pvap_c[i]);
      end if;
      end for;
    // ////////////

    gmanew_c = gma.gma;
    gmabubl_c = gmabubl.gma;
    gmadew_c = gmadew.gma;

    for i in 1:Nc loop
      K_c[i] = gmanew_c[i] .* Pvap_c[i] ./ P;
    end for;
  end UNIQUAC;

  model Qnew
  Real T;
    parameter Simulator.Files.ChemsepDatabase.GeneralProperties C[Nc];
    parameter Integer Nc = 3;
    Real r (each start = 2, min = 0, max = 1), q(each start = 2);
    Real R[Nc] = C.UniquacR;
```

```
60      Real Q[Nc] = C.UniquacQ;
61      Real x[Nc] ;
62      parameter Real a[Nc, Nc] = Simulator.Files.ThermodynamicFunctions.BIPUNIQUAC(
            Nc, C.name);
63      Real Theta[Nc];
64      Real S[Nc](each start = 1);
65      Real rho_c[Nc];
66      Real Pvap_c[Nc];
67      Real tow[Nc, Nc];
68      Real sum[Nc](each start = 2);
69      Real Cc[Nc];
70      Real gmar[Nc](each start = 1) ;
71      Real D[Nc];
72      Real E[Nc];
73      Real F[Nc];
74      Real A[Nc];
75      Real B[Nc];
76      Real gmac[Nc](each start = 2);
77      Real gmaold[Nc](each start = 1);
78      Real Z = 10;
79      Real gma[Nc](each start = 1);
80      Real PCF[Nc];
81      Real phi[Nc](each start = 0.5);
82      Real P;
83      extends Simulator.GuessModels.InitialGuess;
84    equation
85
86      tow = Simulator.Files.ThermodynamicFunctions.TowUNIQUAC(Nc, a, T);
87      r = sum(x[:] .* R[:]);
88      q = sum(x[:] .* Q[:]);
89
90      //////////////
91     /* for i in 1:Nc loop
92        if q== 0 or x[ i] == 0 then
93          xliqdew_c[i] = 0;
94        else
95          xliqdew_c[i] = x[i] * P / (gma[i] * Pvap_c[i]);
96        end if;
97    */
98
99      ////////////
100     for i in 1:Nc loop
101       if x[i] == 0 or q == 0 then
102         Theta[i] = 0;
103       else
104         Theta[i] = x[i] * Q[i] * 1 / q;
105       end if;
106       if Theta[i] == 0 then
107         S[i] = 1;
108       else
109         S[i] = sum(Theta[:] .* tow[i, :]);
110       end if;
111     end for;
112 //////////////////////////
113     for i in 1:Nc loop
114       if S[i] == 1 then
115         sum[i] = 0;
116       else
117         sum[i] = sum(Theta[:] .* tow[i, :] ./ S[:]);
118       end if;
119       if S[i] == 1 then
120         Cc[i] = 0;
121       elseif S[i] > 0 then
122         Cc[i] = log(S[i]);
123       else
124         Cc[i] = 0;
125       end if;
```

```
126         gmar[i] = exp(Q[i] * (1 - Cc[i] - sum[i]));
127       end for;
128   //////////////////////////////
129       for i in 1:Nc loop
130         if r == 0 then
131           D[i] = 0;
132         else
133           D[i] = R[i] / r;
134         end if;
135         if q == 0 then
136           E[i] = 0;
137         else
138           E[i] = Q[i] / q;
139         end if;
140         if E[i] == 0 then
141           F[i] = 0;
142         else
143           F[i] = D[i] / E[i];
144         end if;
145         if D[i] > 0 then
146           A[i] = log(D[i]);
147         elseif D[i] == 1 then
148           A[i] = 0;
149         else
150           A[i] = 0;
151         end if;
152         if F[i] > 0 then
153           B[i] = log(F[i]);
154         elseif F[i] == 1 then
155           B[i] = 0;
156         else
157           B[i] = 0;
158         end if;
159         log(gmac[i]) = 1 - D[i] + A[i] + (-Z / 2 * Q[i] * (1 - F[i] + B[i]));
160         gmaold[i] = gmac[i] * gmar[i];
161       end for;
162       for i in 1:Nc loop
163         PCF[i] = Simulator.Files.ThermodynamicFunctions.PoyntingCF(Nc, C[i].Pc, C[i
              ].Tc, C[i].Racketparam, C[i].AF, C[i].MW, T, P, gma[i], Pvap_c[i], rho_c
              [i]);
164         if P == 0 then
165           phi[i] = 1;
166           gma[i] = 1;
167         else
168           phi[i] = gmaold[i] .* Pvap_c[i] ./ P .* PCF[i];
169           phi[i] = gma[i] .* Pvap_c[i] ./ P;
170         end if;
171       end for;
172     end Qnew;
173   end UNIQUACnew;
```

8.6.3 UNIFAC

```
1   package UNIFACnew
2     model UNIFAC2
3     //new varibles
4       Real gma_c[Nc], gmabubl_c[Nc], gmadew_c[Nc];
5       Real Cpres_p[3], Hres_p[3], Sres_p[3];
6       Real K_c[Nc];
7       Real Pvap_c[Nc];
8       Real philiqbubl_c[Nc], phivapdew_c[Nc];
9       ///
10      // Real Psat[Nc];
11      //Real gamma[Nc];
12      //Real K[Nc];
```

```
    //Fugacity coefficient  at the Bubble and Dew Points
    // Real liqfugcoeff_bubl[Nc], vapfugcoeff_dew[Nc];
    //Excess Energy Properties
    // Real resMolSpHeat[3], resMolEnth[3], resMolEntr[3];
    //Activity Coefficient at the Bubble and Dew Points
    //Real gammaBubl[Nc], gammaDew[Nc](each start = 1.5);
  equation
    Cpres_p = zeros(3);
    Hres_p = zeros(3);
    Sres_p = zeros(3);
    for i in 1:Nc loop
      Pvap_c[i] = Simulator.Files.ThermodynamicFunctions.Psat(C[i].VP, T);
// Psat[i] = Simulator.Files.Thermodynamic_Functions.Psat(comp[i].VP[:], T);
    end for;
    for i in 1:Nc loop
      philiqbubl_c[i] = 1;
      phivapdew_c[i] = 1;
    end for;
    (gma_c, gmabubl_c, gmadew_c) = UNIFAC_M(Nc, C.UNIFAC_SubGroup, Pvap_c, T, P,
        x_pc, Pdew);
    for i in 1:Nc loop
      K_c[i] = gma_c[i] * Pvap_c[i] / P;
    end for;
  end UNIFAC2;

  function UNIFAC_gammaNew
  input Integer NOC;
    input Integer length;
    input Integer ID[NOC, 5, 2];
    input Integer ID_v[length];
    input Real Psat[NOC];
    input Real T;
    input Real P;
    input Real x_pc[NOC];
    input Real Pdew;
    input String flag;
    output Real gamma_c[NOC];
  protected
    Real J_c[NOC];
    Real L_c[NOC];
    Real r[NOC];
    Real q[NOC];
    Real X[NOC];
    Real gammac_c[NOC];
    Real gammar_c[NOC];
    Real theta_l[length];
    Real S_l[length];
    Real a;
    Real tau[length, length];
    Real sum_c[NOC];
    Real e[length, NOC];
    Real B[NOC, length];
    Real p;
    Real Xdew[NOC];
  algorithm
    tau := UNIFAC_BIP(length, ID_v, T);
    (r, q, e) := UNIFAC_RQ(NOC, length, ID, ID_v);
    if flag == "dew" then
      for i in 1:NOC loop
        Xdew[i] := x_pc[i] * Pdew / (gamma_c[i] * Psat[i]);
      end for;
      X := Xdew;
    else
      Xdew := zeros(NOC);
      X := x_pc;
    end if;
    for i in 1:NOC loop
```

```
      J_c[i] := r[i] / sum(r[:] .* X[:]);
      L_c[i] := q[i] / sum(q[:] .* X[:]);
      if J_c[i] > 0 and L_c[i] > 0 then
        gammac_c[i] := exp(1 - J_c[i] + log(J_c[i]) + (-5 * q[i] * (1 - J_c[i] /
            L_c[i] + log(J_c[i] / L_c[i]))));
      elseif flag == "dew" then
        gamma_c[i] := exp(1 - 0 + 0 + (-5 * q[i] * (1 - J_c[i] / L_c[i] + 1)));
      else
        gamma_c[i] := 1;
      end if;
    end for;
//changes
    for j in 1:length loop
      theta_l[j] := sum(X[:] .* q[:] .* e[j, :]) / sum(X[:] .* q[:]);
    end for;
    for i in 1:length loop
      S_l[i] := sum(theta_l[:] .* tau[:, i]);
    end for;
    sum_c[:] := fill(0, NOC);
//////
    for i in 1:NOC loop
      for j in 1:length loop
        for l in 1:length loop
          B[i, j] := sum(e[:, i] .* tau[:, j]);
        end for;
      end for;
    end for;
////
    for j in 1:length loop
      if S_l[j] > 0 then
        p := 1;
      elseif S_l[j] < 0 then
        p := -1;
      else
        p := 0;
      end if;
      for i in 1:NOC loop
        if B[i, j] > 0 then
          a := 1;
        elseif B[i, j] < 0 then
          a := -1;
        else
          a := 0;
        end if;
        if a == 1 and p == 1 then
          sum_c[i] := sum_c[i] + theta_l[j] * B[i, j] / S_l[j] - e[j, i] * log(B[
              i, j] / S_l[j]);
          gammar_c[i] := exp(q[i] * (1 - sum_c[i]));
        elseif a == (-1) and p == (-1) then
          sum_c[i] := sum_c[i] + theta_l[j] * B[i, j] / S_l[j] - e[j, i] * log(B[
              i, j] / S_l[j]);
          gammar_c[i] := exp(q[i] * (1 - sum_c[i]));
        else
          sum_c[i] := sum_c[i];
          gammar_c[i] := exp(q[i] * (1 - sum_c[i]));
        end if;
      end for;
    end for;
    for i in 1:NOC loop
      if gammar_c[i] > 0 and gammac_c[i] > 0 then
        gamma_c[i] := exp(log(gammar_c[i]) + log(gammac_c[i]));
      elseif gammar_c[i] > 0 and gammac_c[i] <= 0 then
        gamma_c[i] := exp(log(gammar_c[i]));
      elseif gammar_c[i] <= 0 and gammac_c[i] > 0 then
        gamma_c[i] := exp(log(gammac_c[i]));
      else
        gamma_c[i] := 1;
```

```
143           end if;
144         end for;
145       end UNIFAC_gammaNew;
146     end UNIFACnew;
```