Summer Fellowship Report

On

Modelling and Simulation in OpenModelica

Submitted by

Neel Pulin Modi

Under the guidance of

Prof. Kannan Moudgalya
Chemical Engineering Department
IIT Bombay

June 9, 2020

Acknowledgement

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.

I would also like to extend my gratitude to Priyam Nayak and Rahul Nagaraj of the department of Chemical Engineering, IIT Bombay, FOSSEE Team for their help and guidance.

Contents

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.

2 Assay Manager

2.1 Background

The first step in the process of Crude Distillation is to identify the properties of the various crude cuts present in the feed. This is done by using the Assay Manager. Assay Manager is used to get the pure component properties of crude cut given the property of crude feed. We start by providing the average molecular mass (MW), the average boiling temperature (Tb), the average specific gravity (SG), and the expected number of crude cuts from the feed.

2.2 Modeling

We used a combination of algorithms from DWSIM and from Journal of Petroleum Science and Engineering 42 (2004) 195–207 for calculation of physical properties of individual crude cuts.
The first step in solving a crude mixure into diffrent cuts is to theoritically devide it into infinite number of cuts.
For any general property P the following equation holds true

$$P^* = \left[\frac{A}{B} ln\left(\frac{1}{1-x}\right)\right]^{\frac{1}{B}}$$

Where B is a constant dependent on the property calcualted.

	T_b	MW	SG
B	1.5	1	3

Table 1: Value of B for different properties

The parameter A is calculated using the formula

$$A = \left[\frac{P^*_{avg}}{\Gamma\left(1+\frac{1}{B}\right)}\right]^B B$$

x is the mole, mass or volume fraction, $P^* = (P - P_o)/P_o$, P is any property and P_o is the value of P at x=0. Next step is calculating the mole fraction of i^{th} component (z_i) in the feed.

$$z_i = exp\left(-\frac{B}{A}P^{*B}_{i-1}\right) - exp\left(-\frac{B}{A}P^{*B}_{i}\right)$$

The required value of property is calculated as

$$q_i = \frac{B}{A} P_i^{*B}$$

$$P_{i,av}^{*} = \frac{1}{z_i} \left(\frac{A}{B}\right)^{\frac{1}{B}} \times \left[\Gamma\left(1 + \frac{1}{B}, q_{i-1}\right) - \Gamma\left(1 + \frac{1}{B}, q_i\right)\right]$$

$$P_{i,av} = P_o(1 + P_{i,av}^{*})$$

The critical temperature (T_c) is calcualted as:

$$t1 = -0.00069 * PEMe - 1.4442 * d15 + 0.000491 * PEMe * d15$$

$$T_c = 35.9413 * e^{t1} * PEMe^{0.7293} * d15^{1.2771}$$

The critical pressure (Pc) is calculates as:

$$t1 = -0.008505 * PEMe - 4.8014 * d15 + 0.005749 * PEMe * d15$$

$$P_c = 31958000000.0 * e^{t1} * PEMe^{-0.4844} * d15^{4.0846}$$

And the acentric factor (AF) is calculated as:

$$Tbr = \frac{Tb}{Tc}$$

$$num = -log\left(\frac{Pc}{101325}\right) - 5.92714 + \frac{6.0948}{Tbr} + 1.28862 * log(Tbr) - 0.169347 * Tbr^6$$

$$de = 15.2518 - \frac{15.6875}{Tbr} - 13.472 * log(Tbr) + 0.43577 * Tbr^6$$

$$AF = \frac{num}{de}$$

where PEMe is Median boiling point of a crude cut, d15 is the specific gravity of the fraction and Tb is the average boiling point.

2.3 Results & Observations

The results from OpenModelica were checked for the following specifications of feed.

Parameter	Spec1	Spec2
Molecular Weight(kg/kmol)	85	90
Average Boiling Point (K)	400	450
Specific Gravity	0.85	0.9
Number of crude cuts	10	10

Table 2: Specificatins of the crude feed

Mole fraction		
DWSIM	OM	Error(%)
0.10008	0.0997503	0.003294365
0.10008	0.0997503	0.003294365
0.10008	0.0997503	0.003294365
0.10008	0.0997503	0.003294365
0.10008	0.0997503	0.003294365
0.10008	0.0997503	0.003294365
0.10008	0.0997503	0.003294365
0.10008	0.0997503	0.003294365
0.10008	0.0997503	0.003294365
0.09929	0.0997503	0.004635915

Table 3: The results for mole fraction of each of 10 crude cuts

MW			SG			T_b		
DWSIM	OM	Error(%)	DWSIM	OM	Error(%)	DWSIM	OM	Error(%)
80.259	80.267	0.011	0.766	0.765	0.185	343.858	343.734	0.036
80.815	80.821	0.006	0.802	0.800	0.284	357.419	357.140	0.078
81.442	81.446	0.005	0.824	0.823	0.106	368.814	368.895	0.022
82.159	82.162	0.003	0.842	0.839	0.360	379.905	379.365	0.142
82.996	82.997	0.001	0.858	0.855	0.395	391.378	390.701	0.173
84.003	84.001	0.002	0.875	0.871	0.428	403.825	402.996	0.205
85.266	85.261	0.006	0.891	0.890	0.168	418.039	418.197	0.038
86.965	86.952	0.015	0.910	0.905	0.494	435.456	434.212	0.286
89.581	89.551	0.034	0.933	0.928	0.535	459.676	458.065	0.350
96.320	96.188	0.137	0.977	0.972	0.529	512.533	510.028	0.489

Table 4: Results for the first set of specifications

MW			SG			T_b		
DWSIM	OM	Error(%)	DWSIM	OM	Error(%)	DWSIM	OM	Error(%)
80.518	80.529	0.014	0.788	0.786	0.240	351.961	351.745	0.061
81.631	81.641	0.012	0.836	0.834	0.325	375.642	375.155	0.130
82.884	82.892	0.009	0.865	0.864	0.149	395.541	395.683	0.036
84.318	84.323	0.007	0.889	0.885	0.456	414.909	413.965	0.227
85.992	85.994	0.003	0.911	0.907	0.496	434.943	433.761	0.272
88.006	88.003	0.003	0.933	0.928	0.534	456.679	455.232	0.317
90.532	90.522	0.012	0.955	0.953	0.209	481.501	481.777	0.057
93.930	93.905	0.027	0.980	0.974	0.611	511.915	509.744	0.424
99.163	99.102	0.062	1.011	1.005	0.658	554.210	551.396	0.508
112.641	112.377	0.234	1.069	1.063	0.643	646.512	642.138	0.677

Table 5: Results for the second set of specifications

2.4 OpenModelica Code for Assay Manager

```
within CrudeDistillation;

model AssayManager

parameter Real MW = 85 ;
parameter Real Tb = 400 ;
parameter Real SG = 0.85 ;
parameter Integer n = 10 ;
parameter Integer lim = 800 ;

Real MWA, MWa, MWb, MWB, MWavg;       // variables for MW
Real _MWsi[lim], MWx[lim], MWq[n + 1], MWsi[n + 1 ], MWz[n], MWasi[n
    ], MWai[n] ;
parameter Real MW0 = 80 ;

Real SGA, SGB, SGavg;       // variables for SG
Real _SGsi[lim+1], SGx[lim+1], SGq[n + 1], SGz[n], SGasi[n], SGai[n
    ], SGsi[n+1] ;
parameter Real SG0 = 0.7 ;
Integer SGi[n+1] ;

Real TbA, TbB, Tbavg;       // variables for Tb
Real Tbx[lim+1], _Tbsi[lim+1], Tbsi[n+1], Tbq[n + 1], Tbz[n], Tbasi[
    n], Tbai[n] ;
parameter Real Tb0 = 333 ;
Integer Tbi[n+1] ;

Real Pc[n], d15[n], Tc[n], AF[n], x[n] ; // varialbe for critical
    property calculation and mole fractions

Integer i ;

equation

MWavg = ( MW - MW0 ) / MW0 ; // equation starts for MW calculation

MWB = 1 ;

MWA = (MWavg) ^ MWB;
i = 1;

for i in 1:lim loop

  MWx[i] = i/( lim + 1) ;
  _MWsi[i] = (MWA/MWB) * log ( 1 / (1 - (i)/( lim + 1)) ) ^ (1/MWB)
      ;

end for ;

MWa = 1/n * ( MWx[lim] - MWx[1] ); // converting x as a linear
    function of i in 1:n
MWb = MWx[1] - MWa;

for i in 1:( n + 1) loop
```

```
49
50    MWsi[i] = (MWA/MWB * log ( 1/ ( 1 - MWa*i - MWb ) )) ^ (1/MWB) ;
51    MWq[i] = MWB/MWA * (( MWA/MWB * log ( 1/ ( 1 - MWa*i -MWb ) )) ^
          (1/MWB) ) ^ MWB;
52
53  end for;
54
55  for i in 1:n loop
56
57    MWz[i] = exp( -MWB/MWA * MWsi[i] ^ MWB ) - exp( -MWB/MWA * MWsi[i
          +1] ^ MWB ) ;
58    MWasi[i] = 1/MWz[i] * (MWA/MWB) ^ (1/MWB) *( Gamma.PartialGamma( 1
          + 1/MWB , MWq[i] ) - Gamma.PartialGamma( 1+1/MWB, MWq[i+1]))
          ;
59    MWai[i] = MW0 * ( 1 + MWasi[i] );
60
61  end for; // MW calculation ends
62
63  SGavg = ( SG - SG0 ) / SG0 ; // equation starts for SG calculation
64
65  SGB = 3 ;
66
67  SGA = (SGavg/0.619) ^ SGB;
68
69  for i in 0:lim loop
70
71    SGx[i+1] = i/lim ;
72    _SGsi[i+1] = (SGA/SGB * log ( 1 / (1 - (i-0.001)/lim) )) ^ (1/SGB)
          ;
73
74  end for ;
75
76  for i in 0:n loop
77
78    (SGsi[i+1], SGi[i+1]) = Modelica.Math.Vectors.interpolate(
          SGx,_SGsi,i/(n)) ;
79    SGq[i+1] = SGB/SGA * SGsi[i+1] ^ SGB;
80
81  end for;
82
83  for i in 1:n loop
84
85    SGz[i] = exp( -SGB/SGA * _SGsi[integer((i)*(lim)/(n+1))] ^ SGB ) -
          exp( -SGB/SGA * _SGsi[integer((i+1)*(lim)/(n+1))] ^ SGB ) ;
86    SGasi[i] = 1/SGz[i] * (SGA/SGB) ^ (1/SGB) *( Gamma.PartialGamma( 1
          + 1/SGB , SGq[i] ) - Gamma.PartialGamma( 1+1/SGB, SGq[i+1]))
          ;
87    SGai[i] = SG0 * ( 1 + SGasi[i] ) ;
88
89  end for; //equations for SG ends
90
91
92  Tbavg = ( Tb - Tb0 ) / Tb0 ; // equation starts for Tb calculation
93
94  TbB = 1.5 ;
95
```

```
96  TbA = (Tbavg/0.689) ^ TbB;
97
98   Tbx[1] = 0/lim ;
99    _Tbsi[1] = (TbA/TbB * log ( 1 / (1 - (0-(0-0.001)/lim)/(lim)) ) )
          ^ (1/TbB) ;
100
101 for i in 1:lim loop
102
103   Tbx[i+1] = i/lim ;
104   _Tbsi[i+1] = (TbA/TbB * log ( 1 / (1 - (i-(i-0.001)*0.001)/(lim))
          ) ) ^ (1/TbB) ;
105
106 end for ;
107
108 for i in 0:n loop
109
110   (Tbsi[i+1], Tbi[i+1]) = Modelica.Math.Vectors.interpolate(
          Tbx,_Tbsi,i/(n)) ;
111   Tbq[i+1] = TbB/TbA * Tbsi[i+1] ^ TbB;
112
113 end for;
114
115 for i in 1:n loop
116
117   Tbz[i] = exp( -TbB/TbA * _Tbsi[integer((i)*(lim)/(n+1))] ^ TbB ) -
          exp( -TbB/TbA * _Tbsi[integer((i+1)*(lim)/(n+1))] ^ TbB ) ;
118   Tbasi[i] = 1/Tbz[i] * (TbA/TbB) ^ (1/TbB) *( Gamma.PartialGamma( 1
          + 1/TbB , Tbq[i] ) - Gamma.PartialGamma( 1+1/TbB, Tbq[i+1]))
          ;
119   Tbai[i] = Tb0 * ( 1 + Tbasi[i] ) ;
120
121 end for; //equations for Tb ends
122
123 for i in 1:n loop // calcualtion for critical properties
124
125   x[i] = MWz[i] ;
126   d15[i] = SGai[i] ;
127   Tc[i] = CriticalProp.Tc_Riazi( Tbai[i], d15[i] ) ;
128   Pc[i] = CriticalProp.Pc_RiaziDaubert( Tbai[i], d15[i] ) ;
129   AF[i] = CriticalProp.AcentricFactor_LeeKesler2( Tc[i], Pc[i], Tbai
          [i] ) ;
130
131 end for;
132
133 end AssayManager;
```

3 Crude Material Stream

3.1 Background

After the creation of assay manager, there is a need to use the values of critical properties obtained from assay manager in a material stream.

Unlike vanilla material stream, the issue encountered in this approach is the need to define the critical property of the crude cut(s) seperatly like the Chemsep Database. For now, the solution to this problem is to create a compound database from the values we get from DWSIM and use those components in a regular material stream.

4 Distillation Column

4.1 Background

A distillation column is an essential item used in the distillation of liquid mixtures to separate the mixture into its component parts, or fractions, based on the differences in volatilities.

The existing model for Distillation Column requires the user to include all the different models of the column like Condenser, Trays and Reboiler along with the Thermodynamic Packages in a user-defined package/model to use the Distillation Column in any OpenModelica flowsheet. The aim of this new model is to reduce the work of user, reduce the equations and increase the efficiency of the model.

The thermodynamics in this model is integrated by calculating the vapour pressure and the equilibrium constant (K_i) within the model itself. For this method, the thermodynamic package has to be modified a bit to calculate all K_i from i=1 to n in the same package as opposed to calculation of one K for one model like in the orignal model. The advantage of this method is the reduced number of equation due to reduced number of variables because only one model is called.

4.2 Modeling

MESH algorithm is used to solve the variable of distillation column. The main parts of the algorithm include

- Total Mole/Mass balance
- Component Moel/Mass balance
- Summation equations for mole/mass fractions
- Total enthaly balance
- Equilibrium relation for VLE.

In this we consider the condenser to be a total condenser and therfore we don't need the VLE for condenser. To understand MESH algorithm, we need to first work on simple flash operations. The total mole flow balance is

$$M = V + L - F = 0$$

where, M is the net mole flow, V is the vapour mole flow, L is the liquid mole flow, and F is the inlet mole flow. The following equation represents component mole balance.

$$M_i = Vy_i + Lx_i - Fz_i = 0$$

Where M_i is the i^{th} component mole flow, y_i is the i^{th} component vapour mole fraction, x_i is the i^{th} component liquid mole fraction and z_i is the i^{th} component inlet mole fraction.

The enthalpy balance comes next.

$$H = VH^v + LH^l - FH^f = 0$$

Where H^a is the molar enthalpy of the a^{th} material stream.

$$\sum_{i=1}^{n} x_i = 1$$

$$\sum_{i=1}^{n} y_i = 1$$

or

$$\sum_{i=1}^{n} x_i - y_i = 0$$

The first two or just the last equation can be used depending on the degree of freedom.

The last set of equation needed are the equalibrium relations.

$$K_i x_i - y_i = 0$$

where K_i is the equilibrium constant defined for i^{th} species defined as

$$K_i = \frac{y_i}{x_i} = \frac{\phi_{i,L}}{\phi_{i,V}} = \frac{\gamma_i f_{i,L}}{\phi^P_{i,V}}$$

These above equation are modified a bit to use in distillation trays. With N stages in a column, there are N-2 trays in the column with 1^{st} and N^{th} being condenser and reboiler respectively.

The next set of equations known as the MESH equations are used for the N-2 trays in distillation column. For j^{th} tray:

Total mole balance

$$M^T_j = V_j + L_j - V_{j+1} - L_{j-1} - F_j = 0$$

Component mole balance

$$M^T_{j,i} = V_j y_{j,i} + L_j x_{j,i} - V_{j+1} y_{j+1,i} - L_{j-1} x_{j-1,i} - F_j z_{j,i} = 0$$

Total enthaly balance

$$H^T_j = V_j H^V_j + L_j H^L_j - V_{j+1} H^V_{j+1} - L_{j-1} H^L_{j-1} - F_j H^F_j$$

Summation equations

$$\sum_{i=1}^{n} x_{j,i} = 1$$

$$\sum_{i=1}^{n} y_{j,i} = 1$$

$$\sum_{i=1}^{n} x_{j,i} - \sum_{i=1}^{n} y_{j,i} = 0$$

Equilibrium relations

$$K_{j,i} x_{j,i} - y_{j,i} = 0$$

$$K_{j,i} = \frac{y_{j,i}}{x_{j,i}} = \frac{\phi_{j,i,L}}{\phi_{j,i,V}} = \frac{\gamma_{j,i} f_{j,i,L}}{\phi^{P}_{j,i,V}}$$

4.3 Results & Observations

The following conditions were used to verify the column.

Feed Specifications	
Compounds	Benzene-Toluene
Composition(mol/mol)	0.5-0.5
Mole flow rate(mol/s)	100
Pressure (Pa)	101325
Temperature(K)	298.15

Table 6: Specifications of feed

Column Specification	
No. of stages	7
Feed inlet stage	4
Condenser Pressure(Pa)	101325
Reboiler Pressure(Pa)	101325
Condenser Reflux Ratio	2
Bottoms mole flow rate	50
Thermodynamic Package	Raoults Law

Table 7: Specifications of Distillation Column

Streams						
	Distillate			Bottoms		
	DWSIM	OM	error(%)	DWSIM	OM	error(%)
Mole flow	49.999	50.000	0.001	50.000	50.000	0.000
Mole fraction(Benzene)	0.865	0.872	0.841	0.136	0.128	5.425
Mole fraction(Toluene)	0.135	0.128	5.367	0.864	0.872	0.850
Temperature	356.163	356.004	0.045	377.750	378.044	0.078
Pressure	101325	101325	0.000	101325	101325	0.000

Table 8: Results for the output material streams

Column results			
	DWSIM	OM	Error(%)
Condenser duty(kW)	4727.93	4671.51	1.193
Reboiler duty(kW)	-5850.03	-5795.2	0.937

Table 9: Results for the distillation column

4.4 OpenModelica Code for Distillation Column

```
within Distillation.UnitOperations;

model DistillationColumn

//extends Distillation.Files.ThermodynamicPackages.RaoultsLaw;
extends Distillation.GuessModels.GuessInput;

parameter Integer Nt = 7 "number of stages" ;
import Simulator.Files.*;
parameter Simulator.Files.ChemsepDatabase.GeneralProperties C[Nc] "
    Component instances array" ;
parameter Integer Nc = 4 " number of compounds " ;
parameter Integer In_s = 3 " feed entering stage " ;
//V[i] or L[i] are the vapour and liquid mole flows leaving the
    plate respectively
Real V[Nt+1](each start = Fg, each min = 0)"molar flow rates of
    vapour" ;
Real L[Nt](each start = Fg, each min = 0)"Molar flow rates of liquid
    " ;
Real Feed(start = Fg, unit = "mol/s", min = 0)"Mole flow rate of
    feed";
Real F_dis(start = Fg, unit = "mol/s", min = 0)"Mole flow rate of
    distillate";
Real F_bot(start = Fg, unit = "mol/s", min = 0)"Mole flow rate of
    bottoms";
Real PC(start = Pg, unit = "Pa", min = 0) "Pressure of condenser" ;
Real PR(start = Pg, unit = "Pa", min = 0) "Pressure of reboiler" ;
Real Pt[Nt](each start = Pg, each unit = "Pa", each min = 0) "
    Pressure in each tray" ;
Real RLiq(start = Fg, unit = "mol/s", min = 0)"Mole flow rate of
    reflux" ;
Real RR ; // Reflux ratio
Real Tin(start = Tg, unit = "K", min = 0) "Temperature of Inlet
    Stream" ;
Real T_dis(start = Tg, unit = "K", min = 0) "Temperature of
    Distillate Stream" ;
Real T_reb(start = Tg, unit = "K", min = 0) "Temperature of Bottoms
    Stream" ;
Real T_tray[Nt](each start = Tg,each  unit = "K",each  min = 0) "
    Temperature of Trays" ;
Real Hin"enthaly of inlet stream";
Real QC,QR;//energy supplied
Real H_dis,H_bot;//Enthalpy of products
Real H_dis_c[Nc], H_bot_c[Nc];//enthalpy of components in product
Real HV[Nt+1],HL[Nt] ; //enthaly of stream
Real HV_c[Nt+1,Nc] , HL_c[Nt,Nc] ;//enthaly of components in stream
Real x[Nt,Nc](each start = 1/Nc, each min = 0, each max = 1, unit =
    "-") "mole fraction of liquid flow " ;
Real x_tray[Nt,Nc](each start = 1/Nc, each min = 0, each max = 1,
    unit = "-") "mole fraction of tray mixure " ;
Real y[Nt+1,Nc](each start = 1/Nc, each min = 0, each max = 1, unit
    = "-") "mole fraction of vapour flow "  ;
Real x_bot[Nc](each start = 1/Nc, each min = 0, each max = 1, unit =
     "-") " mole fraction of bottoms "  ;
```

```
38  Real x_dis[Nc](each start = 1/Nc, each min = 0, each max = 1, unit =
        "-") "mole fraction of distillate"  ;
39  Real x_reb[Nc](each start = 1/Nc, each min = 0, each max = 1, unit =
        "-") "mole fraction of reboiler"  ;
40  Real z[Nc](each start = 1/Nc, each min = 0, each max = 1, unit = "-"
    ) "mole fraction of feed"  ;
41  Real k_c[Nt+1,Nc](each start = 1) "equlibrium constant for
    components" ;
42  Real xRR[Nc](each start = 1/Nc, each min = 0, each max = 1, unit = "
    -") "mole fraction of reflux";
43
44  Real Pvap_dis[Nc], Pvap_reb[Nc];
45  Real Pvap_tray[Nt,Nc];
46
47  Files.Interfaces.matconn_sim In(Nc = Nc) annotation(
48        Placement(visible = true, transformation(origin = {-248, -40},
              extent = {{-10, -10}, {10, 10}}, rotation = 0),
            iconTransformation(origin = {-250, 0}, extent = {{-10,
            -10}, {10, 10}}, rotation = 0)));
49  Files.Interfaces.matconn_sim Dist(Nc = Nc) annotation(
50        Placement(visible = true, transformation(origin = {250, 316},
            extent = {{-10, -10}, {10, 10}}, rotation = 0),
            iconTransformation(origin = {250, 298}, extent = {{-10,
            -10}, {10, 10}}, rotation = 0)));
51  Files.Interfaces.matconn_sim Bot(Nc = Nc) annotation(
52        Placement(visible = true, transformation(origin = {250, -296},
              extent = {{-10, -10}, {10, 10}}, rotation = 0),
            iconTransformation(origin = {252, -300}, extent = {{-10,
            -10}, {10, 10}}, rotation = 0)));
53
54
55  equation
56
57  //input and solve specs
58
59  In.F = Feed ;
60  In.x_pc = z ;
61  In.T = Tin;
62  In.H = Hin;
63  Dist.F = F_dis ;
64  Dist.x_pc = x_dis ;
65  Dist.T = T_dis;
66  Dist.H = H_dis;
67  Bot.F = F_bot;
68  Bot.x_pc = x_reb ;
69  Bot.T = T_reb;
70  Bot.H = H_bot;
71
72  // calculation of kc
73  for j in 1:Nt loop
74    for i in 1:Nc loop
75      k_c[j,i] = Pvap_tray[j,i]/Pt[j];
76    end for;
77  end for;
78
79  for i in 1:Nc loop
```

```
80    k_c[Nt+1,i] = Pvap_reb[i]/PR;
81  end for;
82
83  //pressure in plates
84
85  for i in 1:Nt loop
86
87     Pt[i] = PC - (PC-PR)/(Nt+1) * i ;
88
89  end for;
90
91  //temperature calculations for products
92
93   for i in 1:Nc loop
94        Pvap_dis[i] = Simulator.Files.ThermodynamicFunctions.Psat(C[i
              ].VP, T_dis);
95        Pvap_reb[i] = Simulator.Files.ThermodynamicFunctions.Psat(C[i
              ].VP, T_reb);
96   end for;
97
98      x_reb[:] = (V[Nt+1] .* y[Nt+1,:] + F_bot.*x_bot[:]) ./ (V[Nt+1]
           + F_bot);
99
100 PC = sum(x_dis[:] .* Pvap_dis[:]);
101 PR = sum(x_reb[:] .* Pvap_reb[:]);
102
103 //enthalpy calculation for products
104
105     for i in 1:Nc loop
106
107       H_dis_c[i] = ThermodynamicFunctions.HLiqId(C[i].SH, C[i].
              VapCp, C[i].HOV, C[i].Tc, T_dis);
108       H_bot_c[i] = ThermodynamicFunctions.HLiqId(C[i].SH, C[i].
              VapCp, C[i].HOV, C[i].Tc, T_reb);
109
110     end for;
111
112   H_dis = sum(x_dis[:] .* H_dis_c[:]) ;
113   H_bot = sum(x_bot[:] .* H_bot_c[:]) ;
114
115 // Total energy balance
116
117   Feed*Hin = QC + QR + F_bot*H_bot + F_dis*H_dis ;
118
119 //condenser heat balance
120
121   V[1]*HV[1] = QC + F_dis*H_dis + RLiq*H_dis ;
122
123 // tray temperature
124
125   for i in 1:Nt loop
126     for j in 1:Nc loop
127       Pvap_tray[i,j] = Simulator.Files.ThermodynamicFunctions.Psat(C
              [j].VP, T_tray[i]);
128     end for;
129   end for;
```

```
130
131     for i in 1:Nt loop
132
133       x_tray[i,:] = (V[i] .* y[i,:] + L[i].*x[i,:]) ./ (V[i] + L[i]);
134       Pt[i] = sum(x_tray[i,:] .* Pvap_tray[i,:]);
135
136     end for;
137
138   //enthalpy calcualtions
139   for j in 1:Nt loop
140     for i in 1:Nc loop
141
142       HL_c[j,i] = ThermodynamicFunctions.HLiqId(C[i].SH, C[i].VapCp, C
              [i].HOV, C[i].Tc, T_tray[j]);
143       HV_c[j,i] = ThermodynamicFunctions.HVapId(C[i].SH, C[i].VapCp, C
              [i].HOV, C[i].Tc, T_tray[j]);
144
145     end for;
146
147     HL[j] = sum(x[j, :] .* HL_c[j,:]) ;
148     HV[j] = sum(y[j, :] .* HV_c[j,:]) ;
149
150   end for;
151
152   for i in 1:Nc loop
153     HV_c[Nt+1,i] = ThermodynamicFunctions.HVapId(C[i].SH, C[i].VapCp,
          C[i].HOV, C[i].Tc, T_reb) ;
154   end for;
155
156   HV[Nt+1] = sum(y[Nt+1, :] .* HV_c[Nt+1]) ;
157
158
159   //total mole balance
160   Feed = F_dis + F_bot ;
161   Feed.*z[:] = F_dis.*x_dis[:] + F_bot.*x_bot[:] ;
162   //sum(x_dis[:]) = 1;
163
164   //condensor
165   V[1] = F_dis + RLiq ;
166   //V[1].*y[1,:] = F_dis.*x_dis[:] + RLiq.*xRR[:] ;
167   RLiq = RR * F_dis ;
168   xRR[:] = x_dis[:] ;
169
170
171   //plate 1
172   //V[1] + L[1] = RLiq + V[2]  ;
173   V[1]*HV[1] + L[1]*HL[1] = RLiq*H_dis + V[2]*HV[2] ;
174   V[1].*y[1,:] + L[1].*x[1,:] = V[2].*y[2,:] + RLiq.*xRR[:] ;
175   y[1,:] = k_c[1,:].*x[1,:];
176   sum(y[1,:]) = 1 ;
177   //sum(x[1,:]) = 1 ;
178   //sum(y[1,:]) - sum(x[1,:]) = 0;
179
180   // for plates exept 1 and nth
181   for i in 2:(Nt-1) loop
182
```

```
183     if i==In_s then
184       //V[i] + L[i] = Feed + L[i-1] + V[i+1];
185       V[i]*y[i,:] + L[i].*x[i,:] = Feed.*z[:] + L[i-1].*x[(i-1),:] + V
              [i+1].*y[i+1,:] ;
186       V[i]*HV[i] + L[i]*HL[i] = Feed*Hin + L[i-1]*HL[i-1] + V[i+1]*HV[
              i+1];
187     else
188       V[i] + L[i] = L[i-1] + V[i+1];
189       V[i]*y[i,:] + L[i].*x[i,:] = L[i-1].*x[(i-1),:] + V[i+1].*y[i
              +1,:] ;
190       //V[i]*HV[i] + L[i]*HL[i] = L[i-1]*HL[i-1] + V[i+1]*HV[i+1];
191     end if;
192
193     y[i,:] = k_c[i,:].*x[i,:];
194     sum(x[i,:]) - sum(y[i,:]) = 0;
195     //sum(y[i,:]) = 1;
196     //sum(x[i,:]) = 1;
197
198
199   end for;
200
201   //for Nt th plate
202   //V[Nt] + L[Nt] = L[Nt-1] + V[Nt+1] ;
203   V[Nt].*y[Nt,:] + L[Nt].*x[Nt,:] = L[Nt-1].*x[Nt-1,:] ;
204   y[Nt,:] = k_c[Nt,:].*x[Nt,:] ;
205   //V[Nt]*HV[Nt] + L[Nt]*HL[Nt] = L[Nt-1]*HL[Nt-1] + V[Nt+1]*HV[Nt+1]
          ;
206   sum(y[Nt,:]) - sum(x[Nt,:]) = 0;
207   //sum(x[Nt,:]) = 1;
208   //sum(y[Nt,:]) = 1;
209
210   //reboiler
211   //L[Nt] = F_bot + V[Nt+1] ;
212   L[Nt].*x[Nt,:] = F_bot.*x_bot[:] + V[Nt+1].*y[Nt+1,:] ;
213   y[Nt+1,:] = x[Nt,:].*k_c[Nt+1,:] ;
214   //sum(y[Nt+1,:]) = 1;
215   sum(x_bot[:]) = 1 ;
216
217   //equilibrium relation
218   for i in 1:Nc loop
219     for j in 1:Nt loop
220       // y[j,i] = x[j,i]*k_c[j,i];
221     end for;
222   end for;
223
224   for i in 1:Nt loop
225
226    //sum(x[i,:]) = 1 ;
227
228   end for;
229
230   end DistillationColumn;
```

5 PFR with a Heterogeneous Catalyst

5.1 Background

The plug flow reactor model (PFR, sometimes called continuous tubular reactor, CTR, or piston flow reactors) is a model used to describe chemical reactions in continuous, flowing systems of cylindrical geometry.

The existing model of PFR does not include a mode to simulate any reaction incolving a heterogeneous catalyst. Therefore, this flat model was created to tacke the vary problem.

5.2 Modeling

The pressure drop for a reaction involving Heterogeneous catalyst is calculated using Ergun equation

$$\Delta p = \frac{150\mu L}{D_p^2}\frac{(1-\epsilon)^2}{\epsilon^3}v_s + \frac{1.75L\rho}{D_p}\frac{(1-\epsilon)}{\epsilon^3}v_s|v_s|$$

where

- μ is the dyanmic viscosity
- L is the length of the reactor
- D_p is the diameter of the reactor
- ϵ is the void fraction of the catalyst
- v_s is the velocity of fluid going through the reactor
- ρ is the density of the fluid

For a catalytic reaction the rate equation is defined by the user. The rate equation can contain T,R1,R2,.....,Rn,P1,P2,.....,Pn.

where T is the temperature of the outlet temperature, Ri is the quantity of reactant in the outlet stream and Pi is the quantity of the product in the outlet stream.

The viscosity of the mixure is also calculated in the model. The values of pure component viscosity (μ_i) at the given temperature T is calculated from the ChemsepDatabase and already created functions for both vapour and liquid.

For a liquid mixure

$$ln(\mu_{mix}) = \sum_{i=1}^{n}(x_i * ln(\mu_i))$$

For a vapour mixure

$$\frac{1}{\mu_{mix}} = \sum_{i=1}^{n} \left(\frac{x_i}{\mu_i} \right)$$

The density of the liquid mixure is calculated using the Racket method and for the vapor part it is calculated using compressibility factor.

5.3 Results & Obserations

Feed Specifications	
Compounds	Methane-Hydrogen-Water-Carbondioxide
Composition(mol/mol)	0.4975-0.0049-0.4975-0
Mole flow rate(mol/s)	5
Pressure (Pa)	202650
Temperature(K)	1000

Table 10: Specifications of feed

Reactor Specification	
Volume of reactor (m^3)	1
Length of reactor (m)	1
Catalyst Loading(kg/m^3)	0.386
Catalyst Diameter (mm)	2
Catalyst Void Fraction	0.4
Thermodynamic Package	Raoults Law

Table 11: Specifications of the PFR

Reactor results			
	DWSIM	OM	Error(%)
Pressure drop (Pa)	1802.99	1636.35	9.242228767
Rate constant	1.40403	1.40791	0.276347371

Table 12: Results for PFR

5.4 OpenModelica code for the reactor

```
within PFR_Catalytic;

model PFR

parameter Real R = 8.3144598;
parameter Real Z = 1;
parameter Real Phase = 1"Use 1 for Vapor and 2 for liquid";

parameter Integer Nc = 4 ;
import data = Simulator.Files.ChemsepDatabase;
parameter data.Methane meth;
parameter data.Water oho;
parameter data.Hydrogen h2;
parameter data.Carbondioxide co2;
parameter data.GeneralProperties C[Nc] = {meth, oho, h2, co2};

//Metahne + Water = H2 + Co2;
parameter Integer Nr = 1 ;
parameter Integer Base = 1 ;
parameter Real dv = 0.01;

//PFR specs

Real Volume"Volume of reactor";
Real Length"Length of reactor";

//Stream specs

Real Fin(unit = "mol/s", start = 3, min = 0)"Inlet mole flow rate";
Real xin_c[Nc](each unit = "-", each start = 1/Nc, each min = 0,
    each max = 1);
Real Tin"Inlet temperature";
Real Pin(unit = "Pa", start = 1, min = 0)"Outlet pressure";
Real Fout(unit = "mol/s", start = 3, min = 0)"Outlet mole flow rate"
    ;
Real xout_c[Nc](each unit = "-", each start = 1/Nc, each min = 0,
    each max = 1)"mole fraction of outlet stream";
Real Tout"Outlet temperature";
Real Pout(unit = "Pa", start = 1, min = 0)"Inlet pressure";

//Phase properties

Real MW[Nc];
Real MWin_avg;
Real MWout_avg;

Real Fin_c[Nc](each unit = "mol/s", each start = 3, each min = 0)"
    Inlet mole flow rate of components" ;
Real Visc_in_c[Nc];
Real Visc_in;
Real rho_in"Inlet density";
Real Finm_c[Nc]"Mass flow of component";
Real Pin_c[Nc]"Partial Pressure of inlet components";
Real Pvapin_c[Nc];
```

```
51  Real Fout_c[Nc](each unit = "mol/s", each start = 3, each min = 0)"
        Outlet mole flow rate of components";
52  Real Visc_out_c[Nc];
53  Real Visc_out;
54  Real rho_out"Outlet density";
55  Real Foutm_c[Nc]"Outlet mass flow of component";
56  Real Pout_c[Nc]"Partial Pressure of outlet components";
57  Real Pvapout_c[Nc];
58
59  //pressure drop variables
60
61  Real eta"Viscosity";
62  Real delp"Pressure drop";
63  Real dp"Catalyst diameter";
64  Real Qin"Inlet volumetric flow";
65  Real Qout"Outlet volumetric flow";
66  Real ev"Void fraction";
67  Real D"Diameter of PFR";
68  Real vel"Velocity of inlet flow";
69
70  //catalyst data
71
72  Real Cat_dia;
73  Real Cat_loading;
74  Real Cat_voidfrac;
75
76  //for reaction variables
77
78  Real Coe[Nc]"Coefficients of reaction";
79  Real Coe_eff[Nc]"Effective coe. of reaction";
80  Real num"Numerator for the rate";
81  Real den"Denominator for the rate";
82  Real Q_c[Nc]"quantity of final products for the reactor";
83  Real rx"Rate of reaction";
84  Real X_base"Conversion of base reactant";
85
86  //heat
87
88  Real Hr"Energy supplied";
89  Real H_extra[Nc];
90  Real Reac_Heat_STP(unit = "kJ/kmol")"Heat of reaction at STP per
        mole base reactant";
91  Real Reac_Heat;
92
93  //
        ------------------------------------------------------------------------------------------


94
95  equation
96
97  //needed inputs
98
99  //reactor specs
100
101 Volume = 1;
102 Length = 1;
```

```
103
104 //input stream
105 Pin = 2*101325;
106 Fin = 5;
107 xin_c[:] = {0.4975124, 0.4975124, 0.0049751, 0};
108 Tin = 1000;
109
110 //coefficients
111 Coe[:] = { -1, -2, 4, 1} ;
112 Coe_eff[:] = Coe./(Coe[Base]) ;
113
114 //conversion
115 X_base = 0.3110202 ;
116
117 //catalyst data
118 Cat_loading = 0.386;
119 Cat_dia = 2;
120 Cat_voidfrac = 0.4;
121
122 //Quantity of proudcts
123 Q_c[:] = Pout/101325 .* xout_c ;
124
125 //rate equations
126 num = 1.02E+15*exp(-243900/8.314/Tout)/Q_c[3]^3.5*(Q_c[1]*Q_c[2]^2-
      Q_c[3]^4*Q_c[4]/(exp(34.218-31266/Tout)));
127 den = (1+6.65E-4*exp(38280/8.314/Tout)*Q_c[1]+1.77E+5*exp
      (-88680/8.314/Tout)*Q_c[2]/Q_c[3]+6.12E-9*exp(82900/8.314/Tout)*
      Q_c[3])^2;
128
129 rx = num/den ;
130
131 //
      ------------------------------------------------------------------------------------------
132
133 //Vapour pressure
134   for i in 1:Nc loop
135     Pvapin_c[i] = Simulator.Files.ThermodynamicFunctions.Psat(C[i].
          VP, Tin);
136     Pvapout_c[i] = Simulator.Files.ThermodynamicFunctions.Psat(C[i].
          VP, Tout);
137   end for;
138
139 //Mole flow of inlet component
140   Fin_c = Fin.*xin_c;
141
142 //Molecular mass
143   MW[:] = C[:].MW;
144   MWin_avg = sum(Finm_c)/Fin * 1E03;
145   MWout_avg = sum(Foutm_c)/Fout * 1E03;
146
147 //Molecular mass of components in stream
148   Finm_c[:] = Fin_c.*MW * 1E-03;
149   Foutm_c[:] = Fout_c.*MW * 1E-03;
150
151 //Calculation of partial pressure
```

```
152     Pin_c[:] = Pin.*xin_c;
153     Pout_c[:] = Pout.*xout_c;
154
155     if Phase == 1 then //vapor part
156
157   //Calculation of density
158     rho_in = Pin * MWin_avg * 1E-03 / ( R * Tin * Z );
159     rho_out = Pout * MWout_avg * 1E-03 / ( R * Tout * Z );
160
161   //calculation of viscsity
162     for i in 1:Nc loop
163
164       Visc_in_c[i] = Simulator.Files.TransportProperties.VapVisc(C[i].
              VapVis[:], Tin);
165       Visc_out_c[i] = Simulator.Files.TransportProperties.VapVisc(C[i
              ].VapVis[:], Tout);
166
167     end for;
168
169     1/Visc_in = sum(xin_c./Visc_in_c);
170     1/Visc_out = sum(xout_c./Visc_out_c);
171
172     else //liquid part
173
174   //Calculation of density
175     rho_in_c = ThermodynamicFunctions.DensityRacket(Nc, Tin, Pin, C
              [:].Pc, C[:].Tc, C[:].Racketparam, C[:].AF, C[:].MW, Pvapin_c
              [:]);
176     rho_in = 1 / sum( (Finm_c./sum(Finm_c)) ./ rho_in_c[:]) / MWin_avg
              ;
177     rho_out_c = ThermodynamicFunctions.DensityRacket(Nc, Tout, Pout, C
              [:].Pc, C[:].Tc, C[:].Racketparam, C[:].AF, C[:].MW, Pvapout_c
              [:]);
178     rho_in = 1 / sum( (Foutm_c./sum(Foutm_c)) ./ rho_out_c[:]) /
            MWout_avg;
179
180   //calculation of viscsity
181     for i in 1:Nc loop
182
183       Visc_in_c[i] = Simulator.Files.TransportProperties.LiqVisc(C[i].
              VapVis[:], Tin);
184       Visc_out_c[i] = Simulator.Files.TransportProperties.LiqVisc(C[i
              ].VapVis[:], Tout);
185
186     end for;
187
188     log(Visc_in) = sum(xin_c.*log(Visc_in_c));
189     log(Visc_out) = sum(xout_c.*log(Visc_out_c));
190
191   end if;
192
193   //calculation of volumetric flow rates
194     Qin = sum(Finm_c)/rho_in;
195     Qout = sum(Foutm_c)/rho_out;
196
197   //outlet mole calcualtion
```

```
198     Fout_c[:] = Fin_c - Fin_c[Base] .* X_base .* Coe_eff ;
199     Fout = sum(Fout_c[:]);
200     xout_c[:] = Fout_c./(sum(Fout_c[:])) ;
201     Tout = Tin;
202
203   //Reaction heat
204   Reac_Heat_STP = sum(Coe .* C[:].IGHF)/abs(Coe[Base]) * 1E-03;
205   for i in 1:Nc loop
206     H_extra[i] = Heat_T(C[i].VapCp[:], Tout);
207   end for;
208   Reac_Heat = Reac_Heat_STP + sum(Coe.*H_extra)/abs(Coe[Base]) ;
209
210   Hr = Reac_Heat * Cat_loading/1000 * Volume ;
211
212   //outlet pressure
213   D = ( 4*Volume / (3.14 * Length) )^(1/2) ;
214   ev = Cat_voidfrac;
215   dp = Cat_dia/1000;
216   vel = Qout/(3.14 * D^2 / 4);
217   eta = Visc_out;
218
219   delp = ( 150 * eta * Length / dp ^ 2 * (1 - ev) ^ 2 / ev ^ 3 * vel +
          1.75 * Length * rho_out / dp * (1 - ev) / ev ^ 3 * vel ^ 2 ) ;
220   Pout = Pin - delp;
221
222
223   end PFR;
```