Problem_5b

September 30, 2021

[1]:
```
using Plots
using JuMP
using Ipopt
```

[2]:
```
#geometric mean of relative volitilies, useful for shortcut column
function geoMean(kLightKeyDist, kLightKeyBot, kHeavyKeyDist, kHeavyKeyBot)
    return(sqrt((kLightKeyDist*kLightKeyBot)/(kHeavyKeyDist*kHeavyKeyBot)))
end
```

[2]:
```
geoMean (generic function with 1 method)
```

[5]:
```
#Fenske equation used in step 3 of column sizing to get the
function fenske(xLightKeyDist, kLightKeyDist, xHeavyKeyDist, kHeavyKeyDist,␣
 ↪xLightKeyBot, kLightKeyBot,
        xHeavyKeyBot, kHeavyKeyBot)
     _avg = geoMean(kLightKeyDist, kLightKeyBot, kHeavyKeyDist, kHeavyKeyBot)
    n = log10(xLightKeyDist*xHeavyKeyBot/(xLightKeyBot*xHeavyKeyDist)) /␣
 ↪log10( _avg)
    return(n) #return number of trays
end
```

[5]:
```
fenske (generic function with 1 method)
```

[6]:
```
function step4obj( , nmin, F, D, B, xHKd, xHKb, xfi, xbi)
    return((F*xfi-B*xbi-D* ( ^nmin* xHKd/xHKb*xbi))^2)
end
```

[6]:
```
step4obj (generic function with 1 method)
```

[8]:
```
function step4( , nmin, F, D, B, xHKd, xHKb, xfi)
    model = Model(with_optimizer(Ipopt.Optimizer))
    set_silent(model)
    register(model, :step4obj, 9, step4obj; autodiff = true)
    @variable(model, xbi)
    @NLconstraint(model, xbi >= 1e-6)
    @NLobjective(model, Min, step4obj( , nmin, F, D, B, xHKd, xHKb, xfi, xbi))
    optimize!(model)
```

```
    xdi = (F*xfi-B*value(xbi))/D
    return(value(xbi), xdi)
end
```

[8]: step4 (generic function with 1 method)

[52]:
```
#Underwood equation for finding theta
function underwood(kDist, kBot, kHeavyKeyDist, kHeavyKeyBot, xFeed, qual, xDist)
    l = length(kDist)
      = zeros(l)
    for i = 1:l
        [i] = geoMean(kDist[i], kBot[i], kHeavyKeyDist, kHeavyKeyBot)
    end

    model = Model(with_optimizer(Ipopt.Optimizer))
    set_silent(model)
    @variable(model, θ)
    @NLconstraint(model, 1<= θ <= maximum( ))
    @NLobjective(model, Min, (sum(( [i]*xFeed[i])/( [i]-θ) for i in 1:l) + qual␣
↪-1)^2)
    optimize!(model)

    rmin = sum((( .* xDist) ./ ( .- value(θ)))) - 1
    return(rmin)
end
```

[52]: underwood (generic function with 1 method)

[69]:
```
#Gilliand correlation for FUG method
function gilliand(ract, rmin, nmin)
    m = Model(with_optimizer(Ipopt.Optimizer))
    set_silent(m)
    @variable(m, Nact)
    @NLconstraint(m, nmin <= Nact)
    @NLobjective(m, Min, (0.75*(1-(ract-rmin/(ract+1))^0.566) - (Nact-nmin)/
↪(Nact+1))^2)
    optimize!(m)
    return(value(Nact))
end
```

[69]: gilliand (generic function with 1 method)

[103]:
```
#Kirkbride function for finding optimal feed stage, uses ipopt solver
function kirkbride(B,D,xHKFeed,xLKFeed,xLKBot,xHKDist, nTot)
    t = ((B/D) * (xHKFeed/xLKFeed) * (xLKBot/xHKDist)^2)^0.206
    model = Model(with_optimizer(Ipopt.Optimizer))
    set_silent(model)
```

```
    @variable(model, m) # where m is number of stages above feed
    @NLconstraint(model, 0.2 <= m <= nTot-0.2)
    @NLobjective(model, Min, (m/(nTot-m) - t)^2)
    optimize!(model)
    return(value(m), nTot-value(m)) #return number stages above, number of␣
 ↪stages below
end
```

[103]: kirkbride (generic function with 1 method)

[112]:
```
function part10(F_ha, F_f, F_st, Cs,  _l,  _g, f, A_d, A_t, G)
    C = F_ha * F_f * F_st * Cs
    Uf = C*sqrt(( _l- _g)/ _g)
    Dt = sqrt(4*G/ (f*Uf*pi*(1-A_d/A_t)* _g))
    return(Dt)
end
```

[112]: part10 (generic function with 1 method)

Step 1 We begin by defining the light and heavy keys. We elected to use the PRSV eos because it is designed for polar chemical systems particularly with dilute solutions. The heavy key is given as 1,2 dichloroethane. The light key was found using HYSYS K values. The next greatest K value is for water this is the light key. We assumed a feed basis of 100 kmol/hr.

	Mixed	Light	Heavy
13-Butadiene	2.217	2.217	<empty>
12-CIC2	0.2638	0.2638	<empty>
112-CIC2	0.1319	0.1319	<empty>
H2O	1.063	1.063	<empty>
VinylCl	2.352	2.352	<empty>

[92]:
```
### define molar flows in feed, bottoms, and distillate by assuming splits␣
 ↪given in problem statement
### and letting all HNK go to distillate and all LNK to the bottoms
feedBasis = 1
xFeed = [0.1493,0.5083,0.0007,0.0096,0.3321]
feedMolFlow = xFeed * feedBasis
distMolFlow = zeros(5)
botMolFlow = zeros(5)
distMolFlow[1] = feedMolFlow[1]; distMolFlow[2] = feedMolFlow[2]*0.005;␣
 ↪distMolFlow[4] = feedMolFlow[4]*0.9;
distMolFlow[5] = feedMolFlow[5]
botMolFlow[2] = feedMolFlow[2]*0.995; botMolFlow[3] = feedMolFlow[3];␣
 ↪botMolFlow[4] = feedMolFlow[4]*0.1;
xDist = zeros(5)
xBot = zeros(5)
```

```
for i = 1:5
    xDist[i] = distMolFlow[i]/sum(distMolFlow)
    xBot[i] = botMolFlow[i]/sum(botMolFlow)
end
print(xDist[2], " x val of HK in distilate " , xBot[4], " x val of LK in␣
 ↪bottoms")
```

```
0.005159552277135865 x val of HK in distilate 0.0018919294428563402 x val of LK
in bottoms
```

Step 2 After using the above ratios in the performance data above and a pressure of 1100kPa in the bottoms(given) and 1090kPa in the distilate using the suggestion that for preliminary design we estimate 0.1 psi drop per tray with an estimated 10-15 trays, with a total condenser and partial reboiler. We went with a total condenser because we do not have any extremely volitile components and a partial reboiler because we are going to need to run this at well above 100 degrees celcius so we would like to keep the energy costs low.

Step 3 Now we can use the K values from the full column to continue with the Fenske equation specified above

Profile Table: K Values Profile

	13-Butadiene	12-CIC2	112-CIC2	H2O	VinylCl
Condenser	0.8793	0.1005	4.605e-002	1.360	1.053
1_Main Tower	0.9176	0.1071	4.944e-002	1.489	1.108
2_Main Tower	1.086	0.1214	5.629e-002	1.316	1.280
3_Main Tower	1.546	0.1696	8.078e-002	1.108	1.727
4_Main Tower	2.055	0.2396	0.1184	1.135	2.218
5_Main Tower	2.298	0.2793	0.1406	1.189	2.451
6_Main Tower	2.375	0.2933	0.1486	1.211	2.526
7_Main Tower	2.397	0.2975	0.1510	1.218	2.547
8_Main Tower	2.402	0.2988	0.1518	1.220	2.552
9_Main Tower	2.860	0.3819	0.1999	1.372	2.964
10_Main Tower	3.543	0.5329	0.2911	1.649	3.578
11_Main Tower	4.193	0.7143	0.4057	1.977	4.166
12_Main Tower	4.594	0.8545	0.4976	2.223	4.543
13_Main Tower	4.777	0.9347	0.5514	2.361	4.729
Reboiler	4.824	0.9746	0.5795	2.426	4.789

[105]:
```
xLKd = xDist[4]
kLKd = 1.36
xHKd = xDist[2]
kHKd = 0.1005
xLKb = xBot[4]
kLKb = 2.426
xHKb = xBot[2]
kHKb = 0.9746
```

```
nmin = fenske(xLKd, kLKd, xHKd, kHKd, xLKb, kLKb, xHKb, kHKb)
print("Theoretical min number of stages is ", nmin)
```

```
Theoretical min number of stages is 4.25510676647357
```

Step 4 We can now find the compositions of the non-key components using the step 4 function above:

[106]:
```
names = ["13-Butadine", "Trichloroethane", "Vinyl chloride"]
kNKd = [0.8793, 4.605e-2, 1.053] #this is 13-Butadiene, 112-ClC2, VinylCl
kNKb = [4.824, 0.5795, 4.789]
xfi = [0.1493, 0.0007, 0.3321]
xHKd = 0.0052
xHKb = 0.9889
alp = zeros(3)
for i = 1:3
     alp[i] = geoMean(kNKd[i], kNKb[i], xHKd, xHKb)
end

F = 4234
B = 2165
D = F-B

for i = 1:3
    xbi, xdi = step4(alp[i], nmin, F, D, B, xHKd, xHKb, xfi[i])
    print(names[i], " bottoms frac ", xbi, " distilate frac ", xdi, '\n')
end
```

```
13-Butadine bottoms frac 0.017978267847049492 distilate frac 0.2867149589710671
Trichloroethane bottoms frac 0.0013685282790636517 distilate frac
4.5252577438097823e-7
Vinyl chloride bottoms frac 0.028211706191844675 distilate frac
0.6500884756378232
```

Step 5 Now we can use the Underwood equation defined above to find the minimum reflux ratio.

[107]:
```
kDist = [0.8793, 0.1005, 4.605e-2, 1.36, 1.053]
kBot = [4.824, 0.9746, 0.5795, 2.426, 4.789]
kHKd = 0.1005
kHKb = 0.9746
xFeed = [0.1493, 0.5083, 0.0007, 0.0096, 0.3321]
qual = 0.7338 #looked at the feed stream in HYSYS
xDist = [0.28686, xHKd, 4.5122e-7, xLKd , 0.65033]

rmin = underwood(kDist, kBot, kHKd, kHKb, xFeed, qual, xDist)
print("The min reflux ratio is ", rmin)
```

```
The min reflux ratio is 0.3764543799630531
```

Part 6 Now we we use the heuristic that the actual reflux ratio should be between 1.1 and 1.4 times the min reflux ratio and we find:

[108]:
```
ract = 1.3*rmin
print("Our actual reflux ratio is ", ract)
```

```
Our actual reflux ratio is 0.48939069395196905
```

Part 7 We can now use the Gilliand correlation to find the actual number of stages needed

[109]:
```
nact = gilliand(rmin, ract, nmin)
print("Our actual number of stages will be ", nact)
```

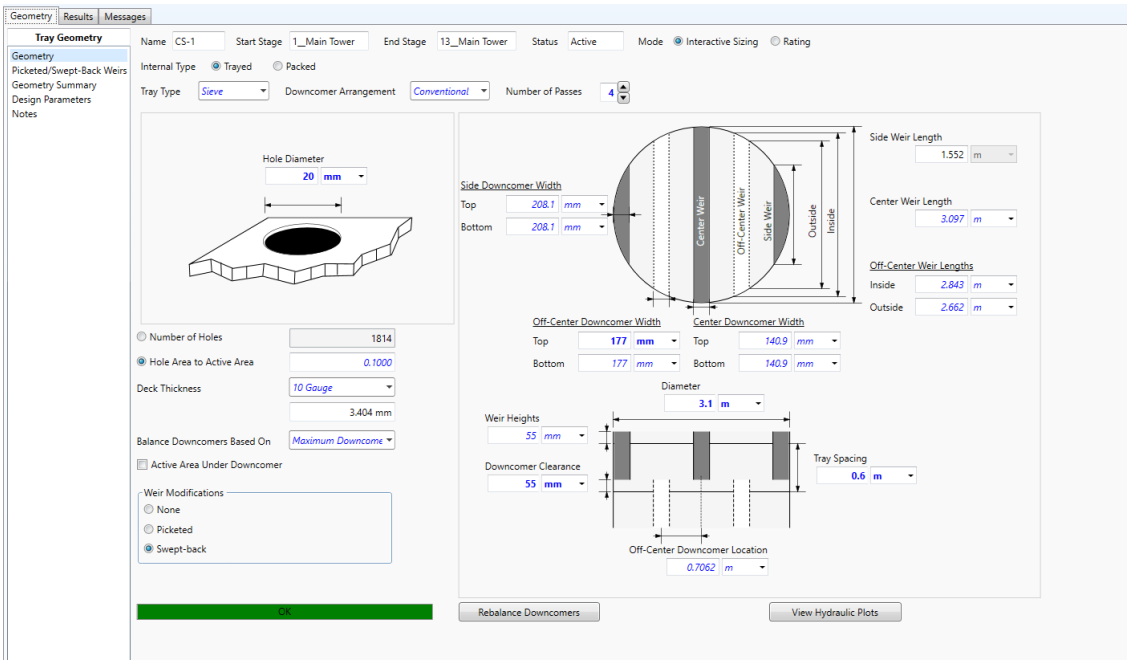
```
Our actual number of stages will be 14.73292374395708
```

Part 8 We are now ready for the Kirkbride equation which will give us the optimal feed stage

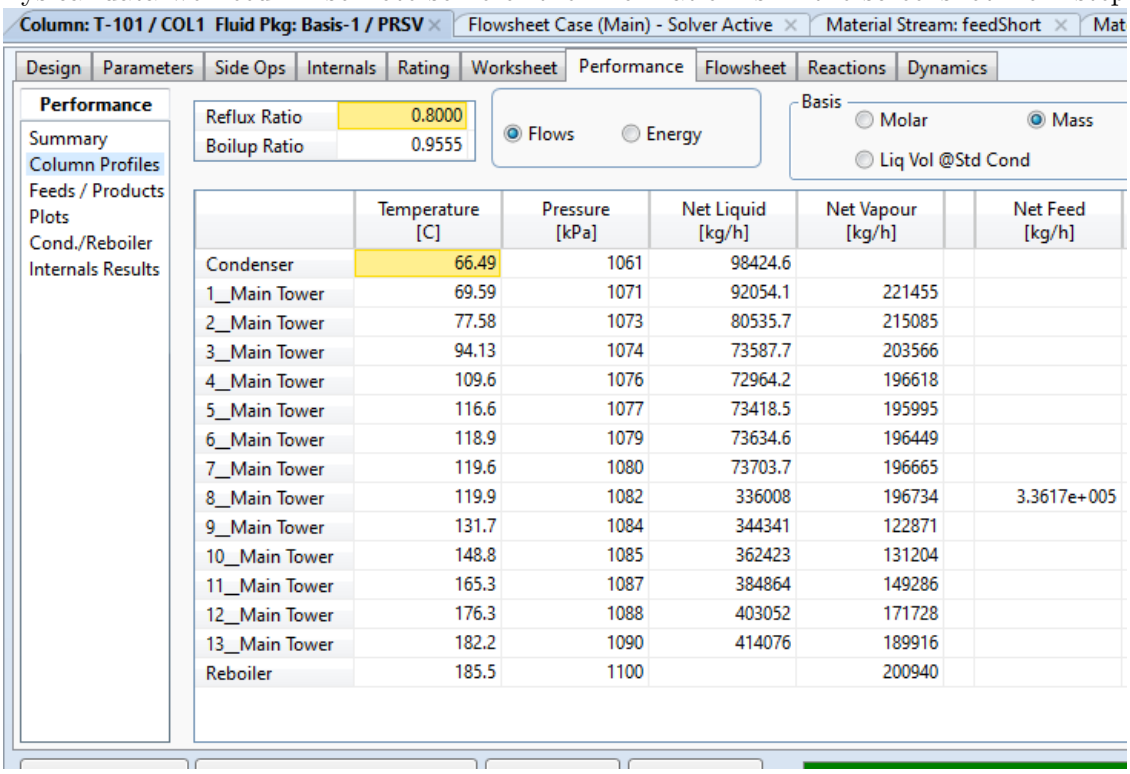
[110]:
```
above, below = kirkbride(B,D,xHKFeed,xFeed[4],xLKb,xHKd, nact)
print("our optimal feed stage is ", above, " from the top and ", below, " from␣
 ↪the bottom")
```

```
our optimal feed stage is 8.845434704354513 from the top and 5.887489039602567
from the bottom
```

Part 9 Our column internals will be informed by the fact that we have a large flow rate. We elected to go with 4 pass sieve trays. These trays are not particulary efficent but they are better for large columns as long as the turndown ratio is not large. Our customer has not given us any information regarding this. When we went to simulate the full column in HYSYS we used the same 4 pass sieve trays with significant internal modification to get the simulation to converge with no warnings. In our simulation we used the following configuration:



Step 10 We will now use the data from our HYSYS simulation to plug into the equation for step 10 to get column diameter. To do this we will need to size the column at both the top and the bottom. The following 3 screenshots are from our HYSYS simulation and will help us to find the physical data we need. Also note some of the information is in the screenshot from step 9.



	Temperature [C]	Pressure [kPa]	Net Liquid [kg/h]	Net Vapour [kg/h]	Net Feed [kg/h]
Condenser	66.49	1061	98424.6		
1_Main Tower	69.59	1071	92054.1	221455	
2_Main Tower	77.58	1073	80535.7	215085	
3_Main Tower	94.13	1074	73587.7	203566	
4_Main Tower	109.6	1076	72964.2	196618	
5_Main Tower	116.6	1077	73418.5	195995	
6_Main Tower	118.9	1079	73634.6	196449	
7_Main Tower	119.6	1080	73703.7	196665	
8_Main Tower	119.9	1082	336008	196734	3.3617e+005
9_Main Tower	131.7	1084	344341	122871	
10_Main Tower	148.8	1085	362423	131204	
11_Main Tower	165.3	1087	384864	149286	
12_Main Tower	176.3	1088	403052	171728	
13_Main Tower	182.2	1090	414076	189916	
Reboiler	185.5	1100		200940	

Tray Number	Liquid Molecular Weight	Vapor Molecular Weight	Liquid Mass Density [kg/m3]	Vapor Mass Density [kg/m3]	Liquid Viscosity [cP]	Vapor Viscosity [cP]	Surface Tension [dyne/cm]
1_Main To...	61.04	60.14	732.5	26.45	0.1175	1.097e-002	9.731
2_Main To...	66.68	62.13	791.7	25.67	0.1406	1.144e-002	11.11
3_Main To...	76.39	64.85	888.7	25.48	0.1799	1.188e-002	13.21
4_Main To...	82.67	66.41	938.5	25.58	0.1976	1.209e-002	13.87
5_Main To...	84.89	66.97	952.0	25.67	0.2010	1.215e-002	13.87
6_Main To...	85.54	67.14	955.3	25.73	0.2016	1.217e-002	13.83
7_Main To...	85.73	67.18	956.1	25.77	0.2017	1.218e-002	13.81
8_Main To...	85.79	68.55	956.3	25.89	0.2017	1.233e-002	13.79
9_Main To...	88.54	76.11	968.0	27.27	0.2021	1.290e-002	13.64
10_Main T...	91.90	83.95	975.5	29.22	0.1930	1.328e-002	12.98
11_Main T...	94.78	90.60	975.3	31.10	0.1829	1.350e-002	12.05
12_Main T...	96.68	94.79	972.3	32.39	0.1758	1.360e-002	11.19
13_Main T...	97.79	97.12	969.7	33.07	0.1717	1.365e-002	10.62

Cs (Net Area) [m/s]	Cs (Bubbling Area) [m/s]
6.602e-002	7.673e-002
6.088e-002	7.076e-002
5.561e-002	6.463e-002
5.379e-002	6.252e-002
5.343e-002	6.210e-002
5.334e-002	6.199e-002
5.329e-002	6.194e-002
5.327e-002	6.191e-002
3.436e-002	3.993e-002
3.765e-002	4.376e-002
4.203e-002	4.885e-002
4.566e-002	5.306e-002
4.789e-002	5.566e-002

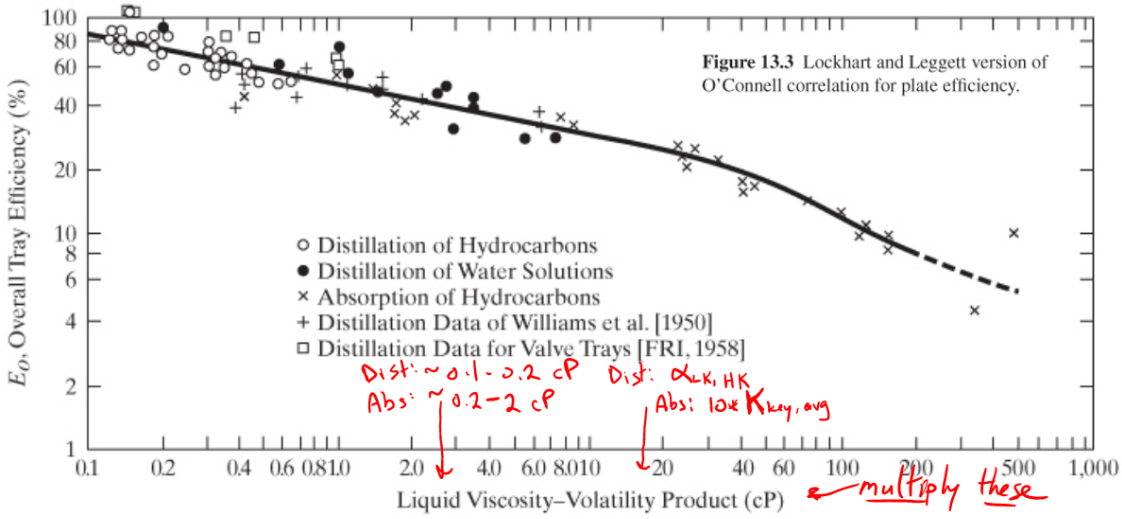
[125]:
```
F_ha = 0.1 #from part 9 screenshot
F_f = 1
F_st_top = (9.731/20)^0.2
```

```
F_st_bot = (10.62/20)^0.2
Cs_top = 7.673e-1
Cs_bot = 5.566e-1
 _l_top = 732.5
 _l_bot = 969.7
 _g_top = 26.45
 _g_bot = 33.07
f = 0.8
A_d = 0.4366
A_t = (3.1/2)^2*pi
G_top = 221455/3600 #convert hours to minutes
G_bot = 189916/3600

Dtop = part10(F_ha, F_f, F_st_top, Cs_top,  _l_top,  _g_top, f, A_d, A_t, G_top)
Dbot = part10(F_ha, F_f, F_st_bot, Cs_bot,  _l_bot,  _g_bot, f, A_d, A_t, G_bot)

print("The diameter at the top of the column is ", Dtop, " m and at the bottom␣
 ↪", Dbot, " m")
```

```
The diameter at the top of the column is 3.383217308862213 m and at the bottom
3.2132679229726624 m
```

We would elect to go with the bigger diameter which in this case is at the top of the column and is 3.38 meters.

Step 11 We can now calculate the height of the column. We have a tray spacing that is on the edge between using 24 and 36 inches, we went with 24 inches in order to try and keep the expense down. We then used the following graph to find the tray efficiency.



[129]:
```
lvvp = 10*geoMean(kLKd, kLKb, kHKd, kHKb)*1.365e-2
```

[129]:
```
0.7922299731371087
```

We can see the overall tray efficiency is about 55% thus we can find the total number of trays, multiply by the tray spacing, and add 14 ft total for bottoms storage and disengagement height at the top.

[130]:
```
height = nact/0.55*0.61 + 14*0.305
print("Our total column height is ", height, " m")
```

```
Our total column height is 20.610151788752397 m
```

We elected to use stainless steel for our column because it is inexpensive, versitile, and none of our componets are acids so we expect it to wear nicely. There are also a wide variety of steel grades that might suit our needs.

	Hand calc	Shortcut HYSYS	Full HYSYS
N_{min}	4.25	4.5	
N	14.7	12.3	13
N_{feed}	8.8	7.4	8
R_{min}	0.376	0.401	0.65
R	0.489	0.51	0.8
Diameter	3.38		3.1
Height	20.61		7.8
Internals	Simple Sieve	Simple Sieve	4 pass sieve, with modified downcomers

We can see from our comparision table the minimum, actual, and feed trays are all quite similar across all columns. The reflux ratio for the hand calcs and the shortcut column were close but the full HYSYS column took a much higher ratio to converge. We ultimately went with 0.8 because it gave us some wiggle room with other parameters. Our actual reflux ratio again was close for the hand calc and shortcut column, but significantly higher for the full column. We think the reflux ratio is higher in the full column becuase of significant non-ideal liquid interaction between polar and non-polar molecules. The diameter was similar between the hand calculations and the full HYSYS column. We also note we had to change the diameter of the HYSYS column because it was initially too small. The biggest difference was between the hand calcs and the full column in HYSYS for height. We are not entirely sure why there is such a large discrepancy here, but one reason is because HYSYS may not be adding the additional space for bottoms storage and disengagement height at the top. The other is HYSYS may be trying to pack the column down as far as possible, which may be cost effective for building it, but makes maintence much more difficult.

In our full HYSYS column we specified the reflux ratio and the recovery of 12-ClC2 in the bottoms. We had an idea about what the reflux ratio might be and the customer gave us the spec for the bottoms recovery. The reflux ratio was significantly higher than we expected, but once we had it converged the rest of the column was easy enough to design.

This column seems feasible from a building standpoint. a column of 20 meters is tall, but not uncommon in the hydrocarbon industry. If the height is closer the full column in HYSYS then it is not tall than a 2 story building, quite feasible indeed. The next steps for us would be an economics study. We have not looked at the energy consumption or the cost of running the column vs. profit. At this point we can say the column is buildable, but not if it make financial sense to build it.

[ ]: