Summer Fellowship Report

On

Custom Unit Operations in DWSIM using Python

Submitted by

Charan R

Under the guidance of

Prof.Kannan M. Moudgalya

Chemical Engineering Department

IIT Bombay

August 7, 2018

Acknowledgment

I wish to express our profound gratitude to our internship guide Dr. Kannan Moudgalya, Professor, Department of Chemical Engineering, IIT Bombay for his constant support and supervision throughout the internship. We have reaped benefits from his wisdom, guidance and patience all along. I wish to thank my professor Dr.P.R.Naren, SASTRA Deemed University, for guiding us through the project. I thank my mentors, Priyam Nayak, A.S.Rahul and Pravin Dalve for giving us invaluable inputs all through the fellowship. I would also like to extend my gratitudeto the FOSSEE Team, IIT-Bombay for providing me with this wonderful oppurtunity.

Contents

Chapter 1

Introduction

DWSIM is a free and open source steady state chemical process simulator. It follows the sequential modular approach. *DWSIM* has more than fifteen thermodynamic property packages built into it along with basic unit operations that can be used to build a process flowsheet. Furthermore, a user can also develop custom unit operations in *DWSIM* using Python scripts. This feature helps to develop unit operations that are otherwise not inherently available in *DWSIM*. This enhances the workability of the user to incorporate various features in *DWSIM* according to the needs of the user. This increasses the versatility of *DWSIM* to be used in commercial process industries without the need for any proprietary tool.

Chapter 2

Inbuilt Functions

2.1 Calculate

Function used to calculate the equilibrium values and phase properties for a material stream based on set parameters.

2.2 Clear

Clear all the pre-existing values present in the material stream.

2.3 CalcEquilibrium

Function used to calculate equilibrium values by performing flash based on set parameters.

2.4 GetNumCompounds

Function that returns the number of compounds present in the given material stream.

2.5 GetPhase

Function that gets the phase object specified from the given material stream.

2.6 GetProp

Function used to extract values from the material streams.

2.7 SetProp

Function used to set values to the material streams.

2.8 WriteMessage

Function that Displays a string in the message box of the DWSIM UI.

Chapter 3

Custom Modelling of a Basic Mixer

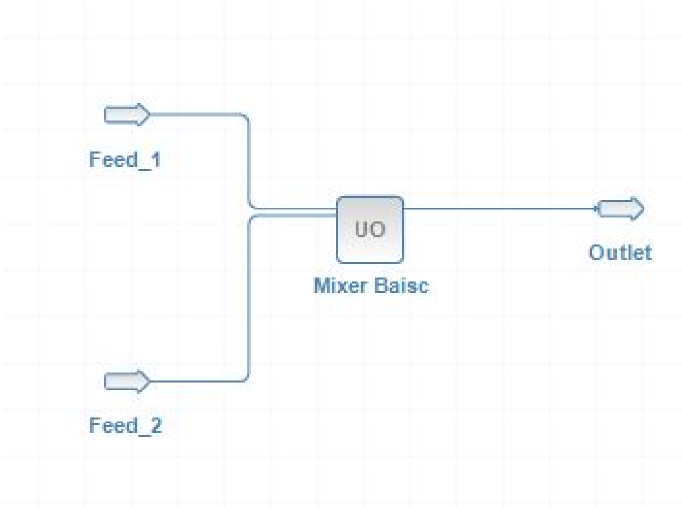
3.1 Objective

The objective is to develop a model that can mix two material streams to obtain one new material stream and calculate its properties based on the inlet streams. This particular model is created only for two inlet materials streams that contain only two components.

3.2 Assumptions

- Steady state.
- Complete mixing.
- Ideal mixing.
- No heat loss.
- No reaction between components.

3.3 Flowsheet



3.4 Equations

3.4.1 Overall Mass Balance

$$M_{outlet} = \sum_{i=1}^{2} M_i \tag{3.1}$$

3.4.2 Overall Molar Balance

$$F_{outlet} = \sum_{i=1}^{2} F_i \tag{3.2}$$

3.4.3 Individual Molar Balance

$$F_{outlet} * x_{outlet} = \sum_{i=1}^{2} (F_i * x_i) \tag{3.3}$$

3.4.4 Overall Energy Balance

$$H_{outlet} * M_{outlet} = \sum_{i=1}^{2} (H_i * M_i) \tag{3.4}$$

3.4.5 Average Pressure

$$P_{outlet} = \sum_{i=1}^{2} P_i/2 \tag{3.5}$$

3.5 Algorithm

- Extract required properties from the inlet streams.
- Calculate output mass flow using input stream mass flows using equation 3.1
- Calculate output composition from input stream molar flow using equations 3.2 and 3.3
- Calculate output enthalpy from input stream enthalpy using equation 3.4
- Calculate output pressure as average of inlet pressure using equation 3.5
- Perform PH flash on output stream to calculate other properties

3.6 Python Script

Listing 3.1: Basic Mixer

```python
#=========================================
#Mixer using custom unit operation
#Charan R
#SASTRA University

from DWSIM.Thermodynamics import *

#=========================================
#Extracting input from stream 1
feed1 = ims1
P_1 = feed1.GetProp("pressure", "Overall", None, "", "")
massflow_1 = feed1.GetProp("totalFlow" ,"Overall", None, "", "mass")
molfrac_1 = feed1.GetProp("fraction", "Overall", None, "", "mole")
molflow_1 = feed1.GetProp("totalFlow" ,"Overall", None, "", "mole")
enthalpy_1 = feed1.GetProp("enthalpy" ,"Overall", None, "Mixture", "mass")


#=========================================
#Extracting input from stream 2
feed2 = ims2
P_2 = feed2.GetProp("pressure", "Overall", None, "", "")
massflow_2 = feed2.GetProp("totalFlow" ,"Overall", None, "", "mass")
molfrac_2 = feed2.GetProp("fraction", "Overall", None, "", "mole")
molflow_2 = feed2.GetProp("totalFlow" ,"Overall", None, "", "mole")
enthalpy_2 = feed2.GetProp("enthalpy" ,"Overall", None, "Mixture", "mass")


#=========================================
#Initiating variables
massflow_3 = [0]
molflow_3 = [0]
enthalpy_3= [0]
P_3=[0]

#=========================================
#Calculation

#Calculating outlet mass flow
massflow_3[0] = massflow_1[0] + massflow_2[0]

#Calculating total outlet molar flow
molflow_3[0] = molflow_1[0] + molflow_2[0]

#Calculating the specific enthalpy of outlet stream
totalenthalpy = (massflow_1[0] * enthalpy_1[0]) + (massflow_2[0] * enthalpy_2[0])
```

```
enthalpy_3[0] = totalenthalpy/massflow_3[0]

#Calculating total molar flow of each component in outlet stream
totalmolflow_comp1= (molfrac_1[0] * molflow_1[0]) + (molfrac_2[0] * molflow_2[0])
totalmolflow_comp2= (molfrac_1[1] * molflow_1[0]) + (molfrac_2[1] * molflow_2[0])

#Calculating mol fraction of each component in the outlet stream
molfrac_comp1 = totalmolflow_comp1 / molflow_3[0]
molfrac_comp2 = totalmolflow_comp2 / molflow_3[0]
molfrac_3 = [molfrac_comp1,molfrac_comp2]

#Calculating outlet pressure by taking the average of the inlet streams
P_3[0] = (P_1[0] + P_2[0]) * 0.5

#========================================
#Setting output stream values
out = oms1
out.Clear()
out.SetProp("enthalpy", "Overall", None, "", "mass",enthalpy_3)
out.SetProp("pressure", "Overall", None, "", "", P_3)
out.SetProp("fraction", "Overall", None, "", "mole", molfrac_3)
out.SetProp("totalFlow", "Overall", None, "", "mass", massflow_3)
out.PropertyPackage.DW_CalcEquilibrium(PropertyPackages.FlashSpec.P,
    PropertyPackages.FlashSpec.H)


#End of script
#============================================
```

3.7 Input Stream Specifications

Input Specifications			
Object	Feed_2	Feed_1	
Temperature	330	300	K
Pressure	202650	101325	Pa
Molar Flow	100	100	mol/s
Molar Fraction (Mixture) / Water	0.3	0.4	
Molar Fraction (Mixture) / Acetic acid	0.7	0.6	

3.8 Results

Results			
Object	Outlet(Custom UO)	DWSIM	
Temperature	314.978	314.978	K
Pressure	151988	151988	Pa
Mass Flow	9.06794	9.06794	kg/s
Molar Flow	200	200	mol/s
Molar Fraction (Mixture) / Acetic acid	0.65	0.65	
Molar Fraction (Mixture) / Water	0.35	0.35	

3.9 Additional Notes

A Tutorial based on this model was created to help beginners get a basic understanding of the functionality and syntax of the Python script that was used.

3.10 Nomenclature

x	Mol Fraction of a component in a stream
F	Molar Flow rate
H	Mass Specific Enthalpy
M	Mass Flow rate
P	Pressure of the streams

Chapter 4

Custom Modelling of a Generic Mixer

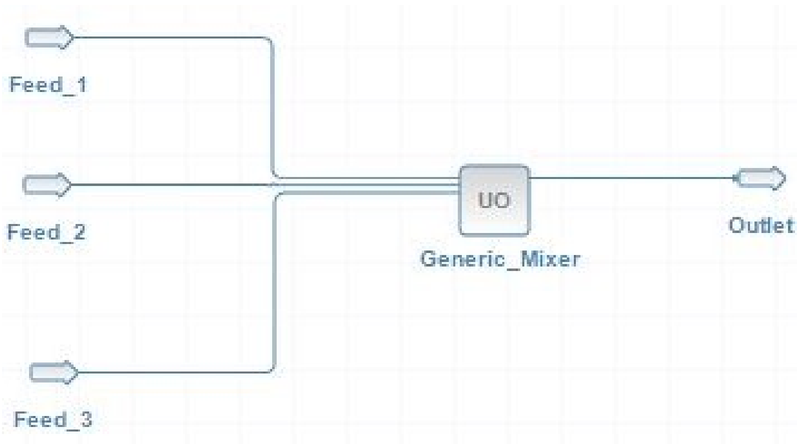
4.1 Objective

The objective is to develop a model that can mix two or more material streams to obtain one new material stream and calculate its properties. This model works without any limitation on the number of the components.

4.2 Assumptions

- Steady state.
- Complete mixing.
- Ideal mixing.
- No heat loss.
- No reaction between components.

4.3 Flowsheet



4.4 Equations

4.4.1 Overall Mass Balance

$$M_{outlet} = \sum_{i=1}^{2} M_i \tag{4.1}$$

4.4.2 Overall Molar Balance

$$F_{outlet} = \sum_{i=1}^{n} F_i \tag{4.2}$$

4.4.3 Individual Molar Balance

$$F_{outlet} * x_{outlet} = \sum_{i=1}^{n} (F_i * x_i) \tag{4.3}$$

4.4.4 Overall Energy Balance

$$H_{outlet} * M_{outlet} = \sum_{i=1}^{n} (H_i * M_i) \tag{4.4}$$

4.4.5 Average Pressure

$$P_{outlet} = \sum_{i=1}^{n} P_i/n \tag{4.5}$$

4.5 Algorithm

- Get input stream properties
- Calculate output mass flow using input stream mass flows of all input streams (Restricted to 6) using equation 4.1
- Calculate output composition from input stream molar flow and component IDs using equations 4.2 and 4.3
- Calculate output enthalpy from input streams enthalpy using equation 4.4
- Calculate output pressure as average of all inlet stream pressures using equation 4.5
- Perform PH flash on output stream to calculate other properties of outlet stream.

4.6 Python Script

Listing 4.1: Generic Mixer

```
#
    ==============================================================================
#Mixer using custom unit operation
#Charan R
#SASTRA University

#Header file to access the thermodynamic Packages
from DWSIM.Thermodynamics import *

#
    ==============================================================================
#Assigning inlet streams
#Hard coded the following segment as could not find a way to make it a generic model
feed = [0] * 6            #6 is the maximum no of inlet connections allowed
for i in range (0,5):
    #Checks if is any stream is attached in the respective port
    if Me.GraphicObject.InputConnectors[i].IsAttached :
        if (i==0):
            feed[i] = ims1
            no_of_feed = i
        if (i==1):
            feed[i] = ims2
            no_of_feed = i
        if (i==2):
            feed[i] = ims3
            no_of_feed = i
        if (i==3):
            feed[i] = ims4
            no_of_feed = i
        if (i==4):
            feed[i] = ims5
            no_of_feed = i
        if (i==5):
            feed[i] = ims6
            no_of_feed = i
no_of_feed = no_of_feed + 1

#
    ==============================================================================
# Initialisation for feed stream
P = [0] * no_of_feed
massflow = [0] * no_of_feed
```

```python
molfrac = [0] * no_of_feed
molflow = [0] * no_of_feed
enthalpy = [0] * no_of_feed

# Initialisation  for  outlet  stream

#Get the number of componenets in the outlet stream
noc = int(feed[0].GetNumCompounds())
massflow_out = [0]
molflow_out = [0]
enthalpy_out= [0]
P_out=[0]
totalmolflow = [0] * noc
molfrac_out = [0] * noc

# Initialisation  for  Calculation  Purposes
totalenthalpy  = 0
P_tot = 0

# ====================================================================

#Extracting input from feed streams

# Get compound IDs in the feed stream
noc = int(feed[0].GetNumCompounds())
ids  = feed[0].ComponentIds

#Extracting input from inlet streams
for i in range (0 , no_of_feed):
    P[i]  = feed[i].GetProp("pressure", "Overall", None, "", "")
    massflow[i]  = feed[i].GetProp("totalFlow" ,"Overall", None, "", "mass")
    molfrac[i]  = feed[i].GetProp("fraction", "Overall", None, "", "mole")
    molflow[i]  = feed[i].GetProp("totalFlow" ,"Overall", None, "", "mole")
    enthalpy[i]  = feed[i].GetProp("enthalpy" ,"Overall", None, "Mixture", "mass")

#NOTE : All the values are returned as vectors (1-D Array) and not as double values.
#        Therefore the above arrays  will  be treated  as  multidimensional arrays

# ====================================================================

#Calculation

for i in range(0,no_of_feed):
    #Calculating outlet mass flow
    massflow_out[0]  = massflow_out[0] + massflow[i][0]
    #Calculating total outlet  molar flow
    molflow_out[0]  = molflow_out[0] + molflow[i][0]
```

```python
    #Calculating the specific enthalpy of outlet stream
    totalenthalpy = totalenthalpy + massflow[i][0] * enthalpy[i][0]
    P_tot = P_tot +P[i][0]
    for j in range (0,noc) :
        #Calculating total molar flow of each component in outlet stream
        totalmolflow[j] = totalmolflow[j] + molfrac[i][j] * molflow[i][0]

#Calculating the specific enthalpy of the outlet stream
enthalpy_out[0] = totalenthalpy / massflow_out[0]
#Calculating the totalmolflow of the outlet stream
totalflow = sum(totalmolflow)

#NOTE : totalflow can also be calculated from molflow_out
#but calculatng the sum of a 2-D array is confusing and
#cumbersome and therefore i used this method

#Calculating the outlet composition
for i in range (0,noc) :
    molfrac_out[i] = totalmolflow[i] / totalflow

#Calculating outlet pressure by taking the average of the inlet streams
P_out[0] = P_tot / no_of_feed

#
    ==================================================================


#Setting output stream values
out = oms1
out.Clear()
out.SetProp("enthalpy", "Overall", None, "", "mass",enthalpy_out)
out.SetProp("pressure", "Overall", None, "", "", P_out)
out.SetProp("fraction", "Overall", None, "", "mole", molfrac_out)
out.SetProp("totalFlow", "Overall", None, "", "mass", massflow_out)
out.PropertyPackage.DW_CalcEquilibrium(PropertyPackages.FlashSpec.P,
    PropertyPackages.FlashSpec.H)


#End of script
#
    ==================================================================
```

4.7 Input Stream Specifications

Input Specifications				
Object	Feed_3	Feed_2	Feed_1	
Temperature	300	330	345	K
Pressure	202650	101325	101325	Pa
Molar Flow	150	200	100	mol/s
Molar Fraction (Mixture) / Methanol	0.5	0.1	0.25	
Molar Fraction (Mixture) / Water	0.25	0.2	0.1	
Molar Fraction (Mixture) / Ethanol	0.15	0.3	0.3	
Molar Fraction (Mixture) / 1-propanol	0.1	0.4	0.35	

4.8 Results

Results			
Object	Outlet(Custom UO)	DWSIM	
Temperature	325.276	325.276	K
Pressure	135100	135100	Pa
Molar Flow	450	450	mol/s
Molar Fraction (Mixture) / Methanol	0.266667	0.266667	
Molar Fraction (Mixture) / Water	0.194444	0.194444	
Molar Fraction (Mixture) / Ethanol	0.25	0.25	
Molar Fraction (Mixture) / 1-propanol	0.288889	0.288889	

4.9 Additional Notes

The number of inlet streams are restricted to 6 due to restrictions in DWSIM.

4.10 Nomenclature

x	Mol Fraction of a component in a stream
F	Molar Flow rate
H	Mass Specific Enthalpy
M	Mass Flow rate
P	Pressure of the streams

Chapter 5

Custom Modelling of Evaporator

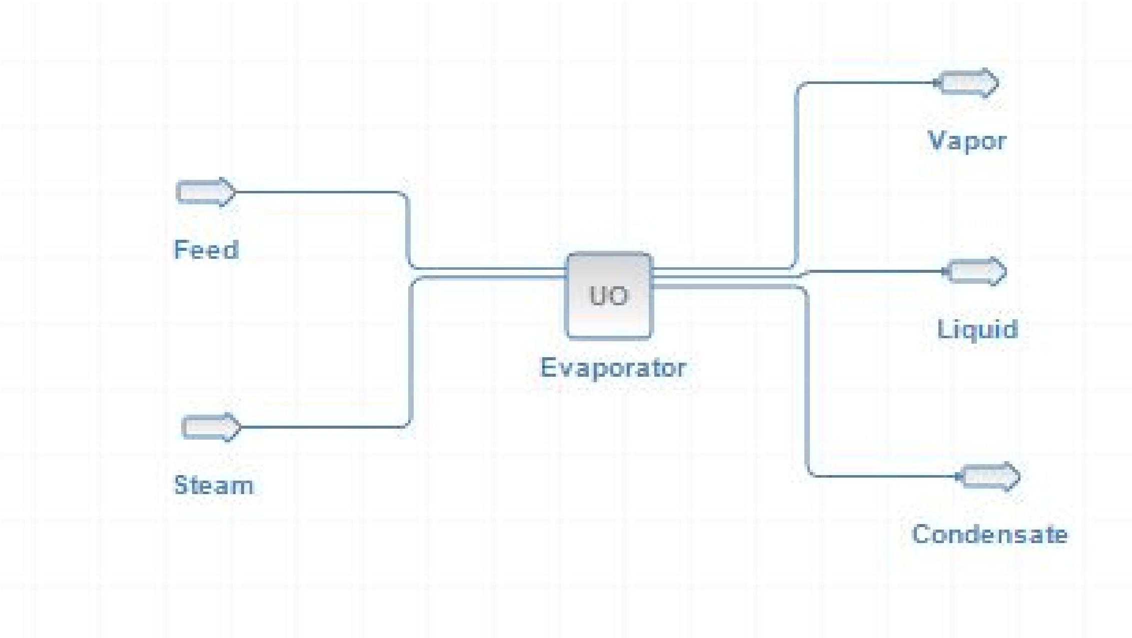
5.1 Objective

The objective of this model is to simulate a simple Evaporator where the feed material stream is heated using another material stream. The resultant heated feed stream is flashed and seperated based on its phases into a liquid stream and a vapor stream. The liquid and vapor streams are the outlet streams from the evaporator model.

5.2 Assumptions

- Steady state.
- Complete heat transfer.
- No loss of energy to surroundings.

5.3 Flowsheet



5.4 Equations

5.4.1 Feed Stream Mass Balance

$$M_{feed} = M_{liquid} + M_{vapor} \tag{5.1}$$

5.4.2 Condensate stream Mass Balance

$$M_{steam} = M_{condensate} \tag{5.2}$$

5.4.3 Steam Energy Balance

$$H_{steam} * M_{steam} = H_{condensate} * M_{condensate} + \Delta E \tag{5.3}$$

5.4.4 Feed Energy Balance

$$H_{feed} * M_{feed} + \Delta E = H_{vapor} * M_{vapor} + H_{liquid} * M_{liquid} \tag{5.4}$$

5.5 Algorithm

- Get input stream properties.
- Get condensate temperature from user.
- The condensate stream is set as same as the steam material stream except for temperature. The temperature is set as the condensate temperature (user input). This is based on equation 5.2
- Check if condensate temperature less than inlet steam temperature. Display error if it is not.
- Get specifc enthalpy of the material stream at the condensate temperature.
- Calculate change in enthalpy by comparing it with inlet steam enthalpy using equation 5.3
- The change in enthalpy is the amount of heat exchanged (assuming no losses).
- Create a temporary clone stream of the feed but only increase the enthalpy by amount of heat exchanged. This is based on equations 5.1 and 5.4
- Perform a PH flash on the clone stream to calculate phase properties.
- Extract the liquid and vapor phase properties from the temporary clone stream.
- The properties extracted are mass flow, specifc enthalpy, composition and pressure.
- Set the extracted values for the Vapor and Liquid material streams.
- Perform a PT flash on both the streams to calculate other properties.

5.6 Python Script

Listing 5.1: Evaporator

```python
#Simple Evaporator
#Charan R
#==============================================================================
#Importing Required Namespaces
import clr
import sys
clr.AddReference("DWSIM.Interfaces")
from DWSIM.Interfaces import *
from DWSIM.Thermodynamics import *
from System import Math
from System import Array

#==============================================================================
#Getting values from Input Feed stream
feed_in = ims1
T_feed_in= feed_in.GetProp("temperature", "Overall", None, "", "")
P_feed_in = feed_in.GetProp("pressure", "Overall", None, "", "")
massflow_feed_in = feed_in.GetProp("totalFlow" ,"Overall", None, "", "mass")
molfrac_feed_in = feed_in.GetProp("fraction", "Overall", None, "", "mole")
enthalpy_feed_in = feed_in.GetProp("enthalpy" ,"Overall", None, "Mixture", "mass")
noc_feed_in = int(feed_in.GetNumCompounds())
ids_feed_in = feed_in.ComponentIds

#Getting values from Input Water Stream
steam_in = ims2
T_steam_in = steam_in.GetProp("temperature", "Overall", None, "", "")
P_steam_in = steam_in.GetProp("pressure", "Overall", None, "", "")
massflow_steam_in = steam_in.GetProp("totalFlow" ,"Overall", None, "", "mass")
molfrac_steam_in = steam_in.GetProp("fraction", "Overall", None, "", "mole")
enthalpy_steam_in = steam_in.GetProp("enthalpy" ,"Overall", None, "Mixture", "mass")
noc_steam_in = int(steam_in.GetNumCompounds())
ids_steam_in = steam_in.ComponentIds

#==============================================================================
# Initialize
T_Condensate = []
enthalpy_feed_out = [0]
pure_stream = False
```

```
# ==================================================================

for i in molfrac_steam_in :
    if (i == 1):
        pure_stream = True

if (pure_stream):
    Flowsheet.WriteMessage("Error : Cannot calculate for Pure water input stream")

else :
    if (Condensate_Temperature <= T_steam_in[0]) :
        T_Condensate.append(Condensate_Temperature)
        #Calculating Condensate Stream
        condensate = oms3
        condensate.Clear()
        condensate.SetProp("temperature", "Overall", None, "", "", Array[float](T_Condensate))
        condensate.SetProp("pressure","Overall",None,"","",P_steam_in)
        condensate.SetProp("fraction", "Overall", None, "", "mole", molfrac_steam_in)
        condensate.SetProp("totalFlow", "Overall", None, "", "mass", massflow_steam_in)
        condensate.SpecType = Enums.StreamSpec.Temperature_and_Pressure
        condensate.Calculate(True,True)

        # ==================================================================

        #Calculating the Change in enthalpy
        H_condensate = condensate.GetProp("enthalpy","Overall",None,"Mixture","mass")
        del_H = (enthalpy_steam_in[0] - H_condensate[0]) * massflow_steam_in[0]
        enthalpy_feed_out[0] = (enthalpy_feed_in[0] * massflow_feed_in[0]) + del_H
        enthalpy_feed_out[0] = enthalpy_feed_out[0] / massflow_feed_in[0]
        Flowsheet.WriteMessage(str(H_condensate))
        # ==================================================================

        #Calculating Properties of Total Feed out
        feed_out = oms1
        feed_out.Clear()
        feed_out.SetProp("enthalpy","Overall",None,"Mixture","mass", enthalpy_feed_out)
        feed_out.SetProp("pressure", "Overall", None, "", "", P_feed_in)
        feed_out.SetProp("fraction", "Overall", None, "", "mole", molfrac_feed_in)
        feed_out.SetProp("totalFlow", "Overall", None, "", "mass", massflow_feed_in)
        feed_out.SpecType = Enums.StreamSpec.Pressure_and_Enthalpy
        feed_out.Calculate(True,True)
```

```
#The first boolean argument of the Calculate function is to tell it to calculate
# the equilibrium, while the second one refers to the phase properties
# =================================================================================

#Extracting the Liquid Data from Feed out stream
H_liquid_out = feed_out.GetProp("enthalpy", "Liquid", None, "Mixture", "mass")
T_liquid_out = feed_out.GetProp("temperature","Liquid",None,"","")
P_liquid_out = feed_out.GetProp("pressure", "Liquid", None, "", "")
massflow_liquid_out = feed_out.GetProp("totalFlow", "Liquid", None,"", "mass")
molfrac_liquid_out = feed_out.GetProp("fraction", "Liquid", None, "", "mole")

#Extracting the Vapor Data from Feed out stream
H_vapor_out = feed_out.GetProp("enthalpy", "Vapor", None, "Mixture", "mass")
T_vapor_out = feed_out.GetProp("temperature","Vapor",None,"","")
P_vapor_out = feed_out.GetProp("pressure", "Vapor", None, "", "")
massflow_vapor_out = feed_out.GetProp("totalFlow", "Vapor", None, "", "mass")
molfrac_vapor_out = feed_out.GetProp("fraction", "Vapor", None, "", "mole")

# =================================================================================

#Setting up Vapor Stream
vap_out = oms1
vap_out.Clear()
vap_out.SetProp("pressure", "Overall", None, "", "",P_vapor_out)
vap_out.SetProp("fraction", "Overall", None, "", "mole", molfrac_vapor_out)
vap_out.SetProp("totalFlow", "Overall", None, "", "mass", massflow_vapor_out)
if (massflow_vapor_out <> 0):
    vap_out.SetProp("enthalpy","Overall",None,"Mixture","mass",H_vapor_out)
    vap_out.SpecType = Enums.StreamSpec.Pressure_and_Enthalpy
    vap_out.Calculate(True,True)

else :
    vap_out.SetProp("temperature","Overall",None,"","",T_vapor_out)
    Flowsheet.WriteMessage("None of the Feed is Vaporised")

#Setting up Liquid Stream
liq_out = oms2
liq_out .Clear()
liq_out .SetProp("pressure", "Overall", None, "", "",P_liquid_out)
liq_out .SetProp("fraction", "Overall", None, "", "mole", molfrac_liquid_out)
liq_out .SetProp("totalFlow", "Overall", None, "", "mass", massflow_liquid_out)
if (massflow_liquid_out <> 0) :
```

```
                liq_out .SetProp("enthalpy","Overall",None,"Mixture","mass",H_liquid_out)
                liq_out .SpecType = Enums.StreamSpec.Pressure_and_Enthalpy
                liq_out .Calculate(True,True)
            else :
                liq_out .SetProp("temperature","Overall",None,"","",T_liquid_out)
                Flowsheet.WriteMessage("Feed is completely Vaporised")

        else :
            Flowsheet.WriteMessage("Error : Condensate Temperature greater than Steam
                Temperature")

#End of Script
#
    ==================================================================
```

5.7 Input Stream Specifications

Input Specifications			
Object	Steam	Feed	
Temperature	353	400	K
Pressure	101325	101325	Pa
Mass Flow	40	100	kg/s
Molar Flow	2205.67	1280.93	mol/s
Molar Fraction (Mixture) / Salicylic acid	0.001	0.5	
Molar Fraction (Mixture) / Water	0.999	0.5	

5.8 User Specifications

Condensate Temperature 323 K

5.9 Results

Input and Output						
Object	Vapor	Steam	Liquid	Feed	Condensate	
Temperature	405.013	353	405.013	400	323	K
Mass Flow	5.4099	40	94.5901	100	40	kg/s
Vapor Phase Mass Flow	5.4099	0	0	3.55162	0	kg/s
Liquid Phase (Mixture) Mass Flow	0	40	94.5901	96.4484	40	kg/s

5.10 Additional Notes

The vapor and liquid streams are composed of only vapor and liquid respectively. This is due to the fact that the streams are calculated by PH flash and not by PT flash.

5.11 Nomenclature

H Mass Specific Enthalpy
M Mass Flow rate
E Energy Flow

Chapter 6

Custom Modelling of Absorption Column

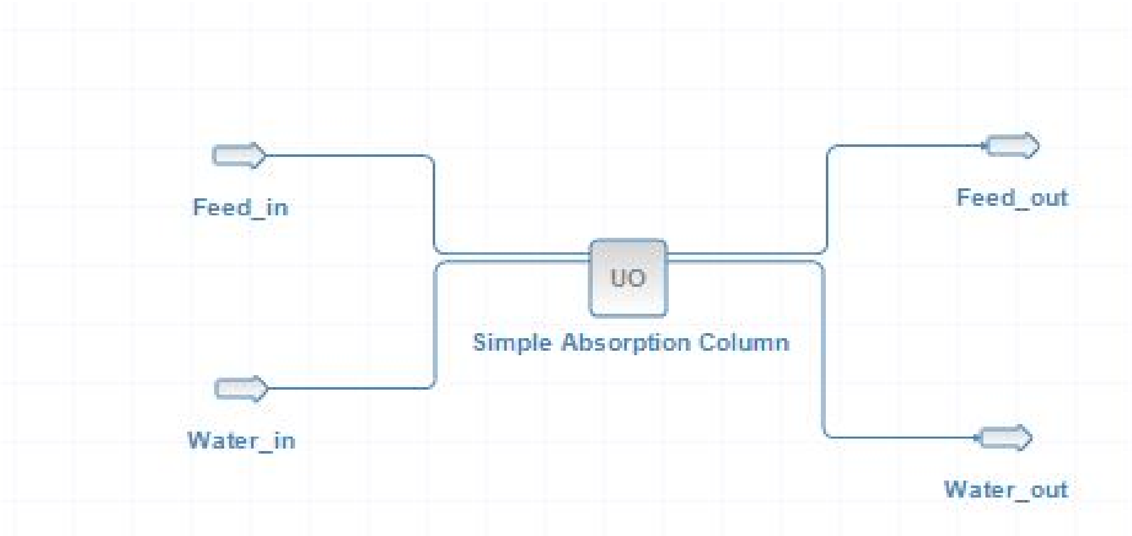
6.1 Objective

The objective is to develop a model which simulates a absorption column. In addition to simulating the absorption column it also calculates the ideal number of stages required using Kremser's equation and Wilson correlation

6.2 Assumptions

- Steady state conditions.
- Stages are Completely Efficient.
- Equilibrium constant is only an Estimate.

6.3 Flowsheet



6.4 Equations

6.4.1 Wilson Correlation

$$k = \frac{P_c}{P_{op}} * exp(5.37 * (1 + a) * (1 - \frac{T_c}{T_{op}})) \tag{6.1}$$

6.4.2 Absorption Factor

$$A = \frac{L_{avg}}{k * V_{avg}} \tag{6.2}$$

6.4.3 Kremsrer Equation

$$N = \frac{Ln(\frac{x_{feedin} - k * x_{waterin})}{x_{feedout} - k * x_{waterout})} * \frac{A-1}{A} + \frac{1}{A})}{Ln(A)} \tag{6.3}$$

6.4.4 Overall Molar Balance

$$M_{feedin} + M_{waterin} = M_{feedout} + M_{waterout} \tag{6.4}$$

6.4.5 Solute Molar Balance

$$M_{feedin} * x_{feedin} + M_{waterin} * x_{waterin} = M_{feedout} * x_{feedout} + M_{waterout} * x_{waterout} \tag{6.5}$$

6.5 Algorithm

- Get input stream properties.
- Get required inputs from user.
- Calculate molar fraction of solute in outlet streams using equation 6.5.
- Calculate outlet flow of outlet streams using equation 6.4.
- Get k value from user input, if k=0 use Wilson's Correlation (6.1).
- Calculate the Absorption factor (6.2)
- Calculate the Ideal number of Equilibrium Stages using Kremser Equation(6.3).
- Set the outlet stream at operating conditions(user input) .
- Performs PT flash on outlet streams to calculate other properties.

6.6 Python Script

```
#Simple Absorption Column

import clr
import sys

clr.AddReference('DWSIM.MathOps.DotNumerics')
from System import Math
from DotNumerics import *
from DotNumerics.ODE import *
from DotNumerics.LinearAlgebra import *
from System import Array

# ======================================================================
#Getting values from Input Feed stream                           # Alternative Method to get inlet stream values
feed_in = ims1                                                   # am_compound = feed_in.GetPhase('Mixture').Compounds[compound]
T_feed_in= feed_in.GetProp("temperature", "Overall", None, "", "")   #temp = feed_in.GetPhase('Mixture').Properties.temperature
P_feed_in = feed_in.GetProp("pressure", "Overall", None, "", "")
massflow_feed_in = feed_in.GetProp("totalFlow" ,"Overall", None, "", "mass")
molfrac_feed_in = feed_in.GetProp("fraction", "Overall", None, "", "mole")
molflow_feed_in = feed_in.GetProp("totalFlow" ,"Overall", None, "", "mole")
noc_feed_in = int(feed_in.GetNumCompounds())
ids_feed_in = feed_in.ComponentIds

#Getting values from Input Water Stream
water_in = ims2
T_water_in = water_in.GetProp("temperature", "Overall", None, "", "")
P_water_in = water_in.GetProp("pressure", "Overall", None, "", "")
massflow_water_in = water_in.GetProp("totalFlow" ,"Overall", None, "", "mass")
molfrac_water_in = water_in.GetProp("fraction", "Overall", None, "", "mole")
molflow_water_in = water_in.GetProp("totalFlow" ,"Overall", None, "", "mole")
noc_water_in = int(water_in.GetNumCompounds())
ids_water_in = water_in.ComponentIds

# ======================================================================
#INITIALISATION
molflow_water_out=[0]
molflow_feed_out=[0]
molfrac_feed_out=[0] * noc_feed_in
molfrac_water_out=[0] * noc_water_in
amt_feed_out = [0] * noc_feed_in
```

```
amt_water_out = [0] * noc_water_in
total_water_out  = 0
total_feed_out  = 0
am_id_feed = 0
am_id_water =0
T_feed_out = [0]
T_water_out = [0]

#USER INPUT
feed_out_ammonia_comp #outlet ammonia compostion
compound #compound that get adsorbed
k_input #Equilibrium k-value
opt_temp #Operating Temperature
opt_pr #Operating Prssure

# ===================================================================
#CALCULATION

#Loop to get compound id in feed stream
for i in range(0,noc_feed_in) :
    if ( ids_feed_in [i] == compound) :
            am_id_feed = i

#Loop to get compound id in water stream
for i in range(0,noc_water_in) :
    if (ids_water_in [i] == compound) :
            am_id_water = i

#NOTE The compound ids are same in the feed and water stream

molfrac_feed_out [am_id_feed] = feed_out_ammonia_comp #Assingning the user input to the mol fraction array
 feed_inert_flow  = (1-molfrac_feed_in[am_id_feed]) * molflow_feed_in [0]  #Calculating the inert flow
molflow_feed_out [0]  = feed_inert_flow  / (1- molfrac_feed_out[am_id_feed])  # Calculating the molflow of outlet gas stream
molflow_water_out[0] = molflow_water_in[0] + molflow_feed_in [0]  - molflow_feed_out[0] #Calculating molflow of the outlet water stream


#Calculating amount of each component in outlet gas stream
for i in range(0,noc_feed_in) :
    if (i==am_id_feed):
        amt_feed_out[i] = molfrac_feed_out[i] * molflow_feed_out [0]
    else :
        amt_feed_out[i] = molfrac_feed_in [i] * molflow_feed_in [0]
```

```
    total_feed_out = total_feed_out + amt_feed_out[i] #Calculating total molflow of outlet feed

#NOTE the total_feed_out should be equal to the molflow_feed_out[0]

#Calculating the molfrac of each component in the outlet gas stream
for i in range(0,noc_feed_in):
    molfrac_feed_out[i] = amt_feed_out[i] / total_feed_out


#Calculating amount of each component in outlet water stream
for i in range(0,noc_water_in):
    if (i==am_id_water):
        amt_water_out[i] = molflow_feed_in[0] - molflow_feed_out[0]
    else :
        amt_water_out[i] = molfrac_water_in[i] * molflow_water_in[0]
    total_water_out = total_water_out + amt_water_out[i]

#Calculating the molfrac of each component in outlet ater stream
for i in range(0,noc_water_in):
    molfrac_water_out[i] = amt_water_out[i] / total_water_out

#Used in calculation of HTU and NTU
#Calculating average liquid and gas flow rate - not tested yet
V_avg = (molflow_feed_out[0] + molflow_feed_in[0]) * 0.5
L_avg = (molflow_water_out[0] + molflow_water_in[0]) * 0.5

# ====================================================================

#Calculating K-value
#Method - 1 : Wilson correlation
am_compound = feed_in.GetPhase('Mixture').Compounds[compound]
crit_temp = am_compound.ConstantProperties.Critical_Temperature
 crit_pr = am_compound.ConstantProperties.Critical_Pressure
 a_factor = am_compound.ConstantProperties.Acentric_Factor
k = (crit_pr/(opt_pr*101325)) * Math.Exp(5.37 * (1 + a_factor) * (1 - (crit_temp / opt_temp)))

# ====================================================================

#Method -2 : Raoults Law (Not - Used)
#Getting vapour pressure constants
am_compound = feed_in.GetPhase('Mixture').Compounds[compound]
A = am_compound.ConstantProperties.Vapor_Pressure_Constant_A
B = am_compound.ConstantProperties.Vapor_Pressure_Constant_B
C = am_compound.ConstantProperties.Vapor_Pressure_Constant_C
D = am_compound.ConstantProperties.Vapor_Pressure_Constant_D
```

```
E = am_compound.ConstantProperties.Vapor_Pressure_Constant_E

#Calculating vapour pressure
vp = Math.Exp (A + B / opt_temp + C * Math.Log(opt_temp) + D * Math.Pow(opt_temp , E))
K = vp /(101325 * opt_pr)

#============================================================

k_value = k #Method -1
#k_value = K #(use this for Method - 2 )

if (k_input <> 0) :
 k_value = k_input

Flowsheet.WriteMessage("k-value : " + str(k_value))

#Calculating Absorption Factor
A = L_avg/(k_value * V_avg)
Flowsheet.WriteMessage("Absorption Factor : " + str(A))

#Calculating No of ideal stages from Kremser Equation
X = (molfrac_feed_in[am_id_feed] - k_value * molfrac_water_in[am_id_feed])/(molfrac_feed_out[am_id_feed] - k_value * molfrac_water_in[am_id_feed])
Y = 1 - 1 / A

check = X*Y + 1/A

if (check < 0) :
    Flowsheet.WriteMessage("Cannot be Solved using Kremser equation")
    Flowsheet.WriteMessage("Error : Negative Logarithmic")

else :
     N = (Math.Log ((X * Y) + 1/A)) / (Math.Log(A))
     N = int(N) + 1 #Converting no of stages to an integer
     Flowsheet.WriteMessage("The Number of ideal stages required : " + str(N)) #Displays the no of stages in the information box

T_feed_out[0] = opt_temp
T_water_out[0] = opt_temp
#============================================================

#Outlet gas streams
feed_out = oms1
feed_out.Clear()
feed_out.SetProp("temperature","Overall",None,"","",T_feed_out)
feed_out.SetProp("pressure","Overall",None,"","",P_feed_in)
```

```
feed_out.SetProp("fraction","Overall",None,"","mole",molfrac_feed_out)
feed_out.SetProp("totalFlow","Overall",None,"","mole",molflow_feed_out)

#Outlet Water stream
water_out = oms2
water_out.Clear()
water_out.SetProp("temperature","Overall",None,"","",T_water_out)
water_out.SetProp("pressure","Overall",None,"","",P_water_in)
water_out.SetProp("fraction","Overall",None,"","mole",molfrac_water_out)
water_out.SetProp("totalFlow","Overall",None,"","mole",molflow_water_out)

Flowsheet.WriteMessage("Enter k_input as Zero to Calculate its value using the Wilson
    Correlation")

#End of Program
#
    ====================================================
```

6.7 Input Stream Specifications

Input Specifications					
Object	Water_out	Water_in	Feed_out	Feed_in	
Temperature	293.15	293.15	294.15	294.15	K
Pressure	101325	101325	101325	101325	Pa
Molar Flow	25.41759	25	11.334583	11.752172	mol/s
Molar Fraction (Mixture) / Water	0.98357083	1	0	0	
Molar Fraction (Mixture) / Air	0	0	0.985	0.95	
Molar Fraction (Mixture) / Ammonia	0.016429169	0	0.015	0.05	

6.8 User Input

Feed out Solute Molfraction	0.015
k input	1.67
opt Pressure	1 atm
opt Temperature	320 K

6.9 Results

Input Output					
Object	Water_out	Water_in	Feed_out	Feed_in	
Mass Flow	0.45748697	0.450375	0.32622136	0.33333333	kg/s
Molar Flow (Mixture) / Ammonia	0.41758988	0	0.17001874	0.58760862	mol/s

6.10 Additional Notes

The Model only gives an approximate number of stages based on Kremser equation. The model also checks for negative log values and gives the appropriate error.

6.11 Nomenclature

6.11.1 Latin Letters

a	Acentric factor
k	Equilibrium constant
x	Molar fraction of the solute
A	Absorption Factor
M	Mass Flow rate

6.11.2 Subscripts

op	Operating condition
c	Critical Value