

**Design and Optimization
of Post-Combustion CO₂ Capture**

Stuart Higgins

**Dissertation submitted to the faculty of
Virginia Polytechnic Institute and State University
in partial fulfillment of the requirements for the
degree of**

**Doctor of Philosophy
in
Chemical Engineering**

**Yih-An Liu, Chair
Luke E. Achenie
Donald G. Baird
Preston L. Durrill**

**April 21, 2016
Blacksburg, VA**

Keywords: Carbon capture, CCS, global warming, climate change, process engineering, process optimization, process modeling, Aspen Plus, C#, automation, COM, acid gas, CO₂, carbon dioxide

**©2016, Stuart Higgins
All rights reserved**

Design and Optimization of Post-Combustion CO₂-Capture

Stuart Higgins

Abstract (academic)

This dissertation describes the design and optimization of a CO₂-capture unit using aqueous amines to remove of carbon dioxide from the flue gas of a coal-fired power plant. In particular we construct a monolithic model of a carbon capture unit and conduct a rigorous optimization to find the lowest solvent regeneration energy yet reported.

Carbon capture is primarily motivated by environmental concerns. The goal of our work is to help make carbon capture and storage (CCS) a more efficient for the sort of universal deployment called for by the Intergovernmental Panel on Climate Change (IPCC) to stabilize anthropomorphic contributions to climate change, though there are commercial applications such as enhanced oil recovery (EOR).

We employ the latest simulation tools from Aspen Tech to rigorously model, design, and optimize acid gas systems. We extend this modeling approach to leverage Aspen Plus in the .NET framework through Microsoft's Component Object Model (COM).

Our work successfully increases the efficiency of acid gas capture. We report a result optimally implementing multiple energy-saving schemes to reach a thermal regeneration energy of $E_t^{\text{regen}} = 1.67 \frac{\text{GJ}_t}{\text{tonne CO}_2}$. By contrast, the IPCC had reported that leading technologies range from 2.7 to $3.3 \frac{\text{GJ}_t}{\text{tonne CO}_2}$ in 2005.² Our work has received significant endorsement for industrial implementation by the senior management from the world's second largest chemical corporation, Sinopec, as being the most efficient technology known today.

Design and Optimization of Post-Combustion CO₂-Capture

Stuart Higgins

General Audience Abstract

This research focuses on designing energy-efficient methods for capturing CO₂ from major polluters like coal-fired power plants. Our primary motivation is to reduce the greenhouse gas emissions that warm Earth and acid gas emissions that make our rain, oceans, lakes, and rivers more acidic. Major chemical companies have also found limited economic applications for this technology; for example, Enhanced Oil Recovery (EOR) sites pump captured CO₂ back underground to help drive more oil back to the surface.

The basic technology in this dissertation has been known since 1930. We focus on improving the technology because the original 1930's version was very wasteful. This research area has attracted major interest, so our research follows earlier research on the same subject. Our major contributions are (1) new improvements not found in earlier literature and (2) an efficient design incorporating multiple improvement methods.

Our research efforts have been very successful, yielding the most efficient technology yet discovered. This technology is to be implemented by a major oil company at one of their oil fields by 2017.

To my mother and father.

Acknowledgements

I would like to thank my Ph.D. advisor, Professor Y.A. Liu of Virginia Tech, for his generous support throughout my graduate career. Above all else, the opportunity to make a real world impact has made this research more than just an academic pursuit. In the same vein, I would like to thank my committee members, Professors Luke Achenie, Donald Baird, Peter Rim, and Preston Durrill for their guidance and instruction throughout my time here at Virginia Tech.

This research has been made possible by the generous support of our industrial partners, including Aspen Technology, China Petroleum and Chemical Corporation (Sinopec), and MATRIC. Beyond simply providing the necessary financial support necessary for this research, our partners have gone above and beyond in furnishing invaluable technical data, powerful software tools, and opportunities for real world application.

Special thanks to Mr. Cao Xianghong, former Senior Vice President and the Chief Technology Officer of Sinopec and an elected foreign member of the U.S. National Academy of Engineering, for his advocacy in implementing our design recommendations in the Shengli Oil Field Power Plant. This demonstration-scale project is one of the world's largest, intended to capture one million tonnes of CO₂ per year beginning in 2018.

Table of Contents

1	Introduction	1
1.1	Process System Engineering	1
1.2	Motivation and Significance.....	1
1.3	Programming Conventions and Presentation Formats for the Chapter.....	2
1.4	Literature overview.....	3
1.4.1	Base-case design	3
1.4.2	Design considerations	7
1.5	General approach: Expand the model space and optimize	15
1.5.1	Heat integration.....	18
1.5.2	Utility integration.....	18
1.6	Tips and Tricks.....	19
1.6.1	Interpolating non-integer values	19
1.6.2	Reconciling estimates	22
1.6.3	Best practices in defensive programming.....	26
1.6.4	Detecting missing tear specifications	33
2	Major schemes.....	37
2.1	Absorber intercooling	37
2.1.1	Modeling approach for absorber intercooling.....	37
2.1.2	Flowsheet construction.....	38
2.1.3	Sensitivity analyses	47
2.1.4	Remarks.....	50
2.2	Stripper interheating.....	50
2.2.1	Modeling approach for stripper interheating.....	51

2.2.2	Flowsheet construction.....	52
2.2.3	Initial run	60
2.2.4	Sensitivity analyses	61
2.2.5	Remarks.....	63
2.3	Stripper condensate rerouting.....	64
2.3.1	Modeling approach.....	64
2.3.2	Flowsheet construction.....	64
2.3.3	Sensitivity analyses	67
2.3.4	Remarks.....	68
2.4	Distributed cross heat exchanger	69
2.4.1	Modeling approach.....	70
2.4.2	Flowsheet construction.....	72
2.5	Heat pump integration.....	86
2.5.1	History of waste heat recovery through heat pump integration.....	87
2.5.2	Understanding a lithium bromide heat pump	87
2.5.3	Aspen Plus flowsheet	90
2.6	Combined carbon capture unit and heat pump simulation	131
2.6.1	Flowsheet layout.....	131
2.6.2	Energy-saving scheme implementations	133
2.6.3	Optimization approach	133
2.6.4	Working with the flowsheet	133
2.6.5	Design conditions.....	138
2.6.6	Optimization process	138
2.6.7	Optimization results.....	148
2.6.8	Comparison	148
2.6.9	Improved methodology	149

3	Using Aspen Plus through its COM interface using C#	150
3.1	Download and install Visual Studio 2015.....	150
3.2	Conceptual preparation	151
3.2.1	Terminology	151
3.2.2	Everything is a program	151
3.3	Working with Aspen Plus using C#.....	153
3.3.1	Setting up the project	154
3.3.2	Opening Aspen Plus through COM	162
3.3.3	Interacting with a basic Aspen Plus simulation.....	164
3.3.4	Run Aspen Plus in a separate thread	169
3.3.5	Adding a custom debug message console	170
3.3.6	Custom design specifications.....	179
3.4	Creating an acid-gas-capture simulation in C#	184
3.4.1	Build the absorber.....	184
3.4.2	Build the stripper	210
3.4.3	Build the distributed cross heat exchanger	224
3.4.4	Build the acid-gas capture unit	233
3.5	Wrapping Aspen Plus	242
3.5.1	Automatic crash recovery	243
3.6	Developing our simulation tools.....	245
3.6.1	Framework structure	245
3.6.2	Using the new framework.....	249
3.7	Summary	251
4	Conclusions	252
4.1	Accomplishments.....	252
4.1.1	Industrial implementation	252

4.1.2	Patent.....	253
4.1.3	Publication	253
4.1.4	Tools.....	253
4.2	Recommendations for future research.....	253
5	References	254
6	Appendices.....	257
Appendix A.	Code	257
Appendix A.1.	CO ₂ -capture unit simulation without a framework	257
Appendix A.2.	Aspen Plus wrapper	285
Appendix A.3.	CO ₂ -capture simulation using the framework	294

List of Figures

Figure 1.1. A series of absorption towers working in concert. This design is from US Patent 1,897,725 (1927).	4
Figure 1.2. Basic acid-gas-capture unit as reported in US Patent 1,783,901 (1930). The tower on the left is the absorption tower (absorber) and the tower on the left is the regeneration tower (stripper). Also shown are the stripper’s reboiler (18), the stripper’s condenser (24), the central cross heat exchanger (22), the lean stream cooler (23), and solution pumps (17 and 20). Modern implementations tend to build on this basic design.	4
Figure 1.3. Conceptual acid-gas-capture unit.	5
Figure 1.4. Thermal loop in a conceptual acid-gas-capture unit.	7
Figure 1.5. Scale showing the range of capture rates from zero to 100%.	8
Figure 1.6. Conceptual layout for a coal-fired power plant without a carbon-capture unit.	11
Figure 1.7. Coal-fired power plant with a carbon-capture unit.	12
Figure 1.8. The production-loss energy penalty plotted against the size-up energy penalty. For very low values, it is not excessively misleading to confuse the two metrics, however the deviation grows with both values. At the limit of a 100% production-loss energy penalty, the size-up energy penalty is infinite.	13
Figure 1.9. Example generalization of a basic process. We consider an arbitrary process design such as in (a). We can redraw this arbitrary process as in (b) with no actual change to the system. We may acknowledge that there is, conceptually, zero heat-exchanging surface area $ACHEX = 0$, without changing the process. Finally, we can draw a HeatX block with zero heat-exchanging surface area without changing the process. Overall, we have made no change to (a) except that the flowsheet can now consider the possibility of adding a cross heat exchanger as shown in (d). Aspen Plus should report the same results for both (a) and (d) in the $ACHEX = 0$ case within the limits of numerical error tolerance.	16
Figure 1.10. Specify zero heat-exchanging surface area in a hypothetical HeatX block.	16
Figure 1.11. Dock panel for Aspen Plus’s Activated Energy Analysis tool.	18
Figure 1.12. Explicit representation of cooling water utility on a flowsheet. We often omit utility streams for brevity.	19

Figure 1.13. We can split a stage before feeding it into a column to implement non-integer stage specifications.	20
Figure 1.14. Flowsheet configuration for non-integer draw specifications.....	21
Figure 1.15. Configuration for non-integer heat stream specification.	22
Figure 1.16. Window for selecting stream values to reconcile. This particular screenshot was for a stream that did not already have input values; for streams that do, “Use input specifications to select variables to reconcile” has the already-provided values updated based on the results.....	24
Figure 1.17. Aspen Plus form for estimates. The “Temperature” form is shown, along with the tabs for the “Flows”, “Liquid Composition”, and “Vapor Composition” forms.	25
Figure 1.18. Holdup specification form for a RadFrac column. The liquid holdup is the amount of liquid considered to be present on a stage for the purpose of calculating reactions, found under Aspen Plus Simulation Blocks → [RadFrac] → Specifications → Reactions “Holdups”. In rate-based mode, the value given here is only an initial estimate.	25
Figure 1.19. Check box to toggle affected block logic. We strongly recommend disabling affected block logic unless you both need the performance boost it might provide and understand the potential errors it may cause.	27
Figure 1.20. Aspen Plus currently provides five options for the behavior of the auto-sequencing algorithm.....	30
Figure 1.21. Bypass checkbox to disable a HeatX block.	31
Figure 1.22. Specifications for C-D-CHEX	32
Figure 1.23. Sequence specification that instructs Aspen Plus to always run the controlling Calculator before its corresponding HeatX	33
Figure 1.24. Regeneration energy results for a sensitivity analysis on stripper interheating area, <i>ASTR</i> – <i>IH/m</i> . The jagged response curve indicates a missing tear specification.	34
Figure 1.25. Comparison between a correctly torn sensitivity analysis and the poorly torn sensitivity analysis. We can see that the correctly torn curve is much smoother and more consistent.....	35
Figure 1.26. Observations on the improperly torn sensitivity analysis results curve.	36
Figure 1.27. Sensitivity analysis results for regeneration energy as a function of the location of the stripper’s interheater. We compare a correctly torn analysis with two poorly torn analyses with differing step sizes. The incorrectly torn analysis with larger step sizes is the most jagged.	36
Figure 2.1. Prior flowsheet for the baseline acid-gas-capture unit.	38
Figure 2.2. Parameters for absorber intercooling to be defined in C-GLOBAL	39

Figure 2.3. Flowsheet modification to implement absorber intercooling. Insert heat streams ABS-IC-1 and ABS-IC-2 as feeds into ABSORBER	41
Figure 2.4. Variable definitions for C-ABS-IC	42
Figure 2.5. Modification for SQ-ABS-3 to run the absorber intercooling offset Heater, ABS-IC-C , between the lean-stream feed and absorber complex.	45
Figure 2.6. Definition for SQ-PRE	45
Figure 2.7. Insert SQ-PRE into SQ-DATA	46
Figure 2.8. New cases for SA-DATA	46
Figure 2.9. Regeneration energy for various absorber intercooling duties on either Stage 20 or Stage 25.	48
Figure 2.10. Results for a sensitivity analysis varying the absorber intercooler’s location over the column. We find poor results for all but the lowest position, at which regeneration energy is just slightly better than in the base case.	48
Figure 2.11. Response curves including Stage 29. The response for Stage 29 is modest but it does outperform the base case in terms of regeneration energy.	49
Figure 2.12. Temperature profile for the bulk liquid and bulk vapor phases.	50
Figure 2.13. Flowsheet modifications for stripper interheating.	52
Figure 2.14. Variable definitions for C-STRIH	54
Figure 2.15. Definition for SQ-STRIH	57
Figure 2.16. Desired order of evaluation. We want the stripper interheating units, SQ-STRIH , to be evaluated after STR-REB and before CHEX	57
Figure 2.17. Modification to SQ-STR-2 that places the interheater, SQ-STRIH , after STR-REB and before CHEX . Note that while SQ-STRIH is beneath CHEX on the list, the Convergence block CV-STR-2 specifies that the list loops, causing SQ-STRIH to also be above CHEX	58
Figure 2.18. Further definitions for C-STRIHR	59
Figure 2.19. Calculator tear specifications for CV-STR-2	59
Figure 2.20. Regeneration energy for various interheater draw stages.	62
Figure 2.21. Regeneration energy from various interheating surface areas on Stage #14.	63
Figure 2.22. Flowsheet configuration for stripper condensate rerouting.	65
Figure 2.23. Input specification for STR-CSPL	65
Figure 2.24. Definitions for C-XSTRCN	66

Figure 2.25. Insert **STR-CSPL** after **CHEX** in **SQ-STR-2**. 67

Figure 2.26. Regeneration energy response to various stripper condensate rerouting portions. 68

Figure 2.27. Conceptual model for the central cross heat exchanger and stripper. We will extend this model to describe the distributed cross heat exchanger. 70

Figure 2.28. Conceptual flowsheet for the distributed cross heat exchanger configuration. 71

Figure 2.29. Definitions for **C-S-HSTR**. 75

Figure 2.30. Current appearance of our acid-gas-capture flowsheet. The flowsheet becomes increasingly involved as we add more energy-saving schemes for the optimizer to consider. 76

Figure 2.31. Our flowsheet after cleaning and formatting. This flowsheet is easier to read, making it easier to understand and work with..... 77

Figure 2.32. The distributed cross heat exchanger part of the flowsheet..... 78

Figure 2.33. Mapping between the conceptual flowsheet and the Aspen Plus flowsheet. We show the conceptual flowsheet in the background with boxes for the Aspen Plus blocks **CHEXMAIN**, **CHEXSIDE**, **CHEX-SPL**, and **STRIPPER**. The streams are also marked in circles, using the shorthand listed in the legend. 79

Figure 2.34. New stream specifications for **STRIPPER**. 80

Figure 2.35. Variable definitions for **C-D-SHEX**. 81

Figure 2.36. Derived object **SQ-DHEX1** for a region of the distributed cross heat exchanger. We will need to include a **Convergence** block in **SQ-DHEX1**'s definition to account for the feedback from **LEAN-6**. 82

Figure 2.37. Definition for **SQ-DHEX1**. 83

Figure 2.38. Four major units in our heat pump. Also shown are a cross heat exchanger, pump, and two valves. 89

Figure 2.39. Conceptual heat pump flowsheet. Units are represented by blocks; streams are arrows; heat-exchanging surfaces are striped rectangles. 90

Figure 2.40. Copy **UNITS** from our prior simulation to save time and avoid potential inconsistency between simulations..... 91

Figure 2.41. Paste **UNITS** into our new simulation. 91

Figure 2.42. Conceptual flowsheet showing how we will integrate the heat pump into our flowsheet. The heat pump provides useful heating to the stripper's bottom stream before that stream is reboiled. 93

Figure 2.43. Property Set block PS-PROPS . Be sure to include temperature, pressure, and enthalpy flow rate for a mixture.....	94
Figure 2.44. Specifications for PT-WATER	95
Figure 2.45. Excel sheet with copy/pasted results from PT-WATER	96
Figure 2.46. Specifications for PT-HPABS	97
Figure 2.47. Maximum pressure for the low-pressure section of the heat pump, P_{low} /bar, as a function of the higher lithium bromide mass portion.	97
Figure 2.48. Minimum temperature that the heat pump can recover waste heat at as a function of the selected mass portion for the full working solution, C_{low}	98
Figure 2.49. Specifications for PT-T-GEN	99
Figure 2.50. Additions to our Excel sheet and the resulting plot for THP – steam against T_{waste}	100
Figure 2.51. Analysis blocks PT-ABS and PT-STR in the CO_2 -capture simulation. These analyses construct waste heat recovery curves.	102
Figure 2.52. Recoverable waste heat at various waste heat temperatures T_{waste}	103
Figure 2.53. Aspen Plus flowsheet for the heat pump. The left image shows the material stream names. The right image shows the blocks' types and names.	104
Figure 2.54. Additional flowsheet units for the heat pump simulation. These units are not connected to the rest of the flowsheet by visible streams, so they may be placed at any location on the flowsheet. We arbitrarily selected to put them on the left side of the simulation.	106
Figure 2.55. Variable definitions for the Calculator blocks that help to provide initial estimates for our heat pump simulation.	112
Figure 2.56. Interpreted Fortran for the Calculator blocks that help to provide initial estimates for our heat pump simulation.....	112
Figure 2.57. Input for the Design Spec , DS-ST2	113
Figure 2.58. Definition for the Sequence coordinating the initial estimation units in our heat pump simulation, SQ-T	113
Figure 2.59. The effective sequence as calculated by Aspen Plus during the initial stages of the simulation's execution. This effective sequence exemplifies how Aspen Plus combines various types of sequence specifications to arrive at a final sequence.	114
Figure 2.60. Aspen Plus auto-generated a sequence for the units that we did not include in Sequence blocks. Here we can see the auto-generated order (in green). Since we avoided flowsheet-level feedback loops, the calculated sequence is fairly straightforward. Aspen Plus did have the opportunity	

to make a selection at HP-GEN as either HP-CON or HP-CHEX was able to follow. Aspen Plus selected HP-CON , falling back to HP-CHEX before HP-COMB since HP-COMB required information from both branches.....	115
Figure 2.61. Results for C-GLOBAL . These results reflect the minimal specifications provided for our simulation, while important information like P_{low} and P_{high} are estimated from the other data. This approach to specifying our simulations helps to make our simulations more robust, reliable, and easier to use.	116
Figure 2.62. Results for C-ST1-1 showing the calculated value for P_{low}	116
Figure 2.63. Results for DS-ST2 showing the calculated value for C_{low} as MANIPULATED	117
Figure 2.64. Results for C-ST3-1 showing the calculated value for P_{high}	117
Figure 2.65. Critical run-time errors due to missing parameters for lithium bromide.....	118
Figure 2.66. Automatic parameter estimation can be enabled using this bubble. We do not use this approach here, but it is an option.	119
Figure 2.67. Specifications for missing property parameters for LiBr to avoid the critical run-time error.	120
Figure 2.68. Specifications for DS-GEN-1	121
Figure 2.69. New flowsheet elements added to calculate VGENMAX from TGENMAX . The material stream HP-15 branches from HP-DUPE to feed the new Heater block, HP-GENF1 , which outputs a material stream HP-16	122
Figure 2.70. Specifications for C-GENM-0 , C-GENM-1 , and SQ-GENM1	123
Figure 2.71. Interpreted Fortran input for C-GENM-0 and C-GENM-1 . Both Calculator blocks require only a single line to set their values. The first sets TGENMAX to the Heater , and the second read the resulting VGENMAX from the Heater	123
Figure 2.72. Sequencing for generator _{HP} using CV-GEN-1 , SQ-GEN-1 , and SQ-GEN-2 . The top-level Sequence block, SQ-GEN-2 , is the derived object for generator _{HP}	124
Figure 2.73. Specifications for the derived pump object, SQ-PUMP1 . Also requires the creation of a new Calculator block, C-PUMP-1	125
Figure 2.74. Specifications for one of the two derived valve objects, SQ-VALV11 . Also requires the creation of a new Calculator block, C-VAL1-1	125
Figure 2.75. Specifications for the second of the two derived valve objects, SQ-VALV21 . Also requires the creation of a new Calculator block, C-VAL2-1	126

Figure 2.76. Derived object for the heat pump’s cross heat exchanger, SQ-CHEX1 . Requires the creation of a new Calculator block, C-HEX-1 .	127
Figure 2.77. Derived object SQ-PREP1 which generates the working solution of aqueous lithium bromide for the primary flowsheet section of the heat pump.	128
Figure 2.78. Calculator block to set CHIGH to its minimum value.	128
Figure 2.79. Derived object SQ-HP for the heat pump and SQ-UNIT for the overall flowsheet.	129
Figure 2.80. Sequence SQ-DATA which wraps SQ-UNIT in our new Sensitivity Analysis, S-DATA .	131
Figure 2.81. Monolithic carbon-capture flowsheet including many energy-saving schemes.	132
Figure 2.82. Actual Aspen Plus flowsheet. This flowsheet incorporates several energy-saving schemes, including the heat pump on the right-hand side.	132
Figure 2.83. The CO ₂ -capture unit is computationally upstream of the heat pump in some situations. In these situations, the CO ₂ -capture unit results are unaffected by the heat pump, providing us with simulation results for designs both with and without the heat pump at the same time. In situations that do not have this property, we only find results for a design with the heat pump. The determining factor is whether or not the semi-waste heat source draws heat for the heat pump.	135
Figure 2.84. Sensitivity analysis varying solvent concentration. We concluded that 30 wt% MEA would be a good solvent concentration.	139
Figure 2.85. Sensitivity analysis varying stripper pressure, <i>PSTR</i> . We recorded both thermal regeneration energy <i>Etregen</i> and reboiler temperature <i>TSTR – REB</i> .	140
Figure 2.86. Sensitivity analysis varying lean solvent loading, <i>Llean</i> . We recorded both thermal regeneration energy <i>Etregen</i> and reboiler temperature <i>TSTR – REB</i> .	141
Figure 2.87. Sensitivity analysis over lean solvent loading. We record thermal regeneration energy at 80% and 90% capture rates.	142
Figure 2.88. We adjust the lean solvent flow rate to meet our target capture rate of 80%. This figure shows the calculated lean solvent flow rate necessary to meet the target capture rate when we varied the absorber’s packing. All else equal, the packings that required a lower lean solvent flow rate are more efficient. In order to do this analysis quickly, we relied on the default packing parameter values provided in Aspen Tech’s databanks. We were unable to validate all of these packing parameters.	143
Figure 2.89. Results for varying amounts of semi-waste heat being used to power the heat pump. The “net” series shows the overall effect; the “benefit” series shows the energy need offset by the heat	

pump; the “harm” series shows the increased energy need caused by the siphoning of semi-waste heat.
..... 144

Figure 2.90. Results for varying the lean solvent cooler (LSC) temperature, *TLSC*. We find small reductions in thermal regeneration energy *Etregen* and *Flean* in this case..... 145

Figure 2.91. Results from varying the absorber packed height, *HABS*, for a process with significant energy savings. We see that *HABS* continues to have a strong impact on the overall thermal regeneration energy, *Etregen*..... 146

Figure 2.92. Sensitivity analysis over two of the distributed cross heat exchanger parameters. 147

Figure 2.93. Assessment of the simulation results by Cao Xianghong, Senior Vice President and Chief Technology Officer (retired) of SINOPEC. 148

Figure 2.94. Comparison of designs reported by Zhang *et al.*,³¹ Lin *et al.*,³³ and this work. 149

Figure 3.1. A simple **Heater** block called **HEATER** with an input stream **IN** and an output stream **OUT**.
..... 152

Figure 3.2. Control Panel from an Aspen Plus simulation run. We can understand Aspen Plus simulations as ordinary computer programs. Components shown here match up with normal computer programs as labeled..... 153

Figure 3.3. Create a new project, “Heat Pump”, as a WPF Application in Visual Studio 2015. 154

Figure 3.4. Add the type-definition library file “happ.tlb” as a reference for your new Visual Studio project. This type-definition library provides Visual Studio with the information needed to interact with Aspen Plus..... 155

Figure 3.5. We can see the newly added reference to “happ.tlb” in Solution Explorer as “Happ”. Visual Studio needs Happ in its References list to interact with Aspen Plus. 156

Figure 3.6. Add a new code file called “_DoSomething.cs” to the directory “DoSomething” in our Heat Pump project..... 157

Figure 3.7. Open _DoSomething.cs from Solution Explorer under the DoSomething folder..... 157

Figure 3.8. Change the code in _DoSomething.cs (left) to the new code shown on the right by removing part of the namespace qualifier and declaring the class **DoSomething** to be public, static, and partial.
..... 158

Figure 3.9. Navigate to MainWindow.xaml.cs and add **DoSomething.Run ()** ; in **MainWindow**’s constructor. Since an object’s constructor runs whenever a new instance of the object type is made, **DoSomething.Run ()** will execute whenever our **MainWindow** is created..... 159

Figure 3.10. Source code for DoSomething001.cs. We note the preprocessing directives and the new **using Happ;** directive. 160

Figure 3.11. We comment-out **#define INCLUDE** by prefixing the line with two slashes, **//**. As a result, all of the code inside of the **#if INCLUDE** and **#endif** directives is no longer part of the project. We can easily reverse this by removing the two slashes. 161

Figure 3.12. **DoSomething.Run()** will now serve as a switch board for the other source code files. This configuration will make it easier for us to test new code. 162

Figure 3.13. Two simple C# commands to start a new Aspen Plus instance and then make that instance visible. 162

Figure 3.14. The Start button to run our program. 163

Figure 3.15. Our program created two new windows. The first is an empty window that doesn't really do anything; this is **MainWindow**, which is still blank because we have not defined it. The second window is an instance of Aspen Plus with a new simulation, just as we instructed in **DoSomething001()**. 164

Figure 3.16. Specifications for our simple test flowsheet. A **Heater** block, **HEATER**, has an input stream **IN** and an output stream **OUT**. **IN** is at 25°C and 1bar with 1kmol/hr of each water and nitrogen. **HEATER** is at 15°C and 1bar. 165

Figure 3.17. Code to open our new simulation. 166

Figure 3.18. Instruct our program to run **DoSomething002()** instead of **DoSomething001()** by adjusting the commenting in **DoSomething.Run()**. 166

Figure 3.19. Amend this code to the end of **DoSomething002**. This code will set **HEATER**'s temperature to 10°C, overwriting the flowsheet-specified 15°C, and then run the simulation. 167

Figure 3.20. The results for **OUT** from **DoSomething002**. We can see that the simulation used 10°C, as specified in our code, rather than 15°C, as we had previously specified in the flowsheet. 167

Figure 3.21. Aspen Plus's Variable Explorer feature allows us to explore its internal data tree. We can use this to find the addresses for information that we want to interact with. Note that many nodes exist only when relevant; for example, many Output nodes will only be visible when simulation results are available. 168

Figure 3.22. The **\Data\Blocks\HEATER\Input\TEMP** node from our basic simulation after it was run by our program. We can learn about this node's properties using Variable Explorer. 169

Figure 3.23. Modified code for MainWindow 's constructor. The new code runs DoSomething.Run () in a separate thread, freeing MainWindow to continue normal operations rather than having to wait for DoSomething.Run () to finish.	170
Figure 3.24. Source code for _Utilities.cs , excluding the default using statements at the top.	171
Figure 3.25. Debug and Release modes in Visual Studio 2015.	171
Figure 3.26. Debug feedback showing the HEATER 's calculated heat duty. We can see that this matches the calculated heat duty shown in Aspen Plus.	179
Figure 3.27. Flowsheet configuration for our new PZ+MDEA simulation's absorption unit.	185
Figure 3.28. Conceptual absorption tower with the feedback from the condenser to the packing, ABS-FC-1	187
Figure 3.29. The new Math folder and its contents.	188
Figure 3.30. Convergence behavior of the 21 mole flow rates involved in the PZ absorber convergence, plotted on a log-10 scale. Some of these errors were exactly zero; for those values, we assigned an order-of-error of -7	196
Figure 3.31. Convergence behavior from DoSomething004 () once we enable reinitialization before each run. This simulation required only six evaluations rather than twelve. When there was no error, a value of -9 was assigned. All 21 component mole flow rates were exact on the final iteration.	198
Figure 3.32. Simulation run time for our PZ absorber packing and condenser. The run without reinitializations took 12 iterations, but most of the iterations required less than 4 seconds. The run with reinitializations took half as many iterations, but due to the greatly increased average-time-per-iteration, it took almost twice as long in real time (403 seconds vs. 202 seconds).	199
Figure 3.33. Definitions for CV-TEMP and SQ-TEMP . These two blocks work together to specify the Wegstein convergence method and order for ABS-FC-1 , allowing Aspen Plus to perform the convergence that we had just done in C#.	200
Figure 3.34. Summary of results and the tear history for CV-TEMP . The "Summary" tab focuses on the final iteration in which the tear variables were all within tolerance. The "Results" tab gives a brief overview of the ten Wegstein iterations, listing the variable with the highest error-over-tolerance for each iteration. Note that the mole flow rate of CO ₂ during the final iteration appears on both tabs. ...	202
Figure 3.35. Run-time-per-iteration for DoSomething005 () . In total, this simulation took ~308.7 seconds and 56 iterations.	210
Figure 3.36. Flowsheet for the stripper simulation.	211

Figure 3.37. Specifications for STRIPPER	213
Figure 3.38. Simulation time for each iteration during the stripper’s convergence in DoSomething006 () . The total run time was about 603.21 seconds over 198 iterations.....	223
Figure 3.39. Flowsheet for the distributed cross heat exchanger section of the PZ+MDEA simulation.	224
Figure 3.40. Specification for CHEXMAIN . Use the same for CHEXSIDE	225
Figure 3.41. Convergence behavior for the distributed cross heat exchanger. We use an error tolerance of 10 – 5 in part to avoid the noise below the precision limit.	232
Figure 3.42. Order of error for each tried tear value, $t_{\text{torn}i}$. We can see that the values with the lowest errors are concentrated around a particular peak, suggesting a more precise value despite the noise.	233
Figure 3.43. Flowsheet logic merging our individual flowsheets together under object-oriented logic.	234
Figure 3.44. The absorber and DHEX are separated by the rich solvent pump. In principle, the rich solvent pump would exist in its own simulation connected to both the absorber simulation (Absorber.apwz) and the DHEX simulation (DHEX.apwz). This configuration would require either (1) an additional license to keep Pump.apwz open or (2) for us to open-and-close Pump.apwz each time we want to run the rich solvent pump.	236
Figure 3.45. Our new absorber complex flowsheet in Absorber.apwz. This flowsheet adds RICHPUMP to avoid having to put RICHPUMP in its own simulation.	238
Figure 3.46. The absorber complex and the DHEX are still separated by RICHPUMP , though we avoid the need for an additional rich solvent pump simulation, e.g. Pump.apwz. This comes at the cost of having to run RICHPUMP every time Absorber.apwz is run, e.g. about 54 unnecessary times in our last test in which we used the Wegstein method to converge the absorber’s condenser feedback (ABS-FC-1 and ABS-FC-2) and the secant method to set the appropriate effective carbon capture rate in the absorber, \mathcal{R}_{ABS} , by adjusting the lean solvent flow rate.	239
Figure 3.47. Additional streams to add to Stripper.apwz, feeding into STRIPPER : RVAP-IN1 , RVAP-IN2 , MVAP-IN1 , and MVAP-IN2	241
Figure 3.48. We use Task Manager, accessible by Ctrl+Alt+Del in Windows, to intentionally crash Aspen Plus while it is running.	243
Figure 3.49. The forced crash caused an exception shown above. This exception is immediately received by the wrapper, and the wrapper recovers by ensuring that the crashed Aspen Plus instance is fully closed, then opens up a fresh instance.	244
Figure 3.50. Part of the simulation interface automatically generated by our simulation framework. This particular screenshot was taken are the run completed. We can see the recursive tree structure of	

blocks, including the number of times each block was executed and how much time that block spent running. Additionally we added a Pause button and an Exit button for basic simulation control.....250

List of Tables

Table 1.1. Explanations for the three absorber intercooling parameters.	39
Table 1.2. Distributed cross heat exchanger configuration parameters to be declared in C-GLOBAL	72
Table 1.3. Major parts of the heat pump.....	88
Table 1.4. Stream results for vapor flows from columns' packed sections to their condensers. ABS-TC-2 is for the absorber while STR-TC-2 is for the stripper.....	101
Table 1.5. Heat pump flowsheet specifications.	105
Table 1.6. Flowsheet-level specifications for the additional estimation units in the heat pump simulation.	106
Table 1.7. Global parameters for C-GLOBAL . Define each parameter as an Export variable.	108
Table 1.8. Parameters to be declared in C-GLOBAL	130
Table 1.9. Flue gas specifications.	138
Table 2.1. Comparison between Aspen Plus programming and C# programming.	152
Table 2.2. Stream table inputs for the PZ+MDEA absorption simulation.	185
Table 2.3. Tear stream results from DoSomething004 () . ABS-FC-2 and ABS-FC-1 represent the same stream, so they should have the same values.	196

List of Abbreviations

- ABS Absorber
- CCS Carbon Capture and Storage
- CHEX Cross Heat Exchanger
- CON Condenser
- DHEX Distributed Central Cross Heat Exchanger
- COM Component Object Model
- CON Condenser
- EOR Enhanced Oil Recovery
- EP Energy Penalty
- GEN Generator
- HP Heat Pump
- IPCC Intergovernmental Panel on Climate Change
- LSC Lean Stream Cooler
- MEA Monoethanolamine
- MDEA Methyldiethanolamine
- PZ Piperazine
- REB Reboiler
- STR Stripper
- VAP Evaporator

Nomenclature

Variable	Units	Description
A_i	$[\text{m}^2]$	Heat-exchanging surface area.
B_i	$[\text{MW}_e]$	Electric duty.
χ_{HP}	$[-]$	Heat pump extent.
D_i	$[\text{MW}_t]$	Heat duty.
E_e^{regen}	$\left[\frac{\text{GJ}_e}{\text{tonne}_{\text{CO}_2}} \right]$	Electric regeneration energy.
E_t^{regen}	$\left[\frac{\text{GJ}_t}{\text{tonne}_{\text{CO}_2}} \right]$	Thermal regeneration energy.
$\mathcal{E}_e^{\text{CO}_2}$	$\left[\frac{\text{tonne}_{\text{CO}_2}}{\text{GJ}_e} \right]$	Electric carbon emission rate.
EP	$[-]$	Energy penalty.
EP _{prod-loss}	$[-]$	Production-loss energy penalty.
EP _{size-up}	$[-]$	Size-up energy penalty.
η_i	$\left[\frac{\text{GJ}_e}{\text{GJ}_t} \right]$	Thermal conversion efficiency.
F_i	$\left[\frac{\text{mol}}{\text{hr}} \right]$	Mole flow rate.
G	$[\text{J}]$	Gibbs energy.
H_i	$[\text{m}]$	Height.
k	$[-]$	Scaling factor.
P	$[\text{bar}]$	Pressure.
R	$[-]$	Carbon capture rate.
T	$[\text{K}]$	Temperature.
W_{eq}	$[\text{MW}]$	Equivalent work

1 Introduction

1.1 Process System Engineering

Process System Engineering (PSE) is the branch of Chemical Engineering that focuses on computer-aided design and operation of chemical systems. The field's name is attributed to an American Institute of Chemical Engineers (AIChE) Symposium Series from 1961, though it was not well established until showcased at a 1982 symposium in Kyoto, Japan.³ While its definition varies, Grossmann and Westerberg offer:

“Process Systems Engineering is concerned with the improvement of decision-making processes for the creation and operation of the chemical supply chain. It deals with the discovery, design, manufacture, and distribution of chemical products in the context of many conflicting goals.”

SimSci launched in 1967 marketing computational tools related to PSE. Later in 1982, MIT Professor Larry Evans launched Aspen Technology with the belief that the field should take a more model-centric approach. Today modern examples of process simulation example include AspenPlus, HYSYS, DWSIM, CHEMCAD, PRO/II, and PROSIM. Literature also contains many examples of models built on more computational software such as Mathematica, MATLAB, and Excel.

1.2 Motivation and Significance

This research aims to enable the climate mitigation strategies called for by the Intergovernmental Panel on Climate Change (IPCC) through PSE while fulfilling the requirements for a Ph.D. in Chemical Engineering at Virginia Tech. In its recent Summary for Policymakers, the IPCC warns that stabilizing climate change at 2°C above pre-industrial levels will require:⁴

- no later than 2050: at most only 20% of power can come from fossil fuels without CCS;
- no later than 2100: 100% deployment of carbon capture on all fossil fuel plants.

While the basic design for a modern absorption-driven CO₂-capture unit was patented⁵ 85 years ago, straightforward implementations can halve the power output at coal-fired power plants. We discuss quantifying the energy requirement for a CO₂-capture unit in Section 1.4.2.3: “Quantifying energy efficiency”.

The goal of this project has been to enable climate change mitigation by providing designs for practical post-combustion carbon capture units appropriate for near-term deployment.

1.3 Programming Conventions and Presentation Formats for the Chapter

This chapter follows the actual process that we have used to simulate and optimize the monoethanolamine (MEA) -based CO₂-capture unit published in our 2015 paper on the subject.⁶ This design process explicitly recognizes the fact that flowsheets like those created in Aspen Plus are computer programs, so we employ font-formatting as a basic element of grammar. These grammatical rules are:

1. General concepts appear in normal text.
2. Class types appear in **case-sensitive Code formatting**.
3. Class instances appear in **UPPER-CASE CODE FORMATTING**.
4. Code sections such as procedural code from a **Calculator** block appear

in case-sensitive code blocks.
5. Vocabulary appears in **vocabulary formatting**.
6. Mathematical expressions appear in equation objects.
 - a. Numbers are mathematical expressions when they are intended as such.
 - b. Numbers are not mathematical expressions when they are meant as an identifier, e.g. the digit in “Chapter 7”.
7. Alphabetic characters in equation objects are italicized when they identify variables and not italicized otherwise.
 - a. Example: F_{lean} italicizes “ F ” because it is a variable while “lean” is not italicized because it is a qualifier.
 - b. Units are treated as qualifiers and thus not italicized, e.g. $N[\equiv] \frac{kg \cdot m}{s^2}$.
 - i. Variables and units are often mixed in well-dimensionalized expressions, e.g. $\frac{T}{K}$ is a common expression for the temperature in Kelvin.

We select these conventions to maintain consistency across related disciplines.

Because this chapter is focuses primarily on Aspen Plus:

- most class types will be block types like `Flash2` and `RadFrac`;
- most class instances will be block instances like `ABSORBER` and `STRIPPER`;
- most code sections will be interpreted Fortran from inside `Calculator` blocks.

We follow these grammatical rules in order to convey meaning rather than simply aid the reader. For example, the absorber and `ABSORBER` are *not* the same thing; the first is the physical unit while the second is a specific instance of a `RadFrac` block.

For convenience, we have several exceptions:

1. Steam types are class types, but I have not put them into class-type format.
 - a. However, stream instances *are* properly put into class-instance format.
2. Aspen Plus is itself a class type, and instances of Aspen Plus are instances, but these are not formatted.
3. Select variables are not italicized, e.g. the energy penalty EP.

1.4 Literature overview

Patents for acid gas separation date back to 1927. Stronger interest has grown starting in the 1980's, increasing with concern for climate change and global industrialization.

1.4.1 Base-case design

The first acid-gas capture system was reported in 1927 in US Patent 1,897,725 as a series of absorption towers as shown in Figure 1.1.⁷ These absorption towers used aqueous ammonia to absorb CO₂ from a flue gas.

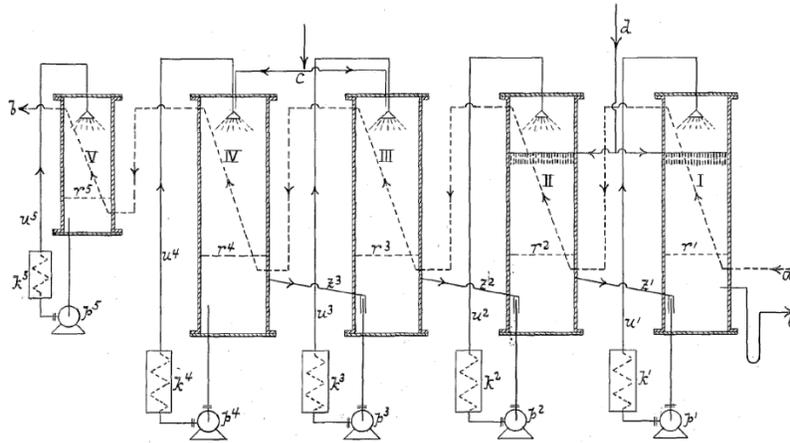


Figure 1.1. A series of absorption towers working in concert. This design is from US Patent 1,897,725 (1927).

In 1930, another patent⁵ reported a significant advancement to the basic acid gas capture system configuration used today. Figure 1.2 shows this baseline design.

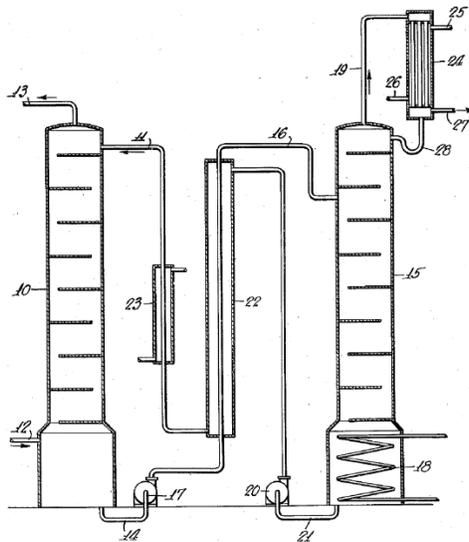


Figure 1.2. Basic acid-gas-capture unit as reported in US Patent 1,783,901 (1930). The tower on the left is the absorption tower (absorber) and the tower on the right is the regeneration tower (stripper). Also shown are the stripper's reboiler (18), the stripper's condenser (24), the central cross heat exchanger (22), the lean stream cooler (23), and solution pumps (17 and 20). Modern implementations tend to build on this basic design.

This design had several major advancements:

1. incorporation of a regeneration tower, allowing solvent reuse;
2. generalization from capturing CO₂ to capturing acid gases (which include both CO₂ and H₂S);

3. listed solvents included alcohol amines with a specific focus on monoethanolamine (MEA).

The features from the 1930's design remain standard today. In particular:

1. **absorption tower:** Removes acid gas from the stream to purify.
2. **regeneration tower:** Regenerates solvent, releasing the captured acid gas.
3. **central cross heat exchanger:** Significantly reduces energy consumption by transferring heat from the freshly regenerated solvent stream to the loaded solvent stream.

While there have been many modifications to the initial design from 1930, this basic design remains. Hereafter we will refer to it as the base-case design.

1.4.1.1 Solvent loop

This unit operates by circulating an acid-gas-capturing solvent. The solvent captures acid gas in the absorption tower (absorber) and then releases that acid gas in the regeneration tower (stripper). Figure 1.3 shows these elements.

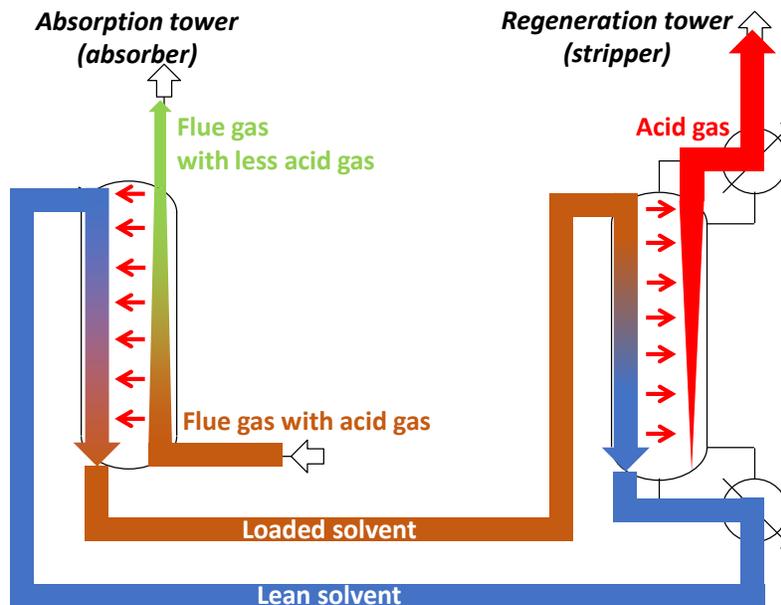


Figure 1.3. Conceptual acid-gas-capture unit.

Solvent leaving the absorber has the maximum amount of acid gas in it. We call this the **rich solvent** or **loaded solvent**.

Solvent leaving the stripper has the minimum amount of acid gas in it. We call this the *lean solvent* or *unloaded solvent*.

1.4.1.2 Thermal loop

The solvent loop works due to the thermal loop:

- in the absorber, acid gas is absorbed since it is relatively cold;
- in the stripper, acid gas is removed since it is relatively hot.

In both cases, Le Châtelier's principle¹ fights our efforts:

- the absorber is warmed by capturing acid gas;
- the stripper is cooled by releasing acid gas.

A working acid gas capture unit could rely on the lean-solvent cooler (LSC) and the stripper's reboiler (STR-REB). However the 1930 design includes the central cross heat exchanger to significantly reduce the burden on both the LSC and STR-REB. Figure 1.4 illustrates the resulting thermal loop.

¹ "Any change in the status quo [of a system at a stable equilibrium] prompts an opposing reaction in the responding system [to reach the new equilibrium]." (8. Gall J. *Systemantics: How Systems Work and Especially How They Fail*: Pocket; 1978.)

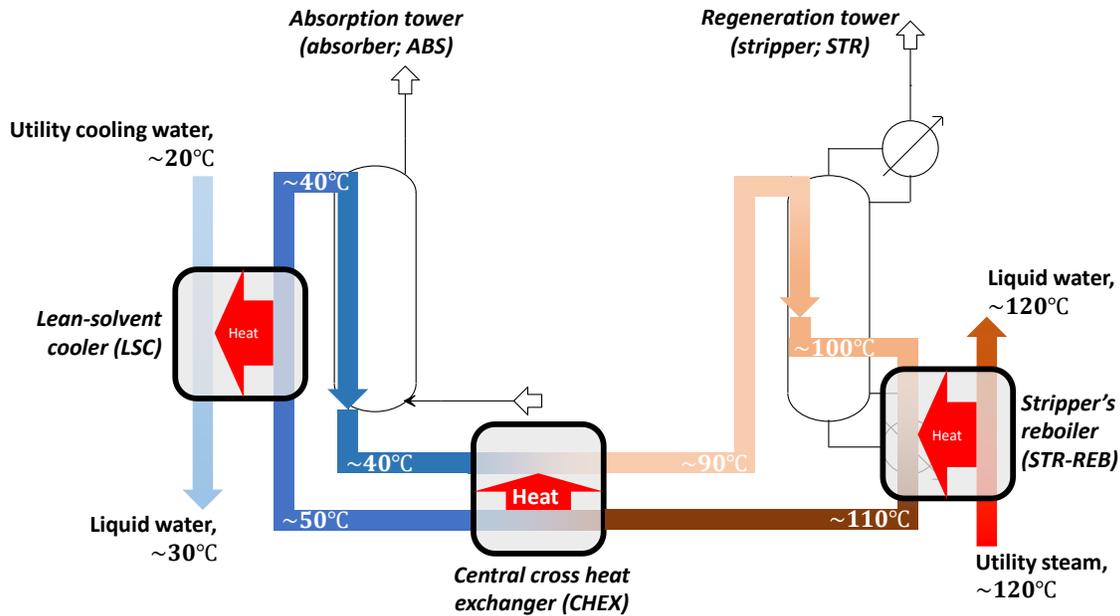


Figure 1.4. Thermal loop in a conceptual acid-gas-capture unit.

The above temperatures are rough estimates for a particular CO₂-capture unit that uses aqueous MEA as a solvent. Variations in design – especially when a different solvent is used – can significantly change these temperatures.

In practice, we also include condensers on both towers. These condensers cool the outgoing vapor stream to around ~40°C to avoid significant solvent loss. Typically these condensers use utility cooling water like the LSC does, however later we will look at using a heat pump instead.

1.4.2 Design considerations

1.4.2.1 Sorbent (solvent) selection

There are many possible choices for acid gas solvents. The default choice for academic research has been monoethanolamine (MEA) as discussed in the 1930 patent⁵ for the basic process design, though today we are aware of hundreds of potential solvents with several popular proprietary offerings.

While there are studies that have looked at the properties of many potential solvents at once, these studies tend to focus on a few characteristic properties rather than provide a complete analysis of each

solvent system. The limited list of solvent systems in the Aspen Plus examples folder reflects the lack of solvent data in the public domain.

We focus on aqueous MEA (monoethanolamine) and PZ (piperazine) because these systems are relatively well understood.

1.4.2.2 Target capture rate, R_{target}

Engineers designing a capture unit usually aim to minimize costs while meeting a capture target. This capture target, called the capture rate R , is usually discussed in terms of the portion of flue acid gas to be captured. Capture rate R is defined in Equation 1.1.

$$R \equiv \frac{[\text{flow rate of captured acid}]}{[\text{flow rate of flue acid gas}]} \quad 1.1$$

Most fossil-fuel power stations do not employ carbon capture, so they have a capture rate $R = 0$. Research often focuses on capture rates from 80% to 95%, though some pilot-scale projects end up capturing lower rates.

Industrial applications are likely to target lower capture rates because:

- i. lower capture rates are cheaper both in terms of absolute cost and unit cost;
- ii. lower capture rates are often sufficient for the two primary industrial needs:
 - a. enhanced oil recovery (EOR);
 - b. meeting regulatory emission limits such as the 2015 Clean Power Plan (US).⁹

Figure 1.5 shows the capture rate scale.

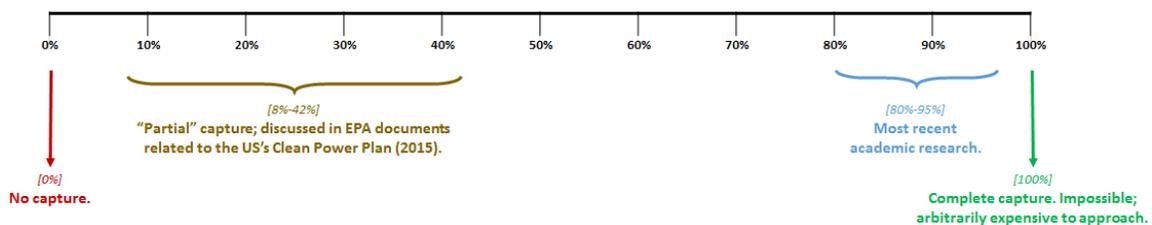


Figure 1.5. Scale showing the range of capture rates from zero to 100%.

1.4.2.3 Quantifying energy efficiency

Popular methods for quantifying energy efficiency include:

- **regeneration energy:** Amount of utility heat or/and electricity required per unit of acid gas captured.
- **energy penalty [production loss]:** The portion of hypothetical energy production lost due to implementing acid gas capture.
- **energy penalty [size-up factor]:** The increase in power plant size needed to obtain the net output as if acid gas capture had not been implemented.

1.4.2.3.1 Regeneration energy

Regeneration energy is the amount of utility energy required to capture a unit of acid gas. We are primarily interested in thermal regeneration energy (often provided by utility steam) and electrical regeneration energy (from electricity).

Equation 1.2.a shows the definition for thermal regeneration energy, E_t^{regen} . Equation 1.2.b shows the common simplification that applies to the baseline capture unit.

$$E_t^{\text{regen}} \equiv \frac{\sum_{\text{consumers}} \text{utility heat}_i D_i}{F_{\text{captured CO}_2}} \quad 1.2.a$$

$$E_t^{\text{regen}} = \frac{D_{\text{STR-REB}}}{F_{\text{captured CO}_2}} \quad 1.2.b$$

Thermal regeneration energy is often given in units of $\frac{\text{GJ}_t}{\text{tonne CO}_2 \text{ captured}}$. In the baseline design, the entirety of this utility heating is used to power the regeneration tower's reboiler. Modified designs would have to add their heating requirements together. The IPCC's 2005 special report on carbon capture and storage (CCS) claimed that leading absorption technologies have thermal regeneration energies from 2.7 to 3.3 $\frac{\text{GJ}_t}{\text{tonne CO}_2}$.² We have reported a design with a thermal regeneration energy of 1.67 $\frac{\text{GJ}_t}{\text{tonne CO}_2}$.⁶

Equation 1.3 shows the definition for electrical regeneration energy, E_e^{regen} , where B_i is the electrical duty of unit i . The electrical reaction energy is an additional cost of operating an acid-gas-capture unit, *not* an alternative cost. Usually the largest electric consumer is the compressor that liquefies captured acid gas for transport, with solvent pumps being a secondary electric consumer. Some designs employ compressors in them, leading to much higher electrical regeneration energies.

$$E_e^{\text{regen}} \equiv \frac{\sum_{\text{consumers}} \text{electricity}_i B_i}{F_{\text{captured CO}_2}} \quad 1.3$$

Electrical regeneration energy is often given in units of $\frac{\text{GJ}_e}{\text{tonne CO}_2 \text{ captured}}$. For most designs, the largest electricity sink is the compressor that compresses the captured acid gas into a liquid or supercritical fluid. Some designs will employ compressors in the process itself. The IPCC's report claimed that leading absorption technologies have electrical regeneration energies from 0.06 to $0.33 \frac{\text{GJ}_e}{\text{tonne CO}_2}$.

1.4.2.3.2 Energy penalties

Carbon-capture units are commonly applied to coal-fired power plants. When applying a carbon-capture to a coal-fired power plant, we can quantify the carbon-capture unit's performance in terms of an energy penalty as an alternative to regeneration energy.

Figure 1.6 shows the base case power plant design. In this basic power plant, a coal burner produces heat that is used to generate steam for a turbine, producing electricity.

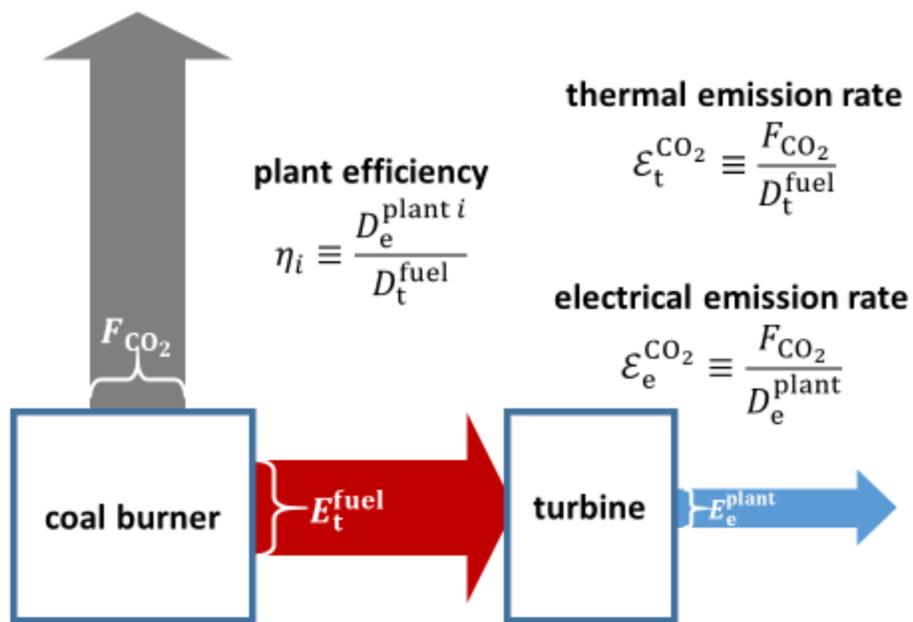


Figure 1.6. Conceptual layout for a coal-fired power plant without a carbon-capture unit.

Figure 1.7 shows this same coal-fired power plant with a carbon-capture unit implemented. The carbon-capture unit draws both heat that would have been used to power the turbine as well as electricity that would have been useful output. We attempt to capture both parasitic effects together as the energy penalty.

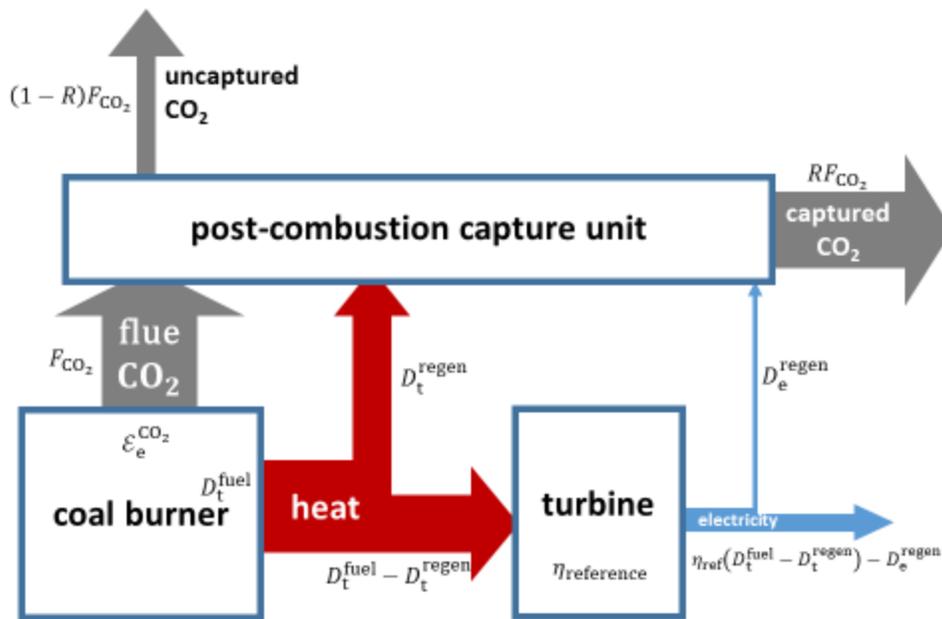


Figure 1.7. Coal-fired power plant with a carbon-capture unit.

The energy penalty is commonly defined in two different ways:¹⁰

1. **production-loss energy penalty, $EP_{\text{prod-loss}}$:** The portion of potential electrical output production not realized due to implementation of CCS.
2. **size-up energy penalty, $EP_{\text{size-up}}$:** The size-up factor applied to the base case to avoid a production loss due to implementation of CCS.

Equations 1.4.a-b give the expressions for the production-loss energy penalty, $EP_{\text{prod-loss}}$, and the size-up energy penalty, $EP_{\text{size-up}}$. In these expressions:

- R is the carbon capture rate defined in Equation 1.1;
- E_t^{regen} is the thermal regeneration energy defined in Equation 1.2;
- E_e^{regen} is the electric regeneration energy defined in Equation 1.3;
- $\mathcal{E}_e^{\text{CO}_2}$ is the electrical emission rate, defined as the mass flow rate of CO_2 emitted per unit of net electrical energy production in the base case plant;

- η_i is the ratio of net electrical energy output of plant i to the thermal energy it obtained from fuel to generate that output, described in Figure 1.6:
 - $\eta_{\text{without CCS}}$ is this ratio for a power plant that has no carbon-capture unit;
 - $\eta_{\text{with CCS}}$ is this ratio for a power plant that does have a carbon-capture unit.

$$EP_{\text{prod-loss}} = \frac{\eta_{\text{without CCS}}}{\eta_{\text{with CCS}}} - 1 = R\mathcal{E}_e^{\text{CO}_2} (\eta_{\text{without CCS}} E_t^{\text{regen}} + E_e^{\text{regen}}) \quad 1.4.a$$

$$EP_{\text{size-up}} = 1 - \frac{\eta_{\text{with CCS}}}{\eta_{\text{without CCS}}} = \frac{1}{\frac{1}{R\mathcal{E}_e^{\text{CO}_2} (\eta_{\text{without CCS}} E_t^{\text{regen}} + E_e^{\text{regen}})} - 1} \quad 1.4.b$$

These two metrics are similar, though the production-loss definition will be quantitatively smaller than the size-up definition. Figure 1.8 illustrates the quantitative difference grows with the value of both metrics.

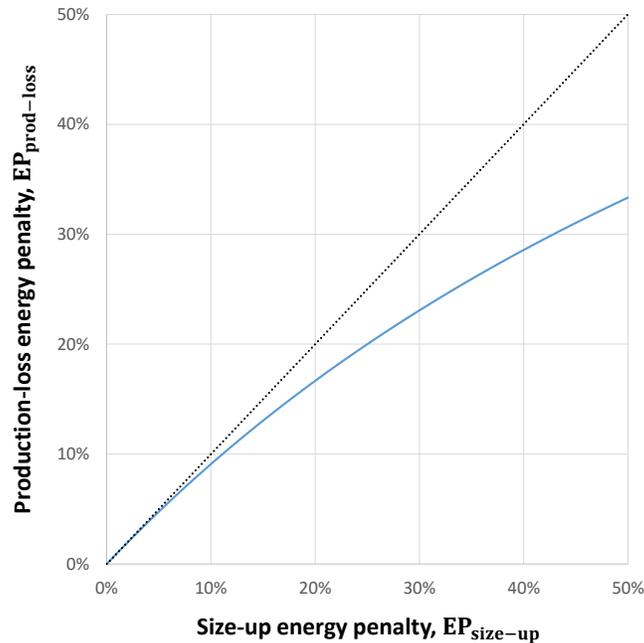


Figure 1.8. The production-loss energy penalty plotted against the size-up energy penalty. For very low values, it is not excessively misleading to confuse the two metrics, however the deviation grows with both values. At the limit of a 100% production-loss energy penalty, the size-up energy penalty is infinite.

To convert:

$$EP_{\text{prod-loss}} = 1 - \frac{1}{EP_{\text{size-up}}} \quad 1.5.a$$

$$EP_{\text{size-up}} = \frac{1}{1 - EP_{\text{prod-loss}}} \quad 1.5.b$$

1.4.2.3.3 Equivalent work

The production-less energy penalty, $EP_{\text{prod-loss}}$, describes the overall production loss at a power plant due to implementing a carbon-capture unit. This information is very useful for engineers considering specific applications, but it is hard to compare carbon-capture units reported in literature this way because the reference power plant greatly affects the energy penalty.

Equivalent work is similar to energy penalty, but it attempts correct for the reference power plant, making it a better metric for comparing between reported designs. Equation 1.6 shows how to calculate equivalent work from either energy penalty or both regeneration energies.

$$W_{\text{eq}} = \frac{EP_{\text{prod-loss}}}{R\mathcal{E}_e^{\text{CO}_2}} = \frac{1 - \frac{1}{EP_{\text{size-up}}}}{R\mathcal{E}_e^{\text{CO}_2}} = \eta_{\text{without CCS}} E_t^{\text{regen}} + E_e^{\text{regen}} \quad 1.6$$

Babatunde and Rochelle suggest estimating $\eta_{\text{without CCS}}$ by assuming that the reference power plant operates as a Carnot heat engine at 75% efficiency.^{11,12,13,14} Equation 1.7 shows this expression.

$$\eta_{\text{without CCS}} \approx \epsilon_{\text{Carnot}} \frac{T_{\text{reboiler}} + \Delta T_{\text{reboiler-approach}} - T_{\text{turbine}}}{T_{\text{reboiler}} + \Delta T_{\text{reboiler-approach}}} \quad 1.7$$

1.4.2.3.4 Recommended quantification metrics

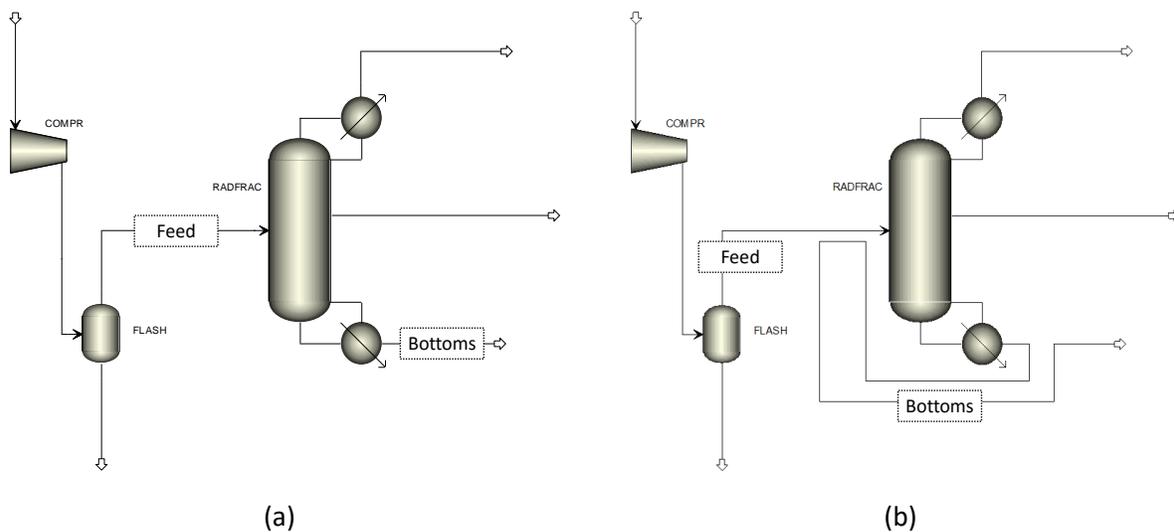
In general, we recommend reporting the thermal regeneration energy E_t^{regen} and electrical regeneration energy E_e^{regen} with the corresponding qualifiers, e.g. heat of steam needed for the thermal regeneration energy.

When a simple, single metric is desired, equivalent work W_{eq} can suffice. Reported values of equivalent work should be qualified with their approach to estimating $\eta_{\text{without CCS}}$.

When working in an applied application, an energy penalty such as $EP_{\text{prod-loss}}$ or $EP_{\text{size-up}}$ can be appropriate. However, we strongly recommend that the definition be clearly provided to avoid ambiguity.

1.5 General approach: Expand the model space and optimize

Let us consider the process flowsheets shown in Figure 1.9. First, say that we have a random chemical process flowsheet such as given in Figure 1.9.a. We are entirely free to draw this process in a contorted way (Figure 1.9.b). We may then assert that, conceptually, there is some heat-exchanging area A_{CHEX} between the **Feed** and **Bottoms** streams (Figure 1.9.c). Finally, we may draw a **HeatX** block to provide a computational unit for A_{CHEX} (Figure 1.9.d). The **HeatX** block can have this specification set as shown in Figure 1.10.



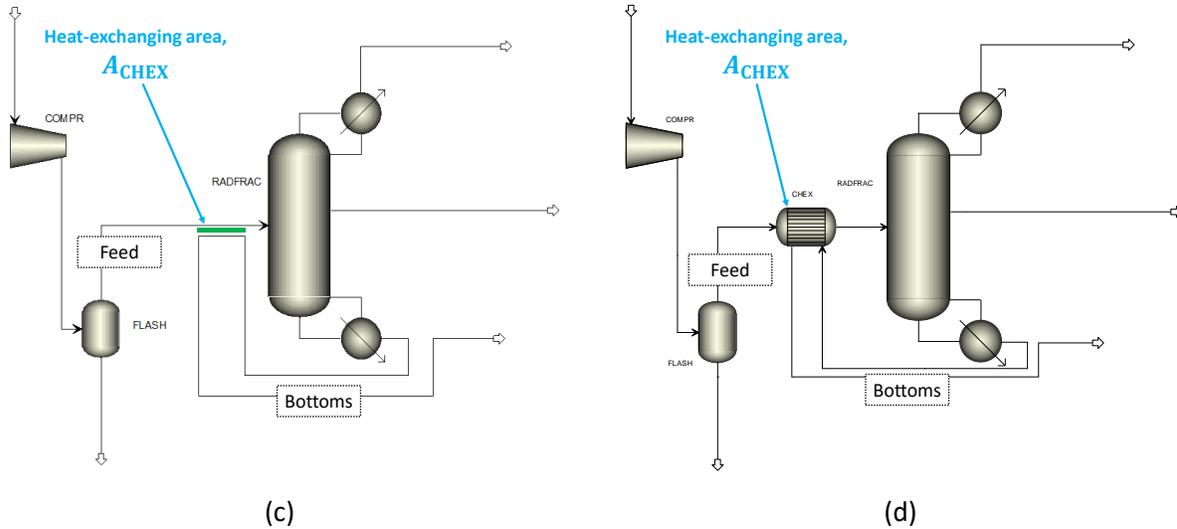


Figure 1.9. Example generalization of a basic process. We consider an arbitrary process design such as in (a). We can redraw this arbitrary process as in (b) with no actual change to the system. We may acknowledge that there is, conceptually, zero heat-exchanging surface area $A_{CHEX} = 0$, without changing the process. Finally, we can draw a **HeatX** block with zero heat-exchanging surface area without changing the process. Overall, we have made no change to (a) except that the flowsheet can now consider the possibility of adding a cross heat exchanger as shown in (d). Aspen Plus should report the same results for both (a) and (d) in the $A_{CHEX} = 0$ case within the limits of numerical error tolerance.

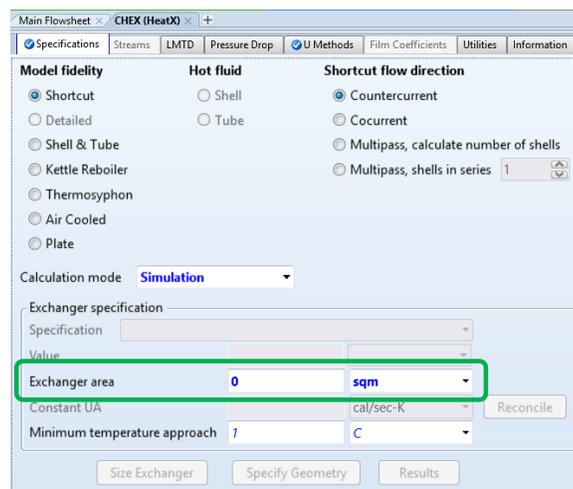


Figure 1.10. Specify zero heat-exchanging surface area in a hypothetical **HeatX** block.

We are free to allow A_{CHEX} to vary in an **Optimization** block. If the **Optimization** finds that $A_{CHEX} = 0$ is optimal, then there is simply no cross heat exchanger there, and our process is effectively

identical to the original flowsheet (Figure 1.9.a). However, if the **optimization** finds that $A_{\text{CHEX}} \gg 0$, then we include the cross heat exchanger (Figure 1.9.d).

Terminology

In the $A_{\text{CHEX}} = 0$ case, we say that the **HeatX** called **CHEX** is *hypothetical*. Hypothetical units are considered in the flowsheet description space, but they have no material existence in the prescribed design. Later we will see examples of other types of hypothetical units.

This approach is fundamental to describing energy-saving schemes and optimizing our process. Rules:

1. Flowsheet modifications should have optimizable descriptions.
 - Example: **CHEX** is described by its area, A_{CHEX} .
2. Flowsheet modifications should not prohibit the optimizer from concluding that the base case is optimal.
 - Example: Adding **CHEX** to our example process does not necessarily change the process because the optimizer is entirely free to conclude $A_{\text{CHEX}} = 0$.
3. Flowsheet modifications should exclude non-physical descriptions.
 - Example: It may be tempting to describe “CHEX” in terms of its transferred heat duty, D_{CHEX} , rather than its surface area, A_{CHEX} . We generally try to avoid this approach because it is often too easy to specify an overly large D_{CHEX} , allowing the cross heat exchanger to transfer heat against a temperature gradient.
4. Flowsheet modifications should be well-behaved.
 - Example: **CHEX** is described by its area, A_{CHEX} . While $\ln(A_{\text{CHEX}})$ might be a reasonable modification because area suffers from diminishing returns, $e^{A_{\text{CHEX}}}$ would worsen the situation.
5. Consider the motivation for flowsheet modifications.
 - Example: Adding **CHEX** to our process greatly increases the computational cost of solving the flowsheet. In addition to adding this unit, we have created a new feedback

loop from **BOTTOMS** to **RADFRAC** that needs to be converged. If another flowsheet modification seems more likely to result in improvements for the same computational cost, we should consider it first.

1.5.1 Heat integration

The above example considering a hypothetical cross heat exchanger **CHEX** is an example of heat integration. Heat integration is one of the oldest, most common, and most important energy-saving approaches. Many of the specific schemes that we will look at below are specific heat integrations.

Aspen Plus provides automated heat integration through its “Activated Energy Analysis” tool, shown in Figure 1.11. At the time of this writing (Aspen Plus V8.8), the Activated Energy Analysis tool is relatively new and not yet a complete replacement for human-guided design, so it is best used as a starting place for more detailed analysis. However, the procedural nature of this work lends itself to automation; we suspect that software tools like Activated Energy Analysis will ultimately obsolete human analyses in most cases.

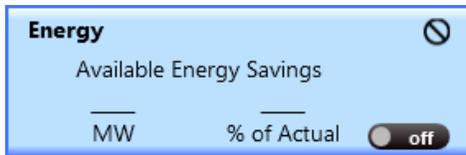


Figure 1.11. Dock panel for Aspen Plus's Activated Energy Analysis tool.

1.5.2 Utility integration

We can make our flowsheet more complete by explicitly drawing utilities as well. For example, we can explicitly represent cooling water utility on a flowsheet as shown in Figure 1.12.

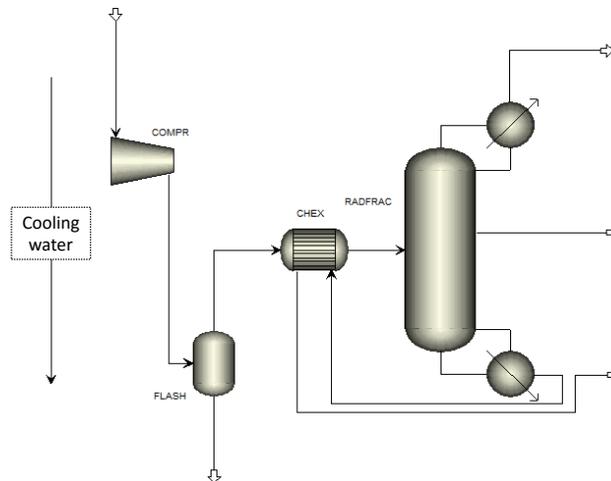


Figure 1.12. Explicit representation of cooling water utility on a flowsheet. We often omit utility streams for brevity.

We can then see that the exact same heat integration approach can apply to the application of utility heating/cooling. This is the basis for one of the oldest, simplest, and most common energy-saving schemes, that we discuss in Section 2.1: “Absorber intercooling”.

Aspen Tech strongly recommends specifying available utilities when using their Activated Energy Analysis tool.¹⁵ If the user does not specify utilities, Aspen Plus will make assumptions.

1.6 Tips and Tricks

1.6.1 Interpolating non-integer values

We can make some artificially integer values continuous by interpreting them as interpolations of two specifications. We usually apply this approach for specifying the stage number that a stream feeds into or comes out of.

1.6.1.1 Example: Interpolating the stage number for a material stream feeding into a packed column

If a stream feeds into a column, we can split it as shown in Figure 1.13.

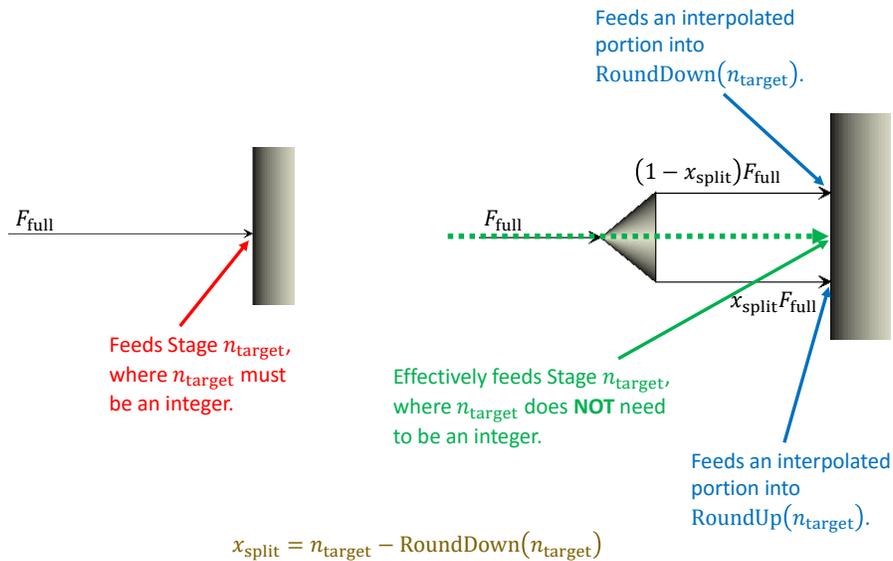


Figure 1.13. We can split a stage before feeding it into a column to implement non-integer stage specifications.

In the above figure, the full flow rate F_{full} of an incoming material stream is split into two streams which feed into consecutive stages on the target packed column. If $F_{\text{full}} = 10 \frac{\text{mol}}{\text{hr}}$ and $n_{\text{target}} = 4.8$, then:

- $x_{\text{split}} = n_{\text{target}} - \text{RoundDown}(n_{\text{target}}) = 0.8$;
- the first split feeds into Stage #4 with $(1 - x_{\text{split}}) \times F_{\text{full}} = 2 \frac{\text{mol}}{\text{hr}}$;
- the second split feeds into Stage #5 with $x_{\text{split}} \times F_{\text{full}} = 8 \frac{\text{mol}}{\text{hr}}$.

In Aspen Plus, we can split a material stream using the **FSplit** block. A **Calculator** block can read the specified target stage n_{target} and then set the split x_{split} and integer stage numbers into the column.

Conceptually, this interpolation splitting process is part of the packed column itself, so a cautious modeler can derive a new column object that includes the splitting blocks at a low level. This is often sufficiently practical because neither the **Calculator** nor **FSplit** is computationally intensive, so it does not take but a very small fraction of a second for them to run.

1.6.1.2 Example: Interpolating the stage number for a material pseudo-stream flowing from a packed column

As a reminder, material pseudo-streams in Aspen Plus do not affect the column that produces them.

We can interpolate a pseudo-stream specification using the approach shown in Figure 1.14.

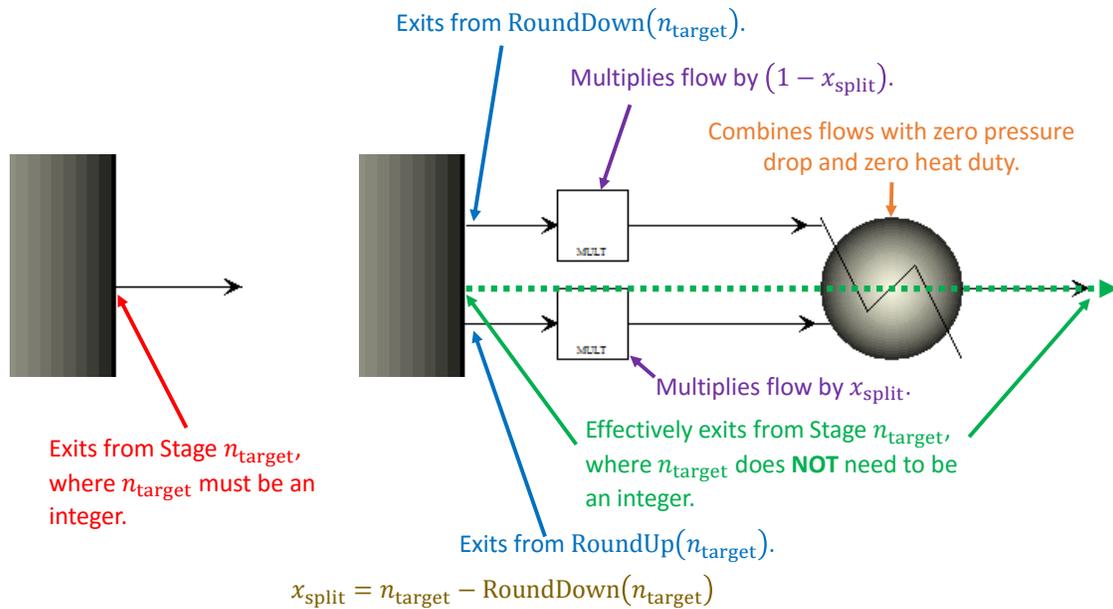


Figure 1.14. Flowsheet configuration for non-integer draw specifications.

Here, we have two material pseudo-streams coming from consecutive integer stages on packed column. Each pseudo-stream goes through a **MULT** block that reduces its flow, and then the sum of both streams is mixed in a **Heater** block with no pressure drop ($\Delta P_{mixer} = 0$) and no heat duty ($D_{mixer} = 0$).

1.6.1.3 Example: Interpolating the stage number for a heat stream flowing into a packed column

In addition to material streams, the approach shown in Section 1.6.1.1, “Example: Interpolating the stage number for a material stream feeding into a packed column”, also works for heat streams and work streams because Aspen Plus allows **FSplit** blocks to split these types of streams as well.

We use an **FSplit** block for material streams because material streams contain many values for thermodynamic state to pass along and material flow rate values to split. By contrast, heat streams and work streams are often much simpler. We often find it more convenient to omit the **FSplit** block

when working with heat streams and work streams. In this case, we can use the approach shown in Figure 1.15.

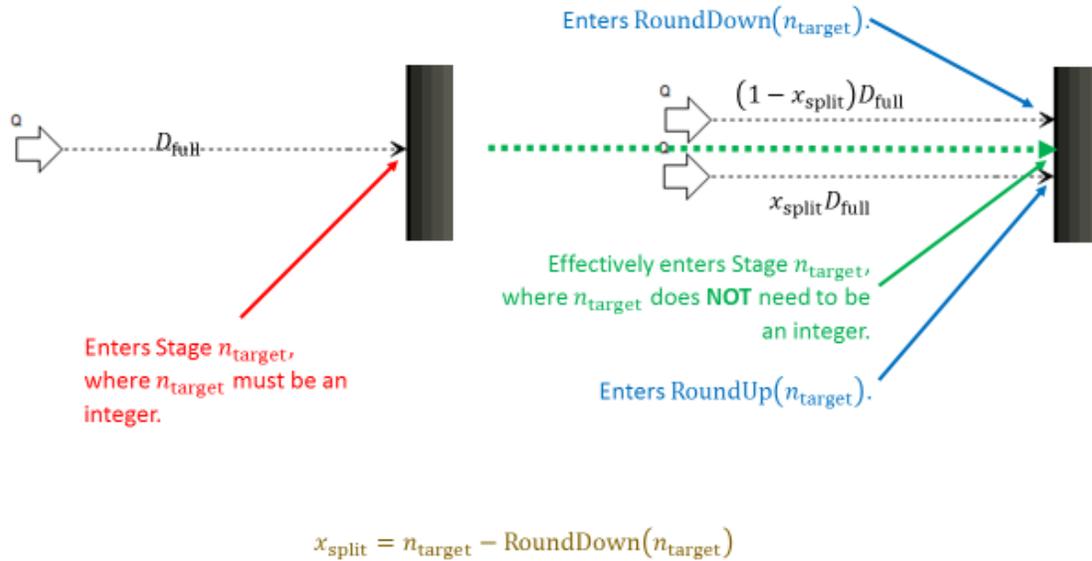


Figure 1.15. Configuration for non-integer heat stream specification.

This works a lot like the material stream example, except that the **Calculator** block is responsible for setting the two duties $(1 - x_{\text{split}})D_{\text{full}}$ and $x_{\text{split}}D_{\text{full}}$ rather than setting x_{split} for an **FSplit** block to perform this same calculation.

1.6.2 Reconciling estimates

Initial estimates are often necessary for complex flowsheets to converge. Better estimates generally result in faster, more reliable convergence. At a minimum, we should provide reasonable estimates that are sufficiently close to the final values in order to put the model in a well-behaved region without computational errors.

There are many different ways to provide good initial estimates. Here we will discuss several methods for specifying initial estimates.

1.6.2.1 Stream values

In Aspen Plus, stream input forms specify a stream's initial value. For example, if you specify that a stream has a temperature of 50°C, the simulation may end with that stream at 100°C without any errors. This is common for tear streams since tear streams are meant to be iteratively written over during convergence; however, other blocks (like **Calculator** blocks) may write over the specifications of a feed stream during convergence.

Though stream values may be written over, they are important and often necessary for convergence. For example, it is important to provide non-zero flow rates for the vapor and liquid going into our **RadFrac** columns (**ABSORBER** and **STRIPPER**); otherwise, the **RadFrac** models may initialize so poorly that the overall simulation fails.

Exact initial estimates are desirable but unnecessary, and qualitatively correct estimates often suffice. Once a flowsheet converges, it can be important to reconcile the calculated tear values for future runs.

To automatically transfer stream results to input after a run, right-click the stream and select "Reconcile". Figure 1.16 shows the resulting window with options for how the results should be transferred.

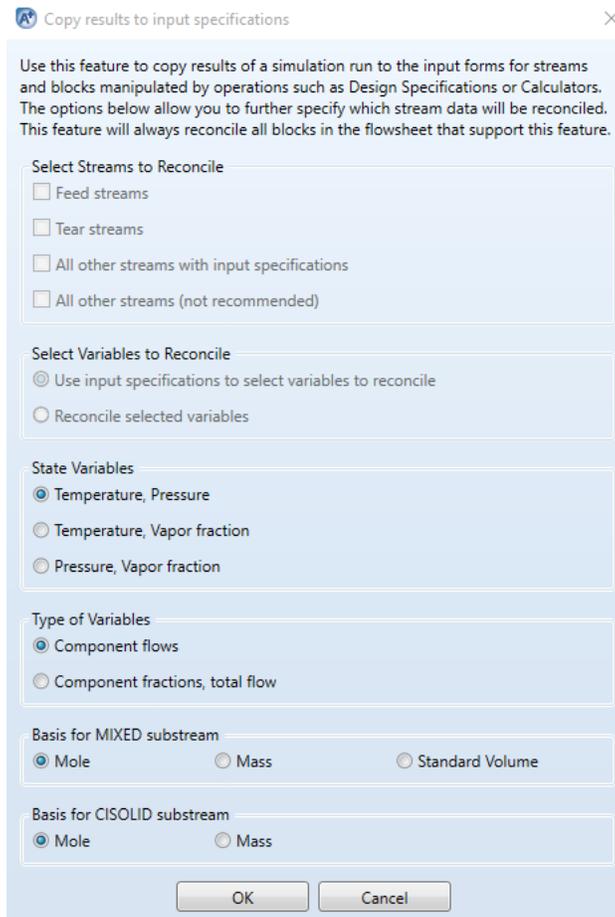


Figure 1.16. Window for selecting stream values to reconcile. This particular screenshot was for a stream that did not already have input values; for streams that do, “Use input specifications to select variables to reconcile” has the already-provided values updated based on the results.

1.6.2.2 RadFrac estimates

RadFrac blocks allow estimates for temperature, profile, liquid flow rate, vapor flow rate, and mole/mass compositions on each stage. Figure 1.17 shows the input form for temperature, along with the tabs for the other parameters.

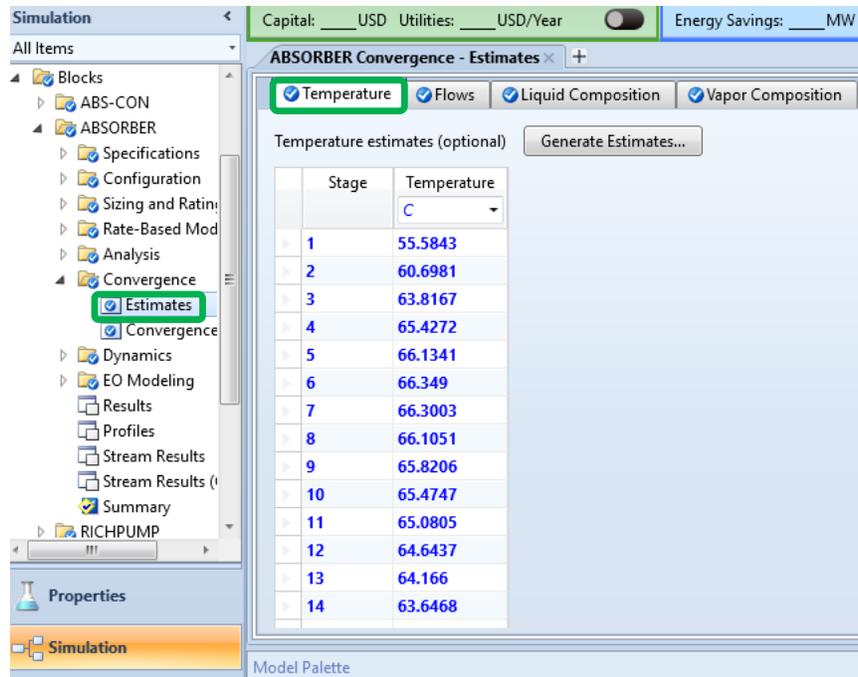


Figure 1.17. Aspen Plus form for estimates. The “Temperature” form is shown, along with the tabs for the “Flows”, “Liquid Composition”, and “Vapor Composition” forms.

In rate-based mode, the holdup specification is also an estimate. Figure 1.18 shows the holdup estimate specification form.

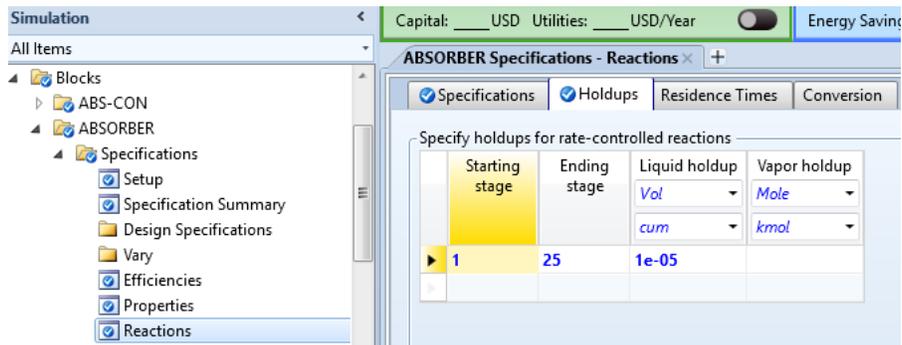


Figure 1.18. Holdup specification form for a **RadFrac** column. The liquid holdup is the amount of liquid considered to be present on a stage for the purpose of calculating reactions, found under Aspen Plus | Simulation | Blocks → [RadFrac] → Specifications → Reactions | “Holdups”. In rate-based mode, the value given here is only an initial estimate.

We have had our best luck reconciling all stages of a column for all values with all available digits. These estimates can significantly improve convergence time and stability.

1.6.3 Best practices in defensive programming

This section focuses on defensive programming. While reading this section, we encourage you to remember that Aspen Plus is a programming platform and that you are acting as a computer programmer. Some of the practices listed here follow from common programming regimes while others are more specific to process flowsheeting.

We focus on the most critical points first.

1.6.3.1 Affected block logic

Aspen Plus's primary solution approach runs flowsheet blocks one-at-a-time. Aspen Plus solves feedback by looping back to earlier units until feedback is solved. This process can cause some blocks to be run many times.

Consider a case in which Aspen Plus has a computationally expensive unit operation, e.g. a rate-based **RadFrac** block, scheduled to run despite:

- a. the block has been previously run;
- b. the block's inputs have not changed since its last run.

Affected block logic is the idea that, in cases like this, we can skip the block's execution because the block's results will not be different from those it found last time. The benefit is reducing simulation time; ideally the results should be the same, just found with less work.

Unfortunately, Aspen Plus's current implementation of affected block logic is incomplete. The current implementation can track changes in stream inputs, but it fails to consider changes to global parameters. It may also fail to consider random numbers (such as in a Monte Carlo method) and external data sources (such as data read from a file). The current implementation considers a block's inputs to be unchanged even when these unconsidered inputs have changed, resulting in Aspen Plus failing to rerun blocks even when the inputs are significantly different.

Aspen Plus enables affected block logic by default. This causes problems in our flowsheets because we make heavy use of global parameters. It is important to disable affected block in all of our simulations

to avoid errors. In general, we recommend disabling affected block logic in all new simulations unless you know that it will not cause errors. Figure 1.19 shows how to disable affected block logic.

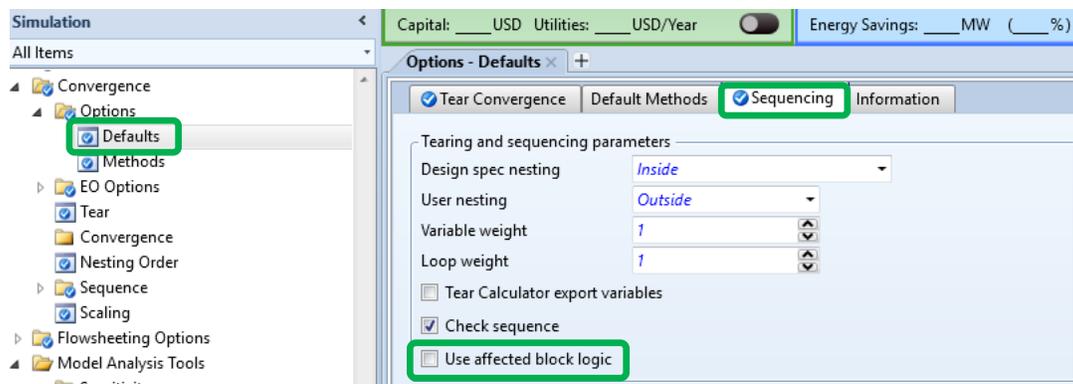


Figure 1.19. Check box to toggle affected block logic. We strongly recommend disabling affected block logic unless you both need the performance boost it might provide and understand the potential errors it may cause.

1.6.3.2 Nesting vs. flat convergence

Our flowsheets include many relationships that need to be converged. In the simplest cases, these problems can be completely isolated from each other and solved separately.

In other cases, problems are interdependent and cannot be separated. For example, we can describe a set of chemical reactions acting in a closed system with extent-of-reaction parameters, ξ_i . Internally, Aspen Plus often solves these problems by minimizing the Gibbs energy by varying the extent-of-reaction parameters ξ_i . The values of ξ_i must be solved together because each affects the optimal value of the others.

Scheme 1.1 shows pseudo-code for a fully nested approach to solving for a set of five chemical reactions at equilibrium by minimizing Gibbs energy. This approach is conceptually simple because the optimizer never has to optimize more than one variable at a time; while tedious, you could easily do this by hand.

```
Guess initial values of  $\{\xi_0, \xi_1, \xi_2, \xi_3, \xi_4\}$ .
```

```
Minimize  $G$  by varying  $\xi_0$ .
```

```
{
```

```

Minimize  $G$  by varying  $\xi_1$ .
{
  Minimize  $G$  by varying  $\xi_2$ .
  {
    Minimize  $G$  by varying  $\xi_3$ .
    {
      Minimize  $G$  by varying  $\xi_4$ .
      {
        Calculate  $G$  based on current values of  $\{\xi_0, \xi_1, \xi_2, \xi_3, \xi_4\}$ .
      }
    }
  }
}

Return the calculated values of  $\{\xi_0, \xi_1, \xi_2, \xi_3, \xi_4\}$ .

```

Scheme 1.1. Nested convergence scheme to find the equilibrium solution for a system of five chemical reactions. Each chemical reaction's extent is described by an extent-of-reaction parameter, ξ_i .

Scheme 1.2 shows pseudo-code for a fully flattened approach to solving the same problem. This approach forces the optimizer to handle the issue of multiple interacting variables. While the optimizer is free to construct a nested approach internally, we generally assume that optimizers employ quasi-Newton methods like Aspen Plus does, e.g. Broyden's method. These methods attempt to modify the values $\{\xi_0, \xi_1, \xi_2, \xi_3, \xi_4\}$ together.

```

Guess initial values of  $\{\xi_0, \xi_1, \xi_2, \xi_3, \xi_4\}$ .

Minimize  $G$  by varying  $\{\xi_0, \xi_1, \xi_2, \xi_3, \xi_4\}$ .
{
  Calculate  $G$  based on current values of  $\{\xi_0, \xi_1, \xi_2, \xi_3, \xi_4\}$ .
}

Return the calculated values of  $\{\xi_0, \xi_1, \xi_2, \xi_3, \xi_4\}$ .

```

Scheme 1.2. Flat convergence scheme to find the equilibrium solution for a system of five chemical reactions. Each chemical reaction's extent is described by an extent-of-reaction parameter, ξ_i .

Usually flat convergence approaches are faster while nested convergence approaches are more reliable. We tend to favor flat convergence approaches when reliability is not an issue. When reliability is an issue, nesting can help provide stability. It is possible to use nesting without using full nesting. Scheme 1.3 shows an example convergence scheme that is neither fully flattened nor fully nested.

```
Guess initial values of  $\{\xi_0, \xi_1, \xi_2, \xi_3, \xi_4\}$ .

Minimize  $G$  by varying  $\{\xi_0, \xi_1, \xi_2\}$ .
{
  Minimize  $G$  by varying  $\{\xi_3, \xi_4\}$ .
  {
    Calculate  $G$  based on current values of  $\{\xi_0, \xi_1, \xi_2, \xi_3, \xi_4\}$ .
  }
}

Return the calculated values of  $\{\xi_0, \xi_1, \xi_2, \xi_3, \xi_4\}$ .
```

Scheme 1.3. Converge scheme to find the equilibrium solution for a system of five chemical reactions. Each chemical reaction's extent is described by an extent-of-reaction parameter, ξ_i .

This convergence scheme is neither fully flat nor fully nested.

If even fully nested convergence approaches result in errors, customized error-handling code can be added to each nested loop to detect and correct problems. Typically we would attempt to implement damping, bounding, and error detection. Exact implementations tend to be specific to the problems encountered, so we will not explore them here.

1.6.3.2.1 Nesting options for automatic sequencing in Aspen Plus

We focus on advanced flowsheet designs that require manual sequencing. However, Aspen Plus does offer a way for users relying on its automatic sequencing algorithm to modify its behavior. Figure 1.20 shows the current set of options.

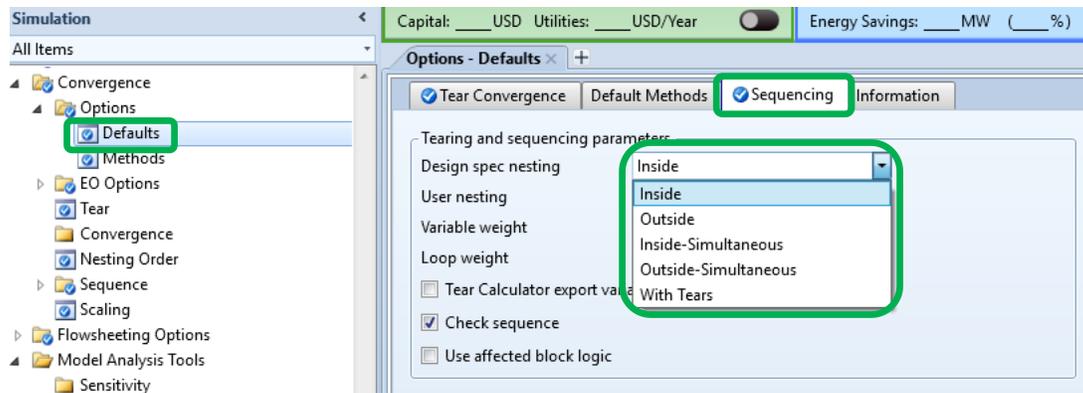


Figure 1.20. Aspen Plus currently provides five options for the behavior of the auto-sequencing algorithm.

“Inside” and “Outside” are highly nested approaches. These options instruct Aspen Plus to converge tears and design specifications in separate loops as we show in Scheme 1.1. “Inside” has design specifications nested inside tears. “Outside” uses the reverse order, nesting tears inside design specifications.

“With Tears” is the flattest approach. This option instructs Aspen Plus to converge all tears and design specifications in a single method as we show in Scheme 1.2.

“Inside-Simultaneous” and “Outside-Simultaneous” are moderate options. They are like “Inside” and “Outside”, except multiple design specifications are flattened into a single loop that is nested with the tears, resulting in a hybrid scheme like that in Scheme 1.3.

1.6.3.3 Object divisions should mirror physical reality

Our flowsheets are constructed as nested hierarchies. Usually we have many different, seemingly reasonable ways to construct the hierarchy. We desire a consistent methodology.

Usually flowsheet designers will tend to group together units that are physically or/and computationally related. For example, most engineers will tend to group the absorber’s packing with the absorber’s condenser rather than the absorber’s packing with the stripper’s condenser; this preferred grouping makes physical sense and is easier to solve.

In more complex flowsheets, we can encounter cases in which there is a tradeoff between physical relationships and computational relationships. When encountered, we stress groupings based on physical relationships even if not computationally optimal. This follows from a desire for modular design: physically-inspired flowsheet hierarchies are easier to modify while computationally-inspired flowsheet hierarchies tend to become entangled, preventing further modification. This trap is an example of *premature optimization*.

1.6.3.4 Disabling HeatX blocks

Figure 1.21 shows how to disable or bypass a **HeatX** block with a check box within Aspen Plus.

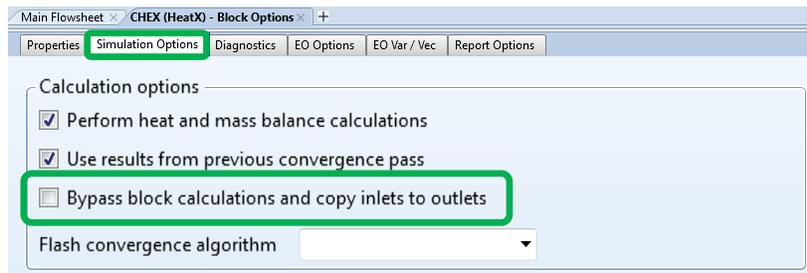


Figure 1.21. Bypass checkbox to disable a **HeatX** block.

This feature is useful in cases where the **HeatX** block would cause simulation errors, most commonly:

1. the assigned heat exchanging surface area A_{CHEX} is zero;
2. the material streams feeding into the cold or/and hot sides have no flow rate.

Then, if used, the **HeatX** block is essentially ignored as though it were hypothetical (Section 1.5: “General approach: Expand the model space and optimize”).

Since we often do not know if one of these conditions will occur in advance, we can set this check box using a **Calculator** block.

First, create a new **Calculator** block. In our simulations we usually prefix these **Calculators** with “C-D-” (the “C” for “Calculator” and the “D” for “Disable”) followed by up to four characters for the corresponding **HeatX** block, e.g. C-D-CHEX. We then define three variables as shown in Figure 1.22.

Variable	Information flow	Definition
ATHRESHD	Import variable	Parameter Parameter no.=90
AREA	Tear variable	Block-Var Block=CHEXMAIN Variable=AREA Sentence=PARAM Units=sqm
DISABLED	Export variable	Block-Var Block=CHEXMAIN Variable=BYPASS Sentence=PARAM

Figure 1.22. Specifications for **C-D-CHEX**.

ATHRESHD, Parameter #90, must be initialized in **C-GLOBAL** to some reasonable threshold, e.g. 10^{-9} m^2 , below which the surface area is considered to be effectively zero for the purposes of disabling the **HeatX**.

Scheme 1.4 shows the interpreted Fortran for the **Calculator** block **C-D-CHEX**.

```

if (AREA .LE. ATHRESHD) then
  DISABLED = 1.0
else
  DISABLED = 0.0
end if

```

Scheme 1.4. Interpreted Fortran for the **HeatX**-disabling **Calculator**, **C-D-CHEX**.

This effectively checks and unchecks the box, as appropriate, whenever the **Calculator** runs. Now we just need to ensure that this **Calculator** always runs *before* its corresponding **HeatX** block.

The surest way to ensure that the **Calculator** is run before its corresponding **HeatX** block is:

1. Declare a **Sequence** block for the **Calculator** and **HeatX**.
2. Replace all references in all other **Sequence** blocks to the **HeatX** with the new **Sequence**.

Conceptually, we are deriving a more automated variant of a **HeatX** by inheriting from **HeatX**.

Hereafter, the actual **HeatX** block is encapsulated by the derived object.

First, create a new **Sequence**, **SQ-DCHEX** as shown in Figure 1.23, and then simply use it in place of the **HeatX** in all other **Sequence** blocks.

	Loop-return	Block type	Block
▶		Calculator	C-D-SHEX
▶		Unit operation	STR-SHEX
▶			

Figure 1.23. **Sequence** specification that instructs Aspen Plus to always run the controlling **Calculator** before its corresponding **HeatX**.

Well-developed flowsheets may have one such **Sequence** per **HeatX** block. When possible, the disabling **Calculator** may also consider zero material flow rates or other erroneous criteria as conditions for disabling the **HeatX**.

1.6.4 Detecting missing tear specifications

Throughout this chapter we have constructed a complex flowsheet with many feedbacks. We execute this model in a sequential manner by iteratively solving for feedbacks with **Convergence** blocks; we call the solved feedbacks “tear variables”. But what happens if we fail to consider a feedback? In other words, what if we omit a tear specification?

While writing Section 2.2: “Stripper interheating”, we initially forgot to include the feedback from the stripper interheater in the list of tear variables for **CV-STR-2**. We quickly realized this omission after running a sensitivity analysis. Figure 1.24 shows the tell-tale jagged response curve.

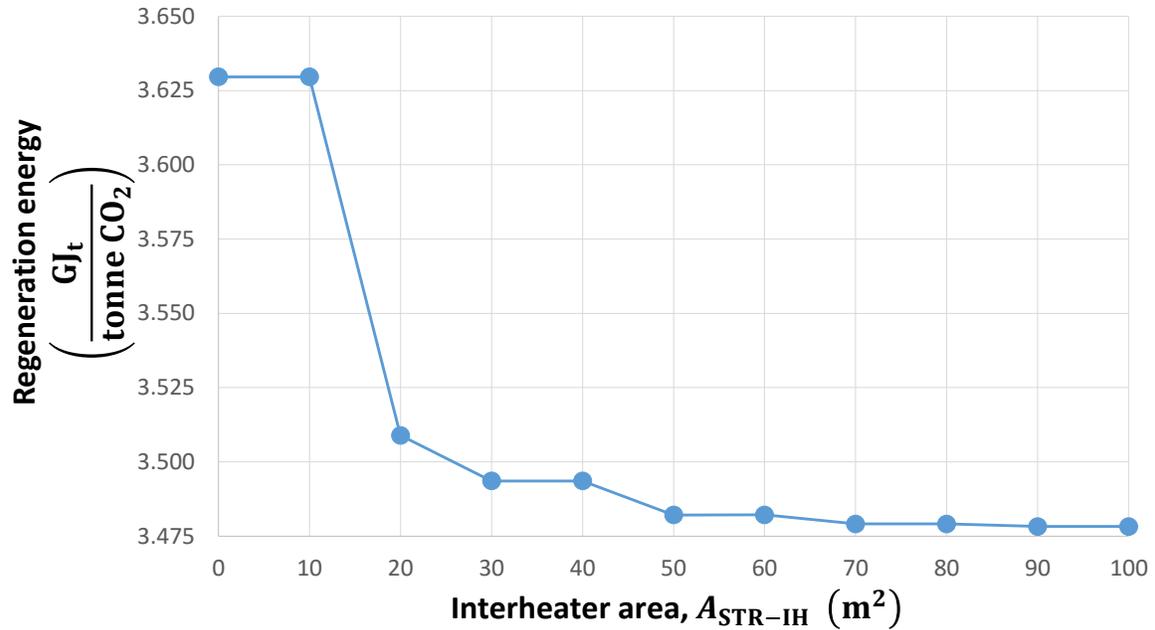


Figure 1.24. Regeneration energy results for a sensitivity analysis on stripper interheating area, A_{STR-IH}/m . The jagged response curve indicates a missing tear specification.

Upon seeing this series, it was immediately obvious that missing tears were the probable cause. The detection criteria are:

1. First value in the series, for the base case, is exactly correct.
2. Later sensitivity results were qualitatively correct.
3. The series is jagged, composed of a point that's correct and then one that's effectively equivalent.

We compare the poorly torn results with the correctly-torn results in Figure 1.25.

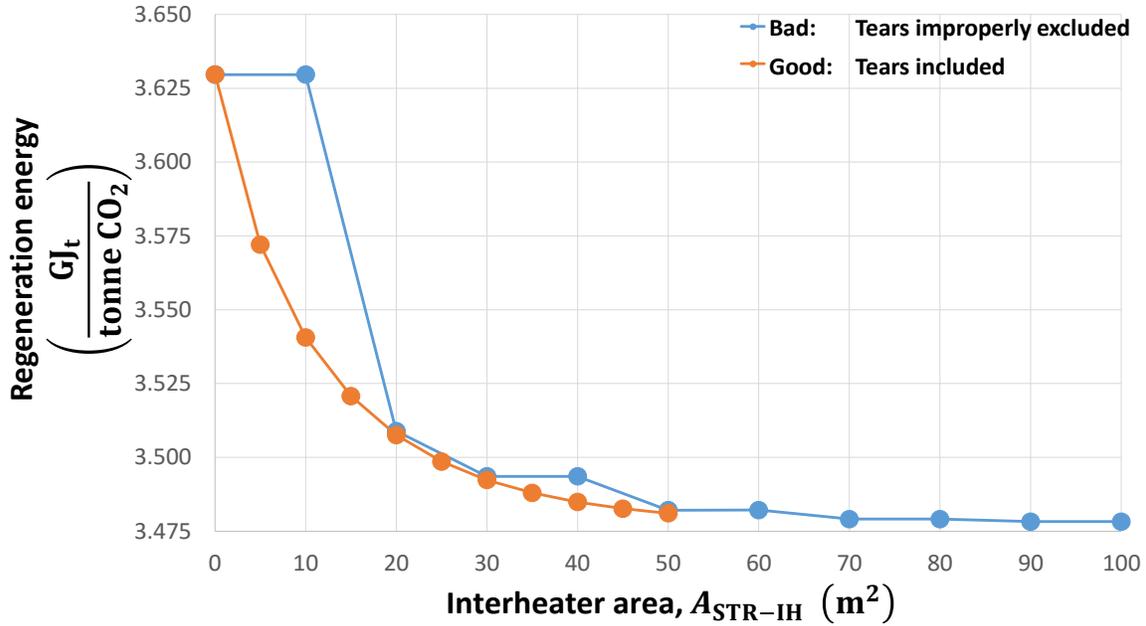


Figure 1.25. Comparison between a correctly torn sensitivity analysis and the poorly torn sensitivity analysis. We can see that the correctly torn curve is much smoother and more consistent.

Some of the major observations are marked in Figure 1.26. The poorly torn analysis shows regular update lags because tear convergence was not checked. The correctly torn analysis did check for tear convergence, performing the additional evaluations necessary to arrive at the correct results.

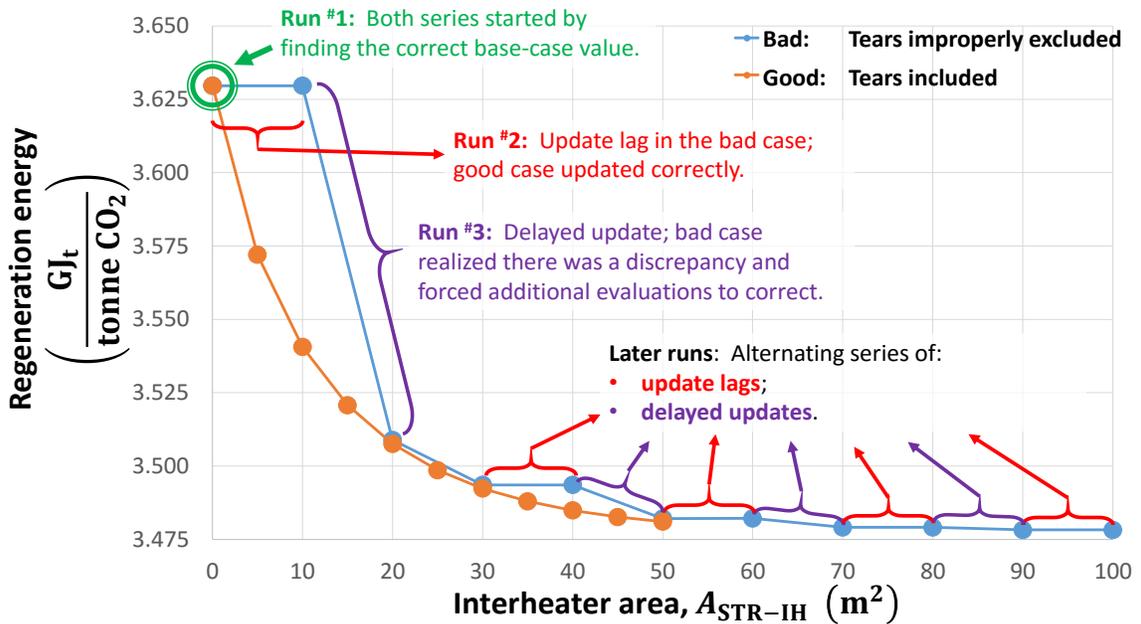


Figure 1.26. Observations on the improperly torn sensitivity analysis results curve.

The jagged corners on this response curve are due to the loss of tearing.

Figure 1.27 displays the same symptom in the sensitivity analysis over interheater draw stage.

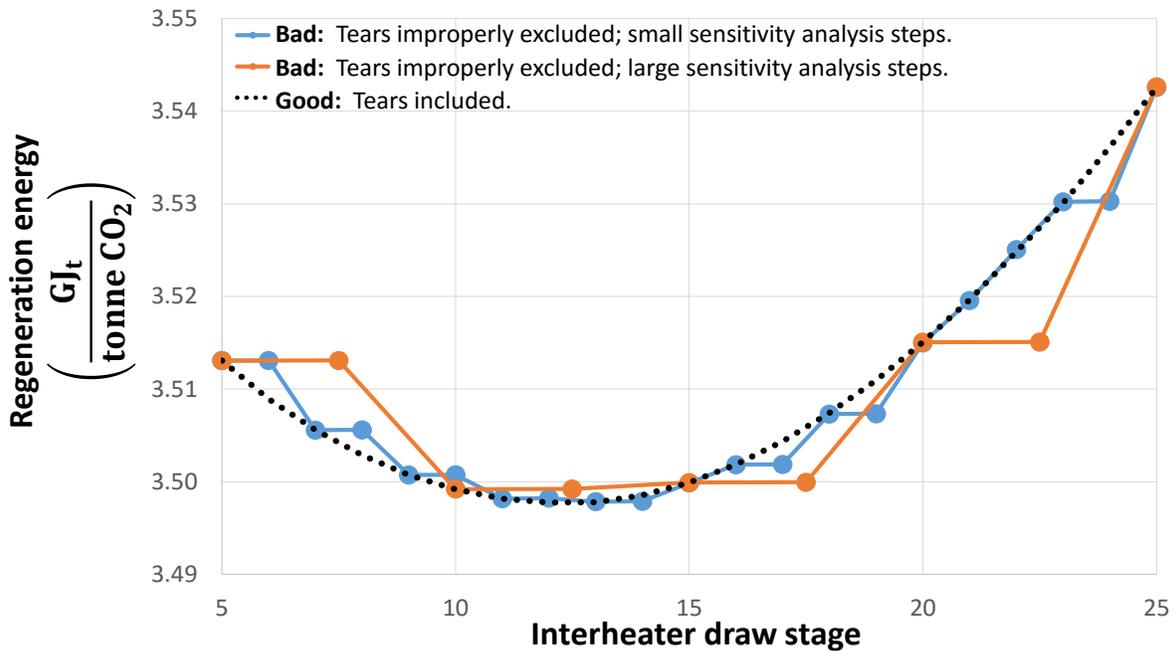


Figure 1.27. Sensitivity analysis results for regeneration energy as a function of the location of the stripper's interheater. We compare a correctly torn analysis with two poorly torn analyses with differing step sizes. The incorrectly torn analysis with larger step sizes is the most jagged.

Figure 1.27 demonstrates another symptom of missing tear specifications: smaller sensitivity analysis steps result in smoother curves. While missing tears cause full updates to be less-than-every-evaluation, updates can still happen regularly. The increased number of updates reduces the noise from lagging updates.

By contrast, we are not plotting the series that includes tears with any smoothing; we merely hide the markers. We can see that the response curve from the teared simulation is very smooth despite having the same resolution as the small-stepping untorn simulation, $1 \frac{\text{run}}{\text{stage}}$.

2 Major schemes

We will discuss common energy-saving schemes by building up a flowsheet demonstrating them. The starting point will be the base case simulation provided in the associated electronic resources.

2.1 Absorber intercooling

Absorber intercooling appears to be the oldest energy-saving scheme, patented in 1931¹⁶ and then again in 1967¹⁷. Absorber intercooling became part of the proprietary Fluor Econamine process between 1992 and 2003.^{18,19} This energy-saving scheme is still widely discussed and deployed today.^{20,21,22} Because absorber intercooling is also very simple, we will explore it as our first energy-saving scheme.

Acid gas absorption is more thermodynamically favored at cooler temperatures. So, if the solvent exiting the bottom of the absorber is near its maximum loading, we can drive further absorption by lowering the temperature. This energy-saving scheme calls for a side cooler (“intercooler”) on the absorption column to help increase the rich solvent loading.

Absorber intercooling has several apparent drawbacks:

1. Capital cost of the absorber intercooler.
2. Utility cost for cooling water to power the absorber intercooler.
3. The rich solvent is colder, reducing the heat that would have otherwise gone into the stripper.
4. Cooling the solvent slows reaction kinetics, potentially slowing absorption in solvents that are sufficiently close to equilibrium capacity.

2.1.1 Modeling approach for absorber intercooling

A somewhat rigorous approach would be to:

1. Draw material pseudo-streams from the absorption tower.
2. Cool the pseudo-streams using utility cooling water streams.
3. Sequence a **Calculator** block to run after the **HeatX** block representing the absorber’s intercooler. Have this **Calculator** block read the calculated heat duty, D_{ABS-IC} , and set it to the stage with the intercooler.
4. Optimize:
 - a. the location of the intercooler along the column;

- b. the heat exchanging surface area of the cross heat exchanger;
- c. the portion of duty offset from the lean stream cooler.

This approach might be appropriate for a more detailed analysis, though for now we will stick with a simpler approach:

1. Feed a heat stream into the absorber.
2. Optimize:
 - a. the location of the intercooler along the column;
 - b. the heat duty of the side heat stream;
 - c. the portion of duty offset from the lean stream cooler.

Additionally, we will use interpolated stage numbers as discussed in Section 1.6.1: “Interpolating non-integer values”.

2.1.2 Flowsheet construction

Figure 2.1 shows the base-case flowsheet provided in the electronic resources.

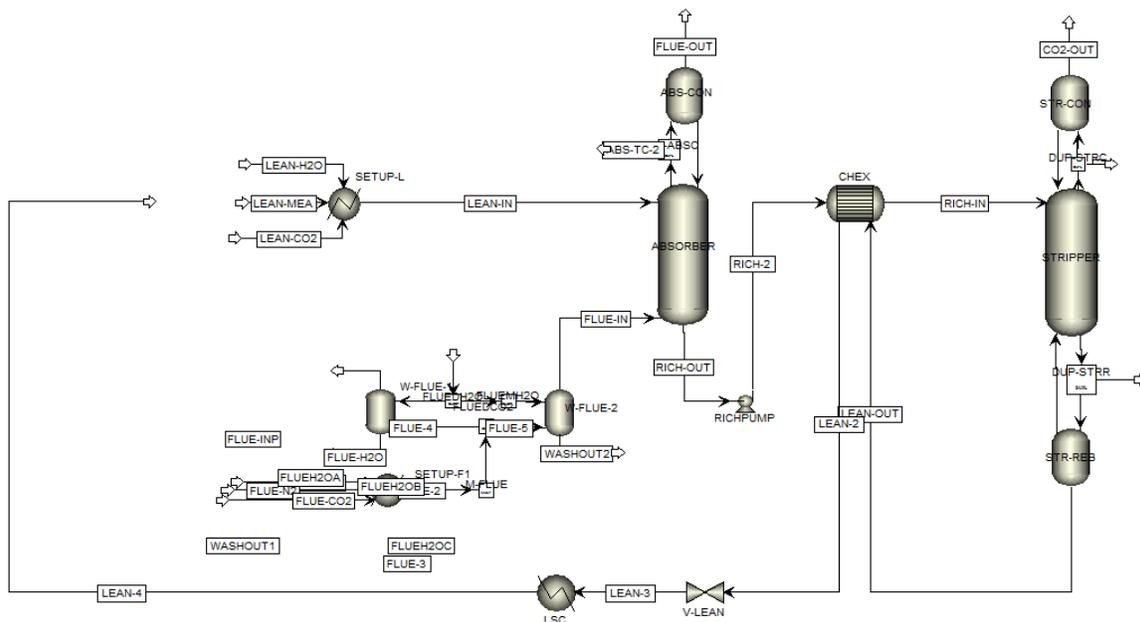


Figure 2.1. Prior flowsheet for the baseline acid-gas-capture unit.

First, define three new global parameters in **C-GLOBAL** as shown in Figure 2.2 and Table 2.1.

ABSINTCS	Export variable	Parameter Parameter no.=20050
ABSINTCD	Export variable	Parameter Parameter no.=20051
ABSICMOD	Export variable	Parameter Parameter no.=20060

Figure 2.2. Parameters for absorber intercooling to be defined in C-GLOBAL.

Table 2.1. Explanations for the three absorber intercooling parameters.

Fortran	Meaning	Direction	Definition	Explanation
ABSINTCS	<u>A</u> BSorber <u>I</u> NTer- <u>C</u> ooling <u>S</u> tage	Export	Parameter #20,050	Stage number for the absorber intercooler. Non-integer between 1 and $N_{\text{ABS stage}} - 1$.
ABSINTCD	<u>A</u> BSorber <u>I</u> NTer- <u>C</u> ooling <u>D</u> uty	Export	Parameter #20,051	Duty for the absorber intercooler. Usually should be negative; positive values represent heating.
ABSICMOD	<u>A</u> BSorber <u>I</u> nter- <u>C</u> ooling <u>M</u> ODE	Export	Parameter #20,060	Mode of operation.

Next, amend the interpreted Fortran in Scheme 2.1 to the end of C-GLOBAL.

```

C =====
C ===== START:  ABSORBER INTERCOOLING =====
C =====

```

```

C Absorber intercooling stage (non-integer values
C are valid; interpolated):
      ABSINTCS = 1

C Absorber intercooling duty (MW):
      ABSINTCD = 0.0

C Absorber intercooling modes:
C   1 --> Duty only sets side cooler.
C   2 --> Duty is counter-balanced on LSC.
      ABSICMOD = 2.0

C =====
C =====      END:  ABSORBER INTERCOOLING  =====
C =====

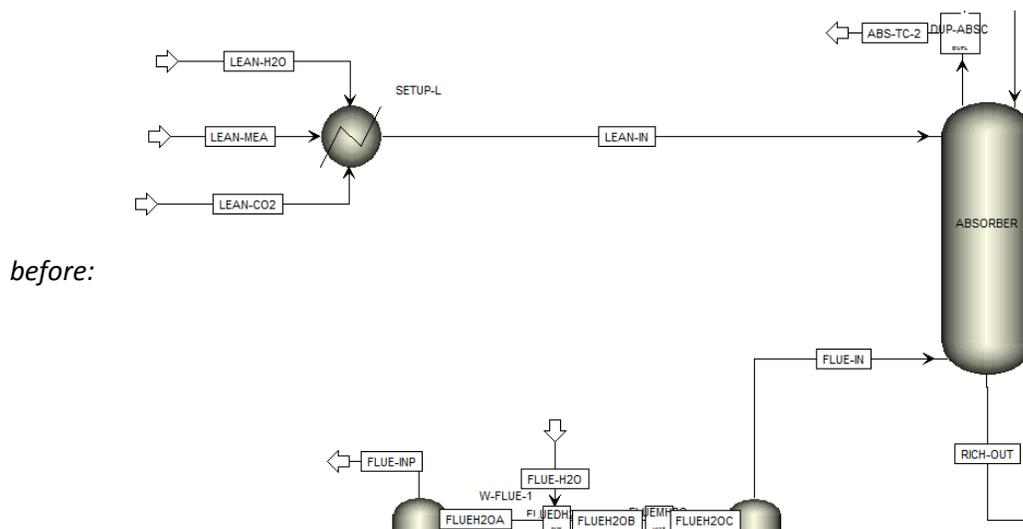
```

Scheme 2.1. Interpreted Fortran to be inserted into C-GLOBAL to specify absorber intercooling.

Next, add:

1. Two heat streams going into **ABSORBER**: **ABS-IC-1** and **ABS-IC-2**.
2. A **Heater** block **ABS-IC-C** and a material stream **P-LEANIN**.

Figure 2.3 shows the flowsheet layouts before and after implementing absorber intercooling.



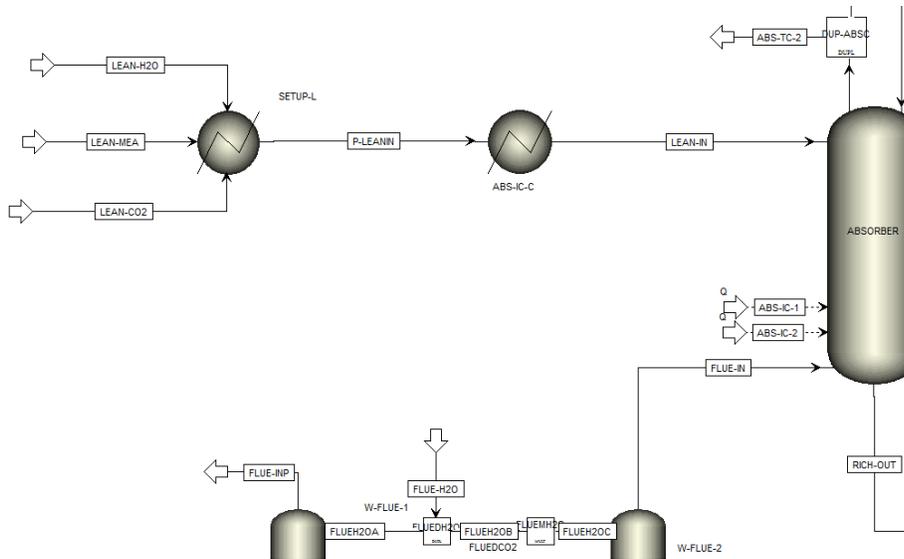


Figure 2.3. Flowsheet modification to implement absorber intercooling. Insert heat streams **ABS-IC-1** and **ABS-IC-2** as feeds into **ABSORBER**.

Then specify the flowsheet inputs:

1. **ABS-IC-C**
 - a. Pressure: 0 bar
 - b. Duty: 0 MW
2. **ABS-IC-1**
 - a. Duty: 0 MW
3. **ABS-IC-2**
 - a. Duty: 0 MW
4. **ABSORBER**
 - a. Configuration → Heats and Coolers → Heat Streams
 - i. **ABS-IC-1**: Stage #1
 - ii. **ABS-IC-2**: Stage #2

Note that these three flowsheet elements (**ABS-IC-C**, **ABS-IC-1**, and **ABS-IC-2**) have no effect on the flowsheet with the above input specifications:

- **ABS-IC-C** has no effect because it is a **Heater** with no heat exchanger and no pressure drop.
- **ABS-IC-1** and **ABS-IC-2** have no effect because they are empty heat streams.

This transparency satisfies our design rule from Section 1.5 that flowsheet modifications must contain the base case.

Next, create a new Calculator called **C-ABS-IC**. Define the variables in Figure 2.4.

Variable	Information flow	Definition
ABSINTCS	Import variable	Parameter Parameter no.=20050
ABSINTCD	Import variable	Parameter Parameter no.=20051
ABSICMOD	Import variable	Parameter Parameter no.=20060
STAGEH1D	Export variable	Heat-Duty Stream=ABS-IC-1 Units=MW
STAGEH2D	Export variable	Heat-Duty Stream=ABS-IC-2 Units=MW
STAGEH1L	Export variable	Block-Var Block=ABSORBER Variable=H-FEED-STAGE Sentence=H-FEEDS ID1=ABS-IC-1
STAGEH2L	Export variable	Block-Var Block=ABSORBER Variable=H-FEED-STAGE Sentence=H-FEEDS ID1=ABS-IC-2
SDLSC1	Export variable	Block-Var Block=ABS-IC-C Variable=DUTY Sentence=PARAM Units=MW

Figure 2.4. Variable definitions for **C-ABS-IC**.

Define the code for **C-ABS-IC** as given in Scheme 2.2.

```

C =====
C === PART 1: Bound the stage specification. ===
C =====
C Bound the target stage between the minimum
C and maximum valid stage numbers.
C   if (ABSINTCS .LT. 1.0) then
C     ZSET = 1.0
C   else if (ABSINTCS .GT. 29.0) then
C     ZSET = 29.0
C   else
C     ZSET = ABSINTCS
C   end if
C
C =====
C === PART 2: Interpolate the stage specification. ===
C =====
C Is the stage specification an integer?
C   ZSETINT = INT(ZSET)

```

```

    if (ZSET .EQ. ZSETINT) then

C   Yes, the stage specification is an integer.
C
C   The effective target stage is an integer.
C   So, first feed heat stream targets it with the full
C   intercooling duty while the second heat stream gets
C   zero duty:
        STAGEH1D = ABSINTCD
        STAGEH2D = 0.0
        STAGEH1L = ZSETINT

C   The second stage has a duty of zero, so its stage
C   would be irrelevant, except Aspen Plus won't allow
C   two heat streams to feed into the same stage. So,
C   we arbitrarily feed it into Stage #1 unless that's
C   the stage that the first stream happens to be feeding
C   into, in which case we feed into Stage #2.
        if (ZSETINT .EQ. 1.0) then
            STAGEH2L = 2.0
        else
            STAGEH2L = 1.0
        end if

    else

C   No, the stage specification isn't an integer.
C
C   So, we target the consecutive integers:
        ZROUND = ZSETINT
        if (ZROUND .GE. ZSET) then
            ZROUND = ZROUND - 1.0
        end if
        STAGEH1L = ZROUND
        STAGEH2L = ZROUND + 1.0

C   And now we round the duty:
        ZSPLTTO2 = ZSET - ZROUND

```

```

        STAGEH1D = (1.0 - ZSPLTTO2) * ABSINTCD
        STAGEH2D = ZSPLTTO2 * ABSINTCD

    end if

C =====
C ===  PART 3:  Counter intercooling duty,          ===
C ===          if appropriate.                    ===
C =====
C If set by the option parameter, counter the
C absorber intercooling duty on the lean stream
C cooler (LSC).  The idea here is that the ABS-IC's
C duty isn't added so much as reallocated from
C the duty usually applied in the LSC.

        if ((ABSICMOD .GT. 1.5) .AND. (ABSICMOD .LT. 2.5)) then
            SDLSC1 = -(STAGEH1D + STAGEH2D)
        else
            SDLSC1 = 0.0
        end if

```

Scheme 2.2. Interpreted Fortran for **C-ABS-IC**.

Next, we need to sequence these new flowsheet elements so that they execute at the proper times.

Modify **SQ-ABS-3** as shown in Figure 2.5 by adding the entry for **ABS-IC-C** after **SETUP-L**.

	Loop-return	Block type	Block
▶		Calculator	C-I-ABS3
▶	Begin	Convergence	CV-ABS-3
▶		Calculator	C-LEAN
▶		Calculator	C-LEAN-T
▶		Unit operation	SETUP-L
▶		Unit operation	ABS-IC-C
▶		Sequence	SQ-ABS-2
▶	Return to	Convergence	CV-ABS-3
▶			

Figure 2.5. Modification for **SQ-ABS-3** to run the absorber intercooling offset **Heater**, **ABS-IC-C**, between the lean-stream feed and absorber complex.

Create a new **Sequence**, **SQ-PRE**, and define it as in Figure 2.6.

	Loop-return	Block type	Block
▶		Calculator	C-ABS-IC
▶			

Figure 2.6. Definition for **SQ-PRE**.

Modify **SQ-DATA** as shown in Figure 2.7.

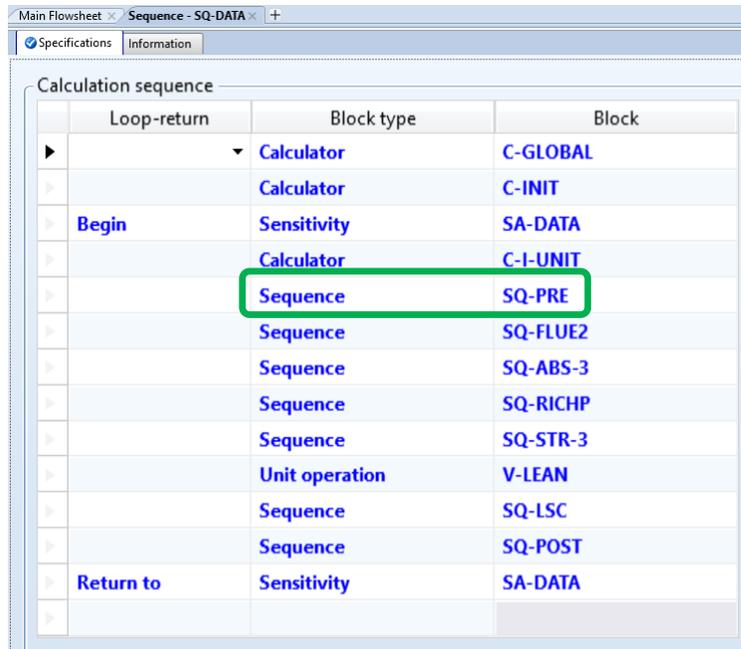


Figure 2.7. Insert SQ-PRE into SQ-DATA.

Next, we will add cases to SA-DATA so that we can perform sensitivity analyses for different absorber intercooling configurations as shown in Figure 2.8.

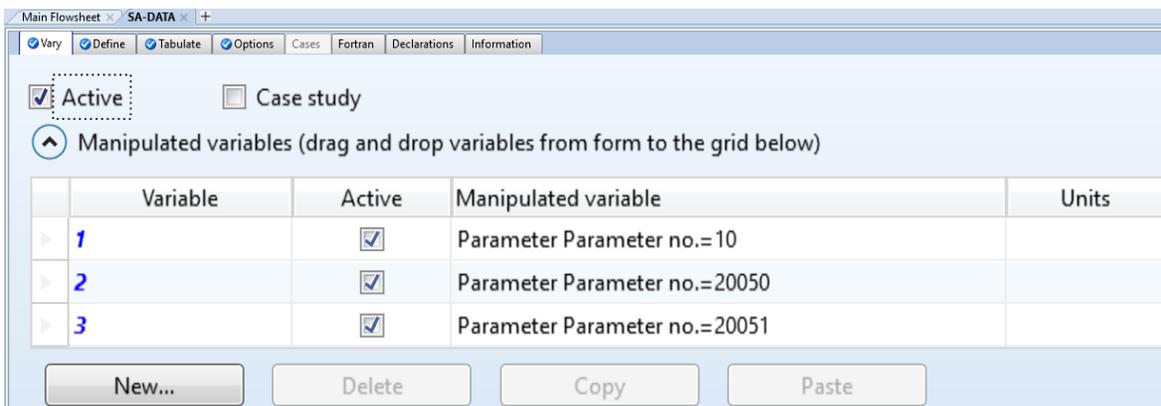


Figure 2.8. New cases for SA-DATA.

For now, we may vary the new parameters (#20,050 and #20,051) over harmless values. The stage number may be set to 1 under "List of values" while the duty can be set to zero under its list of values.

Finally, we want to update the tabulated data to include the new global parameters. It is usually best to simply clear and recreate the list to avoid any potential inconsistency.

1. Go to Aspen Plus | Simulation | Model Analysis Tools → Sensitivity → SA-DATA → Input | “Define” tab.
2. Select the entire list and delete it.
3. Repeat this deletion for the rows in the “Tabulate” tab.
4. Go to Aspen Plus | Simulation | Flowsheeting Options → Calculator → C-GLOBAL → Input | “Define” tab.
5. Select entire list and copy.
6. Return to **SA-DATA**’s defined variable list and paste the copied values.
7. Return to the “Tabulate” tab and click the “Fill variables” button.

2.1.3 Sensitivity analyses

Now that our model includes absorber intercooling, we can perform sensitivity analyses on this energy-saving scheme.

For the first analysis, we vary the absorber intercooling duty from 0 MW (as base case) to -0.05 MW, finding the results in Figure 2.9. This analysis is in Mode #1 which means that the absorber intercooling duty does not offset the lean stream cooler duty.

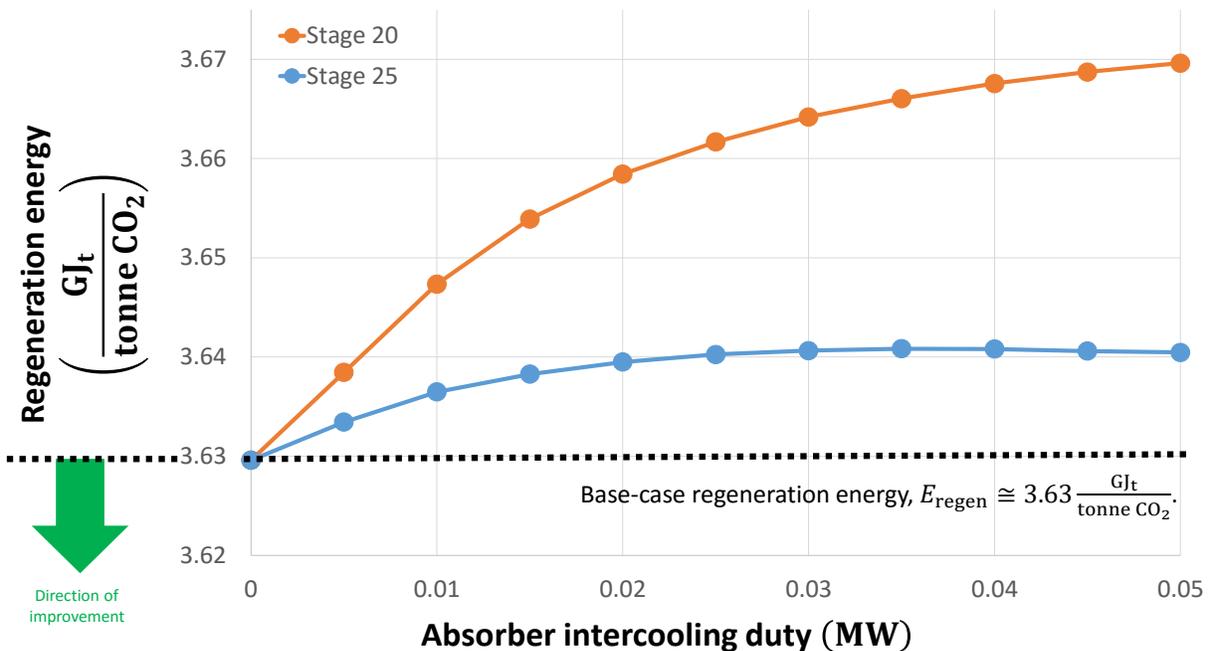


Figure 2.9. Regeneration energy for various absorber intercooling duties on either Stage 20 or Stage 25.

The results show that these particular parameters actually hurt energy efficiency. Examining the flowsheet, we see that absorber intercooling is successful in increasing the rich solvent loading L_{rich} , however the loss of sensible heat before entering the stripper causes more damage than the increased rich solvent loading has helped.

For the next analysis, we try intercooling with a duty $D_{ABS-IC} = -0.025$ MW from Stage 20 to Stage 29, finding the response in Figure 2.10. We plot the results of this analysis for both Mode #1 and Mode #2. In Mode #2, the absorber intercooling duty does offset the lean stream cooler duty.

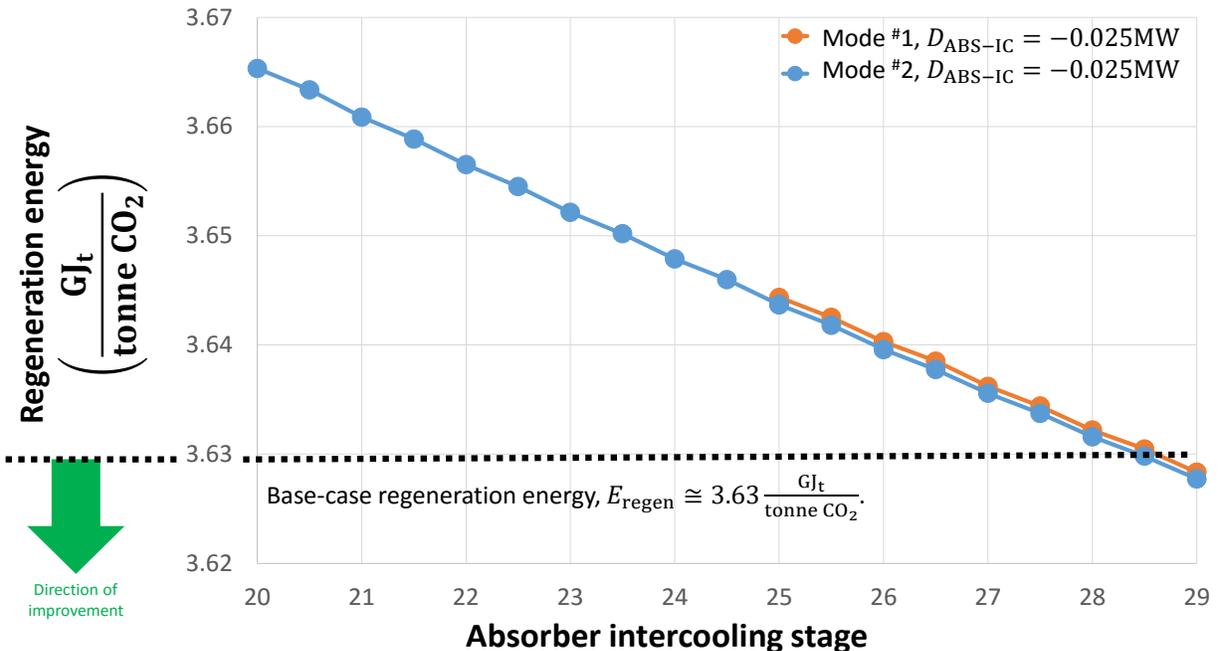


Figure 2.10. Results for a sensitivity analysis varying the absorber intercooler's location over the column. We find poor results for all but the lowest position, at which regeneration energy is just slightly better than in the base case.

We can see that Mode #2 performs a tad better than Mode #1. In a rigorous optimization, we would include the lean stream cooler duty D_{LSC} , though for now the larger effect appears to be from D_{ABS-IC} .

The correlations between the regeneration energy consumption and the absorber intercooling stage were extremely linear. For Mode #2:

$$\frac{E_{\text{regen}}}{\text{GJ}_t/\text{tonne}} \cong 3.7487 - 0.00419n_{\text{stage}_{\text{ABS-IC}}} \quad 2.1$$

Further, we see that absorber intercooling successfully reduces the required regeneration energy when the intercooler is placed low enough, e.g. at Stage 29.

Because we find that Stage 29 has a better response than either Stage 20 or 25, we amend our first sensitivity analysis as shown in Figure 2.11.

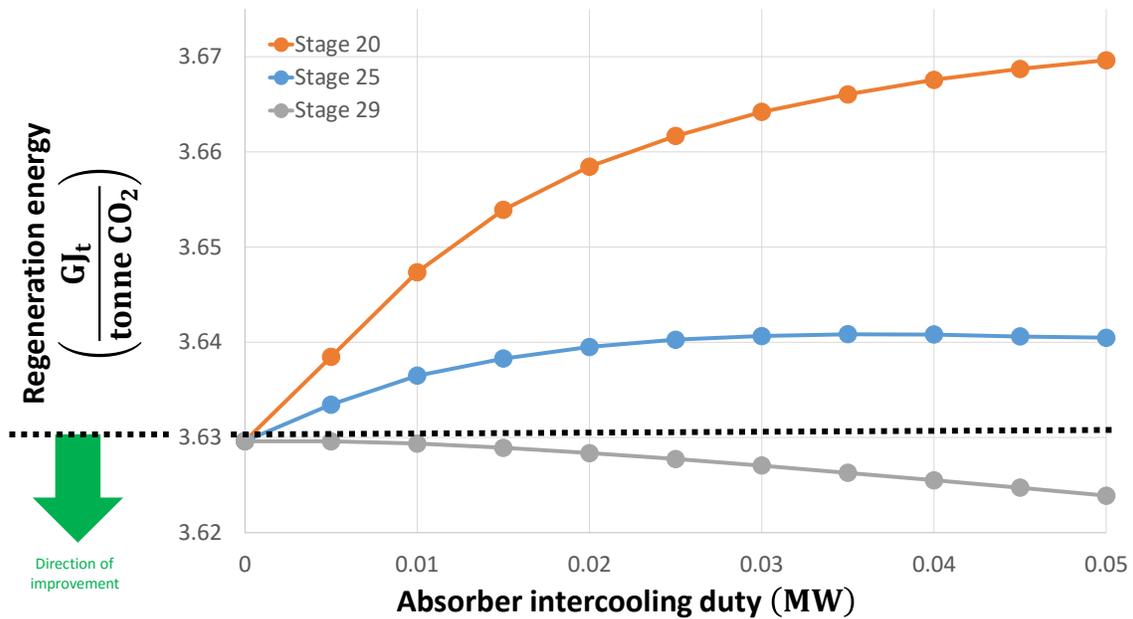


Figure 2.11. Response curves including Stage 29. The response for Stage 29 is modest but it does outperform the base case in terms of regeneration energy.

Checking the temperature profile for the **ABSORBER** as shown in Figure 2.12, we can see that the intercooler at Stage 29 causes the stage temperature to be approximately 30°C. This results in minor numerical errors due to incomplete convergence; the **RADFRAC** model begins to have trouble with such sharp dips. Also, because utility water is often around 20°C, we are nearing the $\Delta T_{\text{approach}} \cong 10^\circ\text{C}$ limit for a reasonably sized intercooler. We note that the small absorber intercooling effect makes it implausible to suggest a larger capital investment.

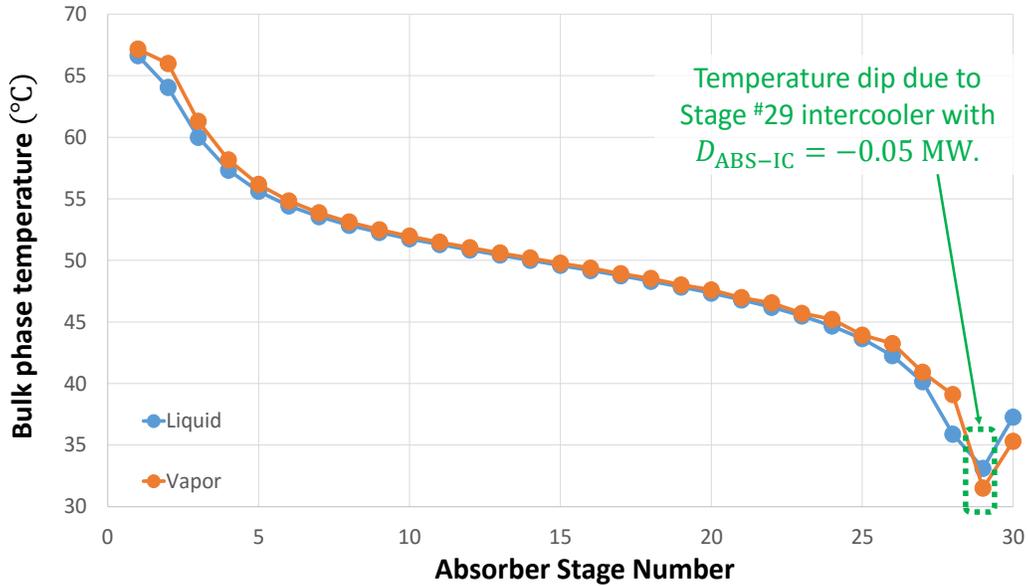


Figure 2.12. Temperature profile for the bulk liquid and bulk vapor phases.

These results are specific to this exact flowsheet and do not necessarily represent a general result. In practice, factors ranging from solvent selection to flue gas composition to other process design selections can fundamentally change the response curves.

2.1.4 Remarks

Numeric errors introduce noise into optimization algorithms, so it is important to have consistent, reliable, precise, accurate models. This reliability is apparent in the sensitivity analyses.

2.2 Stripper interheating

Absorber intercooling used utility cooling water to power a side cooler on the absorber. Stripper interheating uses heat recovered from elsewhere in the process to power a side heater on the stripper. Stripper interheating is a direct application of heat integration (Section 1.5.1).

Stripper interheating is fairly unique among the energy-saving schemes that we have investigated in that it is almost always has a positive impact on energy saving while most other schemes are situational. However, its inclusion does not necessarily have a positive impact on net present value (NPV) due to the associated capital cost.

The first found appearance of stripper interheating comes from a 1982 alkylation process patent.²³ A second appearance in 1986 was reported²⁴ by Leites *et al.* in 2013, though we have not been able to confirm with the original source. This second appearance built the interheater into the column as opposed to attaching it to the side; this modification was termed “internal exchange” by Oyenekan and Rochelle in 2006.^{12,11}

2.2.1 Modeling approach for stripper interheating

When we modeled absorber intercooling, we selected the intercooler duty $D_{\text{ABS-IC}}$ largely because this was a quick, easy solution. Stripper interheating is more prone to complex behavior since it draws heating from within the process (as opposed from utilities like absorber intercooling does). Since we cannot simply use $D_{\text{STR-IH}}$, we instead select the heat-exchanging surface area $A_{\text{STR-IH}}$. We use a **HeatX** block to model this heat-exchanging surface area $A_{\text{STR-IH}}$.

Our approach will be:

1. Draw a material pseudo-stream from the **STRIPPER**.
2. Run the pseudo-stream through a **HeatX** with area $A_{\text{STR-IH}}$ to exchange heat with the stripper’s bottom stream.
3. Converge the feedback between the interheater and the packed column.

We have to make a judgement call on convergence logic. A more cautious approach would involve nesting the interheater in an inner loop with the **STRIPPER**; however, this approach would increase the computational cost of the stripping complex by a factor roughly equal to the average number of iterations that it would take to converge this inner loop. It is potentially faster, though less stable, to converge the interheater along with the rest of the stripping complex.

For the sake of expediency, we will combine the interheater’s convergence with the cross heat exchanger **CHEX**’s. This approach was often too noisy for reliable convergence in earlier versions of Aspen Plus, though Aspen Plus V8.8 appears to be significantly more stable than pre-V8 versions.

In practice, you may sometimes wish to try either more cautious or more expedient convergence designs, then modify them later as the need arises. For example, we may move toward the more stable version if this approach proves too unreliable.

2.2.2 Flowsheet construction

We will pick up with the flowsheet that we added absorber intercooling to in the prior section.

First, we will draw the addition units onto the flowsheet as shown in Figure 2.13, including:

1. **STR-SHM1** and **STR-SHM2**
 - a. **Mult** blocks with a factor set to 1.
2. **STR-SHMX**
 - a. **Heater** block with $\Delta P = 0$ and $D = 0$.
3. **STR-SHEX**
 - a. **HeatX** block with a heat-exchanging surface area of $A_{STR-SH} = 0$.
4. **STR-SH-1** and **STR-SH-2**
 - a. Material pseudo-streams from **STRIPPER**
5. **STR-HIN1** and **STR-HI2**
 - a. Heat streams feeding into **STRIPPER** at Stages #1 and #2, respectively.

Also, we draw the material streams **STR-SH-1** through **STR-SH-5** and **STR-SH-R**, drawn as shown in Figure 2.13.

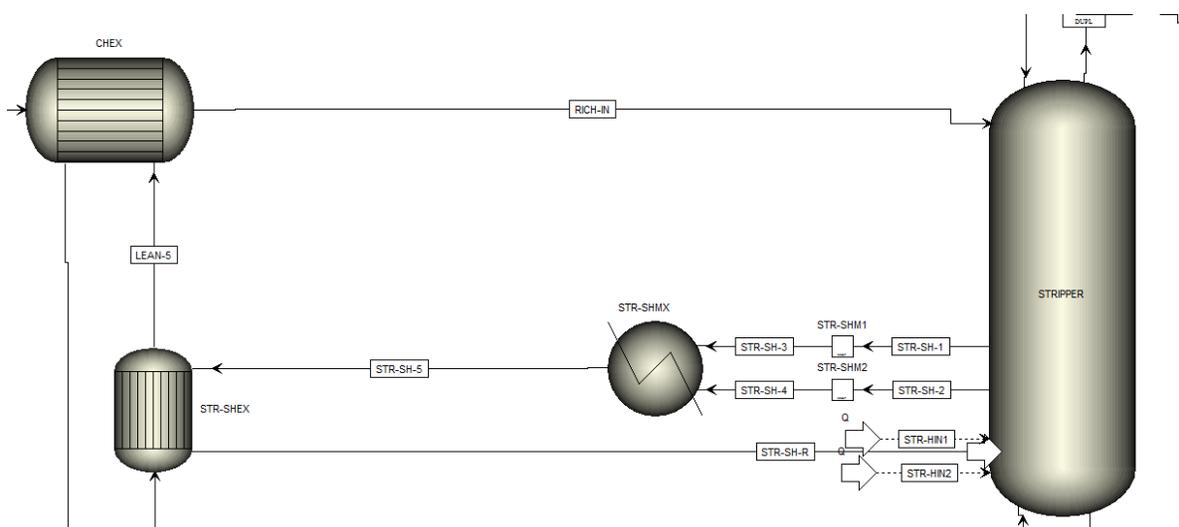


Figure 2.13. Flowsheet modifications for stripper interheating.

Note that the return stream, **STR-SH-R** (“R” for “Return”), does not actually feed back into **STRIPPER**. Because the side draws are pseudo-streams, returning their material content to the process would create a mass imbalance. We draw **STR-SH-R** as though it returns to **STRIPPER** to represent the concept of the design.

While the material in **STR-SH-R** is not computationally returned to the **STRIPPER**, the heat picked up in **STR-SHEX** will be transferred back into **STRIPPER** by a **Calculator** that sets the heat streams **STR-HIN1** and **STR-HIN2**. We note that the two heat streams are conceptually a single heat stream interpolated to allow non-integer stage numbers.

Next, we note the errors Aspen Plus runs into with some **HeatX** specifications, so we employ the solution described in Section 1.6.3.4: “Disabling HeatX blocks”. We used the names:

- **C-D-SHEX** for the disabling **Calculator**;
- **SQ-DSHEX** for the derived-object **Sequence**.

We amend two variables to **C-GLOBAL**:

1. **STRINTHS** as Parameter #20,100;
2. **STRINTHA** as Parameter #20,101.

Initialize these parameters at the end of **C-GLOBAL**’s interpreted Fortran using the code shown in Scheme 2.3.

```
C =====  
C ===== START: STRIPPER INTERHEATING =====  
C =====  
  
C Stripper interheating stage (non-integer values  
C are valid; interpolated):  
    STRINTHS = 1  
  
C Stripper interheating heat-exchanging surface  
C area (m^2; non-negative values):  
    STRINTHA = 0.0  
  
C =====
```

```

C ===== END: STRIPPER INTERHEATING =====
C =====

```

Scheme 2.3. Parameter initializations to amend to the end of **C-GLOBAL**.

Next we create two new **Calculator** blocks:

1. **C-STRIH** handles the stripper interheating draws.
2. **C-STRIDR** handles the stripper interheating return.

The code's logic will largely mirror that in **C-ABS-IC** for absorber intercooling, though we have a complication to avoid feedback: the returned heat must be delivered two stages beneath where it is drawn. This approximation is imperfect, though it avoids issues such as the partial draw from Stage #25 getting direct feedback from the heat return to Stage #25 that we could see when $n_{\text{stage}_{\text{STR-IH}}} = 24.5$ if the heat was returned only one stage below. The undesired offset should diminish as the stage count for **STRIPPER**, N_{STR} , increases.

We define the variables for **C-STRIH** as shown in Figure 2.14.

Variable	Information flow	Definition
STRINTHS	Import variable	Parameter Parameter no.=20100
STRINTHA	Import variable	Parameter Parameter no.=20101
FDRAW1	Export variable	Block-Var Block=STR-SHM1 Variable=FACTOR Sentence=PARAM
FDRAW2	Export variable	Block-Var Block=STR-SHM2 Variable=FACTOR Sentence=PARAM
NDRAW1	Export variable	Block-Var Block=STRIPPER Variable=PSEUDO-STAGE Sentence=PSEUDO-STREA ID1=STR-SH-1
NDRAW2	Export variable	Block-Var Block=STRIPPER Variable=PSEUDO-STAGE Sentence=PSEUDO-STREA ID1=STR-SH-2
AREA	Export variable	Block-Var Block=STR-SHEX Variable=AREA Sentence=PARAM Units=sqm

Figure 2.14. Variable definitions for **C-STRIH**.

We want to:

1. Bound, interpolate, and then set the stage specification for the draw for the interheater.
2. Set the heat-exchanging surface area specification for the interheater, $A_{\text{STR-IH}}$.

We cannot yet read and transfer the calculated heat transfer, $Q_{\text{STR-IH}}$, because we first need to run **STR-SHEX**. We give this task to the second **Calculator**, **C-STRIDR**.

Scheme 2.4 specifies the interpreted Fortran for C-STRIHHR.

```
C =====
C === PART 1: Bound the stage specification.      ===
C =====
C Bound the target stage between the minimum
C and maximum valid stage numbers.
C   if (STRINTHS .LT. 1.0) then
C     ZSET = 1.0
C   else if (STRINTHS .GT. 28.0) then
C     ZSET = 28.0
C   else
C     ZSET = STRINTHS
C   end if
C
C =====
C === PART 2: Interpolate the stage specification.  ===
C =====
C Is the stage specification an integer?
C   ZSETINT = INT(ZSET)
C   if (ZSET .EQ. ZSETINT) then
C
C Yes, the stage specification is an integer.
C
C The effective draw stage is an integer.
C So, first draw multiplier works to full effect,
C 100%, while the second is entirely disabled.
C   FDRAW1 = 1.0
C   FDRAW2 = 0.0
C   NDRAW1 = ZSETINT
C
C The second draw stage is disabled, but it
C still can't match the first draw stage's source.
C We arbitrarily draw from Stage #1 unless the first
C draw is already at Stage #1, in which case we draw
```

```

C from Stage #2 to avoid the collision.
    if (ZSETINT .EQ. 1.0) then
        NDRAW2 = 2.0
    else
        NDRAW2 = 1.0
    end if

    else

C No, the stage specification isn't an integer.
C
C So, we target the consecutive integers:
    ZROUND = ZSETINT
    if (ZROUND .GE. ZSET) then
        ZROUND = ZROUND - 1.0
    end if
    NDRAW1 = ZROUND
    NDRAW2 = ZROUND + 1.0

C And now we round the duty:
    ZSPLTTO2 = ZSET - ZROUND

    FDRAW1 = 1.0 - ZSPLTTO2
    FDRAW2 = ZSPLTTO2

    end if

C =====
C === PART 3: Set HeatX surface area. ===
C =====

    AREA = STRINTHA

```

Scheme 2.4. Interpreted Fortran for C-STR1HD.

Next, we need to sequence our new blocks. First, the amend **Calculator C-STR1HD** to the end of the preparation **Sequence, SQ-PRE**.

Next, we create a new **Sequence**, **SQ-STRIH**, as shown in Figure 2.15.

Calculation sequence		
	Block type	Block
▶	Unit operation	STR-SHM1
▶	Unit operation	STR-SHM2
▶	Unit operation	STR-SHMX
▶	Unit operation	STR-SHEX
▶	Calculator	C-STRIHR

Figure 2.15. Definition for **SQ-STRIH**.

Considering the interheating complex's location on the flowsheet depicted in Figure 2.16, we order **SQ-STRIH** between **STR-REB** and **CHEX**.

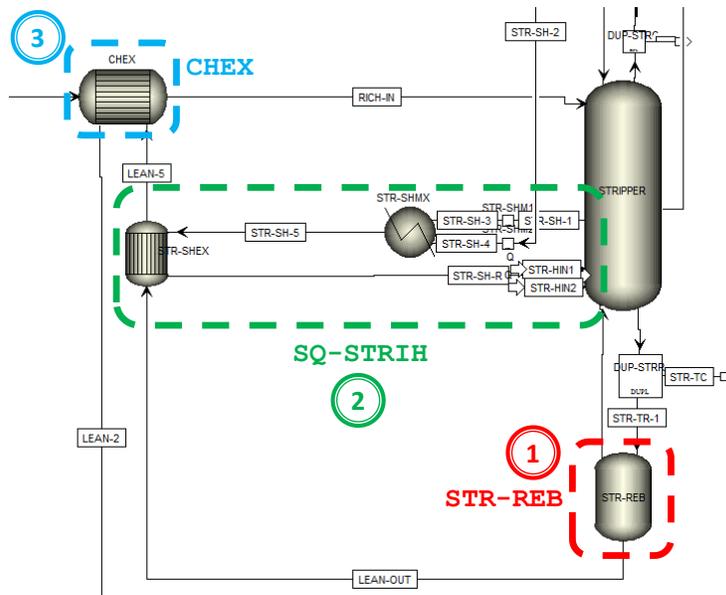


Figure 2.16. Desired order of evaluation. We want the stripper interheating units, **SQ-STRIH**, to be evaluated after **STR-REB** and before **CHEX**.

We modify the **Sequence** **SQ-STR-2** to implement this order as shown in Figure 2.17.

Loop-return	Block type	Block
	Calculator	C-I-STR2
Begin	Convergence	CV-STR-2
	Unit operation	CHEX
	Sequence	SQ-STR-1
	Unit operation	DUP-STRR
	Unit operation	STR-REB
	Sequence	SQ-STRIH
	Unit operation	DUP-STRC
	Unit operation	STR-CON
	Calculator	C-R-STR
Return to	Convergence	CV-STR-2

Figure 2.17. Modification to **SQ-STR-2** that places the interheater, **SQ-STRIH**, after **STR-REB** and before **CHEX**. Note that while **SQ-STRIH** is beneath **CHEX** on the list, the **Convergence** block **CV-STR-2** specifies that the list loops, causing **SQ-STRIH** to also be above **CHEX**.

As a reminder, this sequence places **CHEX** before **SQ-STR-1** so that **CHEX** can determine the primary feed to **STRIPPER** before **STRIPPER**'s first run. **CHEX** is able to perform an initial run since we can provide a reasonably good estimate for the lean solvent stream before the **STRIPPER** is run, forming the initial estimates for **LEAN-OUT**. Now that the information in **LEAN-OUT** can no longer serve this purpose, we must move it to **LEAN-5**. In this particular case, a direct copy/paste will do.

After sequencing this operation, we need to consider if any aspect of it needs to be torn, i.e. considered as a convergence criterion. The stripper interheater complex feeds the heat duty back into the stripper, so this feedback should be torn. Section 1.6.4: "Detecting missing tear specifications" analyzes the consequences of failing to specify this tear.

To tear this feedback, we note that the feedback is adequately described by the combination of the position and duty of the stripper's intercooler. These attributes are each split into two variables for non-

integer stage specification, so we have four variables to tear: **DHEAT1**, **DHEAT2**, **NHEAT1**, and **NHEAT2**. Figure 2.18 defines these variables.

Variable	Information flow	Definition
DISABLED	Import variable	Block-Var Block=STR-SHEX Variable=BYPASS Sentence=PARAM
FDRAW1	Import variable	Block-Var Block=STR-SHM1 Variable=FACTOR Sentence=PARAM
FDRAW2	Import variable	Block-Var Block=STR-SHM2 Variable=FACTOR Sentence=PARAM
NDRAW1	Import variable	Block-Var Block=STRIPPER Variable=PSEUDO-STAGE Sentence=PSEUDO-STREA ID1=STR-SH-1
NDRAW2	Import variable	Block-Var Block=STRIPPER Variable=PSEUDO-STAGE Sentence=PSEUDO-STREA ID1=STR-SH-2
DUTYCALC	Import variable	Block-Var Block=STR-SHEX Variable=QCALC Sentence=RESULTS Units=MW
DHEAT1	Tear variable	Heat-Duty Stream=STR-HIN1 Units=MW
DHEAT2	Tear variable	Heat-Duty Stream=STR-HIN2 Units=MW
NHEAT1	Tear variable	Block-Var Block=STRIPPER Variable=H-FEED-STAGE Sentence=H-FEEDS ID1=STR-HIN1
NHEAT2	Tear variable	Block-Var Block=STRIPPER Variable=H-FEED-STAGE Sentence=H-FEEDS ID1=STR-HIN2

Figure 2.18. Further definitions for **C-STRIHR**.

Next, we need to tell the appropriate **Convergence** block to tear these tear variables. **C-STRIHR** is a member of **SQ-STRIH** which is nested in **CV-STR-2** as specified in **SQ-STR-2**. This means that we need to specify these tears in **CV-STR-2** as shown in Figure 2.19.

Calculator ID	Tear variable	Lower bound	Upper bound	Step size	Maximum step size	Scale
C-STRIHR	DHEAT1			0.01	1	1
C-STRIHR	DHEAT2			0.01	1	1
C-STRIHR	NHEAT1			0.01	1	1
C-STRIHR	NHEAT2			0.01	1	1

Figure 2.19. Calculator tear specifications for **CV-STR-2**.

Workaround

We sometimes encounter a bug in the “Calculator Tears” sheet depicted in Figure 2.19. This bug prevents the user from typing full-length tear variable names in the appropriate column; valid variable names may be up to eight-characters long, but input is restricted to four or five characters. To work around this bug, we rename the tear variables in the **Calculator** block that defines them such that they are within the allowed length limit.

Finally, we update **SA-DATA**:

1. Add Parameter #20,100 as a new Vary variable.
 - a. Single default value of 15, which would put the interheater about midway on the **STRIPPER**.
 - b. Not strictly necessary to pick a reasonable value since, if an unreasonable one is selected, the setting code in Part 1 of **C-STRIHID** would bound it.
2. Add Parameter #20,101 as a new Vary variable.
 - a. Single default value of 0, which would mean no heat-exchanging surface area in the interheater. This makes the interheater hypothetical.
3. Clear all entries in the “Define” and “Tabulate” tabs.
4. Repopulate the “Define” tab using a copy/paste of the defined variables for **C-GLOBAL**.
5. Repopulate the “Tabulate” tab using the “Fill Variables” button.

2.2.3 Initial run

After constructing the flowsheet, we can attempt a quick base-case run. For the five Vary variables in

SA-DATA:

1. Parameter #10, **DUMMY**, can have any single value.
 - a. We usually select 0, however its value is completely irrelevant.
2. Parameter #20,050, **ABSINTCS**, can have any single value.
 - a. Since it is a stage number in the **ABSORBER**, we set the value of this parameter between 1 and 29, though **C-ABS-IC** will interpret any out-of-bounds specification as being at the nearest bound (either 1 or 29).
 - b. Since the duty will be zero, the exact value is irrelevant.
3. Parameter #20,051, **ABSINTCD**, should have a single value of 0.

- a. Since this is a base-case run, the absorber intercooler duty $D_{\text{ABS-IC}}$ should be 0.
4. Parameter #20,100, **STRINTHS**, can have any single value.
 - a. Just like Parameter #20,050, **ABSINTCS**, except it is for the stripper's interheater. The one exception is that the bounds are [1,28] rather than [1,29].
5. Parameter #20,101, **STRINTHA**, should have a single value of 0.
 - a. Since this is a base-case run, the stripper interheating area $A_{\text{STR-IH}}$ should be 0.

If properly constructed, the simulation should run without issue. All relevant results should be exactly as the original base-case simulation within the limits of numerical error.

- Our base-case gives $E_{\text{regen}} \cong 3.62960 \frac{\text{GJ}_t}{\text{tonne CO}_2}$.
- This simulation gives $E_{\text{regen}} \cong 3.62961 \frac{\text{GJ}_t}{\text{tonne CO}_2}$.

We reconcile the relevant streams, blocks, and **C-INIT** parameters to improve initial estimates for future runs.

1. Reconcile **ABSORBER** and **STRIPPER** as described in Section 1.6.2.2: “**RadFrac** estimates”.
2. Reconcile streams:
 - a. **ABS-FCON**
 - b. **LEAN-5**
 - c. **LEAN-OUT**
 - d. **STR-FCON**
3. Using the values reported in **SA-DATA** (**not** the values found in the results of **C-INIT**), reconcile all parameters in **C-INIT** except for the iterators which should continue to be initialized at 0.

2.2.4 Sensitivity analyses

First, we test a 25 m² interheating from Stage #5 to Stage #25, finding the response in Figure 2.20.

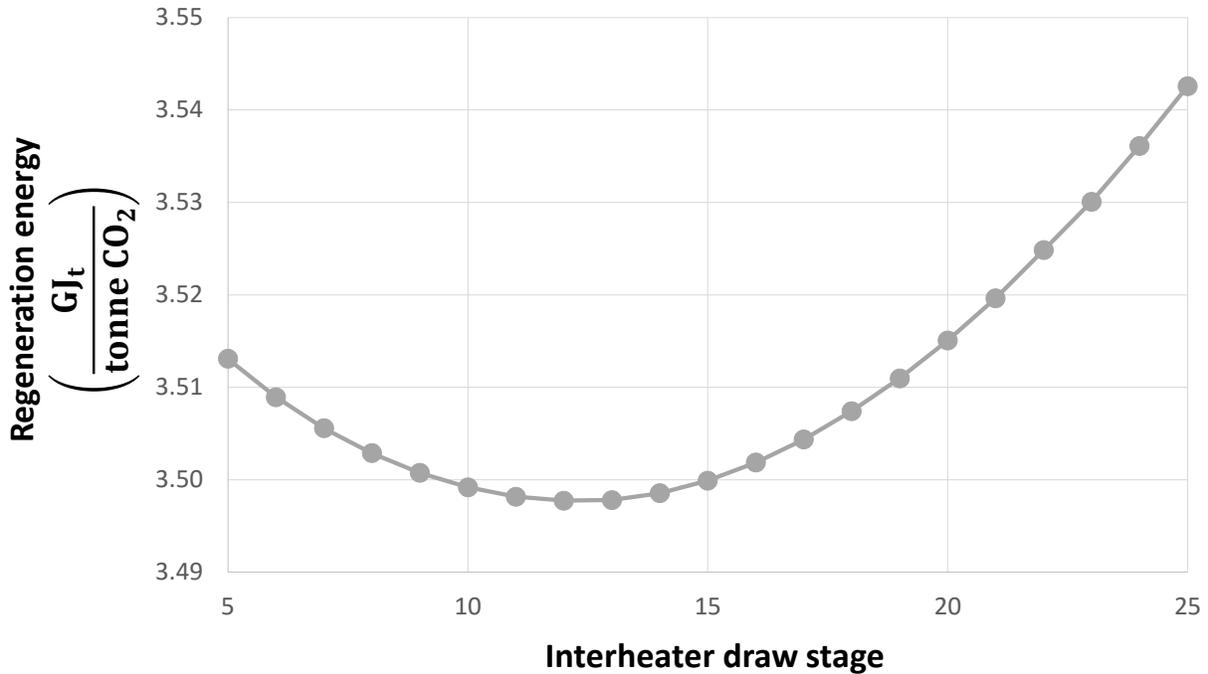


Figure 2.20. Regeneration energy for various interheater draw stages.

Without the interheater, the base-case regeneration energy is $\sim 3.63 \frac{\text{GJ}}{\text{tonne}}$, so we find a reduction of 2% to 4% depending on the stage.

Noting that the optimal is around Stage #12, we can then do a sensitivity for the heat-exchanging surface area from 0m^2 to 50m^2 (compared to the 25m^2 used on different stages in the prior analysis). We started this sensitivity before concluding the prior analysis; therefore, having guessed Stage #14 to be near-optimal, we based our sensitivity analysis of the stripper interheating area at Stage 14. Figure 2.21 shows the results of the sensitivity analysis over heat-exchanging surface area at Stage 14.

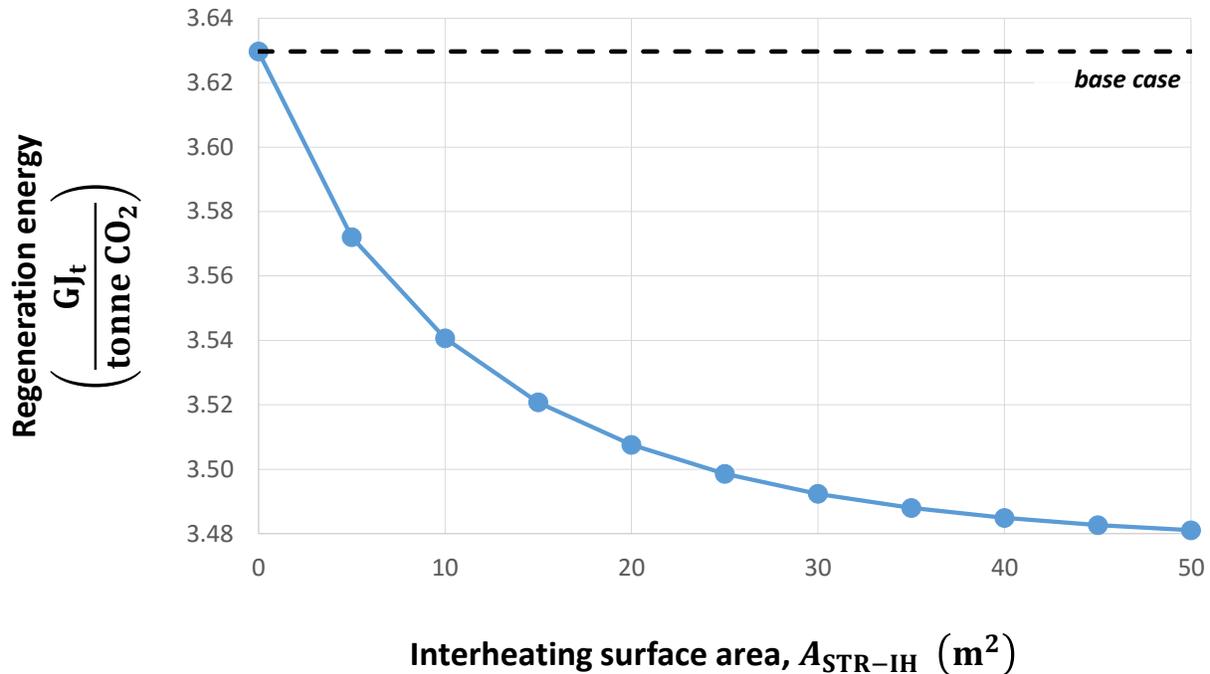


Figure 2.21. Regeneration energy from various interheating surface areas on Stage #14.

The effect of additional surface area drops off as $\frac{A_{STR-IH}}{m^2} \rightarrow \infty$ since the interheater effectively reaches zero temperature approach such that further surface area has no effect. At the asymptotic limit, we see a $\sim 4.2\%$ reduction in regeneration energy, E_t^{regen} , to approximately $3.478 \frac{GJ_t}{tonne CO_2}$.

2.2.5 Remarks

In the baseline configuration, the central cross heat exchanger, **CHEX**, transfers some of the heat in the regenerated solvent to the loaded solvent just before the loaded solvent enters the stripper. A stripper interheater on Stage #1 has almost the same effect as just making **CHEX** larger.

However, we can move the interheater lower, thus moving more of the heat toward a midpoint on the column. This placement reduces solvent vapor being lost to the condenser, increasing the amount of heat going into stripping. We can see from the sensitivity analysis that this midpoint application is the most efficient approach.

2.3 Stripper condensate rerouting

Stripper condensate rerouting is a very simple energy-saving scheme. The core principle is that we should put the cold condensate from the stripper's condenser into a part of the process that it supposed to be cold. The first appearance of this energy-saving scheme comes from a 1961 patent.²⁵

As a benefit, stripper condensate rerouting avoids the energy cost necessary to bring the cold condensate stream back to the stripper's operating temperature. As a drawback, stripper condensate rerouting also allows the top of the stripper to operate at a higher temperature, promoting vaporization.

We often find that stripper condensate rerouting is useful when combined with other energy-saving schemes but harmful in the base case. This result stresses the need to optimize energy-saving schemes together rather than separately.

2.3.1 Modeling approach

Stripper condensate rerouting is another fairly straightforward energy-saving scheme. Here we will attach a single parameter for the portion of the condensate to be rerouted, $\mathcal{X}_{\text{STR-CON}} \in [0,1]$.

2.3.2 Flowsheet construction

First, we want to draw the split on the stripper condensate using an `FSplit` block named `STR-CSPL` as shown in Figure 2.22.

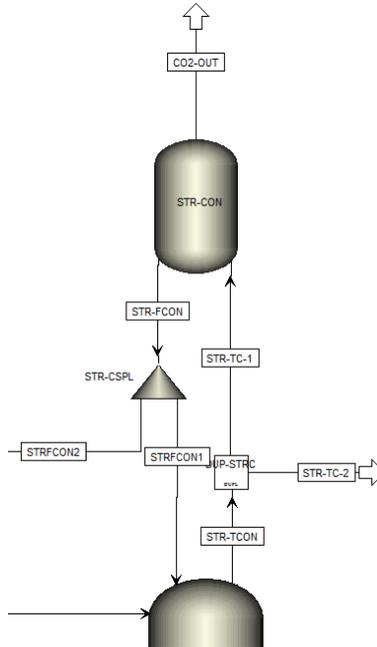


Figure 2.22. Flowsheet configuration for stripper condensate rerouting.

The new stream **STRFCOIN1** restores the connection to **STRIPPER** while **STRFCOIN2** leads to the lean stream cooler, **LSC**.

We specify the splitter with a split fraction of 0 to the **LSC** as shown in Figure 2.23.

in Flowsheet > STR-CSPL (FSplit) > +							
Specifications Flash Options Key Components Information							
Flow split specification for outlet streams							
	Stream	Specification	Basis	Value	Units	Key Comp No	Stream order
▶	STRFCOIN1						
▶	STRFCOIN2	Split fraction		0			

Figure 2.23. Input specification for **STR-CSPL**.

By splitting zero to the new route, we have effectively redescribed the base-case design. We add a **calculator** to set $\mathcal{X}_{\text{STR-CON}}$ to replace the flowsheet-level zero specified in Figure 2.23. The flowsheet-level value does not actually matter as it will never be used.

If it were not already present, we would define a new global parameter **XSTRCON** in **C-GLOBAL**; however, this variable is already defined. It will serve as $\mathcal{X}_{\text{STR-CON}}$. We remove all references to **XSTRCON** from **C-GLOBAL** and amend the code in Scheme 2.5 to the end of **C-GLOBAL**.

```

C =====
C =====  START: STRIPPER CONDENSATE REROUTING  =====
C =====
C Vapor condensate rerouting portion:
C           XSTRCON = 0.0
C
C =====
C =====  END: STRIPPER CONDENSATE REROUTING  =====
C =====

```

Scheme 2.5. Interpreted Fortran to amend to the end of **C-GLOBAL** to initialize **XSTRCON**.

Next, we create a new **Calculator**, **C-XSTRCN**, with the definitions in Figure 2.24 and interpreted Fortran in Scheme 2.6.

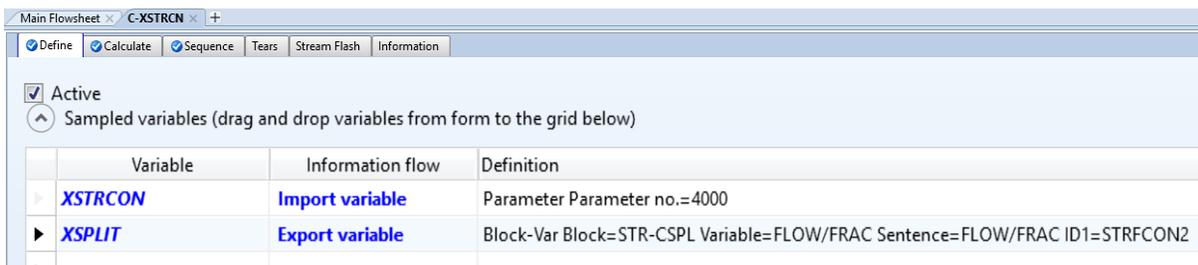


Figure 2.24. Definitions for **C-XSTRCN**.

```

C Bound the specified split fraction between
C zero and one, then set:
C           if (XSTRCON .LT. 0.0) then
C               XSPPLIT = 0.0
C           else if (XSTRCON .GT. 1.0) then
C               XSPPLIT = 1.0
C           else
C               XSPPLIT = XSTRCON
C           end if

```

Scheme 2.6. Interpreted Fortran for **C-XSTRCN**.

We amend this new **Calculator**, **C-XSTRCN**, to the end of **SQ-PRE**.

For the actual splitter, **STR-CSPL**, we sequence it just before the stripper's packing. The idea is that we have a single initial estimate for the condensate stream, **STR-FCON**, that applies for all split fractions, because it is immediately run through the splitter **STR-CSPL** before going into the packed column. This means entering the splitter into **SQ-STR-2** as shown in Figure 2.25.

Loop-return	Block type	Block
	Calculator	C-I-STR2
Begin	Convergence	CV-STR-2
	Unit operation	CHEX
	Unit operation	STR-CSPL
	Sequence	SQ-STR-1
	Unit operation	DUP-STRR
	Unit operation	STR-REB
	Sequence	SQ-STRIH
	Unit operation	DUP-STRC
	Unit operation	STR-CON
	Calculator	C-R-STR
Return to	Convergence	CV-STR-2

Figure 2.25. Insert **STR-CSPL** after **CHEX** in **SQ-STR-2**.

Finally, we add a **SA-DATA** Vary variable for Parameter #4,000 for a single default value, 0, to represent the base case $\mathcal{X}_{\text{STR-CON}} = 0$.

2.3.3 Sensitivity analyses

Since we have a single variable associated with this energy-saving scheme, we can perform a straightforward sensitivity against the base-case configuration by changing the Vary for Parameter #4,000 to record 11 points in $\mathcal{X}_{\text{STR-CON}} \in [0,1]$. Figure 2.26 shows the resulting response.



Figure 2.26. Regeneration energy response to various stripper condensate rerouting portions.

We again stress that the response curve in Figure 2.26 is specific to this exact case. Different flue gas streams, capture rates, incorporation of other energy-saving schemes, etc., may all change the response curve.

2.3.4 Remarks

2.3.4.1 Regarding process optimization

The base case is defined such that $X_{STR-CON} = 0\%$. If we simply made a simulation drawn with full condensate rerouting, i.e. $X_{STR-CON} = 100\%$, we would have found that the energy-saving scheme actually hurts performance.

However, by considering a generalized model, we find an optimal solution near $X_{STR-CON} \cong 30\%$, exemplifying another advantage of model generalization: finding optimal solutions outside of the basic set of configurations.

While not demonstrated in this dissertation, our research has shown that the optimal $X_{STR-CON}$ varies significantly with the exact process configurations. In some cases, it is best to either go completely with or without condensate rerouting, while in others, it is best to go with a hybrid approach such as found above.

Ultimately, we will suggest an automated optimization process; this manual sensitivity analysis process is merely for illustration. The generalized modeling approach not only enables the optimizer, but also allows us to find the wide space of hybrid solutions that more naïve modeling approaches fail to even consider.

2.3.4.2 Regarding stripper condensate rerouting in specific

Stripper condensate rerouting tends to yield more desirable results in optimal designs. This is because it involves a trade-off:

- **good:** Avoids putting cool liquid (condensate, $\sim 40^\circ\text{C}$) into the stripper.
- **bad:** Allows the top of the stripper to remain warmer, increasing harmful, inefficient vaporization.

The optimal point in the above simulation results from balancing the good and bad effects, which just happens to be at $X_{\text{STR-CON}} \cong 30\%$ in this particular case.

However, this section considers stripper condensate rerouting in isolation. In real designs, we will incorporate other energy-saving schemes, many of which work by reducing that harmful, inefficient vaporization (see Section 2.4: “Distributed cross heat exchanger”). These other energy-saving schemes will greatly reduce the negative effect of stripper condensate rerouting, which improves our reason for employing it.

In general, it is not worth being overly concerned with trying to reproduce the consequences of a scheme like this in your own mind; that is what we have Aspen Plus for. In practice, we will allow the optimizer to add or remove energy-saving schemes based on the rigorous optimization process that we are in the process of constructing.

2.4 Distributed cross heat exchanger

The base-case design from 1930 employs a central cross heat exchanger (CHEX) to greatly improve thermal efficiency (Section 1.4.1.2: “Thermal loop”). However, we have seen that, when we add additional heat to the stripper, putting it at the very top of the stripper is sub-optimal; it is better to try for a midpoint (Section 2.2: “Stripper interheating”).

Now, we will attempt to improve the CHEX such that it delivers heat to the stripper in an optimal way, i.e. to midpoints. Stripper interheating can accomplish this by delivering heat directly to the liquid phase already inside of the stripper, so we split a fraction of the CHEX's feed to a midpoint on the stripper and provide it with preferential heating compared to the stream still being fed into the top of the stripper.

2.4.1 Modeling approach

This energy-saving scheme involves three parameters, making it more complex than the previously discussed energy-saving schemes. First, to establish context, we have the base-case design shown in Figure 2.27.

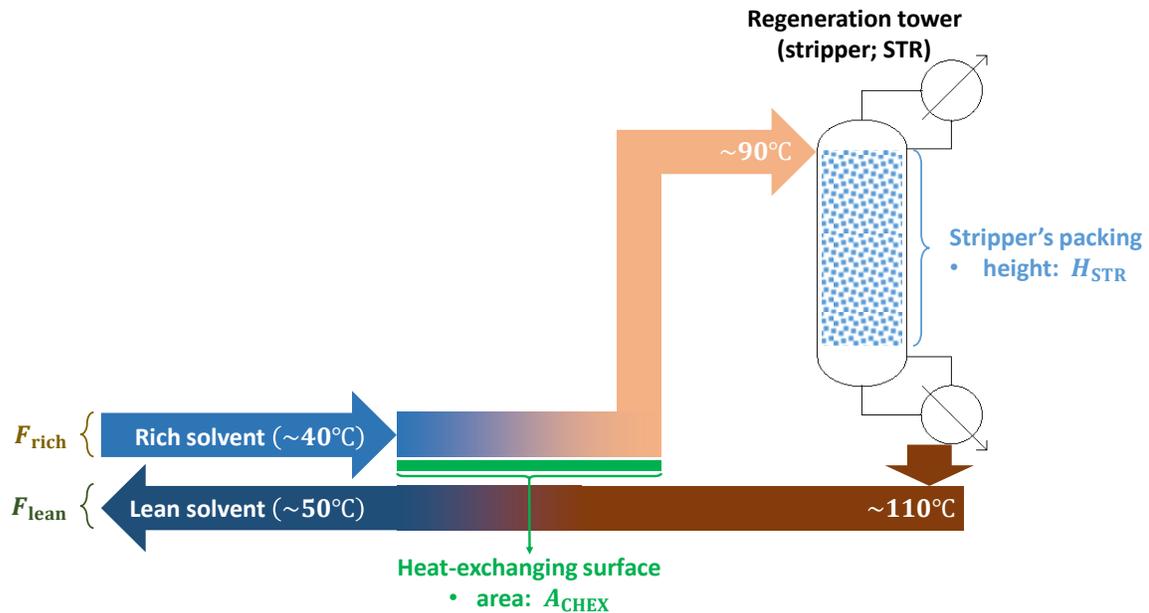


Figure 2.27. Conceptual model for the central cross heat exchanger and stripper. We will extend this model to describe the distributed cross heat exchanger.

We modify this design with three adjustable parameters:

1. a_{DHEX} : the portion of heat-exchanging surface area allocated to the distributed portion;
2. f_{DHEX} : the portion of rich solvent split off to a midpoint on the stripper;
3. h_{DHEX} : the portion of packed height that the split-off portion flows through.

Figure 2.28 shows these three parameters in the conceptual flowsheet for the distributed cross heat exchanger configuration.

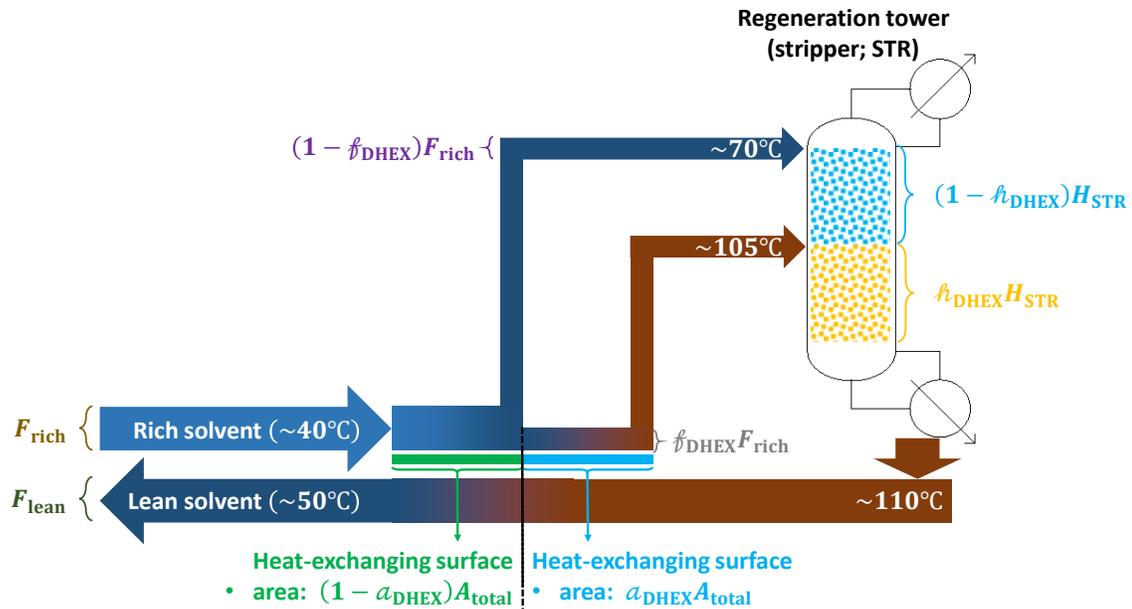


Figure 2.28. Conceptual flowsheet for the distributed cross heat exchanger configuration.

Our description variables $\{a_{DHEX}, f_{DHEX}, h_{DHEX}\}$ are all unitless fractional numbers varying between 0 and 1, with $\{a_{DHEX}, f_{DHEX}, h_{DHEX}\} = 0$ representing the base-case design. As with all of our energy-saving schemes, we want to ensure that the description includes the base-case design so that the optimizer can eliminate this scheme if it is impractical.

As for convergence, we follow the principle that computational object design should directly mirror reality until optimization (Section 1.6.3.3: “Object divisions should mirror physical reality”). Therefore, we will derive an object to replace the current computational analog of the cross heat exchanger, the **HeatX** block **CHEX**.

Our new **CHEX** object contains two **HeatX** blocks in it: one for each of the two heat-exchanging surfaces in the above figure. Because the current **HeatX** block may now realistically encounter zero surface area, $A_{CHEX} = 0$, when $a_{DHEX} = 1$, we need to build in a disabling calculator for it as we did for the stripper’s inteating **HeatX**, **STR-SHEX**. The second **HeatX** block requires this and a condition to check the flow

rate because it may encounter zero cold-side flowrate, e.g. as-in the base-case design in which $\dot{f}_{DHEX} = 0$.

If any of the above is unclear, it may help to walk through the flowsheet construction below.

2.4.2 Flowsheet construction

First, we declare our new parameters in **C-GLOBAL**. These include $\{a_{DHEX}, \dot{f}_{DHEX}, h_{DHEX}\}$. However, a more robust base-case simulation would have already included A_{total} and H_{STR} as global parameters. Since we have not yet made those, we will include them now. Their current values are set at the flowsheet level:

- A_{total} at Aspen Plus | Simulation | Blocks → CHEX → Setup | “Specifications” tab as 50m²;
- H_{STR} at Aspen Plus | Simulation | Blocks → STRIPPER → Sizing and Rating → Pack Rating → 1 → Setup | “Specifications” tab as 25m.

Declare the parameters in Table 2.2 for **C-GLOBAL**.

Table 2.2. Distributed cross heat exchanger configuration parameters to be declared in **C-GLOBAL**.

Fortran	Meaning	Direction	Definition	Explanation
MMINIMUM	<u>Mass flow rate</u> <u>MINIMUM</u>	Export	Parameter #3,102	Minimum mass flow rate for a stream to be considered as non-zero.
HSTR	<u>Height</u> <u>STRipper</u>	Export	Parameter #20,200	Total packing height in stripper.
ATOTAL	<u>Area</u> <u>TOTAL</u>	Export	Parameter #21,000	Total heat-exchanging surface area.
ADHEX	<u>Area portion</u> for the	Export	Parameter #21,010	Portion of the heat-exchanging

	<u>D</u> istributed <u>H</u> eat <u>E</u> Xchanger			surface area assigned to the distributed portion of the central cross heat exchanger.
FDHEX	<u>F</u> low portion for the <u>D</u> istributed <u>H</u> eat <u>E</u> Xchanger	Export	Parameter #21,020	Portion of the rich solvent flow rate sent to the distributed portion of the central cross heat exchanger.
HDHEX	<u>H</u> eight portion for the <u>D</u> istributed <u>H</u> eat <u>E</u> Xchanger	Export	Parameter #21,030	Portion of the height up the stripper's packing that the distributed rich solvent is fed into.

We insert setters for the numeric cut-off variables **AMINIMUM** and **MMINIMUM** in **C-GLOBAL** after the **MATH CONSTANTS** section as shown in Scheme 2.7.

```

C =====
C ===== START:  NUMERIC CUT-OFFS =====
C =====

C Minimum heat-exchanging surface area to be
C considered non-zero (m^2):
      AMINIMUM = 0.0001

C Minimum mass flow rate to be
C considered non-zero (tonne/hr):

```

```

      MMINIMUM = 0.0001

C =====
C =====  END:  NUMERIC CUT-OFFS  =====
C =====

```

Scheme 2.7. Numeric cut-off parameters are initialized in **C-GLOBAL**.

We also insert a setter for **HSTR** into the **STRIPPER** section in **C-GLOBAL**. We modify the stripper section in **C-GLOBAL** to begin as shown in Scheme 2.8.

```

C =====
C =====  START:  STRIPPER           =====
C =====

C Stripper pressure (bar):
      PSTR = 1.4

C Stripper packed height (m):
      HSTR = 25.0

```

Scheme 2.8. Setter for **HSTR** added into the stripper section of **C-GLOBAL**.

As a final modification to **C-GLOBAL**, we amend the code for the cross heat exchanger initializers shown in Scheme 2.9 to the end.

```

C =====
C =====  START:  CROSS HEAT EXCHANGER           =====
C =====

C Total cross heat exchanger surface area (m^2):
      ATOTAL = 50.0

C Portion of heat-exchanging surface area distributed:
      ADHEX = 0.0

C Portion of rich solvent flow distributed:
      FDHEX = 0.0

```

```

C Portion of height from the top of the stripper
C that the distributed rich solvent flows into:
      HDHEX = 0.0

C =====
C =====      END: CROSS HEAT EXCHANGER      =====
C =====

```

Scheme 2.9. Cross heat exchanger section to be amended to the end of **C-GLOBAL**'s interpreted Fortran.

Next, we create a new **Calculator**, **C-S-HSTR**, with the definitions in Figure 2.29 and interpreted Fortran in Scheme 2.10.

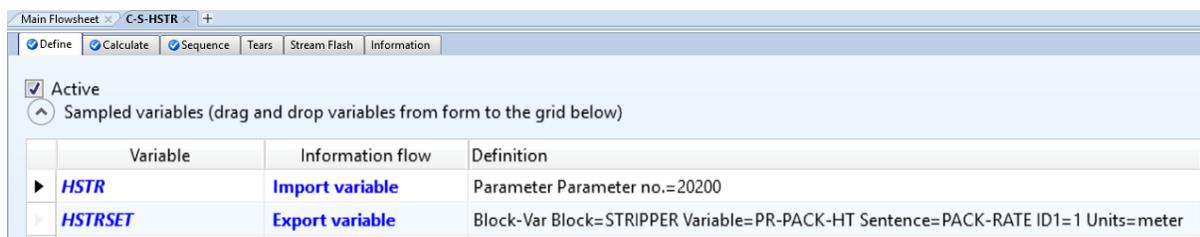


Figure 2.29. Definitions for **C-S-HSTR**.

```

HSTRSET = HSTR

```

Scheme 2.10. Single line of interpreted Fortran for **C-S-HSTR**.

According to a rigorous object-oriented approach, we should create a new derived object as a **Sequence** of **C-S-HSTR** followed by the packed section of the stripper, **STRIPPER**. However, we will take a quicker route in just amending **C-S-HSTR** to the end of **SQ-PRE**, assuming that **HSTR** does not change during the execution of the unit. We may change this at some later point, but it should suffice without any drawbacks until we have developed the flowsheet in a way that enables **HSTR** to be modified within the unit.

After establishing the previously excluded setter, we now move on to the actual distributed cross heat exchanger configuration. More precisely, we want to generalize the prior model (a simple **HeatX**) into a more derived object using the descriptors that we have declared in **C-GLOBAL**.

With the addition of the prior energy-saving schemes, our flowsheet as shown in Figure 2.30 is starting to look a bit messy.

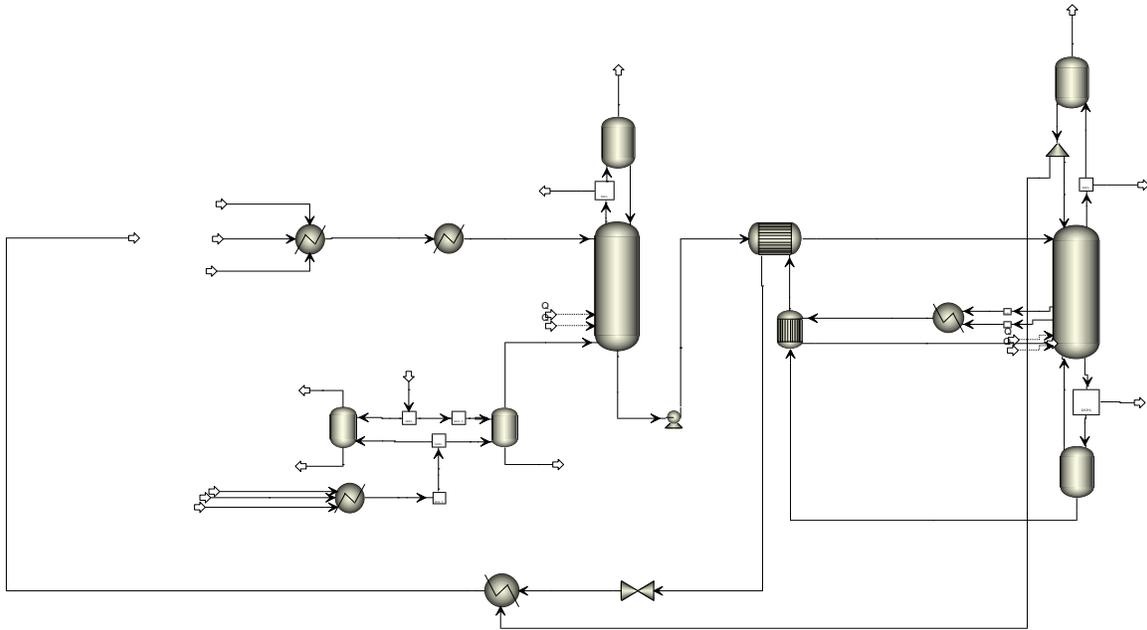


Figure 2.30. Current appearance of our acid-gas-capture flowsheet. The flowsheet becomes increasingly involved as we add more energy-saving schemes for the optimizer to consider.

However, we will clean it up later. For now, let us just make some quick changes:

1. increase the sizes of the icons for **ABSORBER** and **STRIPPER**:
 - a. primarily because they have a lot of connections, so it is cleaner to draw them as very large objects;
 - b. secondarily because they are physically large objects;
2. increase the flowsheet's area, especially by dragging **STRIPPER** to the right:
 - a. to make room for the new blocks;
3. do a quick clean-up:
 - a. recall that you can select blocks and streams and then use **Ctrl+B** to align blocks based on flow direction (avoid selecting loops while doing this);
 - b. if streams become hopelessly twisted, you can reconnect them to their source/destination to refresh them.

Warning

Avoid using **Ctrl+Z** to reverse modifications to the flowsheet. This feature seems to lead to corruption. We suspect that the reversal process causes the internal flowsheet model and the displayed flowsheet model to diverge. If you accidentally hit **Ctrl+Z** as a force of habit, it may be advisable to close the flowsheet without saving and reload.

Take a little while to do this decently well. In the long-term, a well-organized flowsheet will save you far more time than the few minutes it takes to organize it. Additionally, a well-structured image is very helpful to imagining and understanding the design. However, we will make further changes soon, so fine-tuning is not necessary.

After adjusting the flowsheet, we arrive at the flowsheet shown in Figure 2.31.

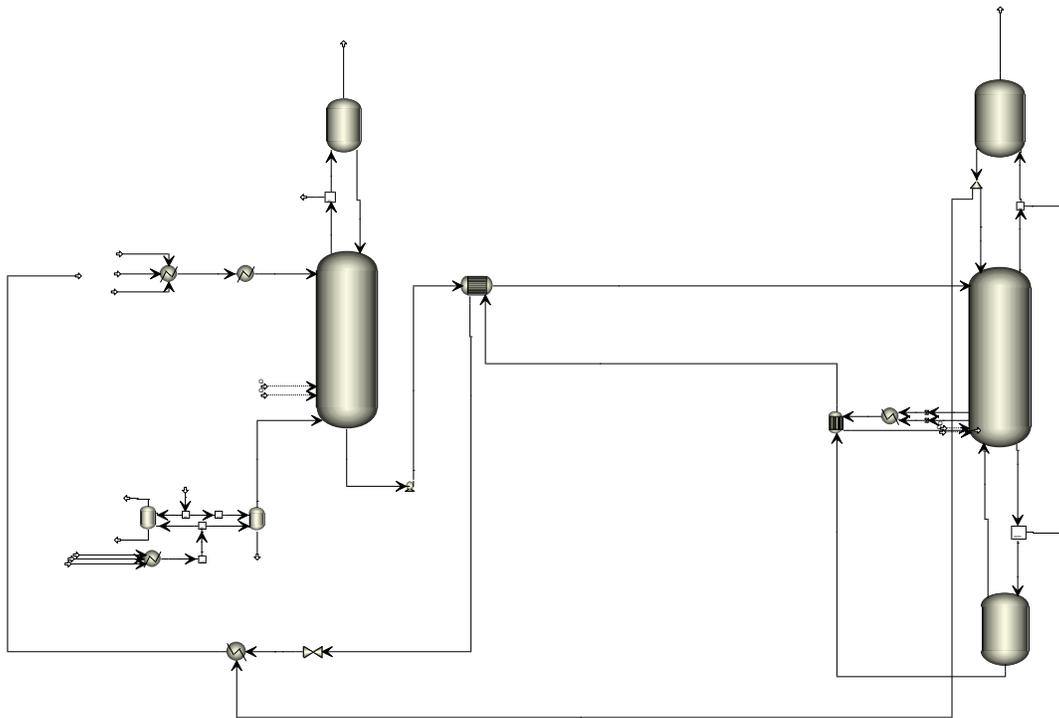


Figure 2.31. Our flowsheet after cleaning and formatting. This flowsheet is easier to read, making it easier to understand and work with.

Now, we draw the new blocks for the distributed central cross heat exchanger configuration:

1. Rename **CHEX** as "**CHEXMAIN**".

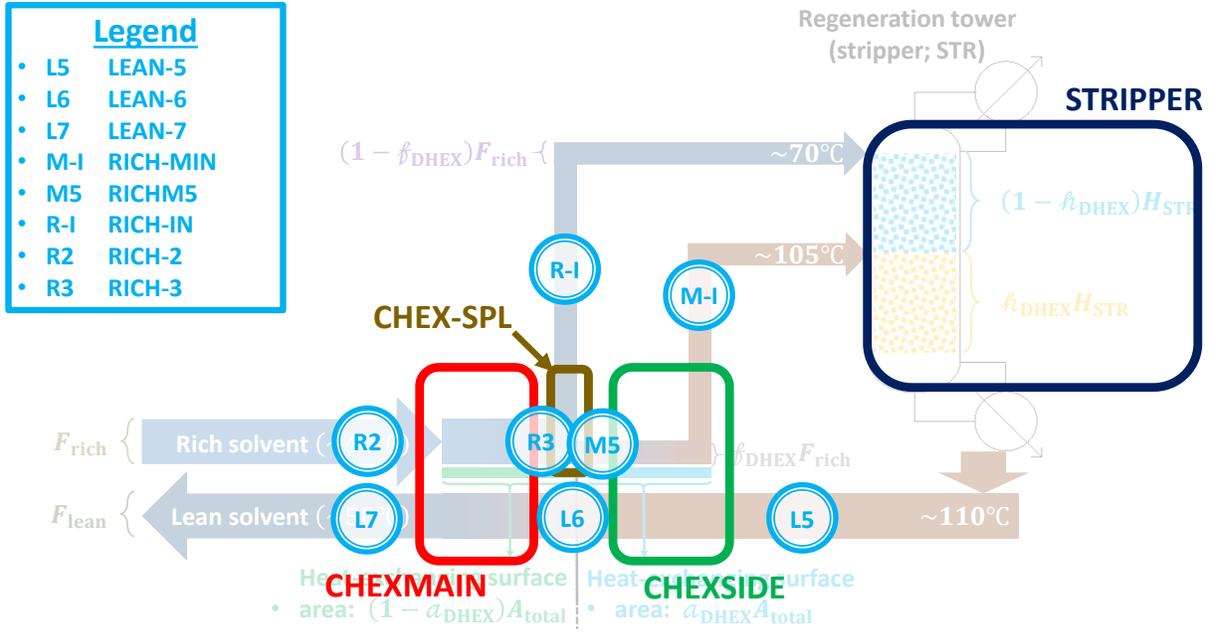


Figure 2.33. Mapping between the conceptual flowsheet and the Aspen Plus flowsheet. We show the conceptual flowsheet in the background with boxes for the Aspen Plus blocks **CHEXMAIN**, **CHEXSIDE**, **CHEX-SPL**, and **STRIPPER**. The streams are also marked in circles, using the shorthand listed in the legend.

Not pictured in the above figure are **CHEX-M-F** and **CHEX-S-F**. These **Flash2** blocks take any vapor produced by the **HeatX** blocks that feed them and split that vapor into the bottom stage of **STRIPPER**, effectively using that vapor as the stripper's reboiler **STR-REB** does.

Go to

Aspen Plus | Simulation | Blocks → STRIPPER → Specifications → Setup | "Streams" tab, and specify the stream stages shown in Figure 2.34.

	Name	Stage	Convention
▶	RICH-IN	1	<i>Above-Stage</i>
▶	STR-FREB	30	On-Stage
▶	STRFCON1	1	<i>Above-Stage</i>
▶	RICH-V-2	30	On-Stage
▶	RICH-V-1	30	On-Stage
▶	RICH-MI1	1	<i>Above-Stage</i>
▶	RICH-MI2	1	<i>Above-Stage</i>

Figure 2.34. New stream specifications for **STRIPPER**.

As a reminder, the “On-Stage” convention means that vapor flows into the specified stage. If we use the default “Above-Stage” for the feed, these vapor streams would effectively flow into Stage #29 instead of the intended Stage #30. Because **RICH-IN**, **RICH-M1**, and **RICH-M2** are all necessarily 100% liquid, the choice of convention for their vapor portions is moot.

Next, we need to derive new objects based on **CHEXMAIN** and **CHEXSIDE** because these blocks may encounter zero-area or, in the case of **CHEXSIDE**, zero flow rate. We will follow the prior pattern in which we created **C-D-SHEX** to derive **SQ-DSHEX** from **STR-SHEX**. This consistency greatly increases robustness and clarity; however, because **C-D-SHEX** does not yet consider incoming mass flow rates, we need to update it first:

1. Copy the definition of **MMINIMUM** from **C-GLOBAL** and paste it into the definition list for **C-D-SHEX**. Set its information flow to “Import variable”.
2. Define **MCOLD** as an import of the mass flowrate of **STR-SHEX**’s incoming cold stream, **STR-SH-5**.
3. Define **MHOT** as an import of the mass flowrate of **STR-SHEX**’s incoming hot stream, **LEAN-OUT**.

Figure 2.35 shows the variable definitions for **C-D-SHEX**.

Variable	Information flow	Definition
AMINIMUM	Import variable	Parameter Parameter no.=3101
MMINIMUM	Import variable	Parameter Parameter no.=3102
AREA	Import variable	Block-Var Block=STR-SHEX Variable=AREA Sentence=PARAM Units=sqm
MCOLD	Import variable	Stream-Var Stream=STR-SH-5 Substream=MIXED Variable=MASS-FLOW Units=tonne/hr
MHOT	Import variable	Stream-Var Stream=LEAN-OUT Substream=MIXED Variable=MASS-FLOW Units=tonne/hr
DISABLED	Export variable	Block-Var Block=STR-SHEX Variable=BYPASS Sentence=PARAM

Figure 2.35. Variable definitions for C-D-SHEX.

We also update C-D-SHEX's interpreted Fortran, replacing the prior code with that shown in Scheme 2.11.

```

if (AREA .LE. AMINIMUM) then
  DISABLED = 1.0
else if (MCOLD .LE. MMINIMUM) then
  DISABLED = 1.0
else if (MHOT .LE. MMINIMUM) then
  DISABLED = 1.0
else
  DISABLED = 0.0
end if

```

Scheme 2.11. New interpreted Fortran for C-D-SHEX.

Compared to the prior version, this now considers the mass flow rate of the incoming streams as a criterion for disabling the HeatX block:

$$\text{Enabled} = \begin{cases} \text{false} & \text{if } A \leq A_{\min} \\ \text{false} & \text{if } M_{\text{cold in}} \leq M_{\min} \\ \text{false} & \text{if } M_{\text{hot in}} \leq M_{\min} \\ \text{true} & \text{else} \end{cases}$$

Now, we apply this template to derive smarter variants of the HeatX block:

1. Create two new Calculator blocks, C-D-MAIN and C-D-SIDE, by copying and pasting C-D-SHEX twice with the corresponding name changes.

2. Change the definitions for **AREA** and **DISABLED** such that they refer to the correct **HeatX** blocks.
 - a. **CHEXMAIN** in **C-D-MAIN** and **CHEXSIDE** in **C-D-SIDE**.
3. Change the definitions for **MCOLD** and **MHOT** such that they refer to the correct streams corresponding to the **HeatX** blocks.
 - a. **RICH-2** and **LEAN-6**, respectively, for **C-D-MAIN**.
 - b. **RICHM5** and **LEAN-5**, respectively, for **C-D-SIDE**.
4. Create two new **Sequence** blocks, **SQ-DMAIN** and **SQ-DSIDE**., as analogs to **SQ-DSHEX**.
 - a. Define **SQ-DMAIN**'s sequence as:
 - i. **C-D-MAIN**;
 - ii. **CHEXMAIN**.
 - b. Define **SQ-DSIDE**'s sequence as:
 - i. **C-D-SIDE**;
 - ii. **CHEXSIDE**.

Next, we define a new **Sequence** block, **SQ-DHEX1**, to describe the derived object shown in Figure 2.36. Figure 2.37 gives the details of the **Sequence** block, **SQ-DHEX1**.

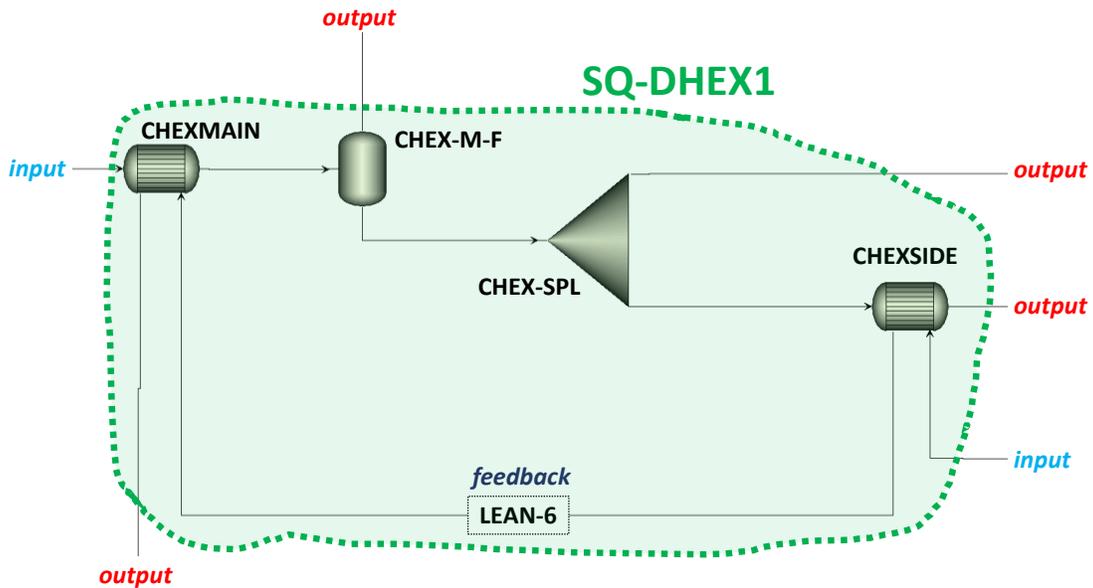


Figure 2.36. Derived object **SQ-DHEX1** for a region of the distributed cross heat exchanger.

We will need to include a **Convergence** block in **SQ-DHEX1**'s definition to account for the feedback from **LEAN-6**.

	Loop-return	Block type	Block
▶	Begin	Convergence	CV-DHEX1
▶		Sequence	SQ-DMAIN
▶		Unit operation	CHEX-M-F
▶		Unit operation	CHEX-SPL
▶		Sequence	SQ-DSIDE
▶	Return to	Convergence	CV-DHEX1
▶			

Figure 2.37. Definition for **SQ-DHEX1**.

The **Convergence** block, **CV-DHEX1**, should use the Wegstein method and have **LEAN-6** specified as a tear stream.

We further define **SQ-DHEX2** to include the remaining blocks. Its procedure is simply:

1. **SQ-DHEX1**
2. **CHEX-S-F**
3. **STR-SPLM**

Our new cross heat exchanger object is now **SQ-DHEX2**. **SQ-DHEX2** effectively replaces the **HeatX** block **CHEX** from our earlier simulations. We need to specify this in **SQ-STR-2** by replacing **CHEX** (or actually **CHEXMAIN**, since Aspen Plus automatically updated the **HeatX**'s name when we changed it earlier) with **SQ-DHEX2**.

Additionally, we should provide initial estimates for the **LEAN-6** by copying the initial estimates for **LEAN-5**.

Our final step will be to create **Calculator** blocks to set the specifications for the cross heat exchanger and then sequence these Calculator blocks in **SQ-PRE**:

1. **C-ADHEX**;
2. **C-FDHEX**;

3. **C-HDHEX.**

For **C-ADHEX**:

1. Copy/paste the definitions of **ATOTAL** and **ADHEX** from **C-GLOBAL** as import variables.
2. Define the specified area of **CHEXMAIN** as an export variable **AMAIN**.
3. Define the specified area of **CHEXSIDE** as an export variable **ASIDE**.
4. Code:

```
C Set the areas for the two HeatX's in
C the distributed cross heat exchanger configuration:
    AMAIN = (1.0 - ADHEX) * ATOTAL
    ASIDE = ADHEX * ATOTAL
```

For **C-FDHEX**:

1. Copy/paste the definition of **FDHEX** from **C-GLOBAL** as an import variable.
2. Define the split-to-**RICHM5** portion of **CHEX-SPL** as an export variable **SPLIT**.
3. Code:

```
C Set the split-to-mid portion for the
C distributed cross heat exchanger configuration:
    SPLIT = FDHEX
```

For **C-HDHEX**:

1. Copy/paste the definition of **HDHEX** from **C-GLOBAL** as an import variable.
2. Define **NFEED1** and **NFEED2** as export variables for the stages that **RICH-MI1** and **RICH-MI2** feed into **STRIPPER**.
 - a. Note that the feed stage variable is called "FEED-STAGE" in the variable list.
3. Define **STAGE0** and **STAGEN** as import variables for the first and last stage numbers of the first packed section in **STRIPPER**.
 - a. "PR-STAGE1" and "PR-STAGE2" in the variable list.
 - b. For both, the "ID1" field is simply "1" because **STRIPPER** contains only one section of packing called "1".
4. Define the split-to-**RICH-MI2** for **STR-SPLM** as the export variable **SPLIT**.

5. Code:

```
C =====
C === PART 1: Bound the portion. ===
C =====
C Bound the portion between 0 and 1:
  if (HDHEX .LT. 0.0) then
    ZSET = 0.0
  else if (HDHEX .GT. 1.0) then
    ZSET = 1.0
  else
    ZSET = HDHEX
  end if

C =====
C === PART 2: Calculate the target stage. ===
C =====
      ZSTAGE = 1.0 + ZSET * (STAGEN - STAGE0)

C =====
C === PART 3: Interpolate the stage specification. ===
C =====

C Is the stage specification an integer?
      ZSTAGEI = INT(ZSTAGE)
      if (ZSET .EQ. ZSTAGEI) then

C Yes, the stage specification is an integer.
C The effective target stage is an integer.
C So, set both streams to feed into that stage.
C Give both streams a 50% of the flow to help
C mitigate numerical errors from a zero-flow-rate stream.
      NFEED1 = ZSTAGEI
      NFEED2 = ZSTAGEI
      SPLIT = 0.5
```

```
else
C No, the stage specification isn't an integer.
C So, we target the consecutive integers:
    ZROUND = ZSTAGEI
    if (ZSTAGEI .GE. ZSET) then
        ZROUND = ZROUND - 1.0
    end if
    NFEED1 = ZROUND
    NFEED2 = ZROUND + 1.0

C And weight the split-to-second:
    SPLIT = ZSTAGE - ZROUND
end if
```

Be sure to have sequenced these new **Calculator** blocks into **SQ-PRE**.

Finally, we update **SA-DATA**:

1. Create Vary variables for **ADHEX**, **FDHEX**, and **HDHEX**.
 - a. Recall each varies between 0 and 1 with the base case for each at 0.
 - b. For now, a single value of 0 will allow us to test the flowsheet to confirm that it converges on the base-case without error, except that we use 0.5 for the single value of **HDHEX**; this corresponds to the midpoint on the **STRIPPER** without having any practical effect while the other two values are at 0.
2. Clear the definitions and tabulations, and then reset them by copy/pasting from **C-GLOBAL** and filling, as we have done before.

2.5 Heat pump integration

Heat exchangers are limited to moving thermal energy from higher temperature to lower temperature. Heat pumps can effectively move thermal energy against the temperature gradient – from lower temperature to higher temperature – effectively doing what simple heat exchangers cannot. We exploit this capability to enhance heat integration.

Caution

Throughout most of this dissertation, “absorber” refers to an absorption column in an acid-gas-capture unit. In this section, “absorber” refers to a unit within the heat pump.

These two units are entirely unrelated.

2.5.1 History of waste heat recovery through heat pump integration

The first absorption-driven heat pump was patented in 1860.²⁶ A modern, single-stage absorption-driven heat pump was patented in 1924,²⁷ six years before the modern CO₂-capture process.⁵

We have found publications on waste heat recovery starting in the late 1970's. A 1980 patent described a method for augmenting a reboiler using single-stage absorption-driven heat pump.²⁸ Another 1980 patent described designs for single, double-, triple, and quadruple-stage absorption-driven heat pumps.²⁹ The triple-stage absorption-driven heat pump appears to have been patented again in 1993.³⁰

In 2013, Zhang *et al.* reported³¹ a method for augmenting a CO₂ capture process by using an absorption-driven heat pump in conjunction with the regeneration column. Zhang *et al.* reported an energy savings of just 2.62%. Their reported design is fundamentally different from the design discussed in this section.

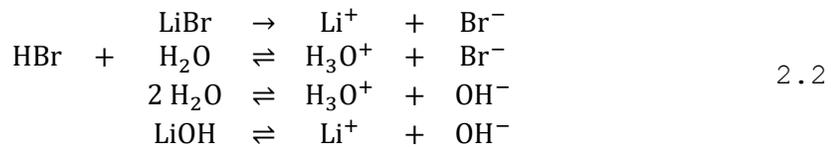
2.5.2 Understanding a lithium bromide heat pump

We will focus on a lithium bromide heat pump. This heat pump is absorption-driven. By contrast, most modern air conditioners use a compression-driven heat pump. From an outsider's point of view, the main difference is driving force; while modern air conditioners need a lot of electricity to run, our absorption-driven heat pump is mostly powered by heat from utility steam.

2.5.2.1 Aqueous lithium bromide

Our working solution is aqueous lithium bromide (H₂O + LiBr). Aqueous lithium bromide is similar to normal salt water (H₂O + NaCl) except lithium bromide is far more soluble than sodium chloride.

Equation 2.2 lists our primary reactions.



2.5.2.2 Anatomy

Table 2.3 summarizes the four primary units of our heat pump.

Table 2.3. Major parts of the heat pump

Name	Pressure	Streams				Description
		Internal		External		
		In	Out	In	Out	
Absorber	low	<ul style="list-style-type: none"> working solution (conc.) water (vapor) 	<ul style="list-style-type: none"> working solution (full) 	<ul style="list-style-type: none"> heating recipient 	<ul style="list-style-type: none"> heating recipient 	Creates the full working solution from the concentrated working solution and water. Provides heat from the water vapor being absorbed to the heating recipient.
Condenser	high	<ul style="list-style-type: none"> water (vapor) 	<ul style="list-style-type: none"> water (liquid) 	<ul style="list-style-type: none"> heating recipient 	<ul style="list-style-type: none"> heating recipient 	Condenses stream into liquid water. Provides heat from condensation to the heating recipient.
Evaporator	low	<ul style="list-style-type: none"> water (vapor + liquid) 	<ul style="list-style-type: none"> water (vapor) 	<ul style="list-style-type: none"> cooling recipient 	<ul style="list-style-type: none"> cooling recipient 	Vaporizes water using waste heat. Provides cooling to the waste heat source.
Generator	high	<ul style="list-style-type: none"> working solution (full) 	<ul style="list-style-type: none"> working solution (conc.) water (vapor) 	<ul style="list-style-type: none"> utility steam 	<ul style="list-style-type: none"> liquid water 	Concentrates working solution by boiling off part of its water. Uses high-pressure utility steam.

We show the conceptual flowsheet in Figure 2.38. In addition to the four major parts, the flowsheet has a cross heat exchanger, a pump, and two valves. The cross heat exchanger provides some heat integration. The pump and valves effectively split the heat pump into a high-pressure side and low-pressure side. Figure 2.38 shows the division between the high-pressure and low-pressure sides.

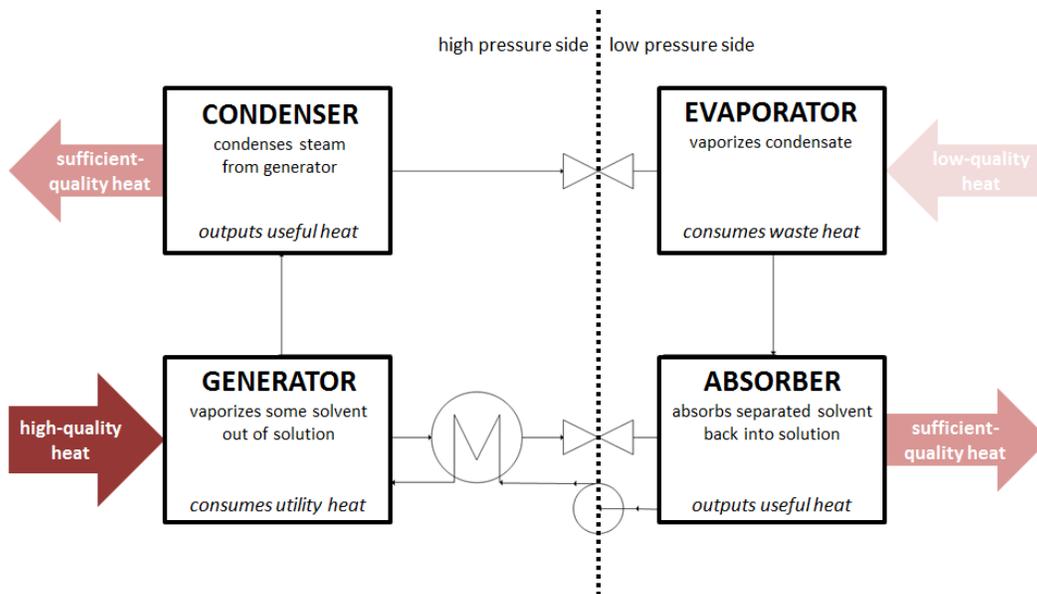


Figure 2.38. Four major units in our heat pump. Also shown are a cross heat exchanger, pump, and two valves.

The pressure differential shown in Figure 2.38 is central to how the heat pump works. By modifying the pressures, we can control the temperatures that the units operate at. For example, the evaporator boils water at $P_{\text{HP-low}}$, so we can select a desired operating temperature $T_{\text{HP-VAP}}$ by selecting the pressure according to water's phase transition function.

Figure 2.39 illustrates our conceptual flowsheet. We see that the material in the heat pump is completely sealed in such that the working solution (aqueous lithium bromide) is never exposed to an outside stream. The working solution interacts with the outside world through heat-exchanging surfaces in the four major unit operations.

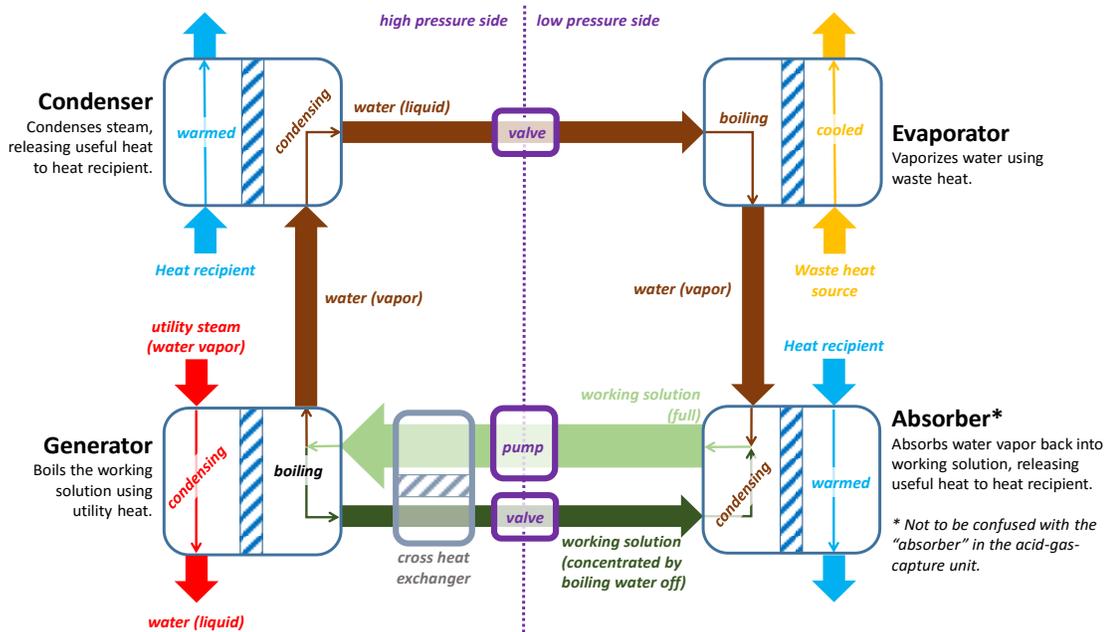


Figure 2.39. Conceptual heat pump flowsheet. Units are represented by blocks; streams are arrows; heat-exchanging surfaces are striped rectangles.

2.5.3 Aspen Plus flowsheet

We will make a quick, simple flowsheet from scratch.

2.5.3.1 Create the basic simulation

First we will copy/paste our unit set from our prior simulation:

1. Open a recent copy of our prior simulation.
2. Copy the unit set **UNITS** as shown in Figure 2.40.
3. Paste the unit set **UNITS** into our new simulation as shown in Figure 2.41.

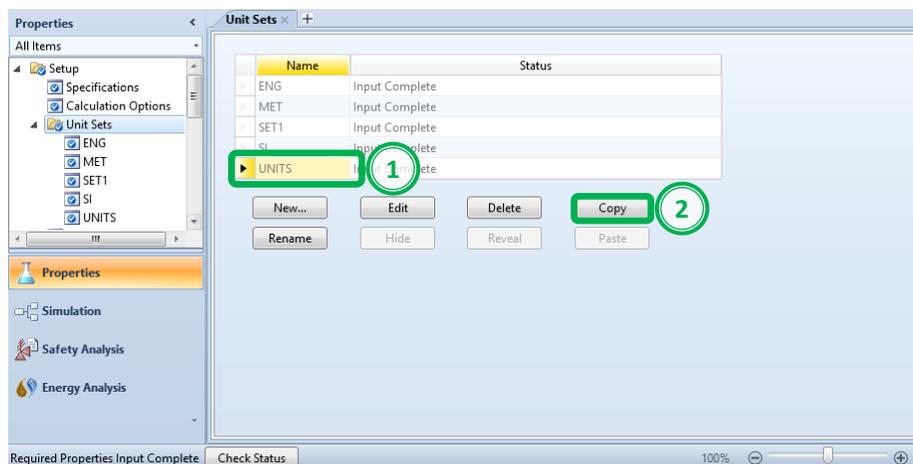


Figure 2.40. Copy **UNITS** from our prior simulation to save time and avoid potential inconsistency between simulations.

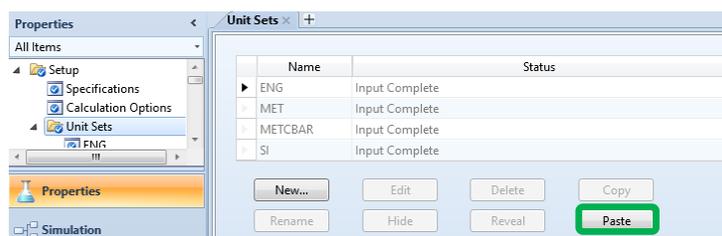


Figure 2.41. Paste **UNITS** into our new simulation.

Tip

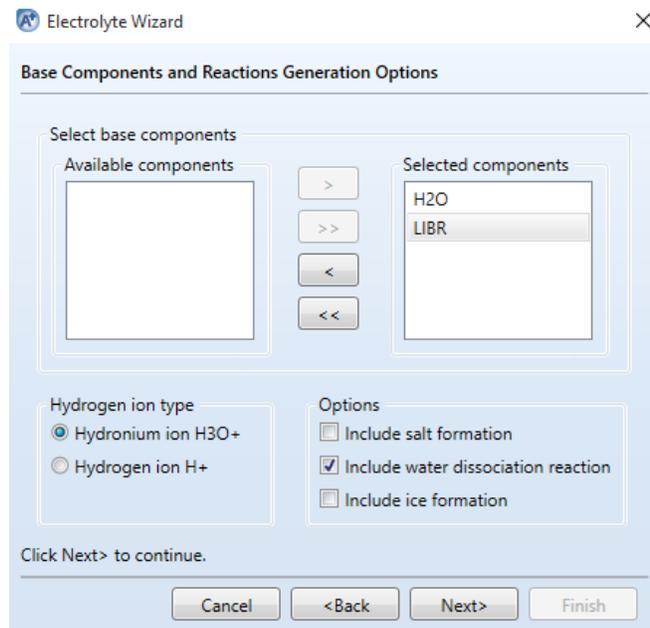
Copying a block from one simulation will not automatically enable the corresponding “Paste” buttons in other simulations. If a Paste button is inappropriately greyed out, navigate away from the page containing the button and then navigate back.

Tip

In 1999, NASA lost the Mars Climate Orbiter (\$327.6 million) due to using two different sets of units in their software.¹ Avoid their costly mistake by using a consistent set of units in all of your simulations!

Next, enter water and lithium bromide as the chemical species. Use the Electrolyte Wizard function on the same tab to generate addition reactions and properties. In the Electrolyte Wizard:

1. Use the unsymmetric reference state.
 - a. This tells Aspen Plus to use infinite dilution in an aqueous solution as the reference state for ionic components.
 - b. By contrast, the symmetric reference state uses pure fused salts as the reference state for ionic components.
2. Set reactions:



3. Set up with the property method **ENRTL-RK**.
4. Use the true component approach.
5. Go to

Aspen Plus | Properties | Methods → Parameters → Electrolyte Pair
and select the tree view nodes **GMENCC-1**, **GMENCD-1**, **GMENCE-1**, and **GMENCN-1** to have Aspen Plus automatically fill out the ENRTL parameters from its databases.

2.5.3.2 Analyze behavior

We now decide the basic behavior for integrating the heat pump.

First, we decide that the heat pump will assist the standard stripper's reboiler, first heating the stripper bottoms before the reboiler does. So, instead of using the stripper's reboiler **STR-REB** to perform all

heating, we instead use the heat pump for part of the heating and then **STR-REB** performs any additional heating that might be needed. Additionally, we note that the heat pump provides useful heating from two internal units: its absorber (not the absorption tower) and its condenser. We use the absorber first and then the condenser second. Figure 2.42 shows this configuration.

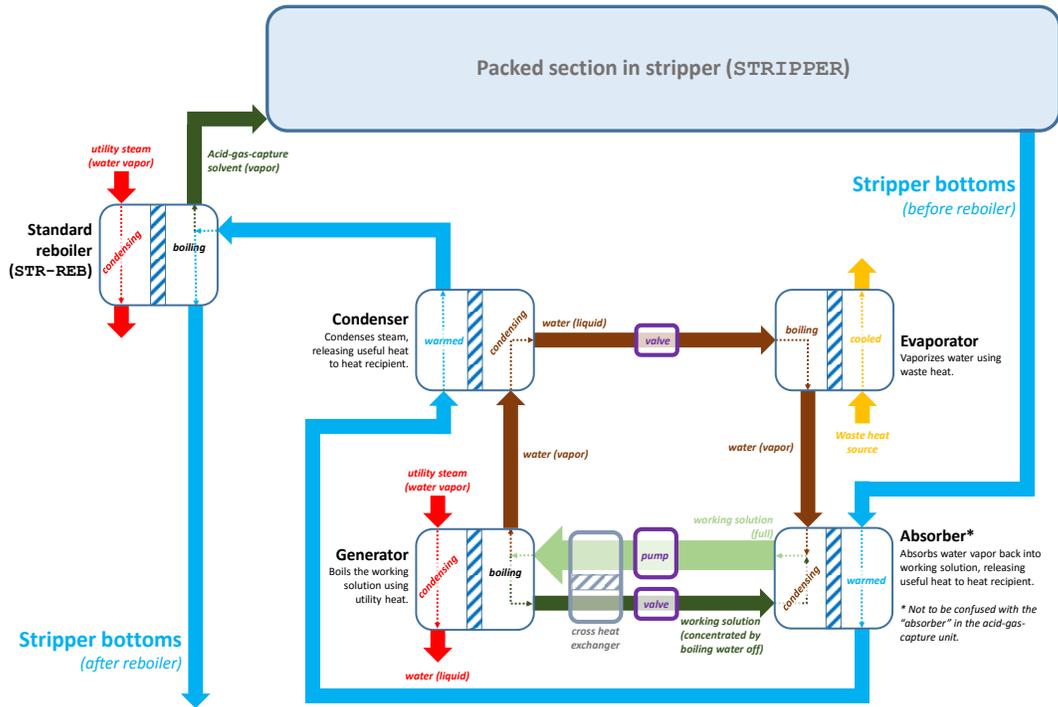


Figure 2.42. Conceptual flowsheet showing how we will integrate the heat pump into our flowsheet. The heat pump provides useful heating to the stripper's bottom stream before that stream is reboiled.

Figure 2.42 shows how to use the heat pump in conjunction with a standard reboiler. Equation 2.3.a defines the heat pump extent χ_{HP} as the portion of effect reboiler duty coming from the heat pump portion of the reboiler. Equation 2.3.b shows the effective calculation for the heat pump extent, χ_{HP} , for our configuration in Figure 2.42 that provides useful heating from absorber_{HP} and condenser_{HP}.

$$\chi_{HP} \equiv \frac{D_{HP}}{D_{STR-REB} + D_{HP}} \quad 2.3.a$$

$$\chi_{HP} = \frac{D_{HP-ABS} + D_{HP-CON}}{D_{STR-REB} + D_{HP-ABS} + D_{HP-CON}} \quad 2.3.b$$

where:

- $D_{HP-ABS} \equiv$ duty of absorber_{HP}
- $D_{HP-CON} \equiv$ duty of condenser_{HP}
- $D_{HP} \equiv$ heat pump's useful duty
- $D_{STR-REB} \equiv$ stripper's reboiler duty

We make the following observations about design parameter selection:

- i. The absorber_{HP} must combine the full solvent with concentration C_{high} at P_{low} to achieve a vapor fraction of zero (fully liquid) at a temperature above the bottoms stream plus a reasonable temperature approach.
- ii. The condenser_{HP} must condense water at P_{high} to achieve a vapor fraction of zero (fully liquid) at a temperature above the bottoms stream plus a reasonable temperature approach.
- iii. The evaporator_{HP} must vaporize water at P_{low} to achieve a vapor fraction of one (fully vapor) at temperature below the waste heat recovery temperature T_{waste}

We want to observe the response curve for P_{high} in the condenser_{HP}. First, create **PS-PROPS** as shown in Figure 2.43. This **Property Set** block includes temperature T , pressure P , and enthalpy H .

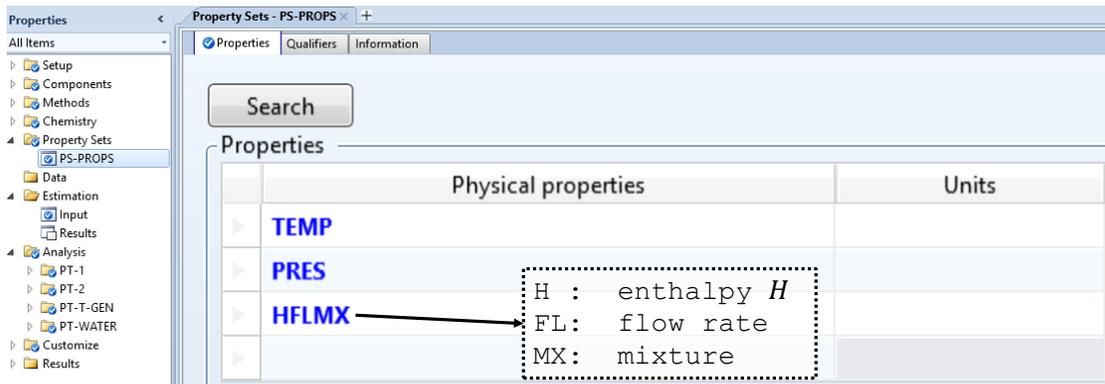


Figure 2.43. **Property Set** block **PS-PROPS**. Be sure to include temperature, pressure, and enthalpy flow rate for a mixture.

Next, create an **Analysis** block **PT-WATER** as shown in Figure 2.44. This analysis will provide the temperature, pressure, and enthalpy flow rate as specified in **PS-PROPS** for water from pressure

0.001bar to 10bar in 0.001bar increments. These results are essentially as you might find on a steam table, though using the currently specified property methods.

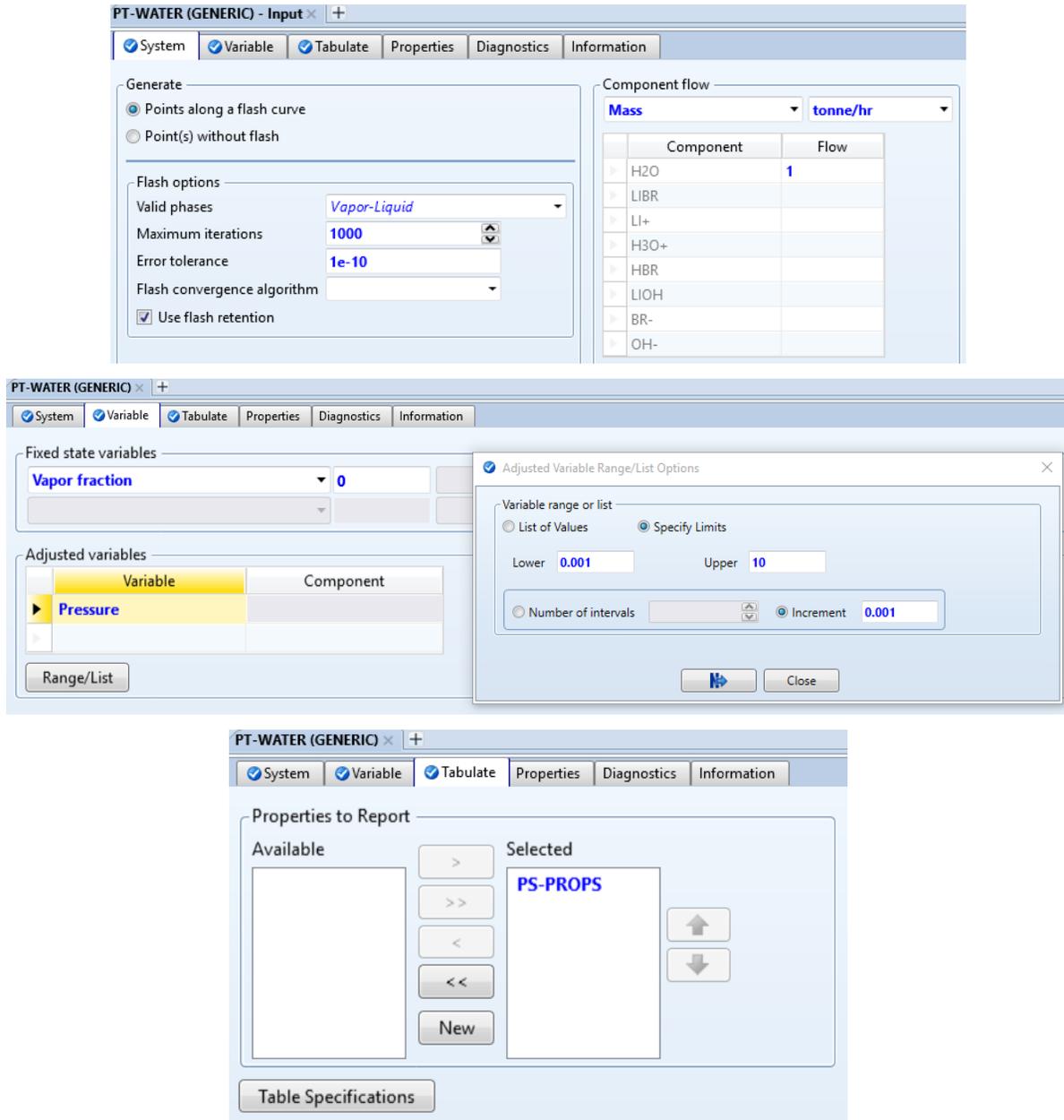


Figure 2.44. Specifications for **PT-WATER**.

Next, run the simulation and copy/paste the results into a new Excel sheet as shown in Figure 2.45. We have fields for $T_{STR-REB} = 110^{\circ}\text{C}$ and $\Delta T = 5\text{K}$, calculating $T_{CON} = T_{STR-REB} + \Delta T$. We then look up the corresponding pressure for T_{CON} using Excel's **VLOOKUP** function, `=VLOOKUP (E13, K14 :`

$\$M\$10013, 2, \text{TRUE}$). This command instructs Excel to look up $T_{\text{CON}} \equiv [E13]$ in the table we just copy/pasted $\{[K14]: [M10013]\}$ and return the value in the second column of the corresponding row. This retrieved value is P_{CON} , and since the condenser_{HP} is in the high-pressure region, $P_{\text{high}} = P_{\text{CON}}$.

	D	E	F	G	H	I	J	K	L	M	N
10							PT-WATER				
11	T_REB:	110	degC								
12	delT:	5	K				PRES	TOTAL TEM	TOTAL PR	TOTAL HFLMX	
13	T_CON:	115	degC	(minimum)			bar	C	bar	MW	
14	P_high:	1.685	bar	(minimum)			0.001	-24.7753	0.001	-4.46803	
15							0.002	-16.2267	0.002	-4.45803	
16							0.003	-10.9478	0.003	-4.45185	
17							0.004	-7.06877	0.004	-4.44731	
18							0.005	-3.98003	0.005	-4.44369	
19							0.006	-1.40261	0.006	-4.44067	
20							0.007	0.815398	0.007	-4.4381	
21							0.008	2.766195	0.008	-4.43582	

Figure 2.45. Excel sheet with copy/pasted results from **PT-WATER**.

Next, we know that the vapor fraction leaving absorber_{HP} must be zero at P_{low} and C_{high} , so we construct a boiling-point curve for P_{low} against C_{high} in a new **Property Set** block, **PT-HPABS**, as shown in Figure 2.46.

PT-HPABS (GENERIC) - Input

System Variable Tabulate Properties Diagnostics Information

Generate

Points along a flash curve
 Point(s) without flash

Flash options

Valid phases: Vapor-Liquid
Maximum iterations: 1000
Error tolerance: 1e-10
Flash convergence algorithm:
 Use flash retention

Component flow

Mass tonne/hr

Component	Flow
H2O	1
LIBR	
LI+	
H3O+	
HBR	
LI0H	
BR-	
OH-	

Fixed state variables

Temperature	115	C
Vapor fraction	0	

Adjusted variables

Variable	Component
Mass fraction	LIBR

Range/List

Adjusted Variable Range/List Options

Variable range or list

List of Values Specify Limits

Lower: 0 Upper: 0.999

Number of intervals Increment: 0.001

Close

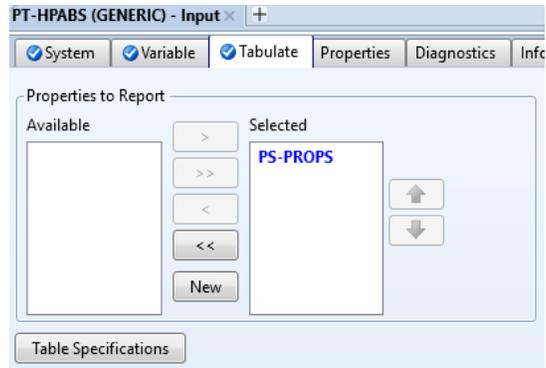


Figure 2.46. Specifications for **PT-HPABS**.

Plotting the results of this **Analysis** gives us the curve shown in Figure 2.47. In general, we are interested in mass portions above $0.5 \frac{\text{kg LiBr}}{\text{kg solution}}$. We also note that our property models do not properly recognize the solubility limited, so it is necessary for us to manually ensure that we do not exceed the solubility limit. Be sure to include these results from **PT-HPABS** in the Excel sheet along with those from **PT-WATER**.

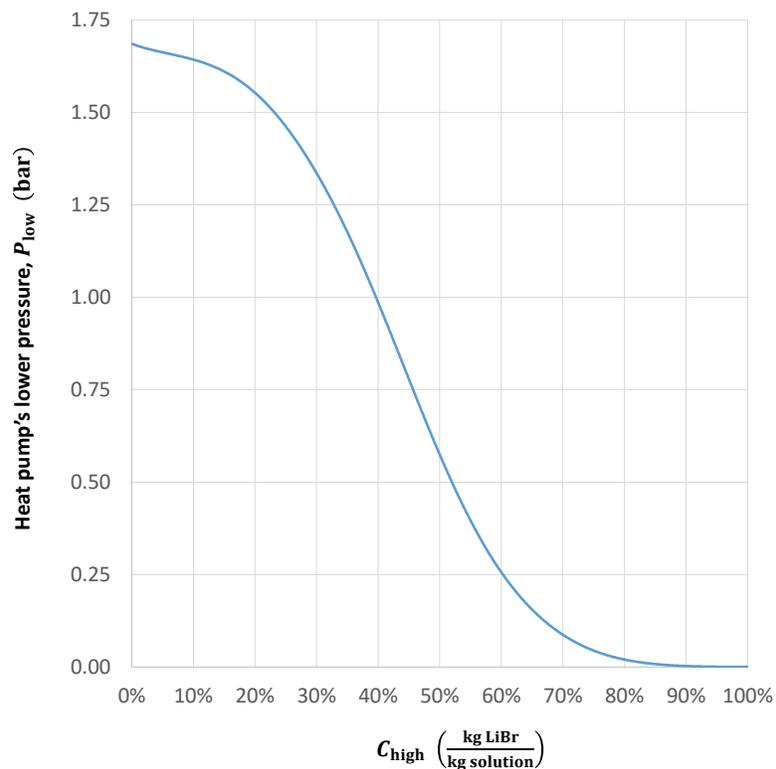


Figure 2.47. Maximum pressure for the low-pressure section of the heat pump, P_{low} /bar, as a function of the higher lithium bromide mass portion.

Next, we note that the evaporator_{HP} operates at P_{low} and $T_{HP-VAP} \approx T_{STR-REB} + \Delta T$, so we can connect our choice of C_{high} to P_{low} using the above plot in conjunction with the steam table previously generated. The Excel sheet and resulting plot are shown in Figure 2.48.

				PT-WATER				PT-HPABS						
T_REB:	110 degC			PRES	TOTAL TEN	TOTAL PRI	TOTAL HFLMX	MASSFRA	TOTAL TEN	TOTAL PRI	TOTAL HFLMX	C_low	T_CON	T_waste
delT:	5 K			bar	C	bar	MW	C	bar	MW				
T_CON:	115 degC	(minimum)		0.001	-24.7753	0.001	-4.46803	0	115	1.685992	-4.30505	0	114.9819	119.9819
P_high:	1.685 bar	(minimum)		0.002	-16.2267	0.002	-4.45803	0.001	115	1.685318	-4.30204	0.001	114.9819	119.9819
				0.003	-10.9478	0.003	-4.45185	0.002	115	1.684671	-4.29903	0.002	114.9637	119.9637
				0.004	-7.06877	0.004	-4.44731	0.003	115	1.684024	-4.29602	0.003	114.9637	119.9637
				0.005	-3.98003	0.005	-4.44369	0.004	115	1.683377	-4.29301	0.004	114.9455	119.9455
				0.006	-1.40261	0.006	-4.44067	0.005	115	1.682730	-4.29001	0.005	114.9272	119.9272
				0.007				0.006	115	1.682083	-4.28700	0.006	114.9272	119.9272
				0.008				0.007	115	1.681436	-4.28400	0.007	114.909	119.909
				0.009	4.510125	0.009	-4.43379	0.008	115	1.681083	-4.28098	0.008	114.909	119.909
				0.01	6.08891	0.01	-4.43194	0.009	115	1.680730	-4.27797	0.009	114.8907	119.8907
				0.011	7.532642	0.011	-4.43026	0.01	115	1.680377	-4.27496	0.01	114.8725	119.8725
				0.012	8.863764	0.012	-4.42871	0.011	115	1.679478	-4.27195	0.011	114.8725	119.8725

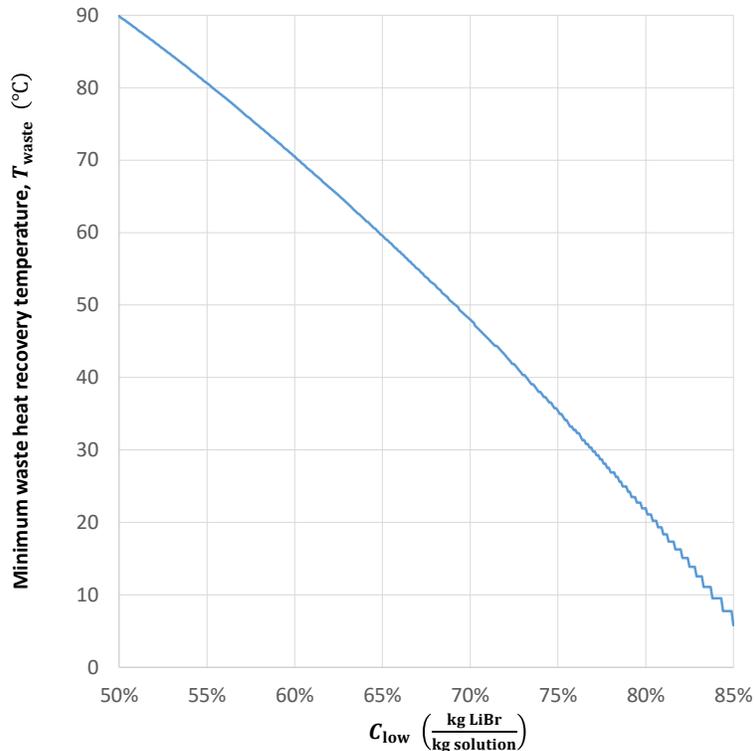


Figure 2.48. Minimum temperature that the heat pump can recover waste heat at as a function of the selected mass portion for the full working solution, C_{low} .

Finally, we can assert that the higher concentration, C_{high} , is $0.05 \frac{\text{kg LiBr}}{\text{kg solution}}$ above C_{low} . Start by creating a new **Property Set** block, **PT-T-GEN**, as shown in Figure 2.49.

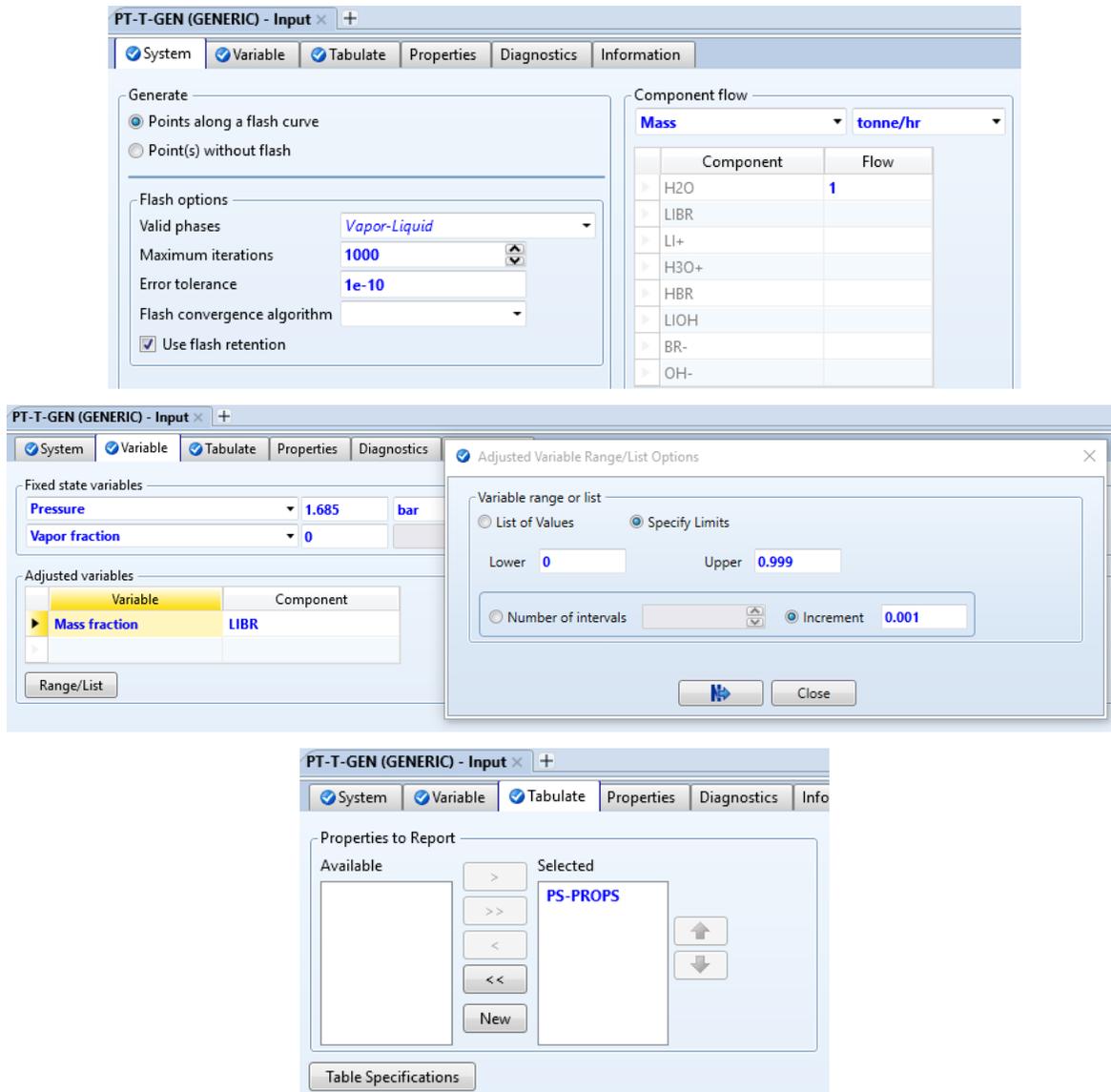


Figure 2.49. Specifications for **PT-T-GEN**.

To update the Excel sheet, we extend the given values, add the results of **PT-T-GEN**, and calculate the curves for T_{steam} against T_{waste} as shown in Figure 2.50.

	C	D	E	F	G	H
10						
11		T_REB:	110 degC			
12		delT:	5 K			
13		T_CON:	115 degC		(minimum)	
14		P_high:	1.685 bar		(minimum)	
15		delC	0.05			
16		eta_Carnot	0.75			
17		T_turbine	40 degC			
18			313.15 K			

	Y	Z	AA	AB	AC	AD	AE	AF	AG	AH
10	PT-T-GEN									
11										
12	MASSFRA	TOTAL TEM	TOTAL PRI	TOTAL HFLMX			T_waste	C_high	T_steam	T_GEN
13		C	bar	MW						
14	0	114.9819	1.685	-4.30507			119.9819	0.05	115.4115	120.4115
15	0.001	114.9942	1.685	-4.30205			119.9819	0.051	115.4183	120.4183
16	0.002	115.006	1.685	-4.29903			119.9637	0.052	115.4251	120.4251
17	0.003	115.0175	1.685	-4.296						
18	0.004	115.0287	1.685	-4.29298						
19	0.005	115.0397	1.685	-4.28996						
20	0.006	115.0505	1.685	-4.28694						
21	0.007	115.0611	1.685	-4.28392			119.909	0.057	115.4589	120.4589
22	0.008	115.0715	1.685	-4.2809			119.909	0.058	115.4656	120.4656
23	0.009	115.0817	1.685	-4.27788			119.8907	0.059	115.4724	120.4724

AE: =W14
 AF: =U14+\$E\$15
 AG: =VLOOKUP(AF14,\$Y\$14:\$Z\$1013,2,TRUE)
 AH: =AG14+\$E\$12

Results from PT-T-GEN

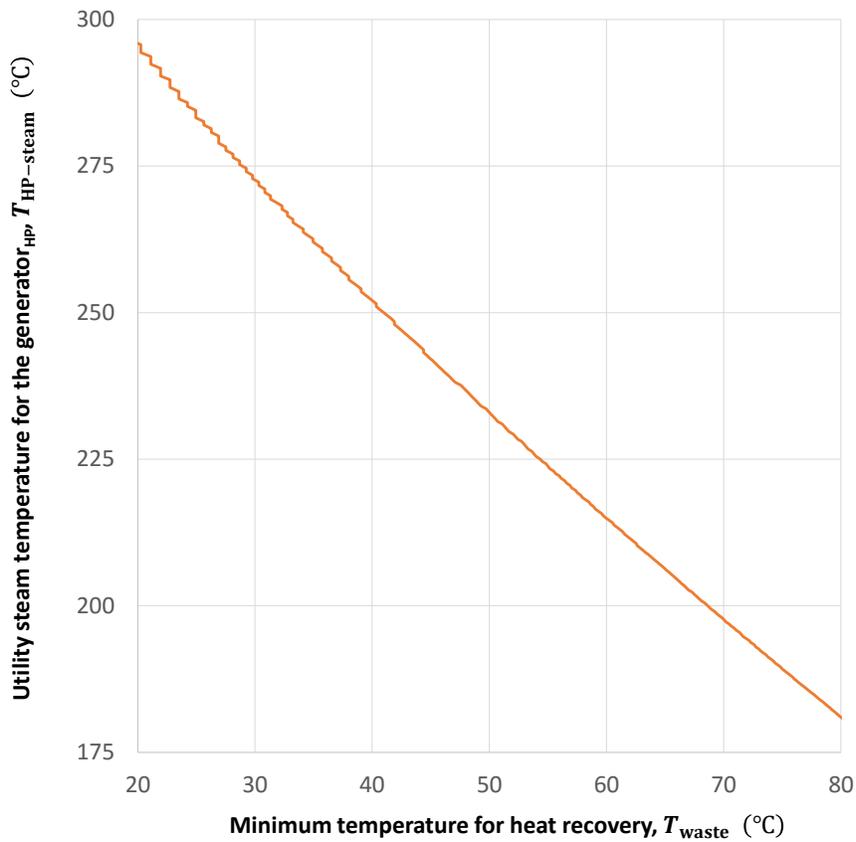


Figure 2.50. Additions to our Excel sheet and the resulting plot for $T_{HP-steam}$ against T_{waste} .

At this point, we have fairly reasonable estimates for the heat pump's specifications as a function of a selected waste heat recovery temperature, $T_{HP-waste}$. Next, we examine our process to determine

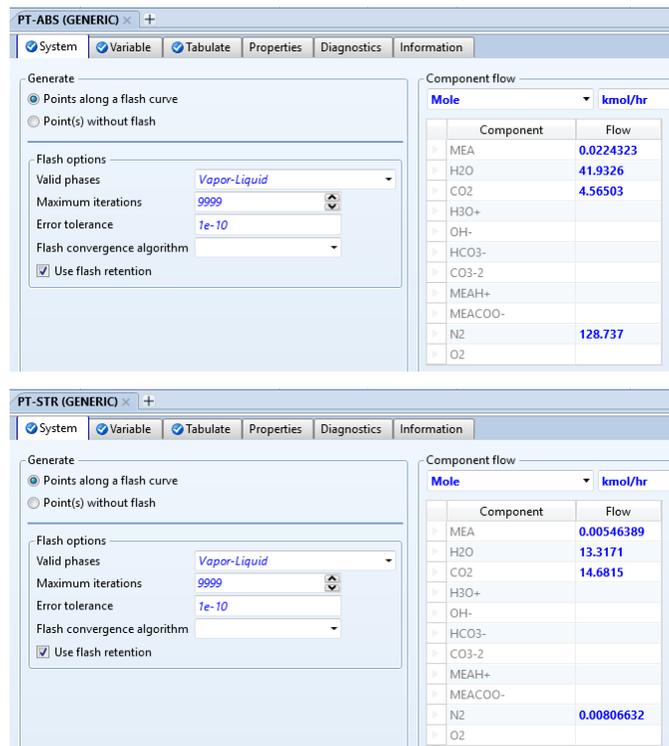
potential sources of recoverable waste heat. We choose to select the vapor streams coming from the columns' packed sections into their condensers. Table 2.4 shows the results from one particular simulation for streams **ABS-TC-2** and **STR-TC-2**.

Table 2.4. Stream results for vapor flows from columns' packed sections to their condensers.

ABS-TC-2 is for the absorber while **STR-TC-2** is for the stripper.

	Mole flow rate $\left(\frac{\text{kmol}}{\text{hr}}\right)$			Pressure (bar)	Temperature (°C)
	MEA	H ₂ O	CO ₂		
ABS-TC-2	0.022	41.933	4.565	1.000	65.797
STR-TC-2	0.005	13.317	14.682	1.400	91.578

Next, we copy/paste the **Property Set** block from our recent heat pump simulation, **PS-PROPS**, into our CO₂-capture simulation and use them in the two new **Analysis** blocks, **PT-ABS** and **PT-STR**, shown in Figure 2.51.



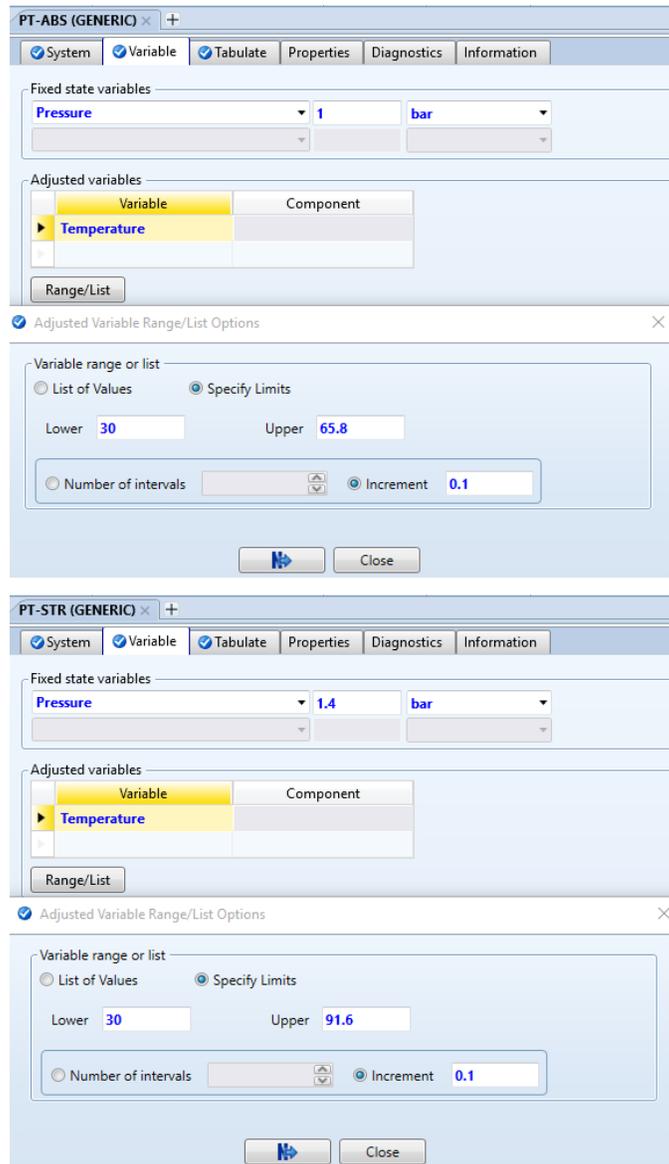


Figure 2.51. **Analysis** blocks **PT-ABS** and **PT-STR** in the CO₂-capture simulation.

These analyses construct waste heat recovery curves.

Once we run these analyses, we can add the results to our Excel sheet and find the recoverable heat duty at possible values for T_{waste} . Figure 2.52 shows the results.

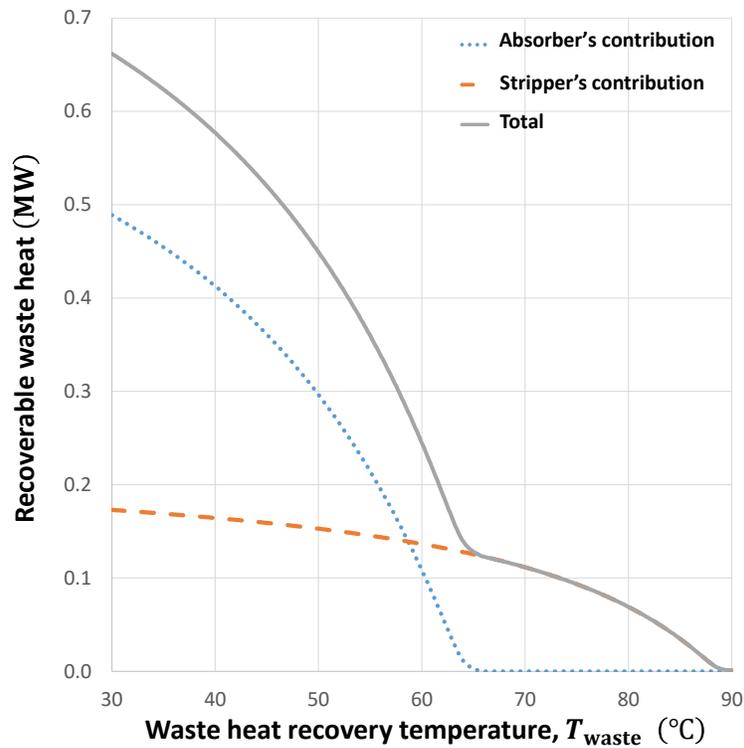


Figure 2.52. Recoverable waste heat at various waste heat temperatures T_{waste} .

2.5.3.3 Draw the flowsheet in Aspen Plus

Now that we have generated estimates and a basic understanding for our heat pump, we will draw the flowsheet in our simulation that already contains the properties. Figure 2.53 shows the flowsheet, including blocks, streams, and names. Table 2.5 gives the corresponding flowsheet-level specifications.

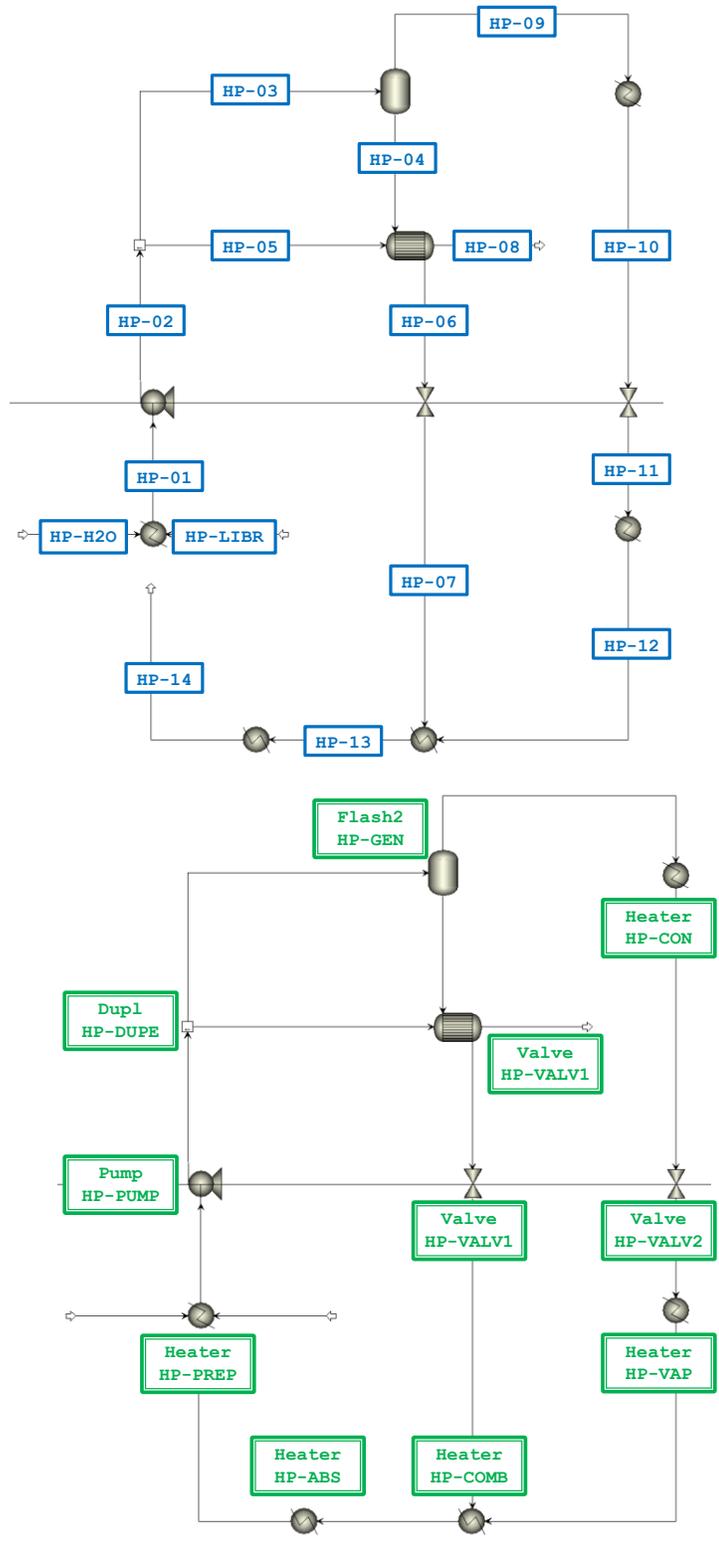


Figure 2.53. Aspen Plus flowsheet for the heat pump. The left image shows the material stream names. The right image shows the blocks' types and names.

Table 2.5. Heat pump flowsheet specifications.

Name	Block Type	Specifications	Description
HP-ABS	Heater	$\Delta P = 0$ $v = 0$	Represents the absorber _{HP} .
HP-CHEX	HeatX	$A = 50\text{m}^2$	Represents the heat pump's cross heat exchanger.
HP-COMB	Heater	$D = 0$ $\Delta P = 0$	A Heater block that flashes the input to the absorber _{HP} . Not an actual physical unit.
HP-CON	Heater	$\Delta P = 0$ $v = 0$	Represents the condenser _{HP} .
HP-DUPE	Dup1	--	Duplicates the stream feeding into the generator _{HP} . Not an actual physical unit.
HP-GEN	Flash2	$\Delta P = 0$ $v = 0.1$	Represents the generator _{HP} .
HP-H2O	Material	$T = 40^\circ\text{C}$ $P = 1\text{bar}$ $F = 1 \frac{\text{tonne}}{\text{hr}}$ $m_{\text{H}_2\text{O}} = 1$ Disable "Calculate stream properties".	Stream specifying the water specification in the full working solution.
HP-LIBR	Material	$T = 40^\circ\text{C}$ $P = 1\text{bar}$ $F = 1 \frac{\text{tonne}}{\text{hr}}$ $m_{\text{LiBr}} = 1$ Disable "Calculate stream properties".	Stream specifying the salt specification in the full working solution.
HP-PREP	Heater	$P = 1\text{bar}$ $v = 0$	A Heater block that flashes the full working solution from the specifying streams. Not an actual physical unit.
HP-PUMP	Pump	$P = 1\text{bar}$	Represents the heat pump's solution pump.

HP-VALV1	Valve	$P = 1\text{bar}$	Represents the flash pressure drop from P_{high} to P_{low} for the concentrated working solution.
HP-VALV2	Valve	$P = 1\text{bar}$	Represents the flash pressure drop from P_{high} to P_{low} for the separated water branch.
HP-VAP	Heater	$\Delta P = 0$ $v = 0$	Represents the evaporator _{HP} .

In addition to these heat pump units, we draw additional units to assist us in calculating the heat pump's properties. Figure 2.54 shows these additional units and we draw these units anywhere on the flowsheet since they do not visibly connect to the other units. Table 2.6 gives the corresponding flowsheet specifications.

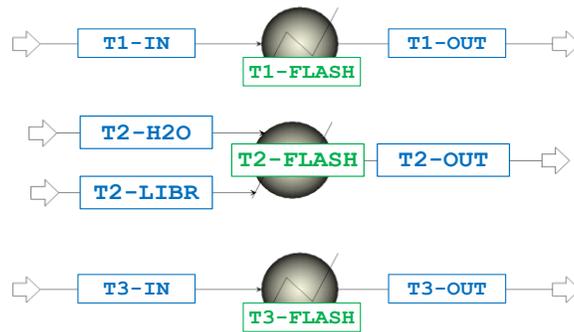


Figure 2.54. Additional flowsheet units for the heat pump simulation. These units are not connected to the rest of the flowsheet by visible streams, so they may be placed at any location on the flowsheet. We arbitrary selected to put them on the left side of the simulation.

Table 2.6. Flowsheet-level specifications for the additional estimation units in the heat pump simulation.

Name	Block Type	Specifications	Description
T1-FLASH	Flash2	$T = 40^{\circ}\text{C}$ $v = 1$	Flash as a proxy for the evaporator _{HP} .

T1-IN	Material	$T = 40^{\circ}\text{C}$ $P = 1\text{bar}$ $F = 1 \frac{\text{tonne}}{\text{hr}}$ $m_{\text{H}_2\text{O}} = 1$ Disable "Calculate stream properties".	Water as a proxy for the feed into the evaporator _{HP} .
T2-FLASH	Flash2	$P = 1\text{bar}$ $v = 0$	Flash as a proxy for the absorber _{HP} .
T2-H2O	Material	$T = 40^{\circ}\text{C}$ $P = 1\text{bar}$ $F = 1 \frac{\text{tonne}}{\text{hr}}$ $m_{\text{H}_2\text{O}} = 1$ Disable "Calculate stream properties".	Water as a proxy for aqueous component of the feed into the absorber _{HP} .
T2-LIBR	Material	$T = 40^{\circ}\text{C}$ $P = 1\text{bar}$ $F = 1 \frac{\text{tonne}}{\text{hr}}$ $m_{\text{LiBr}} = 1$ Disable "Calculate stream properties".	Lithium bromide as a proxy for aqueous component of the feed into the absorber _{HP} .
T3-FLASH	Flash2	$T = 115^{\circ}\text{C}$ $v = 0$	Flash as a proxy for the condenser _{HP} .
T3-IN	Material	$T = 40^{\circ}\text{C}$ $P = 1\text{bar}$ $F = 1 \frac{\text{tonne}}{\text{hr}}$ $m_{\text{H}_2\text{O}} = 1$ Disable "Calculate stream properties".	Water as a proxy for the feed into the condenser _{HP} .

2.5.3.4 Add flowsheet-level logic

First, create our standard **C-GLOBAL Calculator** sequenced to run first with the parameters listed in Table 2.7 with the code shown in Scheme 2.12.

Table 2.7. Global parameters for **C-GLOBAL**. Define each parameter as an Export variable.

Name	Parameter Number
E	1
PI	2
ERROR	10
NEEDINIT	11
DELT	20
TSTRREB	21
TDELIVER	22
FFULL	30
DELCMIN	40
CLOW	41
CHIGH	42
TWASTE	100
PLOW	200
PHIGH	300

```

C =====
C ===== START: MATH CONSTANTS =====
C =====

C Dummy value. Should never be taken as useful.
C Any element of the program may change it without
C cause. No element of the program may read it
C with the intention of using its value as
C information.
      DUMMY = 0.0

C Error value. Should never be changed.
C Usually used to note that an assigned value
C is meaningless.

```

ERROR = -99999

C "NEEDs INITalization". This value indicates
C that a variable should be initilized in C-INIT.
C Usually these variables are NOT design elements,
C but rather useful convergence data, often initial
C estimates. These values are segregated off for
C easy modification in C-INIT. Their values should
C NOT be assigned in C-GLOBAL, other than by NEEDINIT.

NEEDINIT = 777.0

C Math constants for pi and e. Unnecessarily
C many digits are given for each to help maintain
C forward compatibility. Currently Aspen probably
C truncates these extra digits.

PI = 3.14159265358979323846264338328

E = 2.71828182845904523536028747135

C =====
C ===== END: MATH CONSTANTS =====
C =====

C delta-Temperature allowed for
C cross-heat exchangers (K):

DELT = 5.0

C Stripper's reboiler temperature (degC):

TSTRREB = 110.0

TDELIVER = TSTRREB + DELT

C Full working solution flow rate (tonne/hr):

FFULL = 1.0

C Minimum delta-Concentraton of LiBr

```
C (kg LiBr / kg solution):  
    DELCMIN = 0.05  
  
C LiBr concentrations (kg LiBr / kg solution).  
C The lower concentration is for the full  
C working solution while the higher concentration  
C is for the concentrated solution.  
    CLOW = ERROR  
    CHIGH = ERROR  
  
    TWASTE = 50.0  
  
    PLOW = ERROR  
    PHIGH = ERROR
```

Scheme 2.12. Interpreted Fortran for **C-GLOBAL**.

Next, we want to define control logic for the initial estimation units. Create five new **Calculator** blocks **C-ST1-0**, **C-ST-1**, **C-ST2-0**, **C-ST3-0**, and **C-ST3-1**. Define their variables as shown in Figure 2.55 and provide them with the code in Figure 2.56.

(a) C-ST1-0

C-ST1-0 x +

Define Calculate Sequence Tears Stream Flash Information

Active

Sampled variables (drag and drop variables from form to the grid below)

Variable	Information flow	Definition
TWASTE	Import variable	Parameter Parameter no.=100
TSET	Export variable	Block-Var Block=T1-FLASH Variable=TEMP Sentence=PARAM Units=C

New... Delete Copy Paste Move Up Move Down View Variables

(b) C-ST1-1

C-ST1-1 - Input x +

Define Calculate Sequence Tears Stream Flash Information

Active

Sampled variables (drag and drop variables from form to the grid below)

Variable	Information flow	Definition
PCALC	Import variable	Stream-Var Stream=T1-OUT Substream=MIXED Variable=PRES Units=bar
PLOW	Export variable	Parameter Parameter no.=200

New... Delete Copy Paste Move Up Move Down View Variables

(c) C-ST2-0

C-ST2-0 - Input x +

Define Calculate Sequence Tears Stream Flash Information

Active

Sampled variables (drag and drop variables from form to the grid below)

Variable	Information flow	Definition
FFULL	Import variable	Parameter Parameter no.=30
CLOW	Import variable	Parameter Parameter no.=41
FH2O	Export variable	Stream-Var Stream=T2-H2O Substream=MIXED Variable=MASS-FLOW Units=tonne/hr
FLIBR	Export variable	Stream-Var Stream=T2-LIBR Substream=MIXED Variable=MASS-FLOW Units=tonne/hr
PLOW	Import variable	Parameter Parameter no.=200
PSET	Export variable	Block-Var Block=T2-FLASH Variable=PRES Sentence=PARAM Units=bar

New... Delete Copy Paste Move Up Move Down View Variables

(d) C-ST3-0

C-ST3-0 - Input x +

Define Calculate Sequence Tears Stream Flash Information

Active

Sampled variables (drag and drop variables from form to the grid below)

Variable	Information flow	Definition
TDELIVER	Import variable	Parameter Parameter no.=22
TSET	Export variable	Block-Var Block=T3-FLASH Variable=TEMP Sentence=PARAM Units=C

New... Delete Copy Paste Move Up Move Down View Variables

(e) C-ST3-1

C-ST3-1 - Input x +

Define Calculate Sequence Tears Stream Flash Information

Active

Sampled variables (drag and drop variables from form to the grid below)

Variable	Information flow	Definition
PCALC	Import variable	Stream-Var Stream=T3-OUT Substream=MIXED Variable=PRES Units=bar
PHIGH	Export variable	Parameter Parameter no.=300

New... Delete Copy Paste Move Up Move Down View Variables

Figure 2.55. Variable definitions for the **Calculator** blocks that help to provide initial estimates for our heat pump simulation.

(a) C-ST1-0	TSET = TWASTE
(b) C-ST1-1	PLOW = PCALC
(c) C-ST2-0	FLIBR = FFULL * CLOW
	FH2O = FFULL * (1.0 - CLOW)
	PSET = PLOW
(d) C-ST3-0	TSET = TDELIVER
(e) C-ST3-1	PHIGH = PCALC

Figure 2.56. Interpreted Fortran for the Calculator blocks that help to provide initial estimates for our heat pump simulation.

Additionally, we define the **Design Spec**, **DS-ST2**, as shown in Figure 2.57.

The image shows two screenshots of the 'DS-ST2' Design Specification window. The top screenshot shows the 'Active' tab with a table of sampled variables:

Variable	Definition
TDELIVER	Parameter Parameter no.=22
TCALC	Stream-Var Stream=T2-OUT Substream=MIXED Variable=TEMP Units=C

The bottom screenshot shows the 'Design specification expressions' tab with the following values:

Spec	TCALC
Target	TDELIVER
Tolerance	1e-7

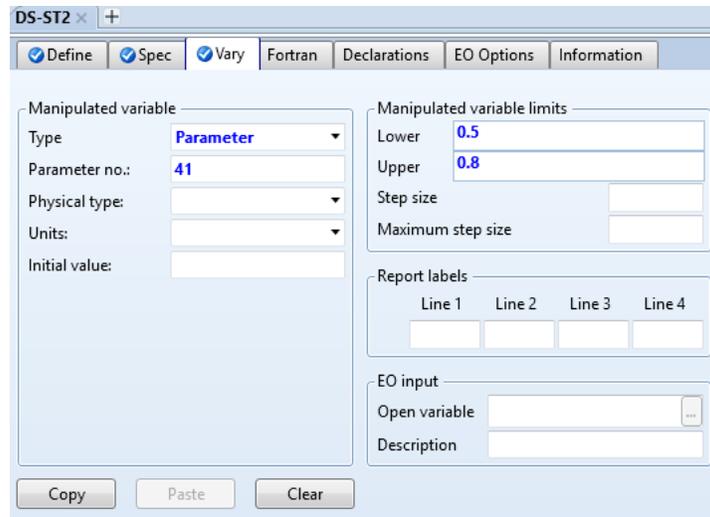


Figure 2.57. Input for the **Design Spec**, **DS-ST2**.

Encapsulate **DS-ST2** in a new **Convergence** block, **CV-T2**, using the secant approach.

We sequence these **Calculator** blocks with their corresponding flowsheet units. We also create the initial estimation **Sequence** block, **SQ-T**, defined as in Figure 2.58.

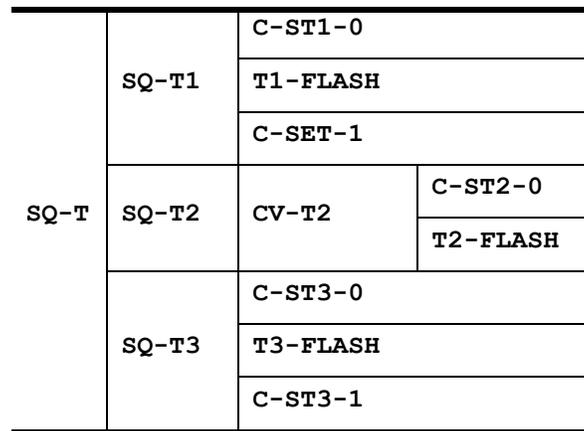


Figure 2.58. Definition for the **Sequence** coordinating the initial estimation units in our heat pump simulation, **SQ-T**.

2.5.3.5 Checkpoint

We have entered a significant amount of input into our simulation. At this point, we will stop to confirm that the specifications so far are correct and describe what we observe about the simulation.

Be sure to save the simulation under a new name, then reinitialize and run the simulation.

As always, Aspen Plus first constructs the overall run sequence. We see this overall sequence in the Control Panel as in Figure 2.59. We construct this overall sequence as:

1. **C-GLOBAL** comes first because it was sequenced as First.
2. **SQ-T** follows the sequencing previously specified in Figure 2.58.
3. Aspen Plus follows the information flow as in Figure 2.60 to sequence the remaining blocks.

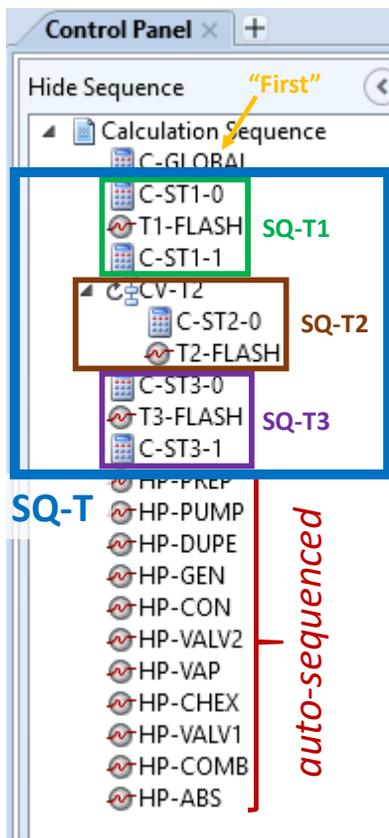


Figure 2.59. The effective sequence as calculated by Aspen Plus during the initial stages of the simulation's execution. This effective sequence exemplifies how Aspen Plus combines various types of sequence specifications to arrive at a final sequence.

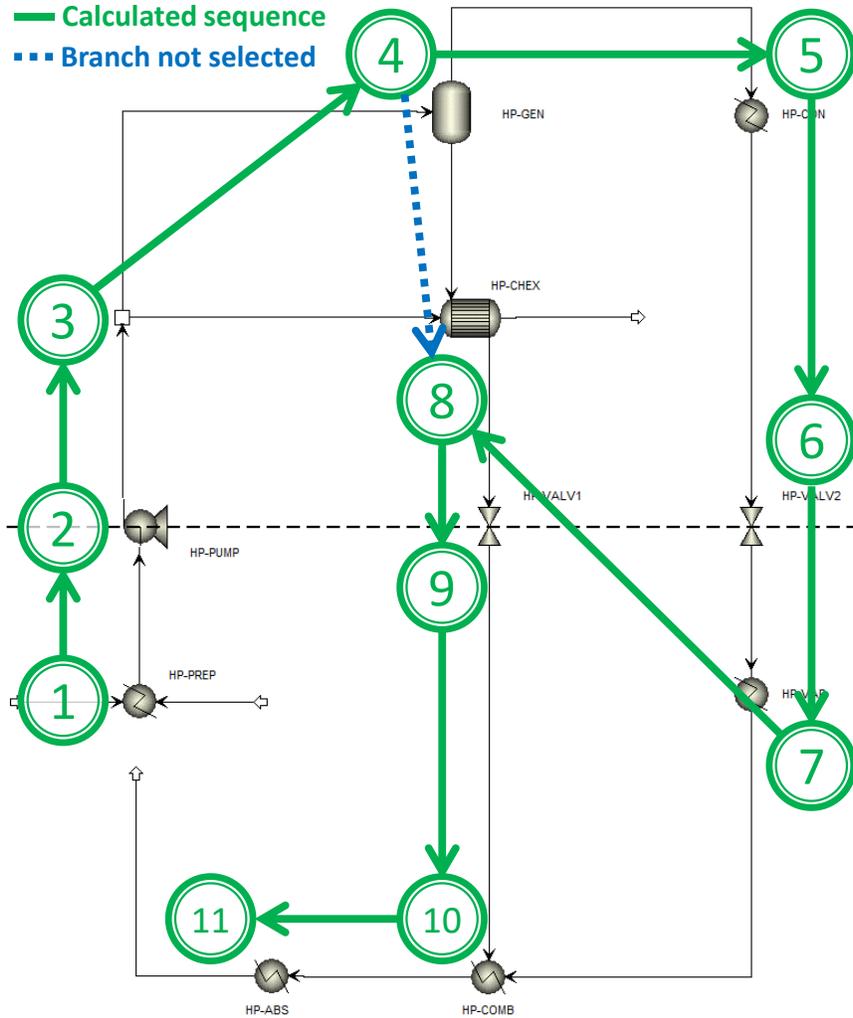


Figure 2.60. Aspen Plus auto-generated a sequence for the units that we did not include in **Sequence** blocks. Here we can see the auto-generated order (in green). Since we avoided flowsheet-level feedback loops, the calculated sequence is fairly straightforward. Aspen Plus did have the opportunity to make a selection at **HP-GEN** as either **HP-CON** or **HP-CHEX** was able to follow. Aspen Plus selected **HP-CON**, falling back to **HP-CHEX** before **HP-COMB** since **HP-COMB** required information from both branches.

The initial estimation components follow the logic we prototyped in Section 2.5.3.2: “Analyze behavior”. Figure 2.61 gives the corresponding specifications.

C-GLOBAL - Results				
Summary		Define Variable	Status	
Variable	Value read	Value written	Units	
▶ E		2.71828		
▶ PI		3.14159		
▶ ERROR		-99999		
▶ NEEDINIT		777		
▶ DELT		5		
▶ TSTRREB		110		
▶ TDELIVER		115		
▶ FFULL		1		
▶ DELCMIN		0.05		
▶ CLOW		-99999		
▶ CHIGH		-99999		
▶ TWASTE		50		
▶ PLOW		-99999		
▶ PHIGH		-99999		

Figure 2.61. Results for **C-GLOBAL**. These results reflect the minimal specifications provided for our simulation, while important information like P_{low} and P_{high} are estimated from the other data. This approach to specifying our simulations helps to make our simulations more robust, reliable, and easier to use.

In the first step of our initialization, **SQ-ST1** calculates P_{low} . Figure 2.62 shows that **C-ST1-1** sets the global parameter for P_{low} with the calculated value. In this case, we find $P_{low} \approx 0.124$ bar; however, this result will change with the simulation input in **C-GLOBAL** as well as our selected property models and parameters. A major advantage of our current approach is that, regardless of any changes in P_{low} that may result from any change in design or modeling parameters, P_{low} is bound to a correct value. Other initialized values from **SQ-ST2** and **SQ-ST3** are similarly automatic.

C-ST1-1 - Results				
Summary		Define Variable	Status	
Variable	Value read	Value written	Units	
▶ PCALC	0.123987		BAR	
▶ PLOW	-99999	0.123987		

Figure 2.62. Results for **C-ST1-1** showing the calculated value for P_{low} .

Next, **SQ-ST2** calculates C_{low} with the results shown in Figure 2.63. This information will allow us to set the input streams **HP-H2O** and **HP-LIBR** for the primary heat pump unit. In this case we find $C_{low} \approx 0.67 \frac{\text{kg LiBr}}{\text{kg solution}}$. We do not need a **Calculator** called **C-ST2-1** because **DS-ST2** already sets the global parameter for C_{low} in the process of manipulating it.

Variable	Initial value	Final value	Units
MANIPULATED	0.53	0.670403	
TDELIVER	115	115	
TCALC	81.5159	115	C

Figure 2.63. Results for **DS-ST2** showing the calculated value for C_{low} as **MANIPULATED**.

Finally, **sq-st3** calculates P_{high} with the results shown in Figure 2.64. In this case we find $P_{high} \approx 1.69$ bar.

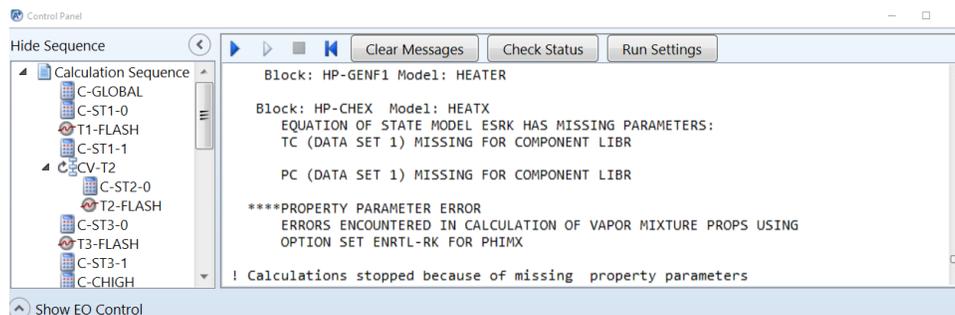
Variable	Value read	Value written	Units
PCALC	1.68599		BAR
PHIGH	-99999	1.68599	

Figure 2.64. Results for **C-ST3-1** showing the calculated value for P_{high} .

If your results are not close to those in Figure 2.62 to Figure 2.64 provided above, take this time to find any errors in your simulation before moving on. There are legitimate reasons that your results may show very minor variation from those above, such as if you use databanks with property parameter values different than those from Aspen Plus V8.8 used to write this dissertation.

2.5.3.6 Specifying work-around parameters for LiBr

While running these simulations, we occasionally run into the error shown in Figure 2.65. This error will cause the simulation to stop without providing any results, even for the sections of the simulation that has run correctly.



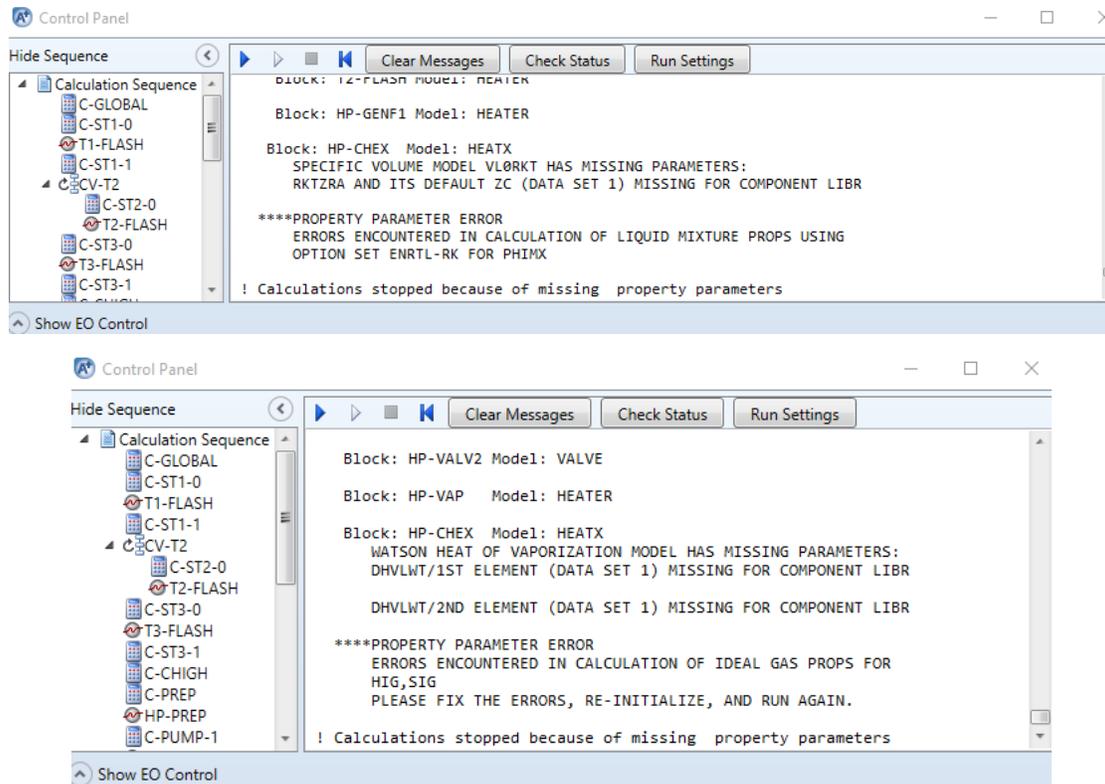


Figure 2.65. Critical run-time errors due to missing parameters for lithium bromide.

The parameters, T_c , P_c , and Z_c , are the critical-point values for the temperature, pressure, and compressibility factor of lithium bromide.

The parameter **RKTZRA** is the Rackett liquid molar volume. We will not need to provide this parameter because, if absent, it will be effectively replaced by a calculation based on Z_c .

The missing parameters **DHVLWT** are for the Watson heat of vaporization, calculated as shown in Equation 2.4. The missing variables referred to in the error are the reference molar enthalpy of vaporization, $\Delta_{\text{vap}}H_{\text{LiBr}}^{\text{ref}}$, and the reference temperature, T^{ref} , respectively.

$$\Delta_{\text{vap}}H_{\text{LiBr}} = \Delta_{\text{vap}}H_{\text{LiBr}}^{\text{ref}} \left(\frac{1 - \frac{T}{T_{\text{LiBr}}^{\text{crit}}}}{1 - \frac{T^{\text{ref}}}{T_{\text{LiBr}}^{\text{crit}}}} \right)^{\left(a_i + b_i \left(1 - \frac{T}{T_{\text{LiBr}}^{\text{crit}}} \right) \right)} \quad 2.4$$

Aspen Plus does not automatically generate these parameters when we create the simulation because it neither has nor requires them for this simulation. Aspen Plus can provide estimates if this option is selected. Figure 2.66 shows how to enable parameter estimation.

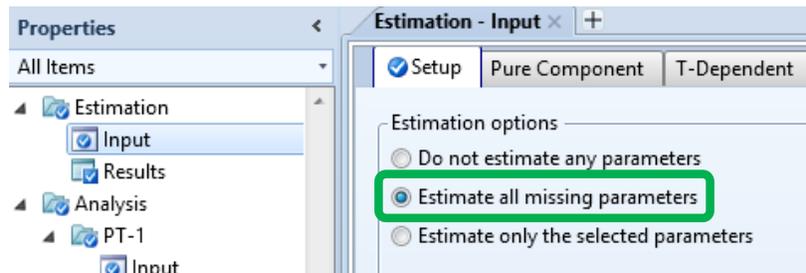
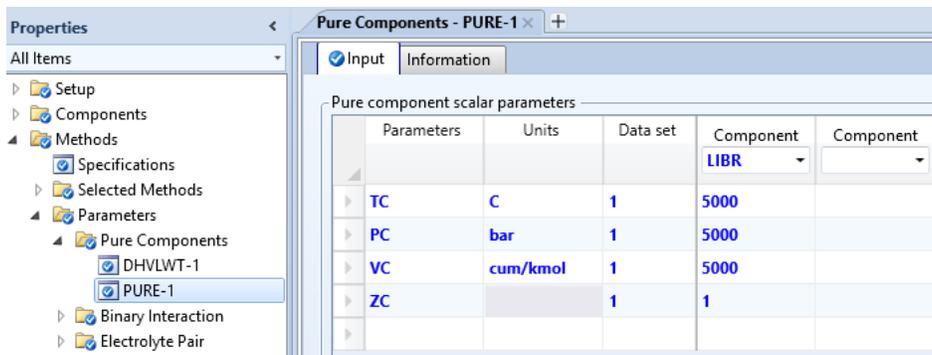


Figure 2.66. Automatic parameter estimation can be enabled using this bubble. We do not use this approach here, but it is an option.

Rather than use automatic parameter estimation, we work around this Aspen Plus error by specifying dummy values for the missing values as shown in Figure 2.67. These values are not correct, though Aspen Plus should not be using them for any calculation.



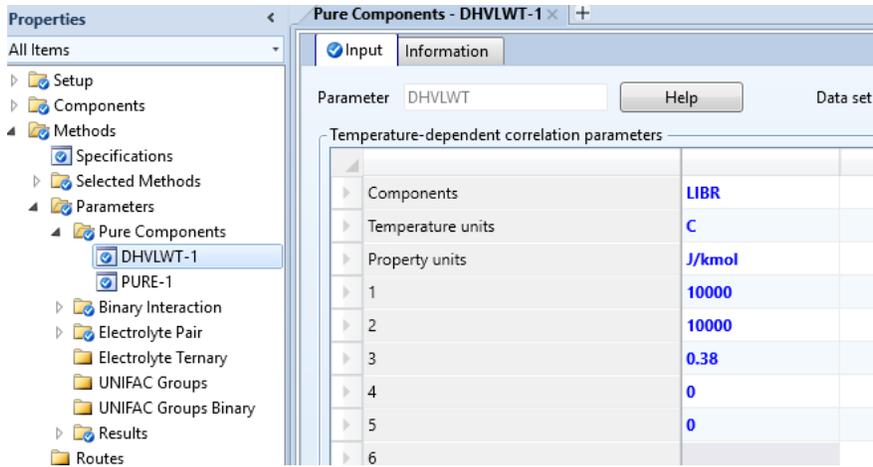


Figure 2.67. Specifications for missing property parameters for LiBr to avoid the critical run-time error.

2.5.3.7 Further flowsheet-level logic

We specify the **Flash2** block for the generator_{HP}, **HP-GEN**, in terms of zero pressure drop, $\Delta P = 0$, and an arbitrary vapor fraction. We need to have the generator_{HP} provide the concentrated working solution at its bottoms by correctly selecting the vapor fraction for **HP-GEN**. So, we will derive a new object **SQ-GEN-1** that varies the vapor fraction such that the desired working solution concentration C_{high} is achieved.

We calculate the concentrated lithium bromide mass fraction using Equation 2.5. Figure 2.68 shows a **Design Spec** block, **DS-GEN-1**, to do this.

$$C_{\text{high}} = \frac{M_{\text{LiBr}}^{\text{eff}}}{M_{\text{solution}}} = \frac{(N_{\text{Br}^-} + N_{\text{HBr}} + N_{\text{LiBr}})MW_{\text{LiBr}}}{M_{\text{solution}}} \quad 2.5$$

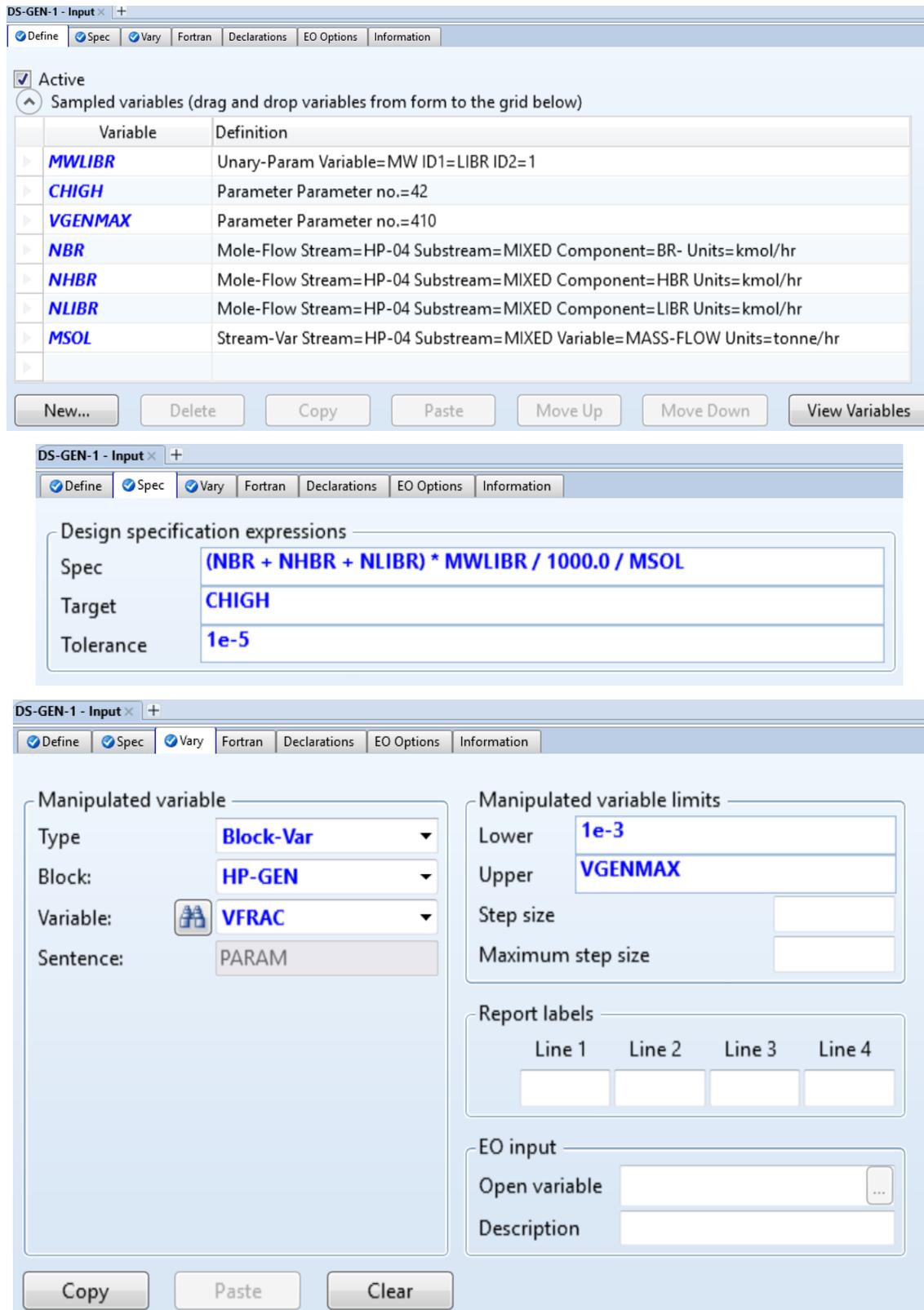


Figure 2.68. Specifications for DS-GEN-1.

We amend the new parameter **VGENMAX** to **C-GLOBAL** as Parameter #410, initialized to **ERROR** in **C-GLOBAL**'s interpreted Fortran. Because the generator contains a large amount of salt, it is not possible for it to reach a vapor fraction of 100%. We use **VGENMAX** to specify the maximum reasonable vapor fraction such that the **Design Spec DS-GEN-1** does not introduce simulation errors by attempting to vary the vapor fraction to unreasonable values.

For now, we will assume that the maximum reasonable vapor fraction is that achieved when the generator_{HP} is at 700°C; any higher vapor fraction would require an even higher temperature. To specify this decision, we amend **TGENMAX** to **C-GLOBAL** as Parameter #400. We initialize **TGENMAX** to 700.0 in the interpreted Fortran of **C-GLOBAL**.

We then add a **Heater** block **HP-GENF1** and two new material streams, **HP-15** and **HP-16**, to our simulation as shown in Figure 2.69. We assume that **HP-GENF1** has a pressure drop of zero, $\Delta P = 0$, and a temperature arbitrarily selected to be $T = 100^\circ\text{C}$. In all cases, **TGENMAX** will overwrite the arbitrary value of 100°C , so this value's specification is irrelevant.

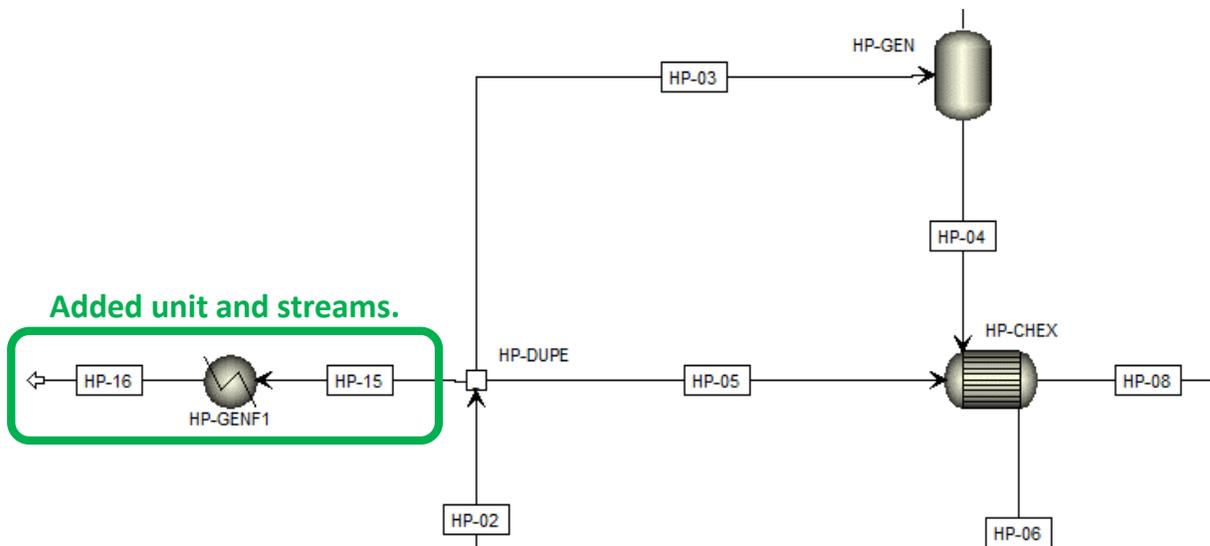


Figure 2.69. New flowsheet elements added to calculate **VGENMAX** from **TGENMAX**. The material stream **HP-15** branches from **HP-DUPE** to feed the new **Heater** block, **HP-GENF1**, which outputs a material stream **HP-16**.

We calculate **VGENMAX** from **TGENMAX** using the specifications shown in Figure 2.70 and Figure 2.71. The cumulative effect of these specifications is:

1. Set the value of **TGENMAX** to the **Heater**, **HP-GENF1**.
2. Run the **Heater**, **HP-GENF1**.
3. Retrieve the calculated vapor fraction from the **Heater** and set it to **VGENMAX**.

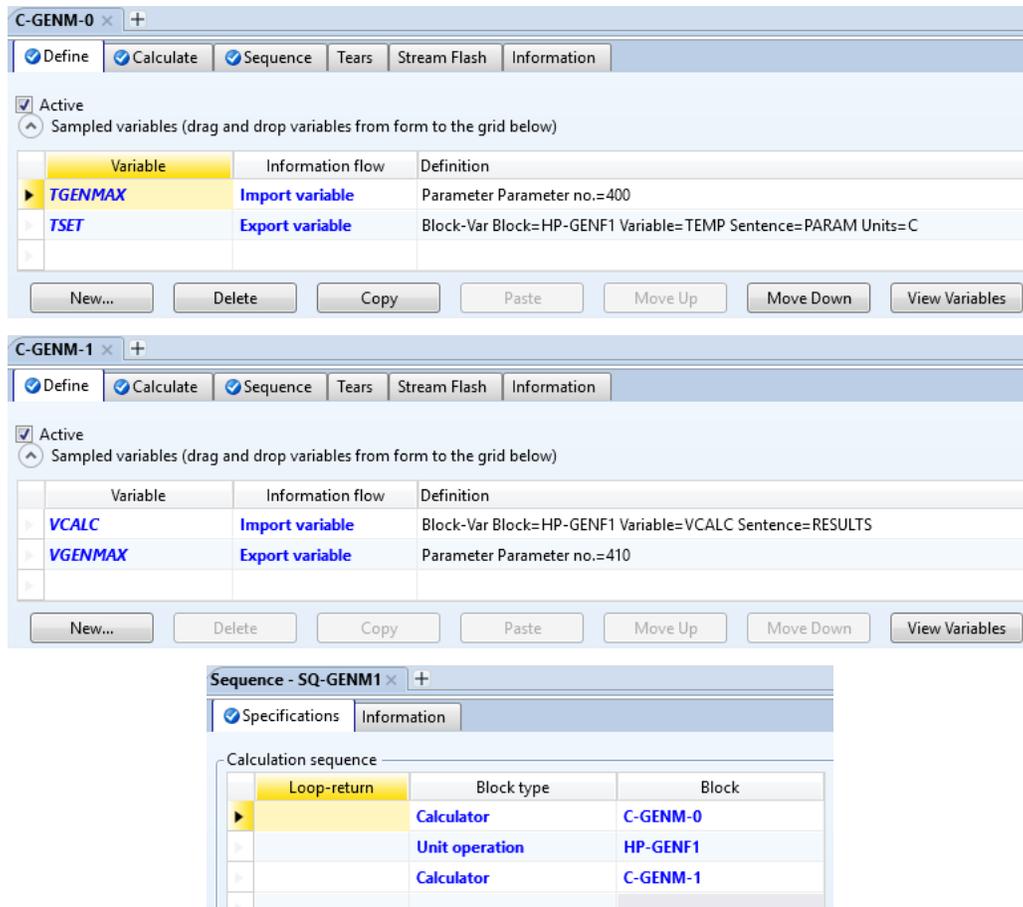


Figure 2.70. Specifications for **C-GENM-0**, **C-GENM-1**, and **SQ-GENM1**.

```

C-GENM-0      TSET = TGENMAX
C-GENM-1      VGENMAX = VCALC
  
```

Figure 2.71. Interpreted Fortran input for **C-GENM-0** and **C-GENM-1**. Both **Calculator** blocks require only a single line to set their values. The first sets **TGENMAX** to the **Heater**, and the second read the resulting **VGENMAX** from the **Heater**.

Figure 2.72 shows the sequencing for generator_{HP}.

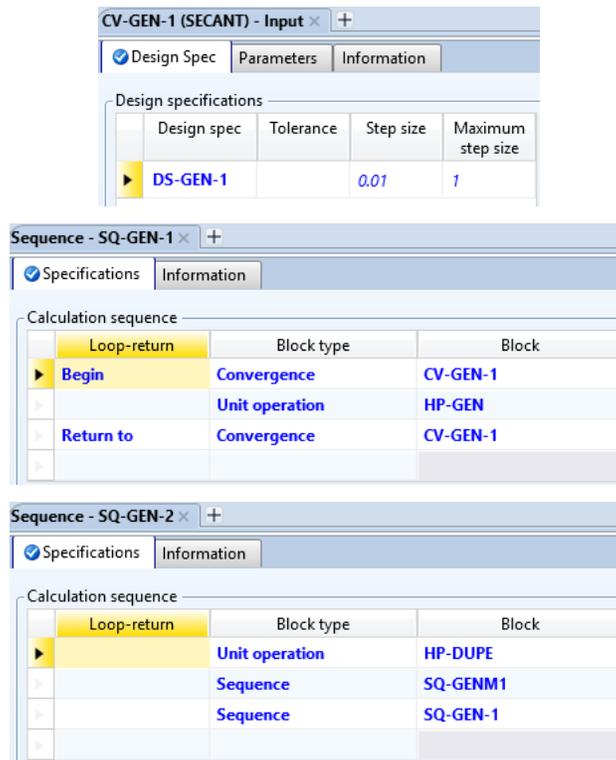
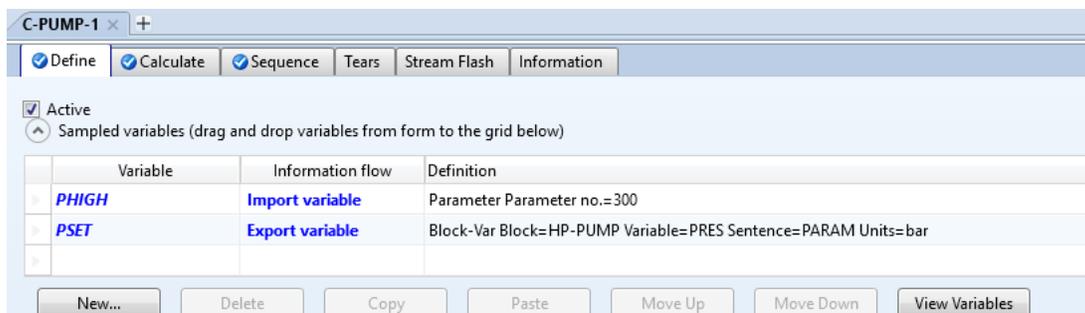


Figure 2.72. Sequencing for generator_{HP} using **CV-GEN-1**, **SQ-GEN-1**, and **SQ-GEN-2**. The top-level **Sequence** block, **SQ-GEN-2**, is the derived object for generator_{HP}.

We bind the global parameters for P_{low} and P_{high} to the pump and valves that effect them. To do this, we will create the derived objects for the pump and valves. Figure 2.73 shows the derived object for the pump while Figure 2.74 and Figure 2.75 show the derived objects for the valves.



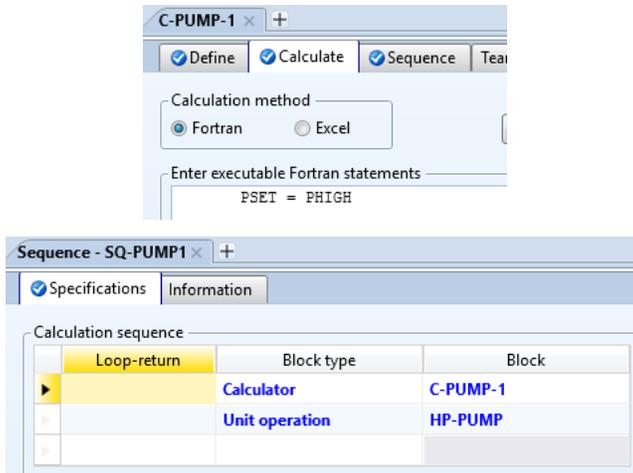


Figure 2.73. Specifications for the derived pump object, **SQ-PUMP1**. Also requires the creation of a new **Calculator** block, **C-PUMP-1**.

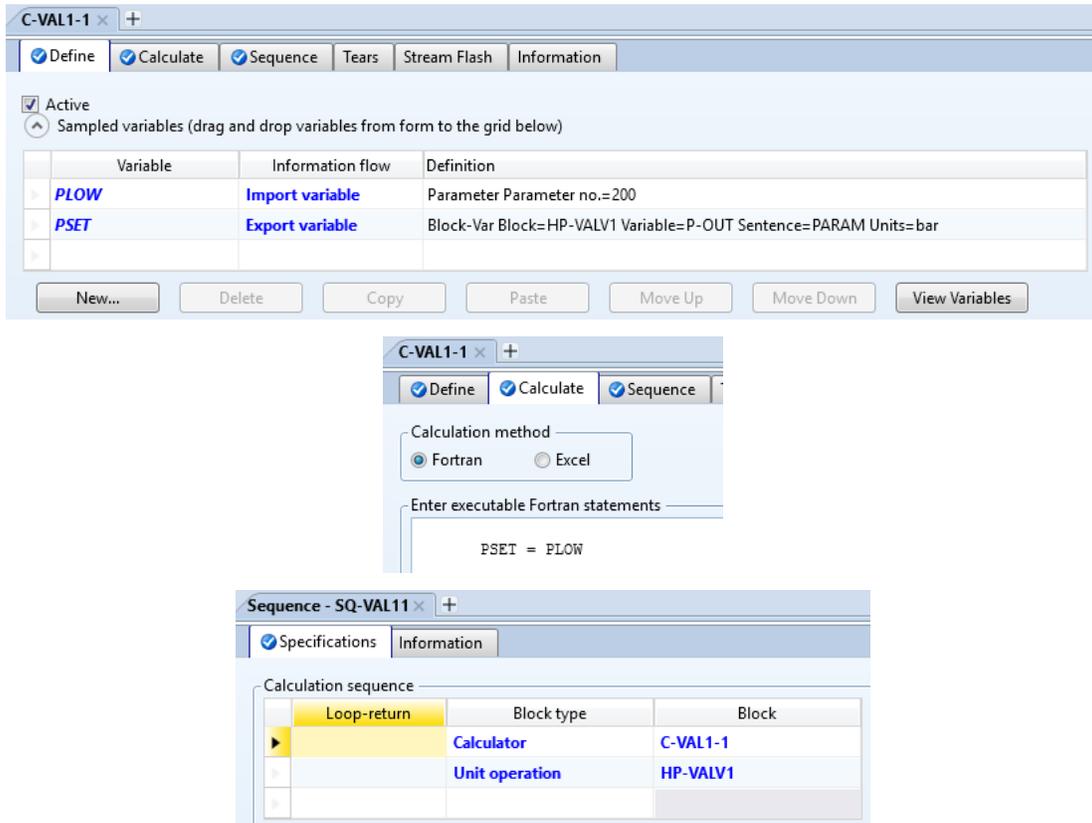


Figure 2.74. Specifications for one of the two derived valve objects, **SQ-VALV11**. Also requires the creation of a new **Calculator** block, **C-VAL1-1**.

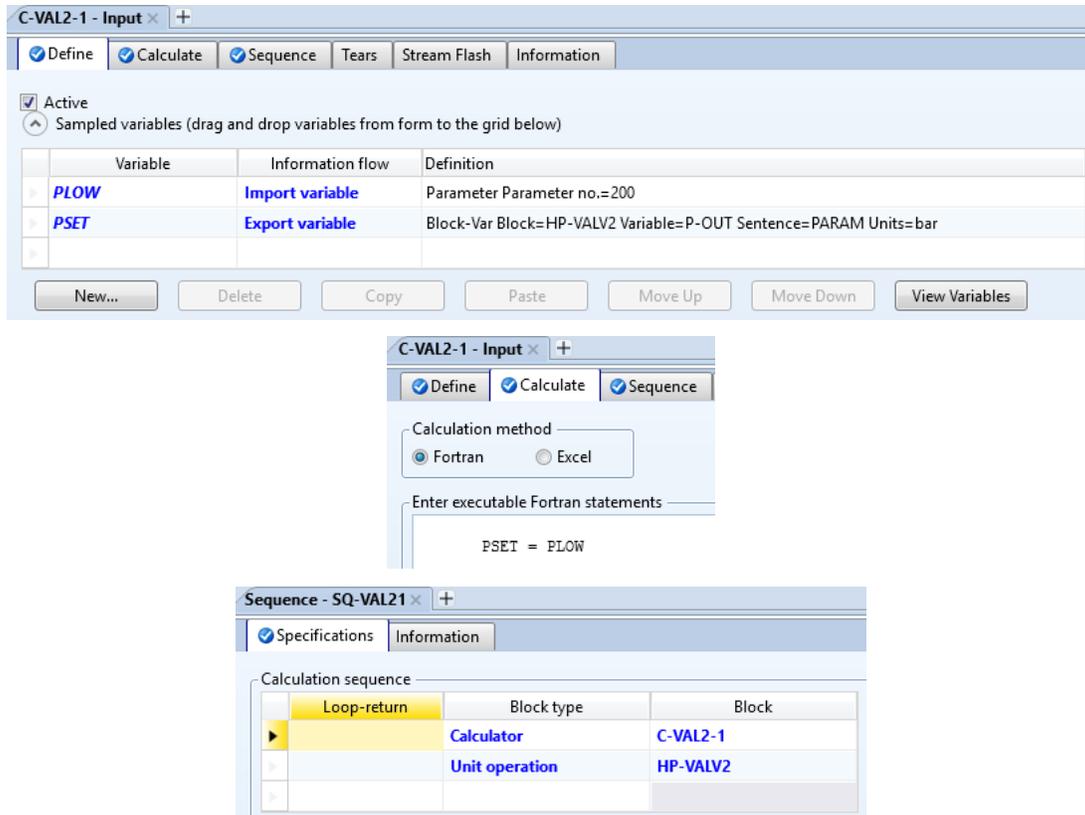
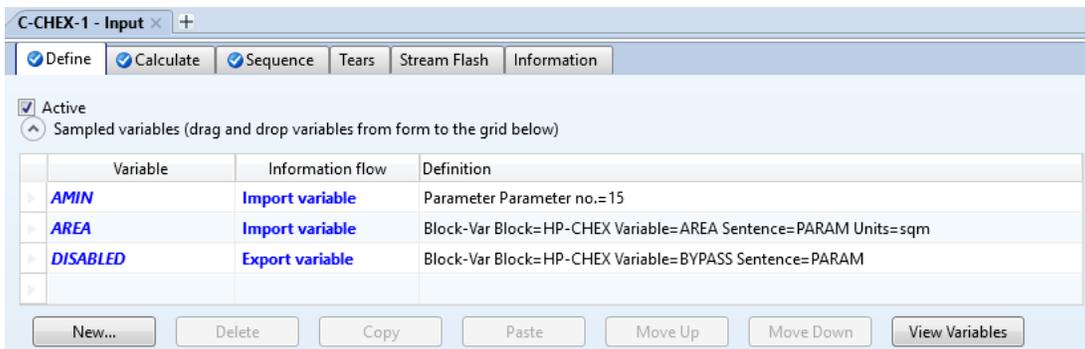


Figure 2.75. Specifications for the second of the two derived valve objects, **SQ-VALV21**. Also requires the creation of a new **Calculator** block, **C-VAL2-1**.

We derive a new cross heat exchanger object that disables the **HeatX** when it has an area specification of zero. We amend **AMIN** to **C-GLOBAL** as Parameter #15 and initialize it to 0.00001 in the interpreted Fortran of **C-GLOBAL**. We also create the additional flowsheeting elements as shown in Figure 2.76.



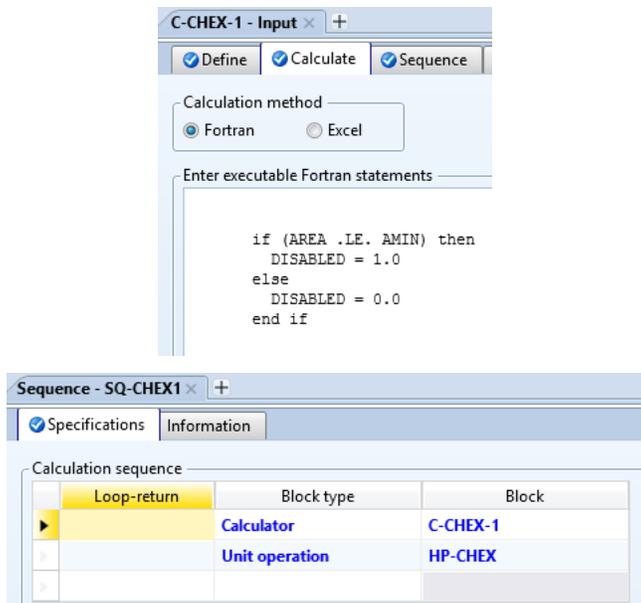
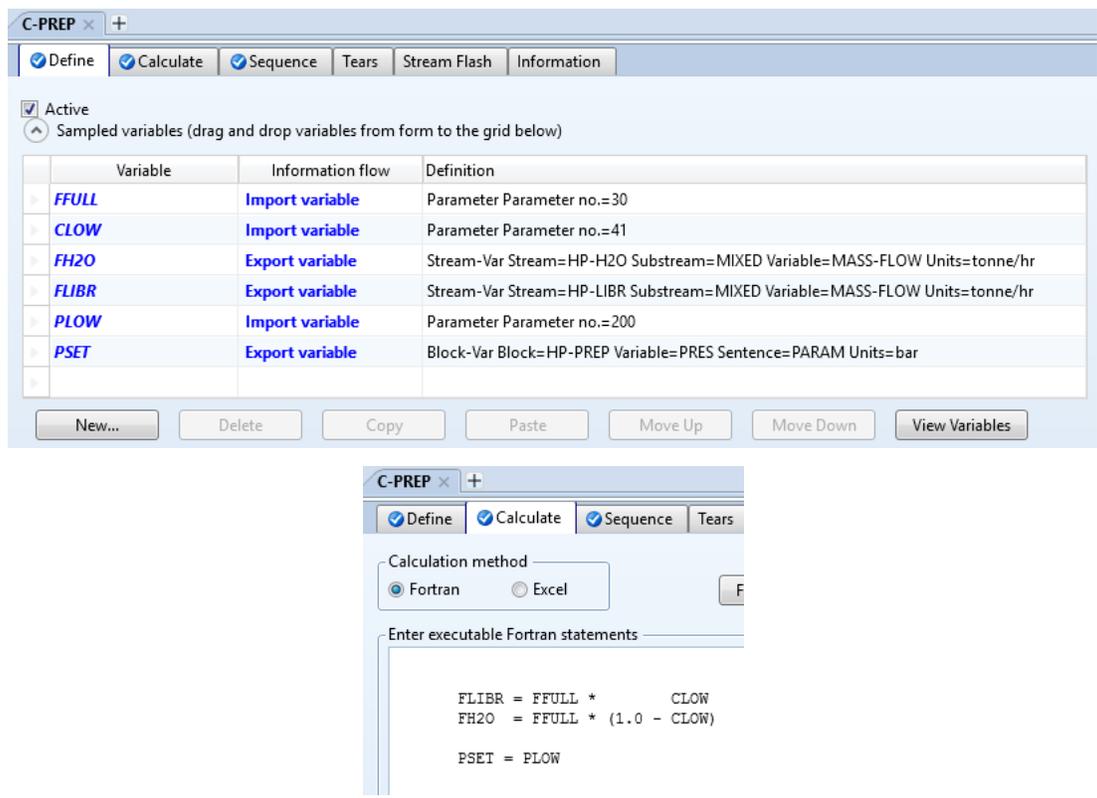


Figure 2.76. Derived object for the heat pump's cross heat exchanger, **SQ-CHEX1**. Requires the creation of a new **Calculator** block, **C-CHEX-1**.

Next, we create the flowsheeting elements for specifying the full working solution. Figure 2.77 shows the details.



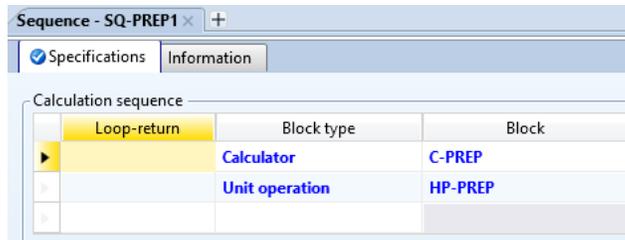


Figure 2.77. Derived object **SQ-PREP1** which generates the working solution of aqueous lithium bromide for the primary flowsheet section of the heat pump.

Finally, we face the C_{high} issue. So far we have not calculated C_{high} , though we have calculated C_{low} and asserted a minimum difference, $\Delta C_{min} \approx 0.05 \frac{\text{kg LiBr}}{\text{kg solution}}$. For now we will arbitrarily select $C_{high} = C_{low} + \Delta C_{min}$ as shown in Figure 2.78.

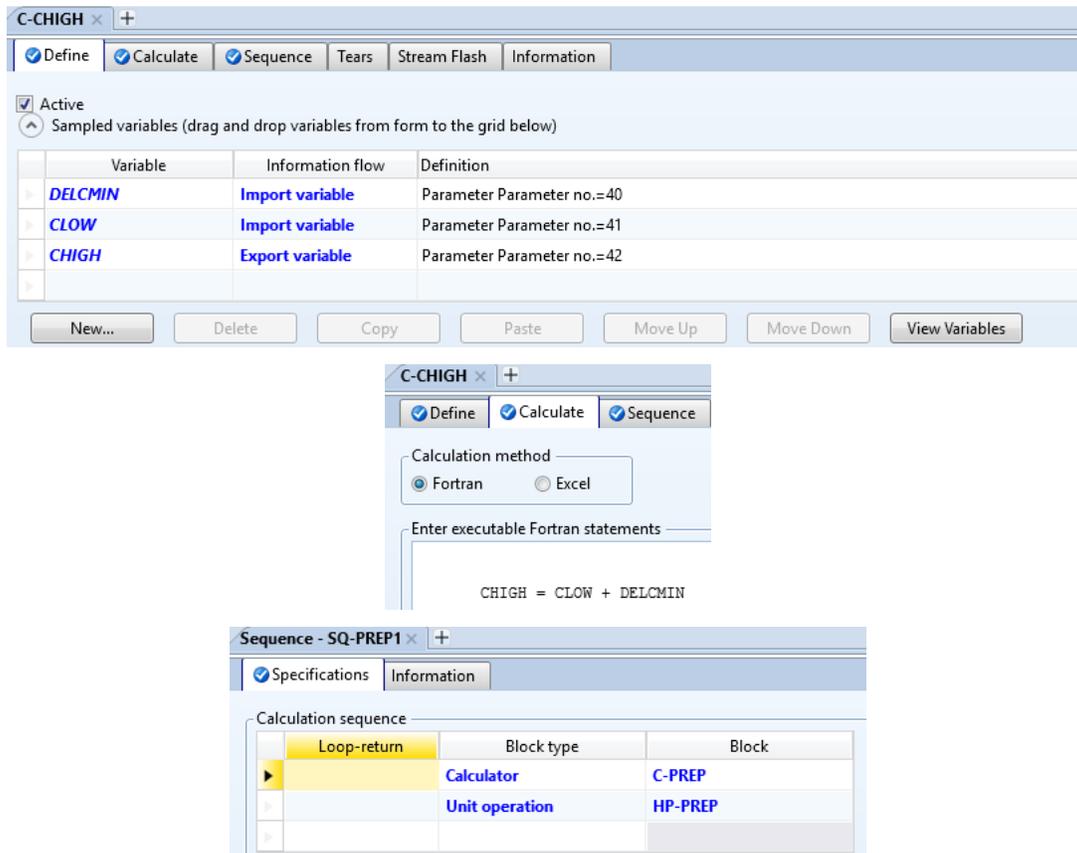


Figure 2.78. **Calculator** block to set **CHIGH** to its minimum value.

Together these specifications are sufficient for the heat pump simulation to run. To ensure proper operation, we will sequence the units together as shown in Figure 2.79.

Sequence - SQ-HP

Loop-return	Block type	Block
▶	Sequence	SQ-PREP1
▶	Sequence	SQ-PUMP1
▶	Sequence	SQ-GEN-2
▶	Sequence	SQ-CHEX1
▶	Unit operation	HP-CON
▶	Sequence	SQ-VAL11
▶	Sequence	SQ-VAL21
▶	Unit operation	HP-VAP
▶	Unit operation	HP-COMB
▶	Unit operation	HP-ABS

Sequence - SQ-UNIT

Loop-return	Block type	Block
▶	Sequence	SQ-T
▶	Calculator	C-CHIGH
▶	Sequence	SQ-HP

Figure 2.79. Derived object **SQ-HP** for the heat pump and **SQ-UNIT** for the overall flowsheet.

2.5.3.8 Adding analysis and control to the flowsheet

We now have a derived object for the overall flowsheet, **SQ-UNIT**. This object accepts values in **C-GLOBAL** and computes the basic heat pump parameters, allowing the heat pump to join with our acid-gas-capture simulation.

While the current simulation is sufficient to perform runs, it is more productive to incorporate basic data recording and analysis functionality.

First, we will specify that we are interested in:

- the major flowsheet parameters already defined in **C-GLOBAL**;
- the temperature and heat duty of each of the four major unit operations;
- the vapor fraction of the generator_{HP};
- the heat duty of the cross heat exchanger;
- the work used by the pump;

- coefficient of performance (COP).

We define these additional parameters in **C-GLOBAL** as shown in Table 2.8, initializing each to **ERROR** in the interpreted Fortran of **C-GLOBAL**.

Table 2.8. Parameters to be declared in **C-GLOBAL**.

Name	Parameter Number
TABS	10,000
DABS	10,002
TCON	10,010
DCON	10,012
TGEN	10,020
VGEN	10,021
DGEN	10,022
TVAP	10,030
DVAP	10,032
DHEX	10,042
WPUMP	10,050
COP	11,000

We then add a new **Calculator** block, **C-RESULT**, which is responsible for setting these values with the calculated results from each run.

We create a new **Sensitivity Analysis** block, **S-DATA**. As with our acid-gas-capture simulations, **S-DATA** will serve as a convenient means to:

1. Specify parameters (as opposed to **C-GLOBAL**).
2. Conduct sensitivity analysis runs.
3. Retrieve tabulated results.

First, we amend a new parameter, **DUMMY**, as Parameter #12 in **C-GLOBAL**, initialized to 0.0. By convention, we always want a dummy variable as the first Vary variable in our sensitivity analyses.

Next, we copy/paste the entire list of all variables defined in **C-GLOBAL** into the definition list of **S-DATA** and have **S-DATA** tabulate all of these defined variables for recording purposes.

Finally, we sequence **S-DATA** as shown in Figure 2.80.

The screenshot shows a software window titled 'Sequence - SQ-DATA' with two tabs: 'Specifications' (selected) and 'Information'. Below the tabs is a section titled 'Calculation sequence' containing a table with three columns: 'Loop-return', 'Block type', and 'Block'. The table contains three rows of data.

Loop-return	Block type	Block
▶ Begin	Sensitivity	S-DATA
▶	▼ Sequence	SQ-UNIT
▶ Return to	Sensitivity	▼ S-DATA

Figure 2.80. **Sequence SQ-DATA** which wraps **SQ-UNIT** in our new **Sensitivity Analysis, S-DATA**.

2.6 Combined carbon capture unit and heat pump simulation

We use the heat pump discussed in Section 2.5: “Heat pump integration” to recover waste heat from a carbon-capture unit. Figure 2.42 shows an example configuration in which recovered waste heat is used to help power the stripper’s reboiler. Now we want to simulate this process.

In this section, we will review a successful monolithic flowsheet that we have previously reported.⁶ We present this for completeness; in practice, we encourage readers to make use of the COM (Microsoft’s “Component Object Model”) interface and employ the more powerful methodologies discussed ahead in Chapter 3: “Using Aspen Plus through its COM interface using C#”.

2.6.1 Flowsheet layout

Figure 2.81 shows an example of a monolithic carbon capture unit flowsheet built on the principles discussed in Section 1.5: “General approach: Expand the model space and optimize”. We use this same approach in our current flowsheet layout.

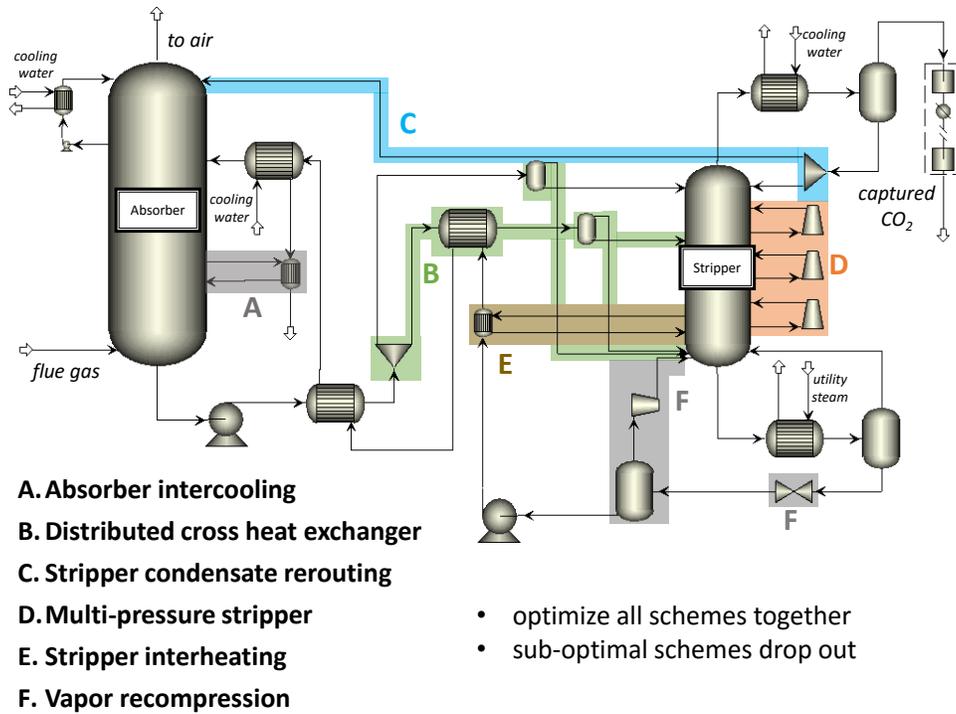


Figure 2.81. Monolithic carbon-capture flowsheet including many energy-saving schemes.

Figure 2.82 shows the actual flowsheet for this Aspen Plus simulation. Since this is a monolithic flowsheet design, it includes many energy-saving schemes including the heat pump.

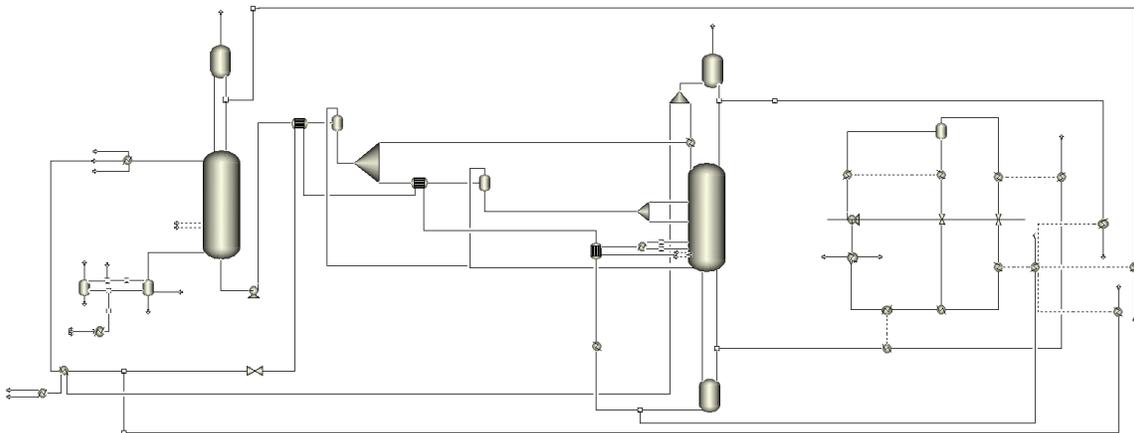


Figure 2.82. Actual Aspen Plus flowsheet. This flowsheet incorporates several energy-saving schemes, including the heat pump on the right-hand side.

2.6.2 Energy-saving scheme implementations

This flowsheet includes:

- absorber intercooling (see Section 2.1: “Absorber intercooling”);
- stripper interheating (see Section 2.2: “Stripper interheating”);
- stripper condensate rerouting (see Section 2.3: “Stripper condensate rerouting”);
- the distributed cross heat exchanger (see Section 2.4: “Distributed cross heat exchanger”);
- a heat-pump-augmented reboiler (see Section 2.5: “Heat pump integration”).

We implement these energy-saving schemes using the approaches discussed in the corresponding sections.

2.6.3 Optimization approach

This flowsheet was too large to optimize using only Aspen Plus’s tools, e.g. the **Optimization** block. However, we did use two **Optimization** blocks inside the simulation:

1. **O-FLUE-W**: Optimizes the H₂O portion in the flue gas to meet a target relative humidity.
2. **O-HEATP**: Optimizes a set of four heat pump parameters to maximize the heat pump objective function.

Both of these **Optimization** blocks are nested deeply within the simulation (see Section 1.6.3.2: “Nesting vs. flat convergence”) due to their limited scope. Specifically, **O-FLUE-W** is nested around the flue gas generation blocks while **O-HEATP** is nested around the heat pump blocks.

We manually optimized the flowsheet-level parameters. A team of chemical engineering professionals from an industrial partner would input trial values in **C-GLOBAL**, run the simulation, and report the results. Whenever a computer in the lab became free, it would be assigned a new set of trial values. We tabulated all results in an Excel spreadsheet for analysis.

2.6.4 Working with the flowsheet

2.6.4.1 We use a **Sensitivity Analysis** block for all runs

Our flowsheet uses the **Sensitivity Analysis** block **S-DATA** to automatically tabulate results.

Whenever a simulation finishes, we go to

Aspen Plus | Simulation | Model Analysis Tools → Sensitivity → S-DATA → Results,

and copy/paste the table into an Excel spreadsheet.

This technique provides us with four major advantages:

1. It is very quick and easy to copy/paste all of a simulation's major results into Excel.
2. We can use **S-DATA** to specify multiple simulations to run in series, or we can simply specify only one simulation at a time.
3. The results tabulated in Excel are structured the same way for all simulation runs, making it very easy to perform analysis and make plots.
4. We maintain records of all major simulation observables for all runs, so if we later become interested in data that we did not previously consider worthwhile, it is already available to us in the Excel sheet for all prior runs.

Our current flowsheet tabulates 215 variables in addition to those automatically tabulated by a **Sensitivity Analysis**, e.g. the varied parameters. While it is possible to specify calculations in the tabulated results, we recommend tabulating only raw data. Tabulating raw data ensures that no information is lost, and it is easy to calculate important derived values such as E_t^{regen} by dragging an equation down on the Excel spreadsheet. By contrast, if you tabulate $E_t^{\text{regen}} = \frac{D_{\text{STR-REB}}}{F_{\text{captured CO}_2}}$, your Excel spreadsheet will fail to track data for $D_{\text{STR-REB}}$ and $F_{\text{captured CO}_2}$ unless you tabulate these separately.

2.6.4.2 The heat pump is computationally downstream of the CO₂-capture unit in some situations

We designed our flowsheet with the heat pump being computationally downstream of the CO₂-capture unit in some situations. In other words, the CO₂ capture unit results did not depend on the heat pump's results in these situations. Figure 2.83 shows this computational flow order.

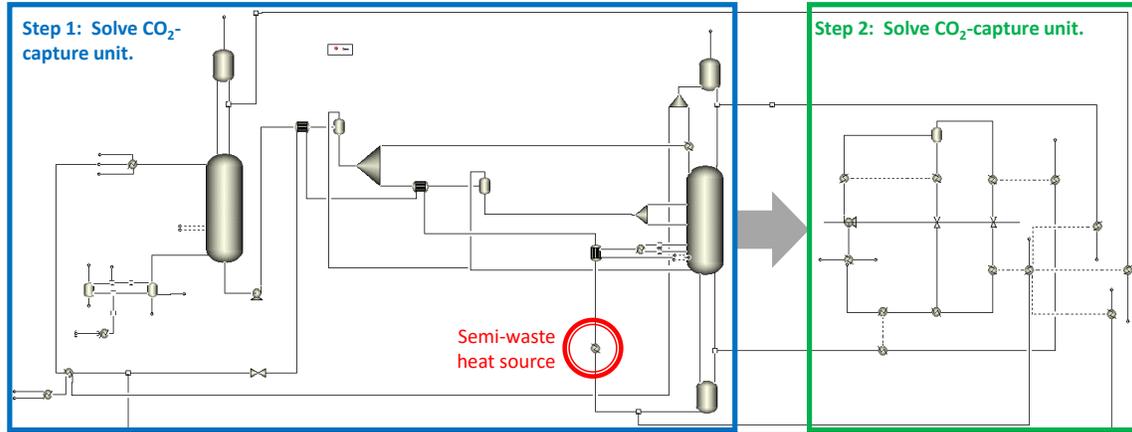


Figure 2.83. The CO₂-capture unit is computationally upstream of the heat pump in some situations. In these situations, the CO₂-capture unit results are unaffected by the heat pump, providing us with simulation results for designs both with and without the heat pump at the same time. In situations that do not have this property, we only find results for a design with the heat pump. The determining factor is whether or not the semi-waste heat source draws heat for the heat pump.

Using a downstream model for the heat pump provides two major advantages:

1. We receive results for both a CO₂-capture unit with a heat pump and results for a CO₂-capture unit without a heat pump.
2. We do not need an outer convergence loop to tear the feedback from the heat pump to the CO₂-capture unit.

The results are computationally entwined if the semi-waste heat source in Figure 2.83 draws non-zero heat. This can occur when other waste heat sources fail to supply adequate waste heat to the heat pump.

As always, we can calculate the regeneration energy E_t^{regen} using Eq. 1.2.a. This general definition reduces to 2.6.a for our current flowsheet. E_t^{regen} can be simplified further in two special cases:

- Eq. 1.2.b when the heat pump extent is zero, $\chi_{\text{HP}} = 0$;
- Eq. 2.6.b when the heat pump extent is one, $\chi_{\text{HP}} = 1$.

$$E_t^{\text{regen}} = \frac{(1 - \chi_{\text{HP}})D_{\text{STR-REB}} + \chi_{\text{HP}}D_{\text{HP-GEN}}}{F_{\text{captured CO}_2}} \quad 2.6.a$$

$$E_t^{\text{regen}} \Big|_{\chi_{\text{HP}}=1} = \frac{D_{\text{HP-GEN}}}{F_{\text{captured CO}_2}} \quad 2.6.b$$

When the semi-waste heat draw is zero, Eq. 1.2.b can be used to calculate the thermal regeneration energy E_t^{regen} as though no heat pump were present, even if $\chi_{\text{HP}} > 0$. This is an example of the downstream computational model providing two results, because we can also use Eq. 2.6.b to calculate the thermal regeneration energy E_t^{regen} with the heat pump present from the same simulation results.

This technique works only when there is no semi-waste heat draw because the semi-waste heat **HEATER, HP-DRAW**, becomes hypothetical as discussed in Section 1.5: “General approach: Expand the model space and optimize”.

2.6.4.3 Specifying run parameters

Our **Sensitivity Analysis, S-DATA**, defines only a single vary variable, Parameter #120. Parameter #120 is defined in **C-GLOBAL** as **DUMMY**, reflecting that it is merely a dummy variable (placeholder). This specification causes **S-DATA** to achieve nothing in the simulation; it merely records data as discussed in Section 2.6.4.1: “We use a **Sensitivity Analysis** block for all runs”.

Because we do not use **S-DATA** for specifications in this simulation, we define all specifications in **C-GLOBAL**. Scheme 2.13 shows an abridged version of the “shortcut” section that we added to the front of **C-GLOBAL**. This section sets the values for the variables that we changed most often in our optimization process for convenience.

```

C =====
C =====START:  SHORTCUT=====
C =====

C Flooding factors for the absorber and stripper.
C Design Spec blocks will vary column diameters to
C meet these specifications.
      ABSFLOOD = 0.7
      STRFLOOD = 0.7

C Set temperature for the lean solvent cooler (degC):

```

```

LEANSETT = 45.0

C Split of stripper's condenser's condensate that goes to the
C lean stream cooler as opposed to returning to the top of
C the stripping column's packed section. Must be between
C 0.0 and 1.0 (inclusive).
    STRCSPLT = 1.0

C Area of the central cross heat exchanger (m^2):
    ACHEX = 35.0

C Area of the side central cross heat exchanger used for the
C distributed cross heat exchanger configuration (m^2):
    ASHEX =100.0

C Split of the rich solvent flow rate to middle of stripper (between 0 and
C 1):
    DHEXSPLT = 0.2

C Rich mid stage: the stage of the stripper that the distributed rich
C solvent flow enters on. (between 2 and 19)
    RICHMIDS = 4.0

C Absorber intercooling duty (MW) (should be zero or negative):
    ABSINTCD = 0.0

C Absorber intercooling stage (should be between 1 and 19):
    ABSINTCS = 15.0

C Stripper pressure (bar):
    PBOTTOM = 1.4

C COP estimate:
    ESTCOP = 1.708358963
C =====
C ===== END: SHORTCUT=====
C =====

```

Scheme 2.13. Abridged version of the “shortcut” section at the top of **C-GLOBAL**. We move variables that we frequently change to the top of this section for convenience. This makes it easier to quickly change the value.

2.6.5 Design conditions

We optimized our flowsheet for a specific site in the Shengli Oil Field. Table 2.9 shows the flue gas specifications.

Table 2.9. Flue gas specifications.

Parameter	Value
Temperature (°C)	40.0
Pressure (bar)	1.06
CO ₂ as portion of the dry flue stream (mol%)	15%
N ₂ as a portion of the dry, inert flue stream (mol%)	10%
O ₂ as a portion of the dry, inert flue stream (mol%)	90%
Relative humidity	100%

Our industrial partner planned to use a proprietary solvent, though they asked us to perform the design work using aqueous MEA. We considered multiple concentrations of MEA as part of our optimization, up to a maximum concentration of 30wt%. Beyond 30wt%, aqueous MEA requires special corrosion inhibitors³² that our industrial partner did not want us to require.

We targeted a 70% maximum approach to flooding for both the absorber and stripper. We tested for capture rates from 80% to 95%, primarily focusing on 80% capture.

2.6.6 Optimization process

We started our flowsheet optimization by searching for reasonable initial parameter values and design choices based on sensitivity analyses.

Caution

The sensitivity analyses in this section show intermediate results obtained while conducting an optimization process. These result curves do not belong to a single, consistent flowsheet specification, nor are they general results applicable to other CO₂-capture contexts.

Figure 2.84 shows a sensitivity analysis that checked for the thermal regeneration energy E_t^{regen} response to the solvent concentration. Based on this analysis, we decided to start with 30 wt% MEA. We note that 30 wt% MEA was our maximum concentration due to corrosivity (see Section 2.6.5: “Design conditions”); the sensitivity explored higher concentrations to satisfy our curiosity.

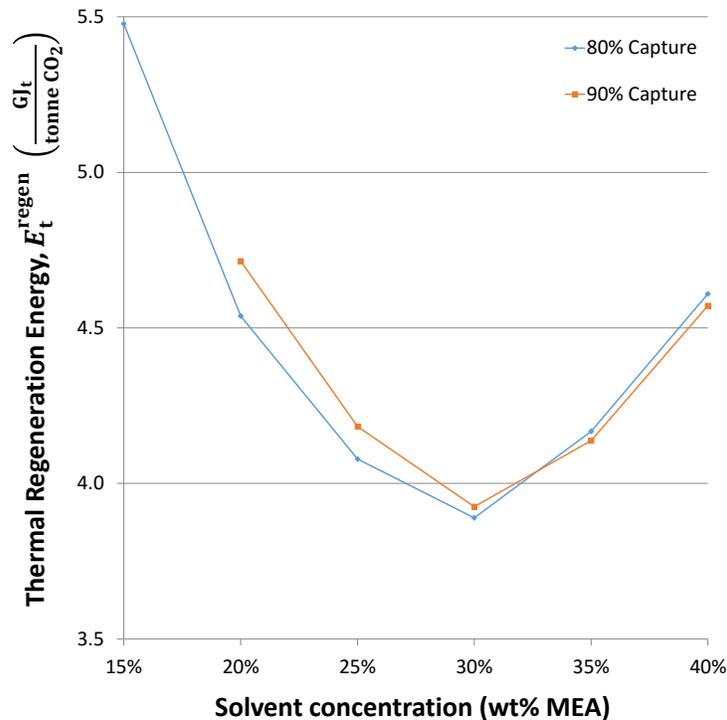


Figure 2.84. Sensitivity analysis varying solvent concentration. We concluded that 30 wt% MEA would be a good solvent concentration.

Figure 2.85 shows a sensitivity analysis over stripper pressure. We tracked both the thermal regeneration energy E_t^{regen} and reboiler temperature $T_{\text{STR-REB}}$ for this analysis. In general, we want to minimize E_t^{regen} while using utility steam temperatures that correspond to what our industrial partner had available at their site. More academic studies that do not consider specific industrial applications often compare metrics like equivalent work, W_{eq} , instead (see Section 1.4.2.3.3: “Equivalent work”).

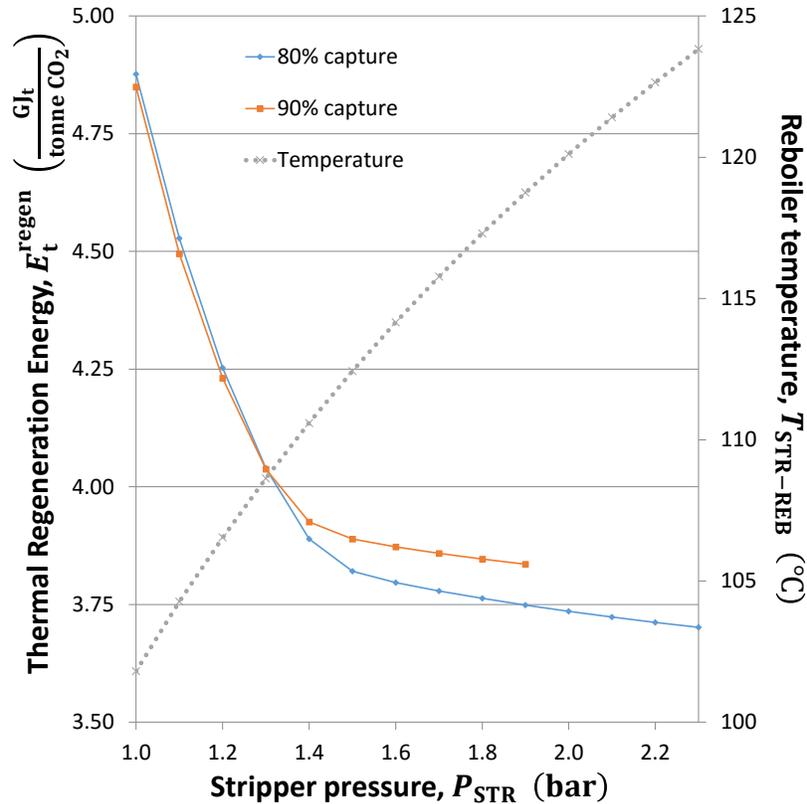


Figure 2.85. Sensitivity analysis varying stripper pressure, P_{STR} . We recorded both thermal regeneration energy E_t^{regen} and reboiler temperature $T_{STR-REB}$.

The thermal regeneration energy results in Figure 2.85 demonstrate that we cannot describe the system behavior in simple terms. For example, while it is generally true that higher capture rates require a higher regeneration energy, we can see that this did not apply at lower stripper pressures in this case because E_t^{regen} was higher in the 80% capture case than in the 90% capture case. Further, we also see that the correlation between E_t^{regen} by capture rate changed over this small range of stripper pressures.

In general, acid gas units like those discussed in this dissertation have strong, complex interactions between parameters. These strong, complex interactions compel us to optimize values together using application-specific design conditions, e.g. the flue gas specifications provided by our industrial partner (Section 2.6.5: “Design conditions”) rather than an arbitrarily guessed flue gas based on seemingly reasonable values.

Figure 2.86 also tracks E_t^{regen} and $T_{\text{STR-REB}}$, varying lean solvent loading $\mathcal{L}_{\text{lean}}$ instead of stripper pressure P_{STR} . In this case, $T_{\text{STR-REB}}$ showed a weak response, but E_t^{regen} showed a very strong response. The optimal lean solvent loading appears to be $0.175 \frac{\text{mol CO}_2}{\text{mol MEA}}$.

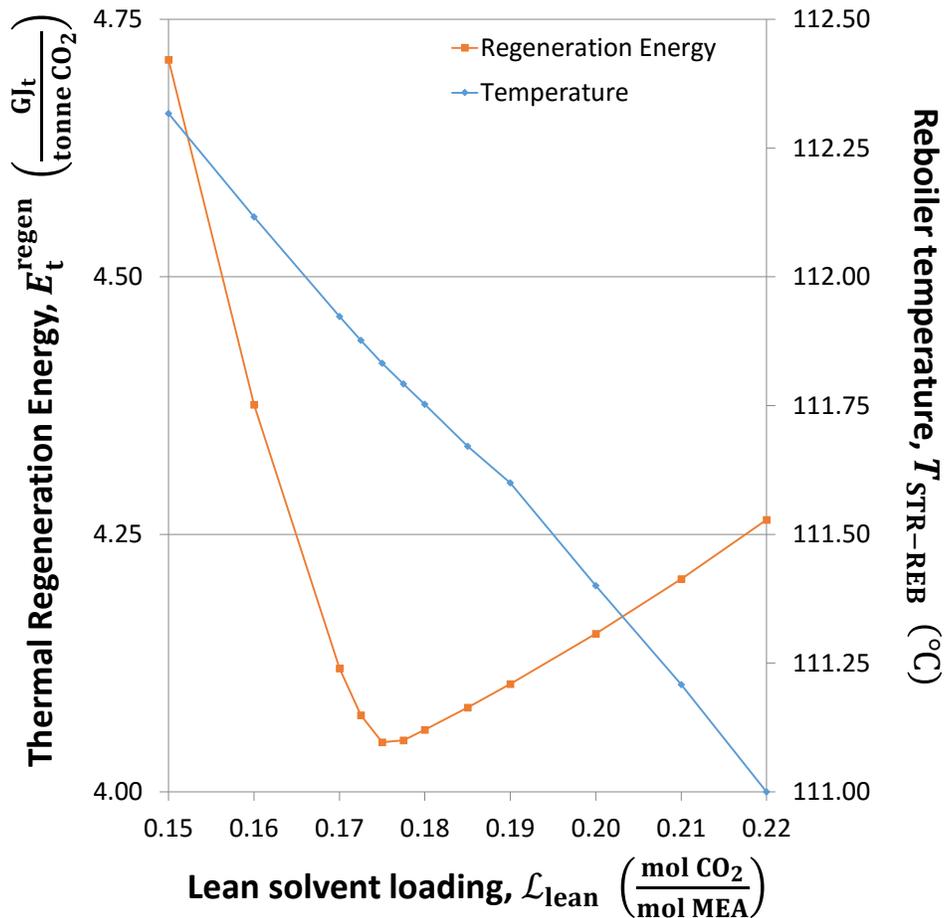


Figure 2.86. Sensitivity analysis varying lean solvent loading, $\mathcal{L}_{\text{lean}}$. We recorded both thermal regeneration energy E_t^{regen} and reboiler temperature $T_{\text{STR-REB}}$.

We moved away from their early lean solvent loading to $0.1825 \frac{\text{mol CO}_2}{\text{mol MEA}}$ in our final design. Apparently, we found that the response curve changed after making other parameter changes and implementing energy-saving schemes. After a brief search through our records, we find Figure 2.87 which shows why we transitioned to $\mathcal{L} = 0.1825 \frac{\text{mol CO}_2}{\text{mol MEA}}$. Note the 4.7% improvement achieved from the best E_t^{regen} in Figure 2.86 to the best E_t^{regen} in Figure 2.87 on our path to $1.67 \frac{\text{GJ}_t}{\text{tonne CO}_2}$.

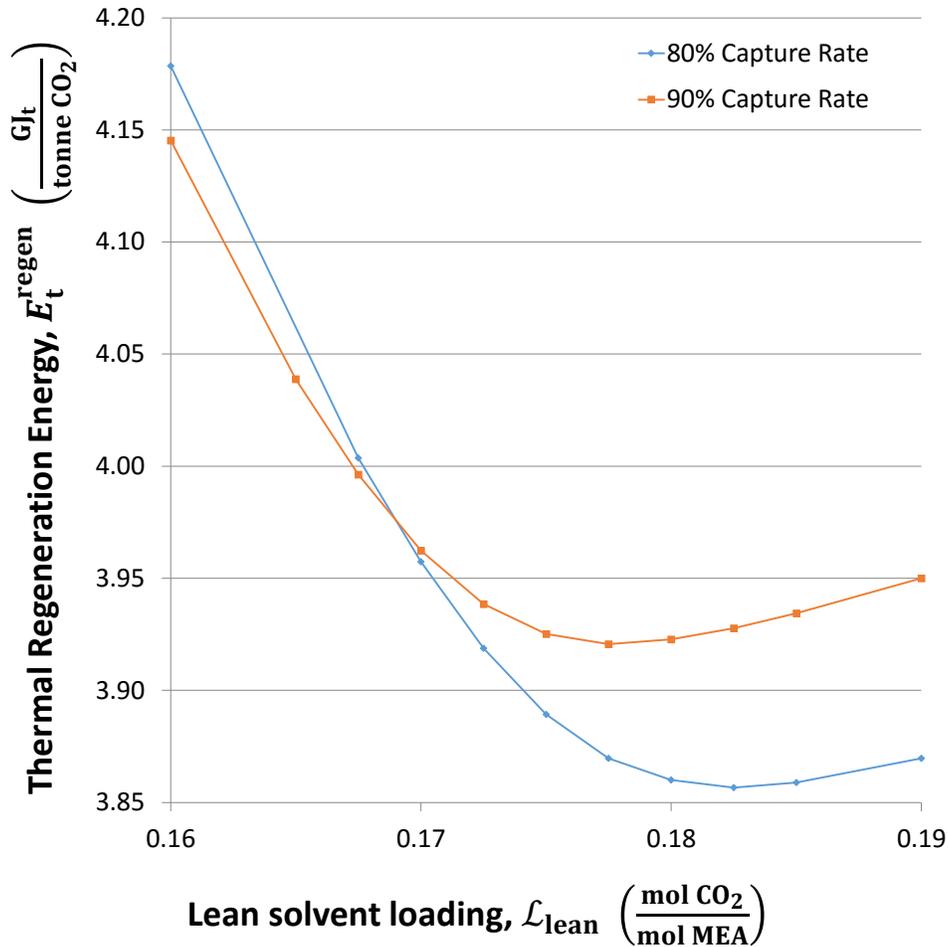


Figure 2.87. Sensitivity analysis over lean solvent loading. We record thermal regeneration energy at 80% and 90% capture rates.

Figure 2.88 shows our quick analysis of packings for the absorption tower. Our simulation adjusts the lean solvent flow rate to ensure that the absorber captures the target capture rate, $R = 80\%$, of the CO_2 in the flue gas. Lower lean solvent flow rates are very desirable; the lower the lean rate, the higher the rich solvent loading and the lower the energy needed to vaporize the solvent in the stripper's reboiler. Additionally lower solvent flow rates cause the same sized cross heat exchanger to provide greater heat transfer.

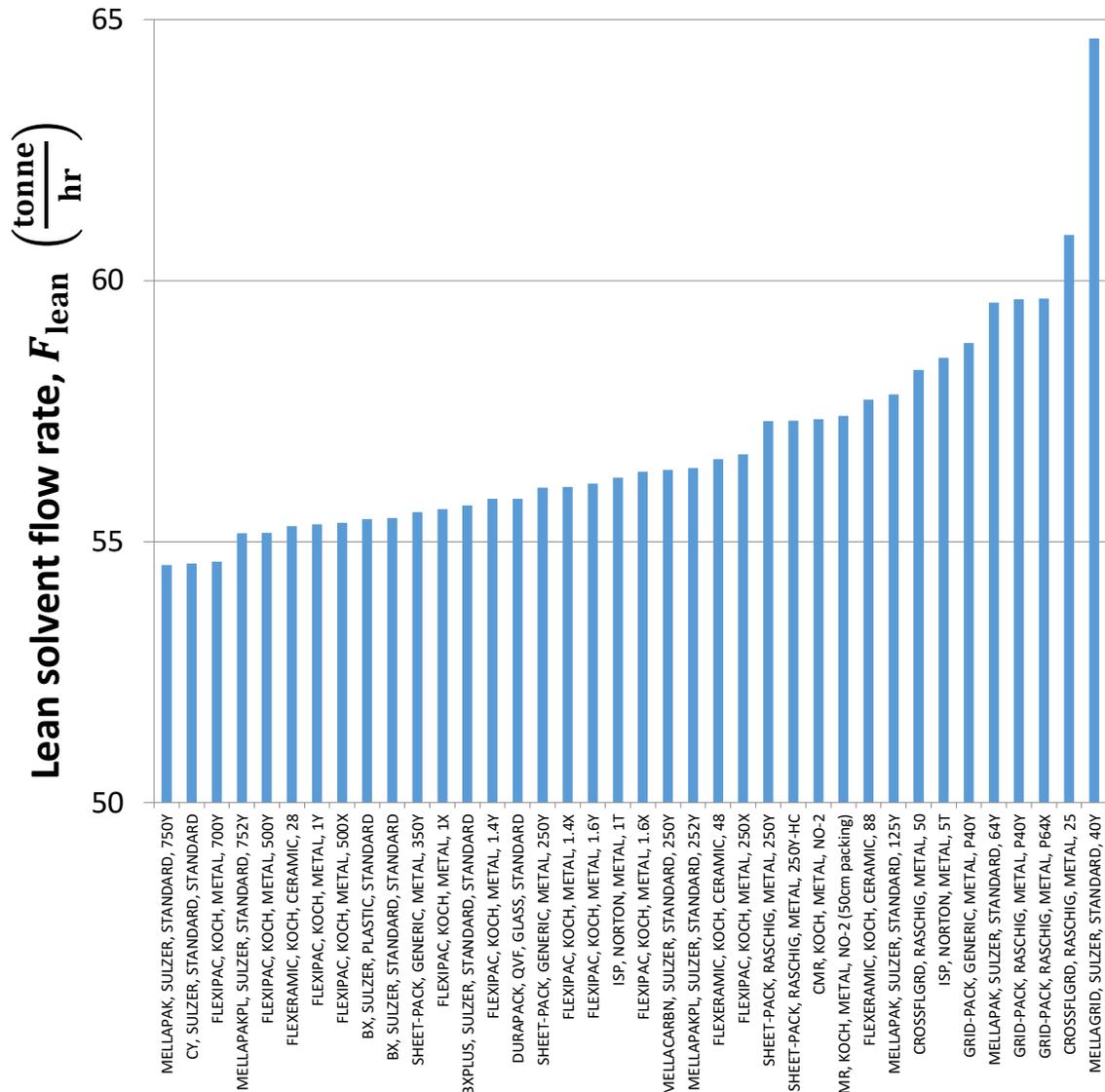


Figure 2.88. We adjust the lean solvent flow rate to meet our target capture rate of 80%. This figure shows the calculated lean solvent flow rate necessary to meet the target capture rate when we varied the absorber's packing. All else equal, the packings that required a lower lean solvent flow rate are more efficient. In order to do this analysis quickly, we relied on the default packing parameter values provided in Aspen Tech's databanks. We were unable to validate all of these packing parameters.

Figure 2.89 shows our analysis of drawing semi-waste heat to power the reboiler. In this figure, we define energy savings as the reduction in thermal regeneration energy. Definitions:

- The benefit is the amount of reboiler duty offset by the heat pump.

- The harm is the increase in the reboiler duty over base-case reboiler duty due to the parasitic cooling of drawing off semi-waste heat.
- The net effect is the benefit minus the harm.

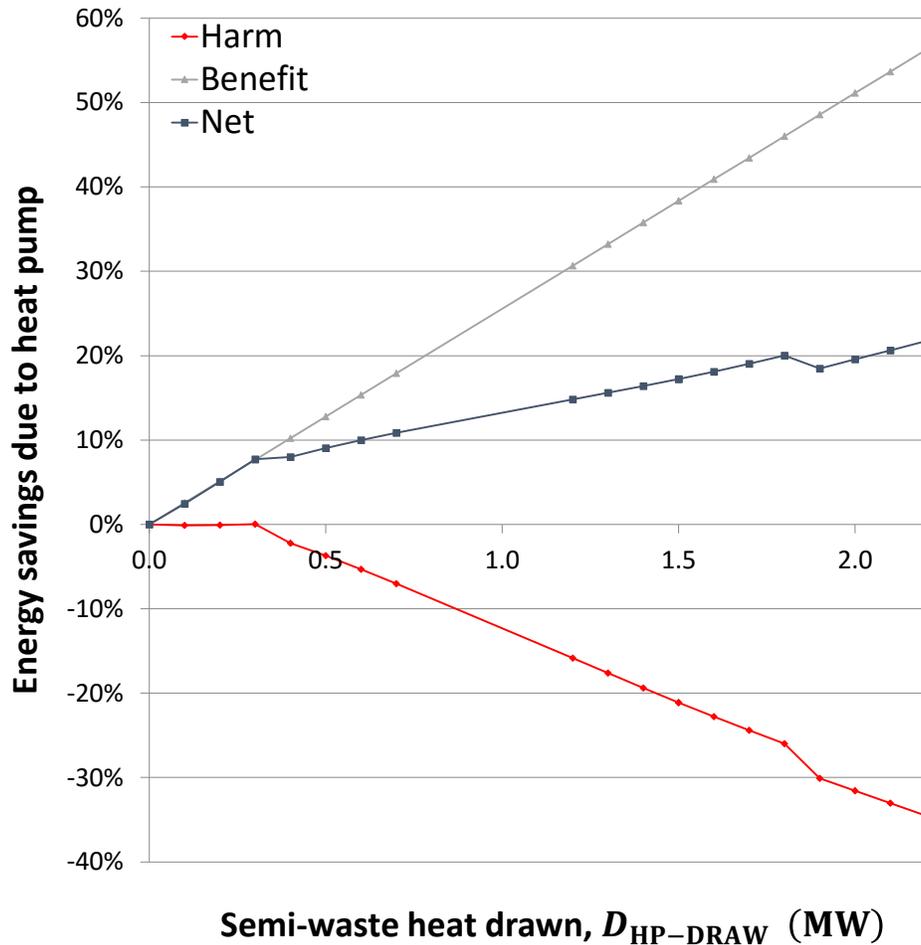


Figure 2.89. Results for varying amounts of semi-waste heat being used to power the heat pump. The “net” series shows the overall effect; the “benefit” series shows the energy need offset by the heat pump; the “harm” series shows the increased energy need caused by the siphoning of semi-waste heat.

In this particular case, the harm was near-zero for low parasitic cooling duties. Once the parasitic cooling duty reached 0.3 MW, further parasitic cooling had a fairly linear contribution to harm. The net benefit had a very positive slope before parasitic cooling started to harm the process; after that point, further use of semi-waste heat did improve the net benefit, but only at a third of the prior rate.

While an economic optimization would have to be performed on a case-by-case basis, we generally suspect that drawing semi-waste heat is unlikely to result in significant benefits past the point that harm begins. We prefer to recover more true waste heat through other process modifications.

Figure 2.90 shows a sensitivity analysis over the lean solvent cooler (LSC) set temperature, T_{LSC} , from the baseline value of 40°C up to 50°C. While we found a larger response to T_{LSC} toward 50°C, in this case we find a fairly small difference.

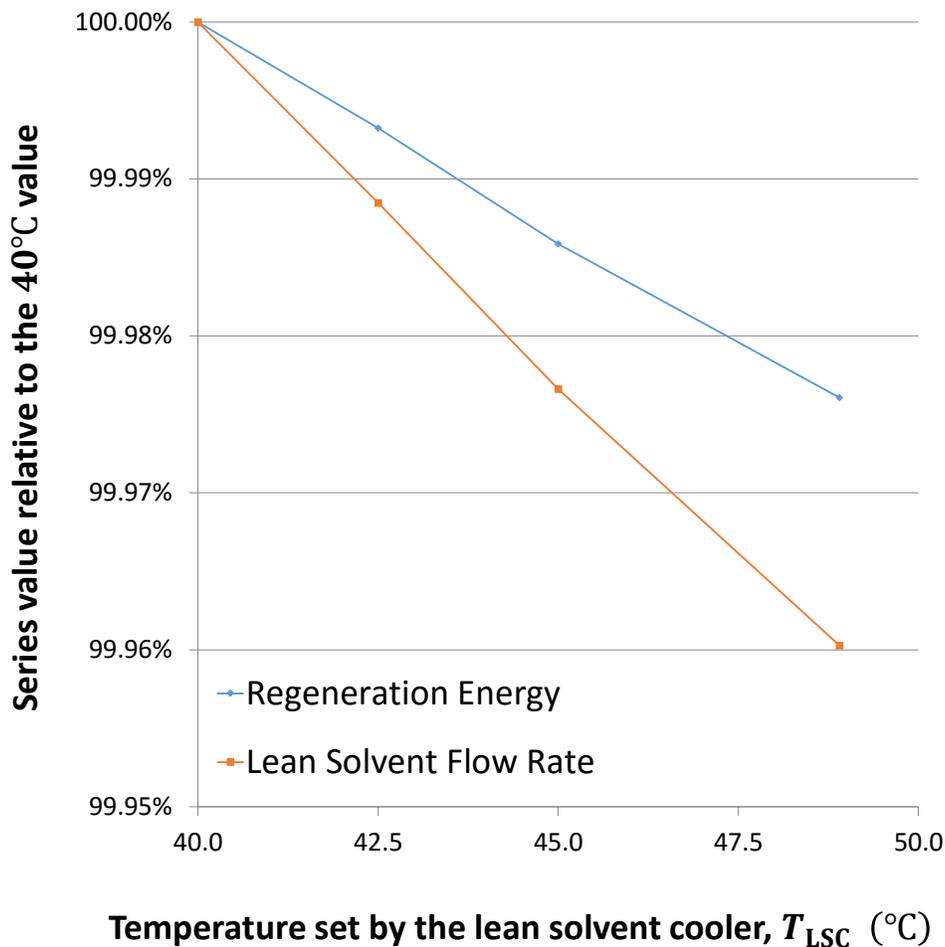


Figure 2.90. Results for varying the lean solvent cooler (LSC) temperature, T_{LSC} . We find small reductions in thermal regeneration energy E_t^{regen} and F_{lean} in this case.

Figure 2.91 shows the results from a sensitivity analysis over absorber packed height, H_{ABS} . Typically we perform this analysis early in the design process, though we checked it again to allow our industrial partners to make a decision based on capital costs versus energy efficiency.

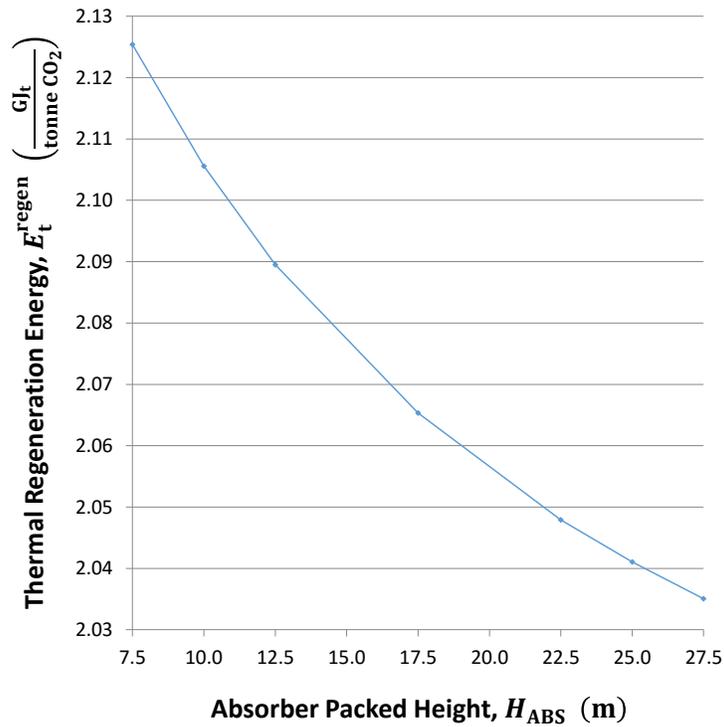


Figure 2.91. Results from varying the absorber packed height, H_{ABS} , for a process with significant energy savings. We see that H_{ABS} continues to have a strong impact on the overall thermal regeneration energy, E_t^{regen} .

Our early optimization approach began with running a simple sensitivity analysis over one variable and selecting a desirable result. We quickly improved our process parameters using this easy approach. After that, we moved on to slightly more complex sensitivity analyses. Figure 2.92 shows a two-variable sensitivity analysis in which we varied two parameters together.

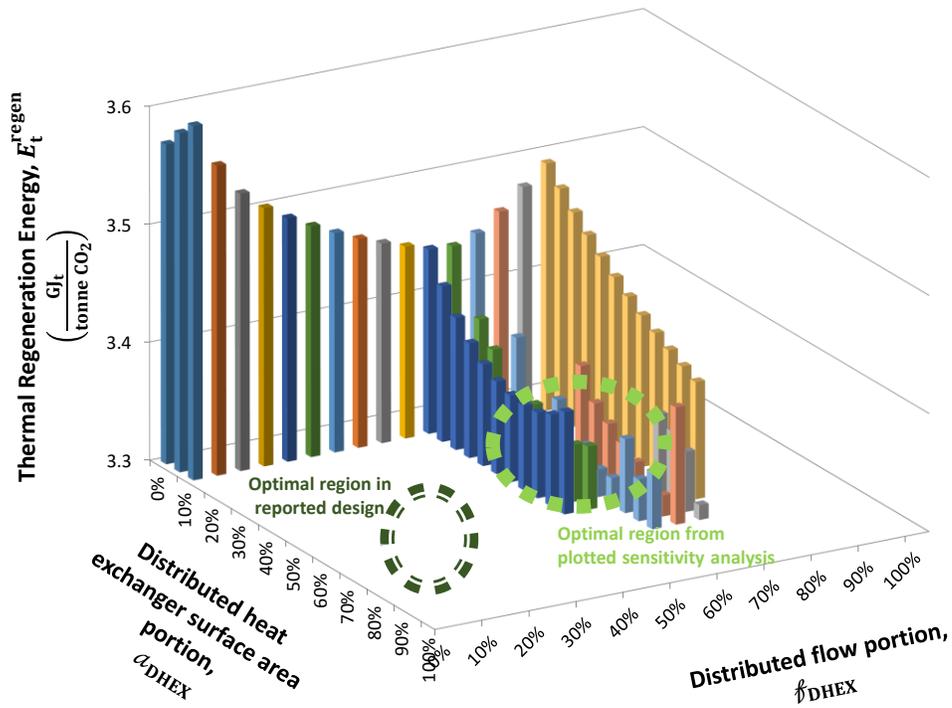


Figure 2.92. Sensitivity analysis over two of the distributed cross heat exchanger parameters.

Not all of our optimization runs were simple sensitivity plots as we attempted to identify trends to predict new optimal values to test. In the end, we found a design with a thermal regeneration energy of

$$E_t^{\text{regen}} \approx 1.67 \frac{\text{GJ}_t}{\text{tonne CO}_2}.$$

2.6.7 Optimization results

This approach was cumbersome, though we were successful. Figure 2.93 provides an assessment from Cao Xianghong of SINOPEC.

“My colleagues and I were so pleased with the lowest solvent regeneration energy requirement of the new process that is leading competing processes internationally, and have recommended its actual implementation in our Shengli Oil Field Power Plant to capture 1 million tonnes of CO₂ per year for enhanced oil recovery.”



-Cao Xianghong
Senior Vice President and Chief Technology Officer (retired),
China Petroleum and Chemical Corporation (SINOPEC)
2014 FORTUNE global top 3 company
Past President, Chinese Institute of Chemical Engineers (2002-2012)
Elected Foreign Member, U.S. National Academy of Engineering

Figure 2.93. Assessment of the simulation results by Cao Xianghong, Senior Vice President and Chief Technology Officer (retired) of SINOPEC.

2.6.8 Comparison

We focus on thermal regeneration energy E_t^{regen} as defined in Section 1.4.2.3.1: “Regeneration energy”.

Figure 2.94 compares our reported thermal regeneration energy to reported thermal regeneration energies from related designs.

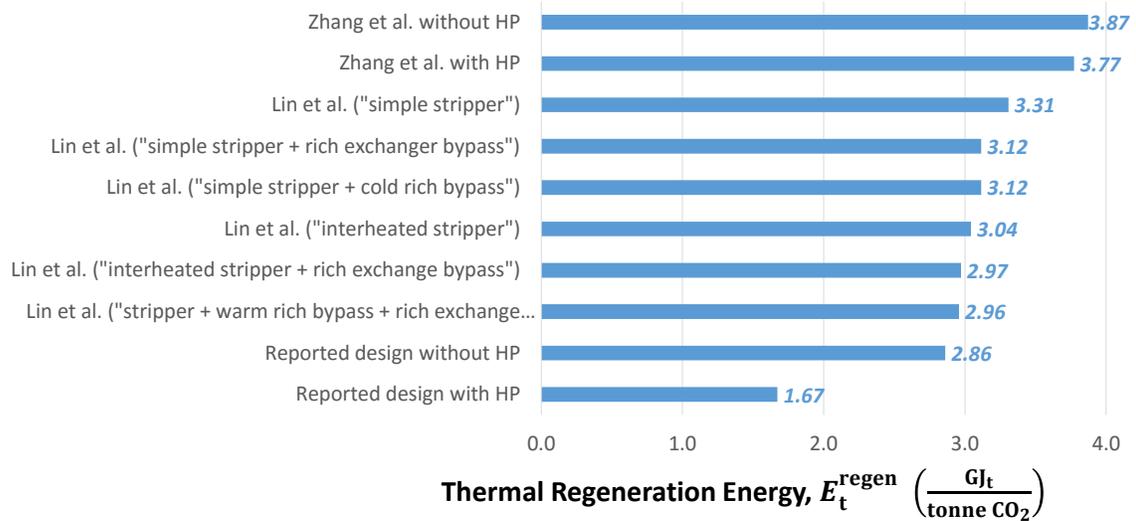


Figure 2.94. Comparison of designs reported by Zhang *et al.*,³¹ Lin *et al.*,³³ and this work.

2.6.9 Improved methodology

The methodology discussed in the preceding sections has proven to be effective, however we are not yet taking advantage of everything Aspen Plus has to offer. The methodology discussed in Chapter 3: "Using Aspen Plus through its COM interface using C#" grants us greatly increased power, flexibility, reliability, and convenience.

3 Using Aspen Plus through its COM interface using C#

In the prior chapter, we generalized our acid-gas-capture model to include many energy-saving schemes. We developed these energy-saving schemes in terms of optimizable parameters. We selected our optimizable parameters such that they contained the base-case flowsheet, allowing the optimizer to effectively add and remove energy-saving schemes automatically.

We can further automate this process by producing our own programs that work with Aspen Plus. Automating Aspen Plus allows us to implement convenient-and-powerful features like automatic error recovery, multi-threading, customized convergence, and more.

In the following, we explore creating our own acid-gas-capture simulation in C# by working with Aspen Plus through its COM (Microsoft's "Component Object Model") interface.

3.1 Download and install Visual Studio 2015

We make our C# program using Microsoft's Visual Studio 2015. Just like Microsoft's Word creates documents and Aspen Plus creates flowsheet simulations, Visual Studio creates programs.

University students and many employees may find the paid versions of Visual Studio available from their organization's site license agreement with Microsoft; however, we assume that readers are using the free "Community" version of Visual Studio 2015.

We recommend using the paid versions of Visual Studio if your organization provides it. However if you need to download the free ("Community") version, you should search for it from an official Microsoft website. Currently Microsoft provides Visual Studio 2015 Community at <https://www.visualstudio.com/en-us/products/vs-2015-product-editions.aspx>.

Visual Studio allows programming in many different languages including C#, Visual Basic, Visual C++, and F#. During installation, you need to tell Visual Studio which language to favor. We select C# and recommend that readers consider doing the same. This selection is largely a matter of common use;

selecting C# will not prevent you from working in other languages, but Visual Studio will tend to present you options that favor creating C# projects.

3.2 Conceptual preparation

3.2.1 Terminology

Important terms to understand:

- **Integrated development environment (IDE):** Just like word processors help users make documents, IDE's help users make programs. Visual Studio is an IDE.
- **Compiler:** A compiler reads source code and turns it into an executable program. Visual Studio includes its own compilers, so you do not need to find a separate product.
- **Programming language:** The language (vocabulary, grammar, spelling, etc.) that you can talk to compilers with. Popular examples include Basic, C, C++, C#, Fortran, Java, and Visual Basic. We will use C#.
- **Component Object Model (COM):** A Microsoft-specific standard for how software components communicate with each other. Aspen Plus relies on COM to communicate both within itself and with other programs like the ones we will make in C#. COM is a complicated subject by itself, but Visual Studio will make COM much easier for us.

3.2.2 Everything is a program

In some sense, just about everything a computer has is a program. Even simple documents can be viewed as programs that specify content to be displayed, though most word-processing environments are relatively dull for general-purpose programming.

If you know how to make Aspen Plus simulations, then you already know how to program. While it may not be immediately obvious, Aspen Plus is a computer-programming platform, and your simulations are computer programs. Understanding the analogy from Aspen Plus simulations to more conventional programming languages will make understanding our use of C# much easier.

Fundamentally, we can describe all computer programs as lists of instructions for the computer to perform. Usually, we think of simple instructions like basic math and calling functions. Aspen Plus is essentially the same thing as any other programming platform, except its instructions and data types are applicable mainly to process engineering. We specify instructions by drawing them on the flowsheet, typing them into **Calculator** blocks, etc. We specify the order of execution with **Sequence** blocks.

For example, when you sequence a **Heater** with $\Delta P = 0$ and $T = 40^\circ\text{C}$ as in Figure 3.1, it is the same thing as writing code in another language (like C#) for the input stream **IN** to be flashed at 40°C and save the result to the output stream **OUT**. Table 3.1 compares Aspen Plus programming and C# programming for the example in Figure 3.1.

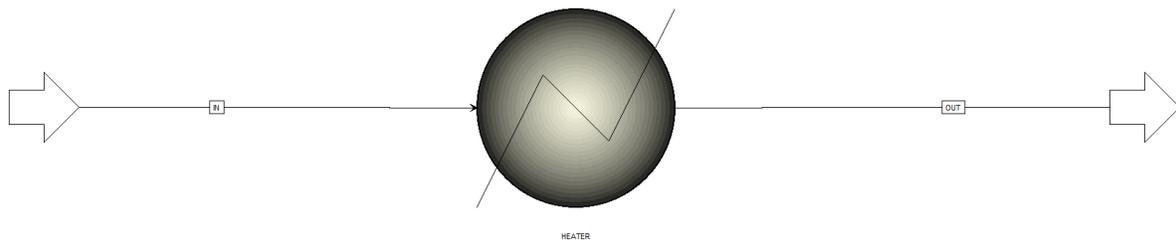


Figure 3.1. A simple **Heater** block called **HEATER** with an input stream **IN** and an output stream **OUT**.

Table 3.1. Comparison between Aspen Plus programming and C# programming.

Aspen Plus programming	<pre>MATERIAL in = new Stream(25°C, 1bar); HEATER heater = new Heater(T = 40°C, ΔP = 0); MATERIAL out = HEATER(IN);</pre>
C# programming	<pre>double X = 5; Func<double, double> CUBE = (double x) => { return x * x * x; }; double Y = CUBE(X);</pre>

Our Aspen Plus simulations are just a list of instructions like any other program. We clearly see this in the Control Panel such as in Figure 3.2. Observations:

- i. An Aspen Plus simulation starts at the beginning of its list of instructions and performs them in order until complete.

- ii. Aspen Plus simulations can loop back to an earlier instruction when in a **Convergence** block. This is exactly the same thing as a loop in other programming languages.
- iii. Aspen Plus simulations can produce console messages while running. C++ programmers can think of this as the `cout` (“console out”) command used from their first “Hello World!” program.

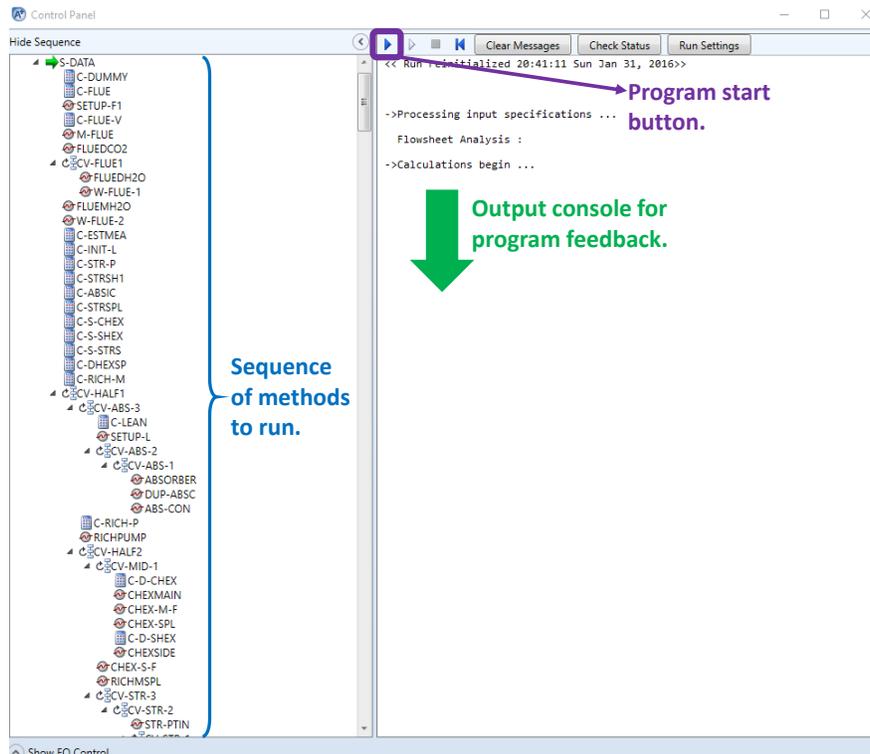


Figure 3.2. Control Panel from an Aspen Plus simulation run. We can understand Aspen Plus simulations as ordinary computer programs. Components shown here match up with normal computer programs as labeled.

All flowsheeting block types, like **Heater**, **Flash2**, **RadFrac**, and **Calculator**, are all class types.

Objects created according to a class type are instances. For example, **HEATER** in Figure 3.1 is an instance of the class type **Heater**. All of the *types* of blocks provided in Aspen Plus are class types while all of the specific examples of those blocks are instances.

3.3 Working with Aspen Plus using C#

This section demonstrates a simple Visual Studio project that controls Aspen Plus.

3.3.1 Setting up the project

Start Visual Studio 2015 and create a new project as shown in Figure 3.3.

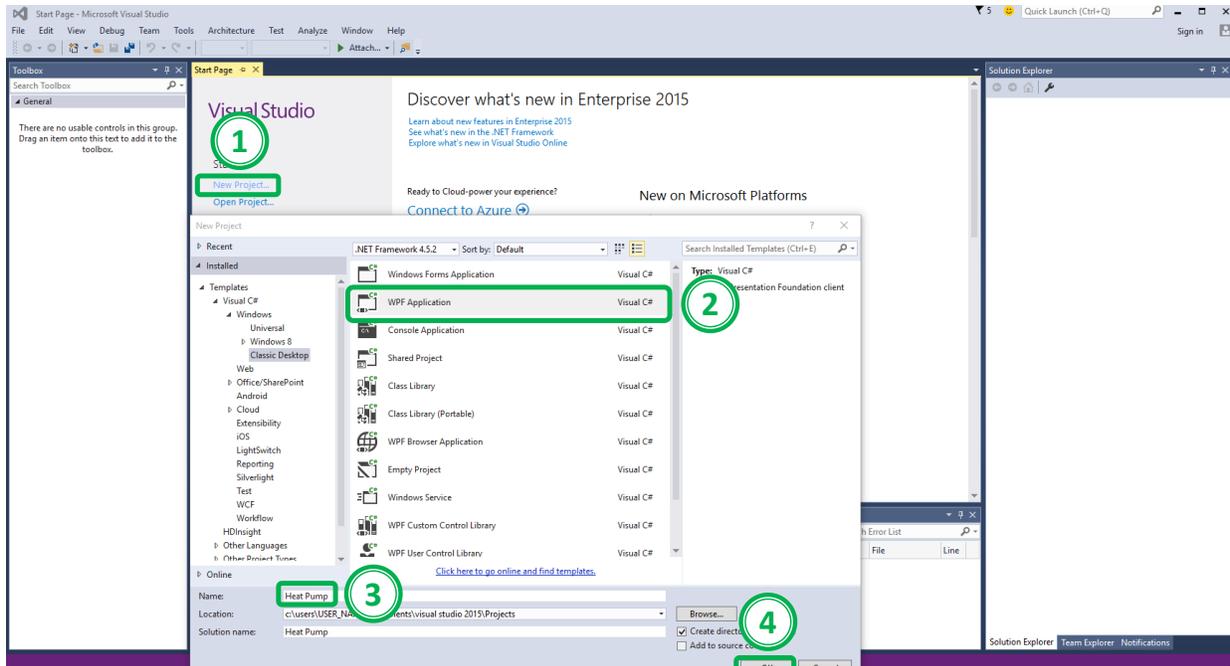


Figure 3.3. Create a new project, “Heat Pump”, as a WPF Application in Visual Studio 2015.

Next, go to

Visual Studio | Project → Add Reference | Browse

and search for the type definition library file, “happ.tlb”, in the directory of your computer in which Aspen Tech software is installed, usually under “Program Files (x86)”. You may find multiple copies. Check the box next to “happ.tlb” and select “OK” as shown in Figure 3.4.

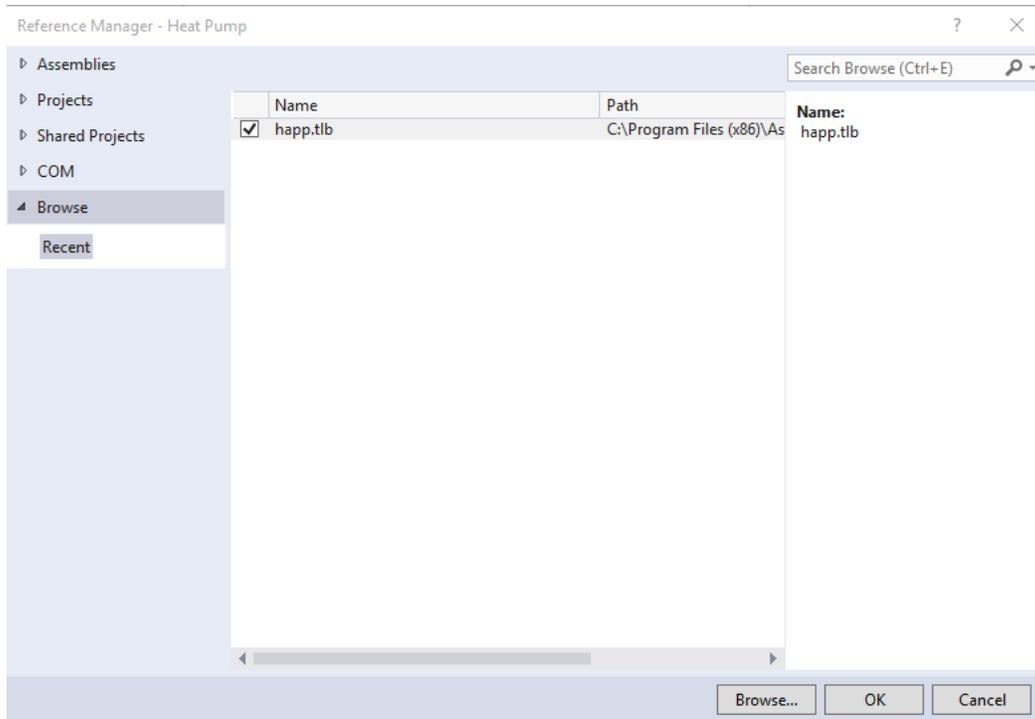


Figure 3.4. Add the type-definition library file “happ.tlb” as a reference for your new Visual Studio project. This type-definition library provides Visual Studio with the information needed to interact with Aspen Plus.

We then see the COM interface “Happ” can be seen in the Solution Explorer as shown in Figure 3.5. Visual Studio constructed this COM interface from the information in “happ.tlb”.

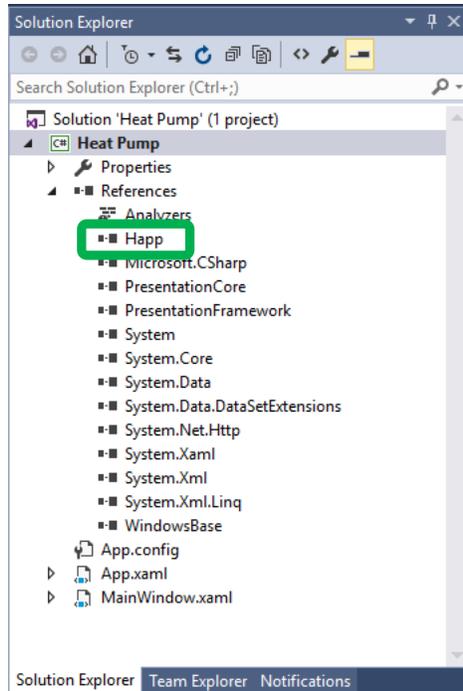


Figure 3.5. We can see the newly added reference to “happ.tlb” in Solution Explorer as “Happ”.
Visual Studio needs Happ in its References list to interact with Aspen Plus.

In the Solution Explorer, right-click “Heat Pump” and select Add → New Folder. Rename the new folder to “DoSomething”.

Next, right-click the new folder, DoSomething, and select Add → New Item. Add a new class code file called “_DoSomething.cs” as shown in Figure 3.6.

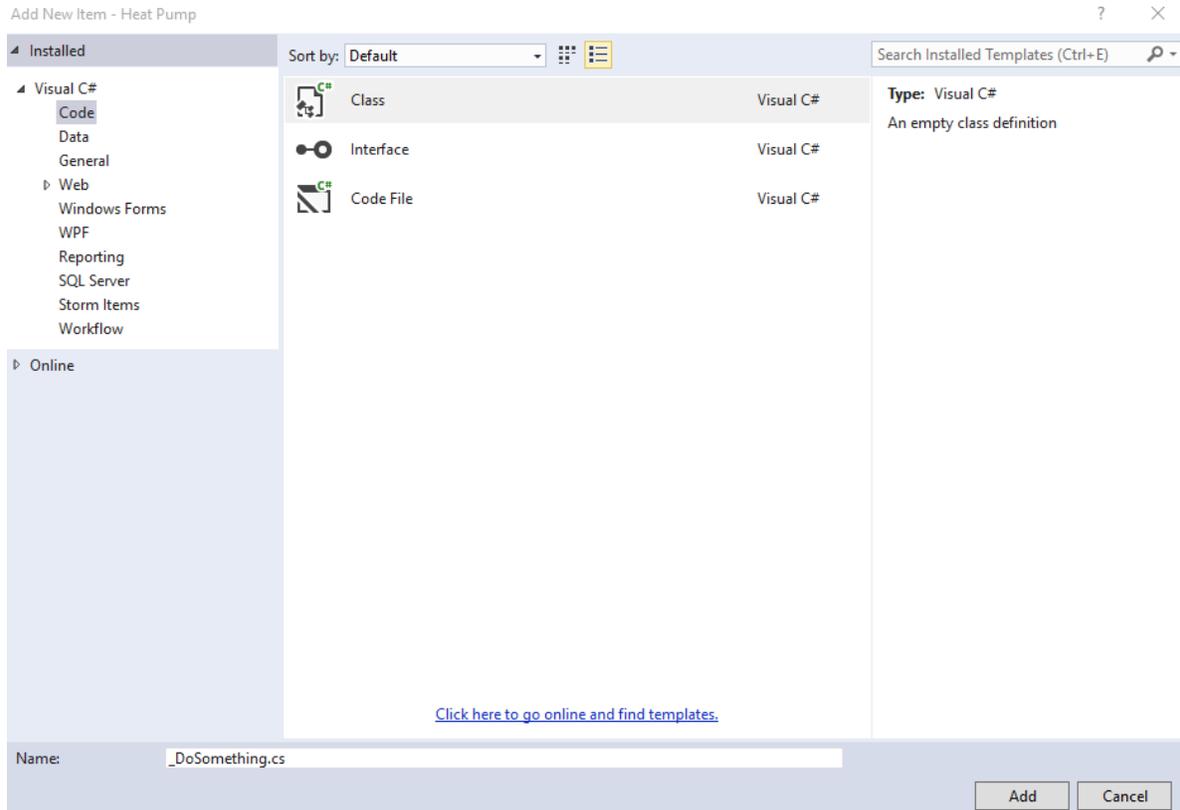


Figure 3.6. Add a new code file called “_DoSomething.cs” to the directory “DoSomething” in our Heat Pump project.

Open the newly created `_DoSomething.cs` as shown in Figure 3.7 and modify the code as shown in Figure 3.8. `DoSomething` will serve as our central control center.

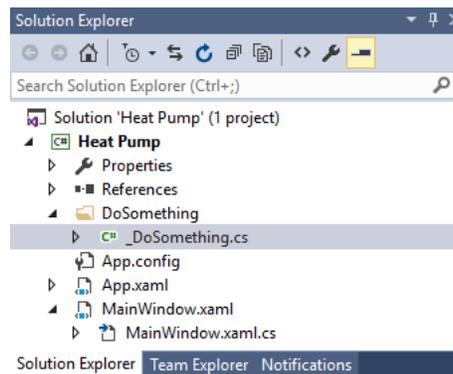


Figure 3.7. Open `_DoSomething.cs` from Solution Explorer under the `DoSomething` folder.

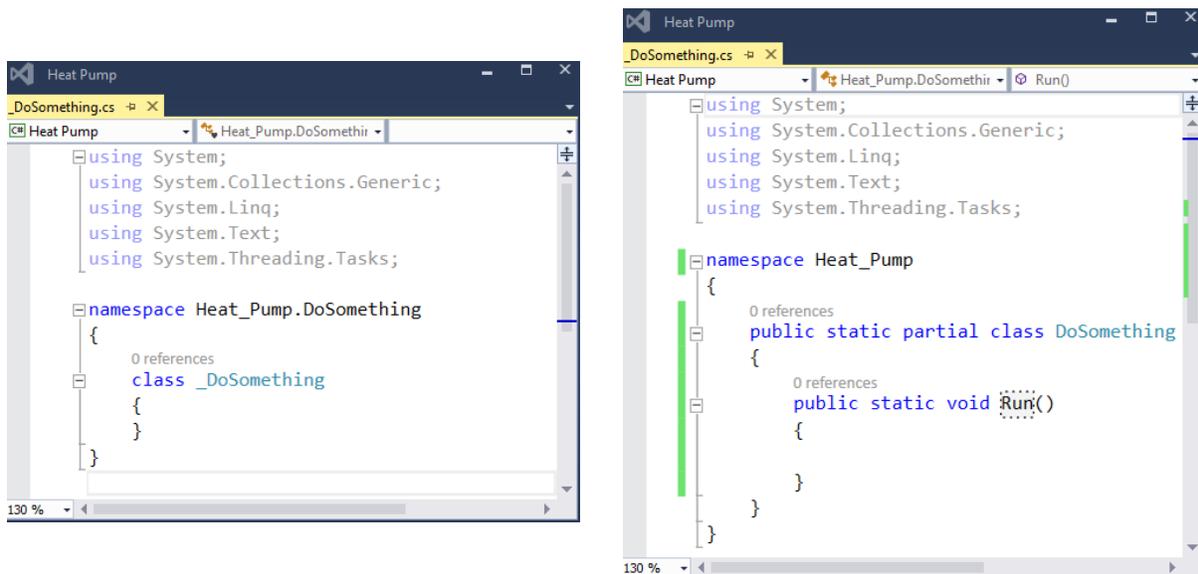


Figure 3.8. Change the code in `_DoSomething.cs` (left) to the new code shown on the right by removing part of the namespace qualifier and declaring the class `DoSomething` to be public, static, and partial.

We need to tell Visual Studio to execute our new method, `DoSomething.Run()`, whenever our program's main window is made. So, we will go to `MainWindow.xaml.cs` and modify it as shown in Figure 3.9.

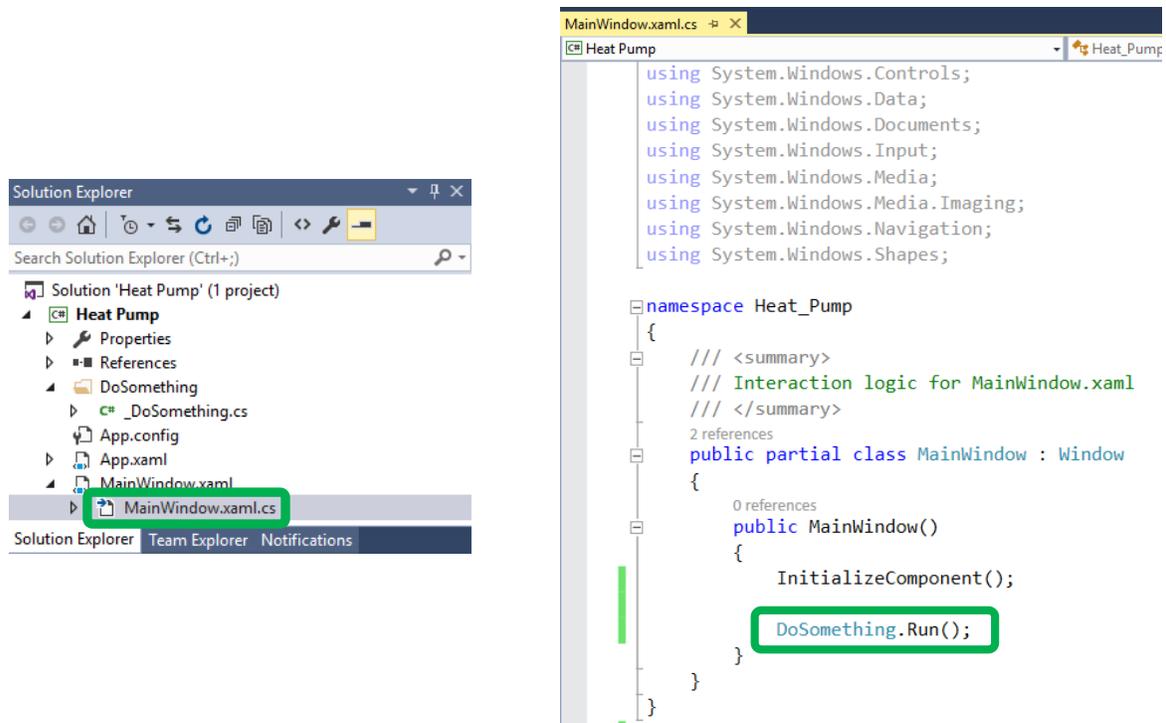


Figure 3.9. Navigate to `MainWindow.xaml.cs` and add `DoSomething.Run()` ; in `MainWindow`'s constructor. Since an object's constructor runs whenever a new instance of the object type is made, `DoSomething.Run()` will execute whenever our `MainWindow` is created.

Now, we will make code for our central control center to call on. Create `DoSomething001.cs` in the same way that we made `_DoSomething.cs`, then set its source code to match that in Figure 3.10. We note the following features of this source:

- i. The `#define` statement is a pre-processing directive that tells Visual Studio that some term, in this case "INCLUDE", is defined.
- ii. The `#if` and `#endif` statements are also pre-processing directives. These pre-processing directives allow the code within them to continue existing as long as "INCLUDE" is defined.
- iii. The `using Happ;` statement informs Visual Studio that we are using content from Aspen Plus's type library.

```
DoSomething001.cs - X
Heat Pump
Heat_Pump.DoSomething

#define INCLUDE
#if INCLUDE
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using Happ;
namespace Heat_Pump
{
    2 references
    partial class DoSomething
    {
        0 references
        public static void DoSomething001()
        {
        }
    }
}
#endif
```

Annotations in the image:

- Red box around `#define INCLUDE` with arrow pointing to "Pre-processing directives."
- Red box around `#if INCLUDE` with arrow pointing to "Pre-processing directives."
- Purple box around `using Happ;` with arrow pointing to "This using statement references Aspen Plus's type library."
- Red box around `#endif` with arrow pointing to "Pre-processing directive."

Figure 3.10. Source code for DoSomething001.cs. We note the preprocessing directives and the new `using Happ;` directive.

If we remove `INCLUDE`'s defining, such as by commenting out `#define INCLUDE`, then `DoSomething001()` effectively ceases to exist within our project. We show the commented-out `#define INCLUDE` directive and its consequences in Figure 3.11.

```
DoSomething001.cs
Heat Pump
//#define INCLUDE
#if INCLUDE
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using Happ;

namespace Heat_Pump
{
    partial class DoSomething
    {
        public static void DoSomething001()
        {

        }
    }
}
#endif
```

This statement is commented out, so INCLUDE is no longer defined in this file.

This code is greyed out and no longer included in our program because #if INCLUDE no longer tests as TRUE.

Figure 3.11. We comment-out `#define INCLUDE` by prefixing the line with two slashes, `//`. As a result, all of the code inside of the `#if INCLUDE` and `#endif` directives is no longer part of the project. We can easily reverse this by removing the two slashes.

We would like a set of `DoSomethingX()` methods to hold our code samples. Copy and paste `DoSomething001.cs` nine times, then rename the resulting files to be “`DoSomething002.cs`” through “`DoSomething010.cs`”. Change the “001” in each file’s `public static void DoSomething001()` to match the number in the file’s filename. For example, “`DoSomething007.cs`” should contain `public static void DoSomething007()`. Then add a commented-out call to each of these new methods in `DoSomething.Run()` as shown in Figure 3.12.

```

namespace Heat_Pump
{
    11 references
    public static partial class DoSomething
    {
        1 reference
        public static void Run()
        {
            DoSomething001();
            //DoSomething002();
            //DoSomething003();
            //DoSomething004();
            //DoSomething005();
            //DoSomething006();
            //DoSomething007();
            //DoSomething008();
            //DoSomething009();
            //DoSomething010();
        }
    }
}

```

Figure 3.12. `DoSomething.Run()` will now serve as a switch board for the other source code files. This configuration will make it easier for us to test new code.

3.3.2 Opening Aspen Plus through COM

Now we will try to open Aspen Plus. Go to `DoSomething001.cs` and insert the code shown in Figure 3.13.

```

1 reference
public static void DoSomething001()
{
    // Create a new instance of Aspen Plus.
    var aspenPlusInstance = new HapPLS();

    // Create a new simulation in this instance.
    aspenPlusInstance.InitNew2();

    // Make this new simulation visible so we
    // can see it.
    aspenPlusInstance.Visible = true;
}

```

Figure 3.13. Two simple C# commands to start a new Aspen Plus instance and then make that instance visible.

Finally, we can start our new program by pressing the “Start” button in the ribbon as shown in Figure 3.14. Our program may take some time to start because it is also starting a new instance of Aspen Plus; we will not see the results until Aspen Plus has started and retrieved a license from the license server.

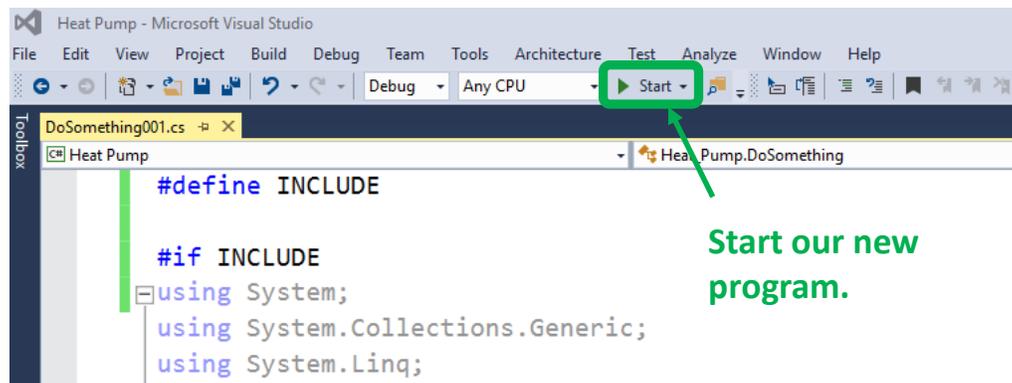


Figure 3.14. The Start button to run our program.

The program creates two new windows as shown in Figure 3.15:

1. A new instance of **MainWindow**. This window is blank because we have not defined any content for it.
2. A new instance of Aspen Plus showing a new simulation created in **DoSomething001**, following the instruction in Figure 3.13.

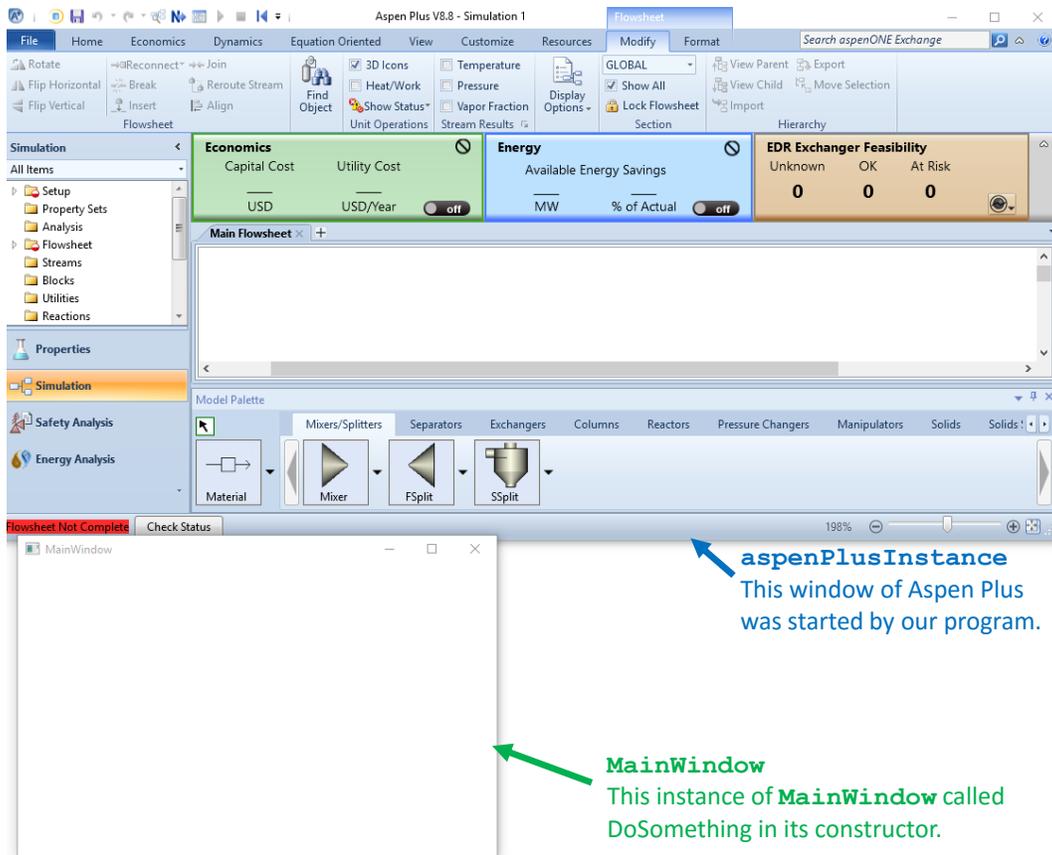


Figure 3.15. Our program created two new windows. The first is an empty window that doesn't really do anything; this is **MainWindow**, which is still blank because we have not defined it. The second window is an instance of Aspen Plus with a new simulation, just as we instructed in `DoSomething001 ()`.

We can close these windows normally. Note that Aspen Plus does not close even if you end the program that started it.

3.3.3 Interacting with a basic Aspen Plus simulation

Now, we try interacting with Aspen Plus, controlling our simulations from C#. To do this, first create a basic simulation for us to control:

1. Make a new Aspen Plus simulation and save it as "Basic.apwz".
2. Go to

Aspen Plus | Properties | Components → Specifications
and H₂O, O₂, N₂, and H₂ as the simulation's components.

3. Go to

Aspen Plus | Properties | Methods → Specifications

and select the “IDEAL” property method for the base method field.

4. Go to

Aspen Plus | Simulation

and open the flowsheet. Draw a **Heater HEATER** with an input material stream called **IN** and an output stream called **OUT**. Figure 3.16 shows this flowsheet along with the input specifications for **HEATER** and **IN**:

- a. **IN** has $1 \frac{\text{kmol}}{\text{hr}}$ of both H₂O and N₂ at 25°C and 1bar;
- b. **HEATER** is at 15°C and 1bar.

5. Save and close the simulation.

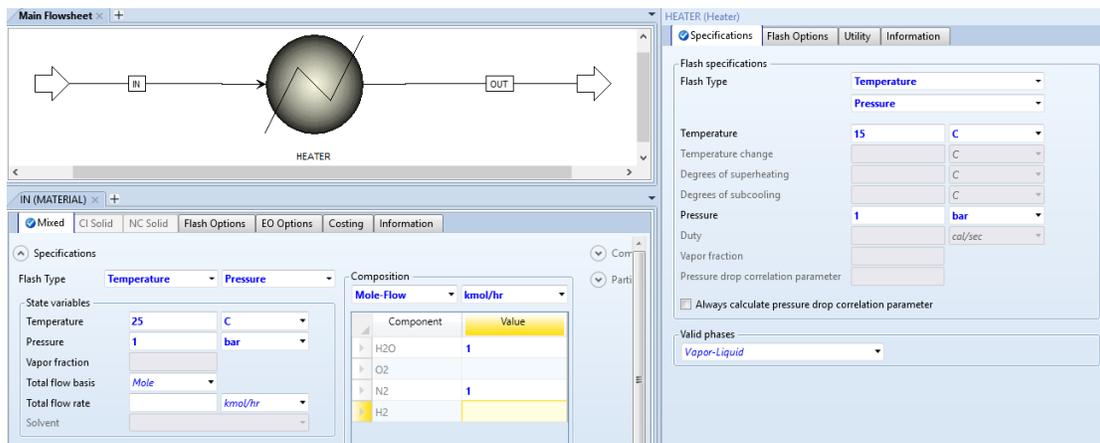


Figure 3.16. Specifications for our simple test flowsheet. A **Heater** block, **HEATER**, has an input stream **IN** and an output stream **OUT**. **IN** is at 25°C and 1bar with $1 \frac{\text{kmol}}{\text{hr}}$ of each water and nitrogen. **HEATER** is at 15°C and 1bar.

We now open this simulation using C# through **DoSomething002**. Set **DoSomething002**'s code as shown in Figure 3.17 and modify the run list as shown in Figure 3.18. Run your program to ensure that it correctly opens the basic simulation just as **DoSomething001** opened a new, blank simulation.

```

public static void DoSomething002()
{
    // Create a new instance of Aspen Plus.
    var aspenPlusInstance = new HappLS();

    // Open a simulation file.
    aspenPlusInstance.InitFromFile2(
        @"C:\Users\UserName\Documents\Simulations\Basic.apwz"
    );
    // Use the correct path and filename for your simulation.

    // Make this new simulation visible so we
    // can see it.
    aspenPlusInstance.Visible = true;
}

```

Figure 3.17. Code to open our new simulation.

```

public static partial class DoSomething
{
    1 reference
    public static void Run()
    {
        //DoSomething001(); // Opens a new simulation.
        DoSomething002(); // Opens our basic simulation with a single HEATER.
        //DoSomething003();
        //DoSomething004();
        //DoSomething005();
        //DoSomething006();
        //DoSomething007();
        //DoSomething008();
        //DoSomething009();
        //DoSomething010();
    }
}

```

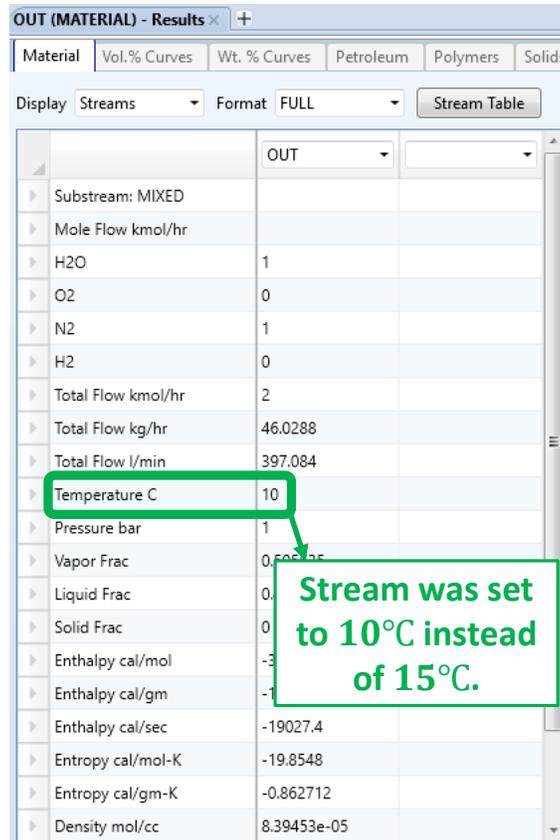
Figure 3.18. Instruct our program to run `DoSomething002()` instead of `DoSomething001()` by adjusting the commenting in `DoSomething.Run()`.

Next, we amend `DoSomething002` to also have it run the simulation with **HEATER** at 10°C instead of the flowsheet-specified 15°C. Add the code in Figure 3.19 to `DoSomething002` and run the C# program. The simulation should automatically open and run. Then, inspect the run results for the material stream **OUT**. **OUT** shows the C#-set 10°C temperature rather than the flowsheet-specified 15°C that we previously set in Figure 3.16. Figure 3.20 shows this result.

```
// Set HEATER's temperature to 10degC.
aspenPlusInstance.Tree.FindNode(@"\Data\Blocks\HEATER\Input\TEMP").Value = 10.0;

// Run the simulation.
aspenPlusInstance.Run2();
```

Figure 3.19. Amend this code to the end of **DoSomething002**. This code will set **HEATER**'s temperature to 10°C, overwriting the flowsheet-specified 15°C, and then run the simulation.



Property	Value
Substream: MIXED	
Mole Flow kmol/hr	
H2O	1
O2	0
N2	1
H2	0
Total Flow kmol/hr	2
Total Flow kg/hr	46.0288
Total Flow l/min	397.084
Temperature C	10
Pressure bar	1
Vapor Frac	0.66235
Liquid Frac	0
Solid Frac	0
Enthalpy cal/mol	-3
Enthalpy cal/gm	-1
Enthalpy cal/sec	-19027.4
Entropy cal/mol-K	-19.8548
Entropy cal/gm-K	-0.862712
Density mol/cc	8.39453e-05

Figure 3.20. The results for **OUT** from **DoSomething002**. We can see that the simulation used 10°C, as specified in our code, rather than 15°C, as we had previously specified in the flowsheet.

In the code, we use `\Data\Blocks\HEATER\Input\TEMP` to access the input temperature of the **HEATER**. Figure 3.21 shows that we can look up these addresses in Variable Explorer in Aspen Plus.

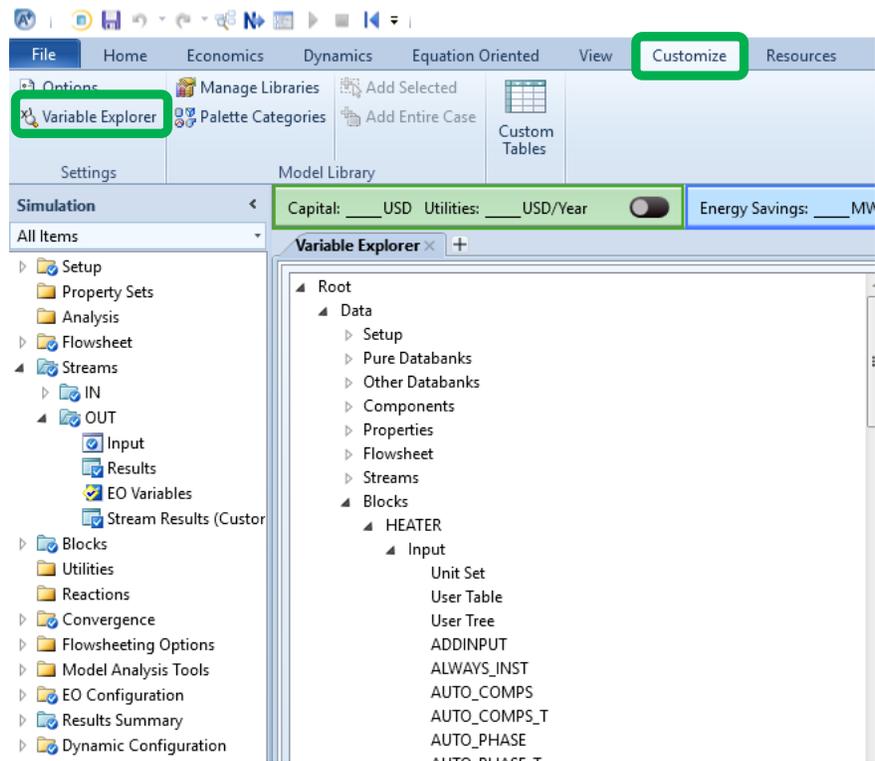


Figure 3.21. Aspen Plus's Variable Explorer feature allows us to explore its internal data tree. We can use this to find the addresses for information that we want to interact with. Note that many nodes exist only when relevant; for example, many Output nodes will only be visible when simulation results are available.

Variable Explorer also gives details about specific nodes. We highlight some of the major points in Figure 3.22.

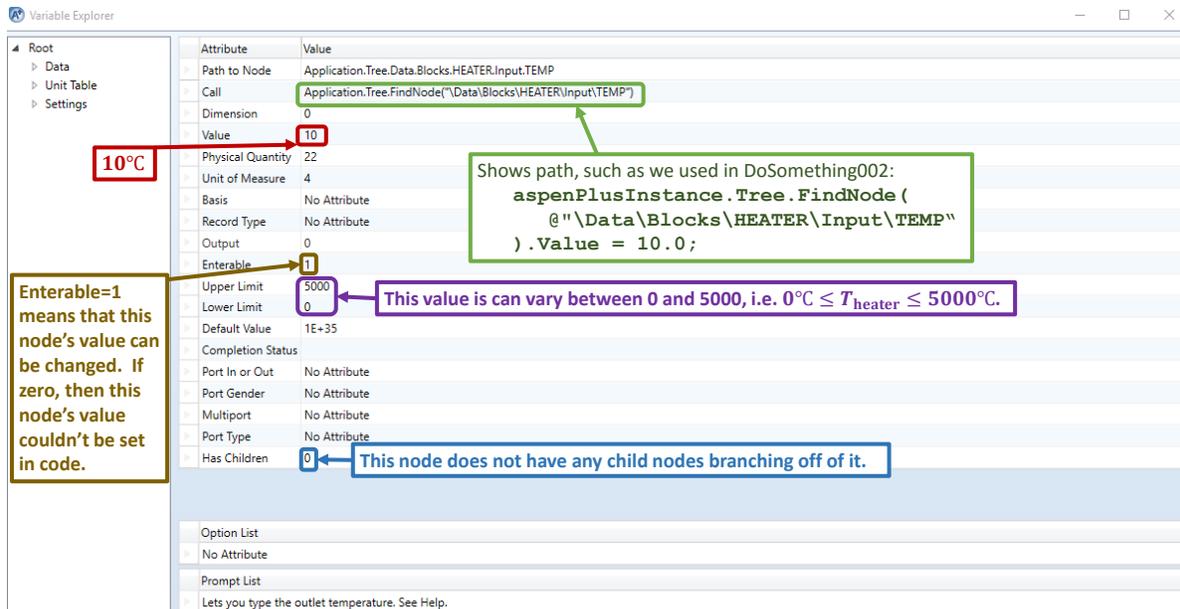


Figure 3.22. The `\Data\Blocks\HEATER\Input\TEMP` node from our basic simulation after it was run by our program. We can learn about this node's properties using Variable Explorer.

3.3.4 Run Aspen Plus in a separate thread

Our program is currently synchronous and single-threaded which means that:

1. We can only make use of a single processor (CPU core) at a time. Most desktop computers have at least four cores, so single-threaded programs fail to use the great majority of available computational power.
2. We have frozen the critical parts of our program, such as the user interface, while long-running simulations are working.

Learning to create multi-threaded programs can require some effort, though it is necessary to both use the computer's full processing power and make responsive programs.

For now we will modify the code in `MainWindow.xaml.cs` as shown in Figure 3.23. The new code calls `DoSomething.Run()` like the old code did, but it also instructs the computer to run the method in a new thread.

```

public partial class MainWindow : Window
{
    0 references
    public MainWindow()
    {
        InitializeComponent();

        (new System.Threading.Thread(() => { DoSomething.Run(); })).Start();
    }
}

```

Figure 3.23. Modified code for **MainWindow**'s constructor. The new code runs **DoSomething.Run ()** in a separate thread, freeing **MainWindow** to continue normal operations rather than having to wait for **DoSomething.Run ()** to finish.

Run the program. Unlike in prior runs, **MainWindow** starts immediately rather than awaiting Aspen Plus. **MainWindow** is now free to interact with the user while the simulation runs.

3.3.5 Adding a custom debug message console

Use Visual Studio's Solution Explorer to add a new folder called "Utilities" to the project and add a two new code files, **_Utilities.cs** and **DebugWindow.cs**. This process is similar to how we created the folder and code files for **DoSomething** in Section 3.3.1: "Setting up the project".

Set the source code for **_Utilities.cs** as shown in Figure 3.24. Note that **static Utilities ()** is a static constructor which will run once more **Utilities** is accessed.

```

namespace Heat_Pump
{
    2 references
    public static partial class Utilities
    {
        0 references
        static Utilities()
        {
            #if DEBUG
                MainDebugWindow = DebugWindow.New();
            #endif
        }
    }
}

```

Figure 3.24. Source code for `_Utilities.cs`, excluding the default `using` statements at the top.

We can change the pre-processor directive `#if DEBUG` as shown in Figure 3.25. If running in Debug mode, `DEBUG` is defined so the code in the `#if DEBUG` block is part of the project. This code is effectively removed from the program when running in Release mode. We use this feature to insert debugging features without bogging down the final product.

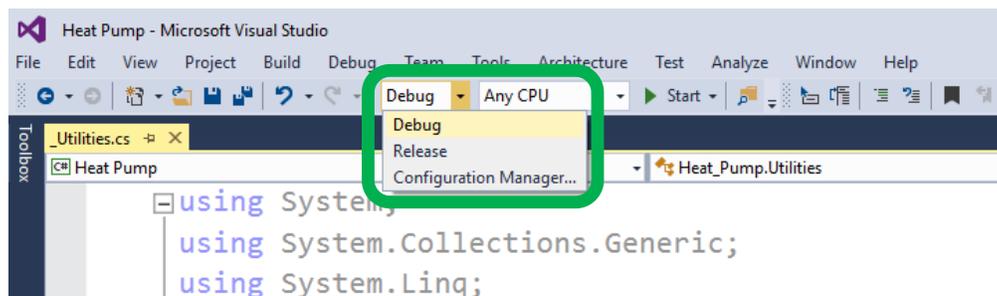


Figure 3.25. Debug and Release modes in Visual Studio 2015.

Next, set the source code for `DebugWindow.cs` to that shown in Scheme 3.1.

```

using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Linq;
using System.Text;

```

```

using System.Threading.Tasks;
using System.Windows;
using System.Windows.Controls;
using System.Windows.Data;
using System.Windows.Documents;
using System.Windows.Input;
using System.Windows.Media;
using System.Windows.Media.Imaging;
using System.Windows.Navigation;
using System.Windows.Shapes;

namespace Heat_Pump
{
    partial class Utilities
    {

#if DEBUG
        private static DebugWindow MainDebugWindow;
        public static void Report(string text)
        {
            MainDebugWindow.Report(text);
        }

        public class DebugWindow
            : Window
        {
            private TextBlock _reportTextBlock;

            protected DebugWindow()
                : base()
            {
                System.Threading.Thread thread =
System.Threading.Thread.CurrentThread;
                this.DataContext = new { ThreadId = thread.ManagedThreadId };

                //this.WindowStyle = WindowStyle.None;
                //this.AllowsTransparency = true;
                this.Background = Brushes.White;
            }
        }
#endif
    }
}

```

```

this.Topmost = true;
this.Title = "Debug Window";

StackPanel contentStackPanel = new StackPanel();
contentStackPanel.Orientation = Orientation.Vertical;
contentStackPanel.Background = Brushes.Transparent;
this.Content = contentStackPanel;

StackPanel headerStackPanel = new StackPanel();
headerStackPanel.Orientation = Orientation.Horizontal;
contentStackPanel.Children.Add(headerStackPanel);

Canvas draggingCanvas = new Canvas();
draggingCanvas.HorizontalAlignment = HorizontalAlignment.Left;
draggingCanvas.Height = 15;
draggingCanvas.Width = 15;
draggingCanvas.Background = Brushes.Green;
headerStackPanel.Children.Add(draggingCanvas);

Button copyToClipboardButton = new Button();
copyToClipboardButton.Content = "Copy";
copyToClipboardButton.Click += CopyToClipboardButton_Click;
headerStackPanel.Children.Add(copyToClipboardButton);

Button clearMessagesButton = new Button();
clearMessagesButton.Content = "Clear messages";
clearMessagesButton.Click += ClearMessagesButton_Click;
headerStackPanel.Children.Add(clearMessagesButton);

draggingCanvas.MouseLeftButtonDown +=
DraggingCanvas_MouseLeftButtonDown;

TextBlock reportTextBlock = new TextBlock();
reportTextBlock.Text = "";
contentStackPanel.Children.Add(reportTextBlock);
this.ReportTextBlock = reportTextBlock;
}

```

```

        private void ClearMessagesButton_Click(object sender,
RoutedEventArgs e)
        {
            var thisReportTextBlock = this.ReportTextBlock;
            if (thisReportTextBlock == null) { return; }

            thisReportTextBlock.Text = "";
        }

        private void CopyToClipboardButton_Click(object sender,
RoutedEventArgs e)
        {
            var thisReportTextBlock = this.ReportTextBlock;
            if (thisReportTextBlock == null) { return; }

            string messages = thisReportTextBlock.Text;
            Clipboard.SetText(messages);
        }

        protected TextBlock ReportTextBlock
        {
            get
            {
                return this._reportTextBlock;
            }
            private set
            {
                this._reportTextBlock = value;
            }
        }

        public void Report(string text)
        {
            Action<string> action = new Action<string>(InternalReport);
            this.Dispatcher.BeginInvoke(action, text);
        }

```

```

protected void InternalReport(string text)
{
    this.ReportTextBlock.Text += Environment.NewLine + text;
}

private void DraggingCanvas_MouseLeftButtonDown(object sender,
    MouseButtonEventArgs e)
{
    this.DragMove();
}

public class AwaitResultContainer<T>
{
    private bool _completed;
    private T _result;
    private System.Threading.ManualResetEvent _manualResetEvent;

    public bool Completed
    {
        get
        {
            return this._completed;
        }
        protected set
        {
            this._completed = value;
        }
    }

    public T Result
    {
        get
        {
            return this._result;
        }
        protected set
        {
            this._result = value;
        }
    }
}

```

```

    }
    protected System.Threading.ManualResetEvent ManualResetEvent
    {
        get { return this._manualResetEvent; }
        private set { this._manualResetEvent = value; }
    }
    protected AwaitResultContainer()
    {
        this.Completed = false;
        this.ManualResetEvent = new
System.Threading.ManualResetEvent(false);
    }
    public static AwaitResultContainer<T> New()
    {
        return new AwaitResultContainer<T>();
    }

    private object ReportResultLockObject = new object();
    public void ReportResult(T result)
    {
        lock (ReportResultLockObject)
        {
            if (this.Completed) { throw new Exception(
                "Error: Result already provided. "
            ); }
            this.Result = result;
            this.Completed = true;
            this.ManualResetEvent.Set();
        }
    }
    public void WaitForResult()
    {
        lock (this)
        {
            if (!this.Completed)
            {
                this.ManualResetEvent.WaitOne();
            }
        }
    }

```

```

        }
    }
}

public static DebugWindow New()
{
    AwaitResultContainer<DebugWindow> toReturnContainer =
        AwaitResultContainer<DebugWindow>.New();

    System.Threading.Thread thread = new System.Threading.Thread(
        () =>
        {
            DebugWindow toReturn = new DebugWindow();

            toReturn.Show();

            toReturn.Closed += (sender2, e2) =>
                toReturn.Dispatcher.InvokeShutdown();

            toReturnContainer.ReportResult(toReturn);
            System.Windows.Threading.Dispatcher.Run();
        }
    );

    thread.SetApartmentState(System.Threading.ApartmentState.STA);
    thread.Start();

    toReturnContainer.WaitForResult();
    return toReturnContainer.Result;
}
}
#endif
}
}

```

Scheme 3.1. Source code for `DebugWindow.cs`.

We can now use `Utilities.Report(string message)` to issue feedback during Debug runs.

To try this, we will replace `DoSomething002` with the code given in Scheme 3.2. Figure 3.26 shows the results.

```
public static void DoSomething002()
{
    #if DEBUG
        Utilities.Report("Creating a new instance of Aspen Plus.");
    #endif

    // Create a new instance of Aspen Plus.
    var aspenPlusInstance = new HappLS();

    #if DEBUG
        Utilities.Report("Opening the basic simulation.");
    #endif

    // Open a simulation file.
    aspenPlusInstance.InitFromFile2(
        @"[insert the path to your simulation file here]"
    );

    #if DEBUG
        Utilities.Report("Making Aspen Plus visible.");
    #endif

    // Make this new simulation visible so we can see it.
    aspenPlusInstance.Visible = true;

    #if DEBUG
        Utilities.Report("Setting HEATER to 10degC.");
    #endif

    // Set HEATER's temperature to 10degC.
    aspenPlusInstance.Tree.FindNode(@"\Data\Blocks\HEATER\Input\TEMP").Value = 10.0;
}
```

```

#if DEBUG
    Utilities.Report("Running the simulation.");
#endif

// Run the simulation.
aspenPlusInstance.Run2();

// Retrieve the duty of HEATER.
var calculatedDuty =

    aspenPlusInstance.Tree.FindNode(@"\Data\Blocks\HEATER\Output\QCALC").Value;

#if DEBUG
    Utilities.Report(
        "Simulation complete! Found the heat duty was "
        + calculatedDuty.ToString()
        + " cal/sec."
    );
#endif

```

Scheme 3.2. Replace **DoSomething002**'s prior source code with the above which also includes feedback through calls to **Utilities.Report(string message)**.

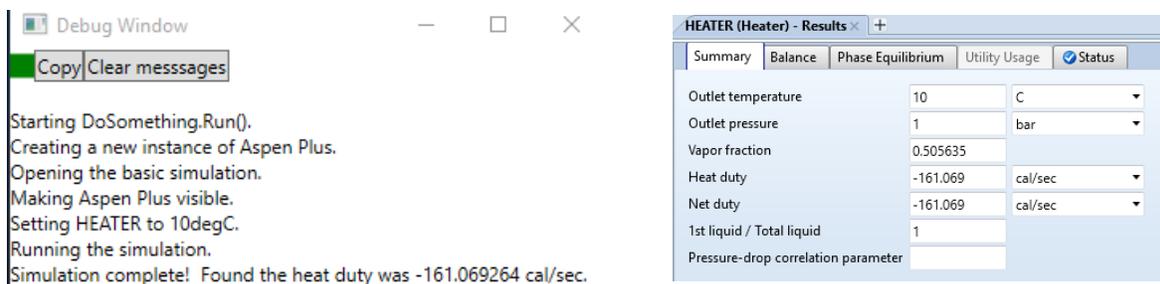


Figure 3.26. Debug feedback showing the **HEATER**'s calculated heat duty. We can see that this matches the calculated heat duty shown in Aspen Plus.

3.3.6 Custom design specifications

Let us consider putting **HEATER** into a **Design Spec** block that varies **HEATER**'s temperature to achieve a target duty such as $-100 \frac{\text{cal}}{\text{s}}$. We can implement the **Design Spec** in C# as shown in Scheme 3.3.

This implementation serves not only as the **Design Spec**, but also as the **Convergence** block that converges the design specification with the secant method and the **Sequence** block that specifies the order.

```
public static void DoSomething003()
{
    #if DEBUG
        Utilities.Report("Creating a new instance of Aspen Plus.");
    #endif

    // Create a new instance of Aspen Plus.
    var aspenPlusInstance = new HappLS();

    #if DEBUG
        Utilities.Report("Opening the basic simulation.");
    #endif

    // Open a simulation file.
    aspenPlusInstance.InitFromFile2(@"[insert the path to your simulation
file here]");

    #if DEBUG
        Utilities.Report("Making Aspen Plus visible.");
    #endif

    // Make this new simulation visible so we can see it.
    aspenPlusInstance.Visible = true;

    double targetDuty = -100.0;           // in cal/sec
    double dutyTolerance = 0.1;          // in cal/sec
    double temperature = 10.0;           // in degC
    double calculatedDuty = double.NaN;  // in cal/sec
    double lastTemperature = double.NaN; // in degC
    double lastError = double.NaN;       // in cal/sec
    bool converged = false;

    for (int i=0; i < 30; ++i)
```

```

    {
    #if DEBUG
        Utilities.Report(
            "Iteration #"
            + i.ToString()
            + ": T_heater = "
            + temperature.ToString()
            + " degC."
        );
    #endif

    aspenPlusInstance.Tree.FindNode(@"\Data\Blocks\HEATER\Input\TEMP").Value
    = temperature;
    aspenPlusInstance.Run2();
    calculatedDuty
    =
    aspenPlusInstance.Tree.FindNode(@"\Data\Blocks\HEATER\Output\QCALC").Value;
    #if DEBUG
        Utilities.Report(
            "Duty calculated to be "
            + calculatedDuty.ToString()
            + " cal/sec."
        );
    #endif

    double error = calculatedDuty - targetDuty;
    if (Math.Abs(error) <= dutyTolerance)
    {
        converged = true;
    #if DEBUG
        Utilities.Report(
            "Converged with an error of "
            + error.ToString()
            + " cal/sec."
        );
    #endif
}

```

```

        break;
    }

    if (i == 0)
    {
        lastError = error;
        lastTemperature = temperature;
        temperature += 5.0;
    }
    else
    {
        double cacheTemp = temperature;
        temperature -= error * (temperature - lastTemperature) / (error -
lastError);
        lastTemperature = cacheTemp;
        lastError = error;
    }
}

#if DEBUG
    if (converged)
    {
        Utilities.Report("Design Spec completed and successfully
converged.");
    }
    else
    {
        Utilities.Report("Design Spec completed but convergence was not
achieved.");
    }
#endif

#if DEBUG
    Utilities.Report(
        "Simulation complete! Found the heat duty was "
        + calculatedDuty.ToString()
        + " cal/sec for a temperature of "
        + temperature.ToString()

```

```

        + " degC."
    );
#endif
}

```

Scheme 3.3. Source code for `DoSomething003 ()` to perform a **Design Spec**-like operation on **HEATER**.

The debug window's copy button provides the feedback shown in Scheme 3.4. We see that the design spec finds $D_{\text{HEATER}} \approx -100.0094 \frac{\text{cal}}{\text{s}}$ at $T_{\text{HEATER}} \approx 16.243^\circ\text{C}$.

```

Starting DoSomething.Run().
Creating a new instance of Aspen Plus.
Opening the basic simulation.
Making Aspen Plus visible.
Iteration #0: T_heater = 10 degC.
Duty calculated to be -161.069264 cal/sec.
Iteration #1: T_heater = 15 degC.
Duty calculated to be -112.731517 cal/sec.
Iteration #2: T_heater = 16.3169332240495 degC.
Duty calculated to be -99.2389755 cal/sec.
Iteration #3: T_heater = 16.2426537824509 degC.
Duty calculated to be -100.009368 cal/sec.
Converged with an error of -0.009367999999999494 cal/sec.
Design Spec completed and successfully converged.
Simulation complete! Found the heat duty was -100.009368 cal/sec for a
temperature of 16.2426537824509 degC.

```

Scheme 3.4. Feedback after running `DoSomething003 ()` showing that our design spec successfully converged after Iteration #3.

We now replace the line specifying `double dutyTolerance = 0.1;` with `double dutyTolerance = 0.0001;`. Running again, we find that improving the tolerance by a thousand fold only requires one additional iteration, resulting in $D_{\text{HEATER}} \approx -100.000007 \frac{\text{cal}}{\text{s}}$ at $T_{\text{HEATER}} \approx 16.243557^\circ\text{C}$.

Our ability to increase tolerance will be limited by Aspen Plus's own tolerance. Aspen Plus is using limited precision, so our own methods relying on Aspen Plus cannot be more precise.

3.4 Creating an acid-gas-capture simulation in C#

Now that we have the basics for interacting with Aspen Plus, we will create a working acid-gas-capture simulation. We use a mixture of piperazine (PZ, $C_4H_{10}N_2$) and methyl diethanolamine (MDEA, $C_5H_{13}NO_2$) in aqueous solution to capture CO_2 .

3.4.1 Build the absorber

In our prior Aspen Plus simulations, we described the absorber by the derived object **SQ-ABS-2**. This object:

1. Uses a **RadFrac** block **ABSORBER** for its packed section.
2. Uses a **Flash2** block **ABS-CON** for its condenser.
3. Adjusts the packing diameter to achieve a target maximum approach to flooding, e.g. $\mathcal{F}_{ABS} \approx 60\%$.
4. Adjusts the lean solvent flow rate to achieve a target carbon capture rate, e.g. $R \approx 80\%$.
5. Uses the Wegstein method to tear the material flow rates and the secant method to adjust the lean solvent flow rate.

We will now create the effective equivalent as a C# program that calls Aspen Plus.

3.4.1.1 Create the Aspen Plus simulation for the absorber

First, open the simulation “ENRTL-RK_Rate_Based_PZ_Model.bkp” under the “\AspenTech\Aspen Plus V8.8\GUI\Examples\Amines_ENRTL-RK” directory. As before, delete the flowsheet-level content from it except for the Reactions and insert the new unit set. Then draw the flowsheet content shown in Figure 3.27. Table 3.2 gives the input stream specifications.

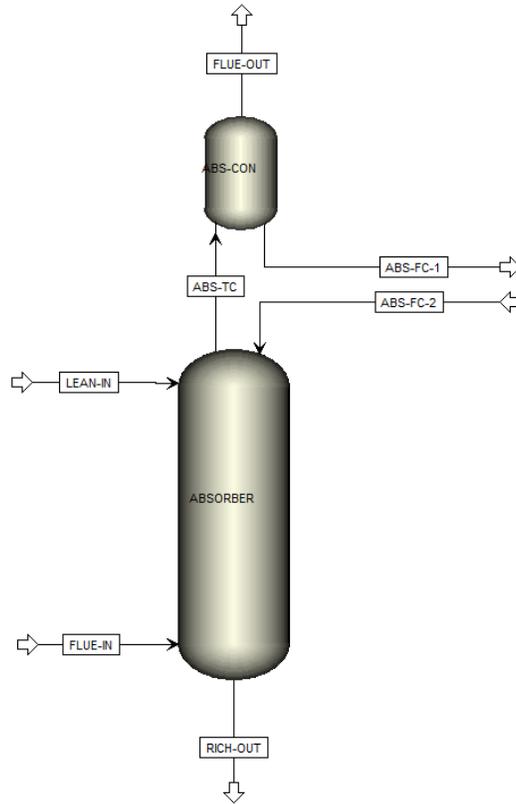


Figure 3.27. Flowsheet configuration for our new PZ+MDEA simulation's absorption unit.

Table 3.2. Stream table inputs for the PZ+MDEA absorption simulation.

		LEAN-IN	FLUE-IN	ABS-FC-2
Temperature (°C)		50.00024	50	40
Pressure (bar)		2	1.01325	1
Component mole flow rate ($\frac{\text{kmol}}{\text{hr}}$)	MDEA	0.426788	0	0
	PZ	0.088733	0	0
	H₂O	10.28188	0.258014	1.8
	CO₂	2.74E-06	0.140246	0
	H₂S	0	0	0
	H₃O⁺	5.92E-11	0	0
	OH⁻	4.86E-05	0	0
	HCO₃⁻	0.008561	0	0
CO₃²⁻	0.002132	0	0	

HS⁻	0	0	0
S²⁻	0	0	0
MDEA⁺	0.008768	0	0
PZH⁺	0.021871	0	0
PZH²⁺	0	0	0
HPZCOO	0.00294	0	0
PZCOO⁻	0.016495	0	0
PZCOO²⁻	0.000635	0	0
N₂	0	2.004728	0
O₂	0	0.288142	0
CO	0	0	0
H₂	0	0	0

For **ABS-CON**, specify a pressure drop of zero and a temperature of 40°C.

For **ABSORBER**, specify 25 stages. Specify the reactions left in the simulation for all stages with the prior holdup, and use a single pressure of 1bar for the entire **RadFrac**. The streams **LEAN-IN** and **ABS-FC-2** as entering above Stage #1 while **FLUE-IN** enters the bottom stage with the On-Stage convention. Keep the same packing type and conventions for all stages that the original AspenTech simulation used for the bulk absorber (as opposed to the wash section).

Run the simulation within Aspen Plus. Once the **RadFrac** block **ABSORBER** converges:

1. Reconcile **ABSORBER**.
2. Turn off equilibrium initialization for **ABSORBER**'s rate-based convergence.
3. Reinitialize the simulation.
4. Save the simulation under a new filename.

3.4.1.2 [Converging the absorption tower](#)

As in our prior simulations, the absorber's condenser is in a closed loop with the absorber's packing as shown in Figure 3.28 rather than as in how we actually draw the flowsheet from Figure 3.27. We will

need to converge this feedback loop as Aspen Plus would have if it was drawn as in Figure 3.28, with **ABS-FC-1** and **ABS-FC-2** being two manifestations of the same stream. Aspen Plus uses the Wegstein method to converge this tear stream, so we will do the same.

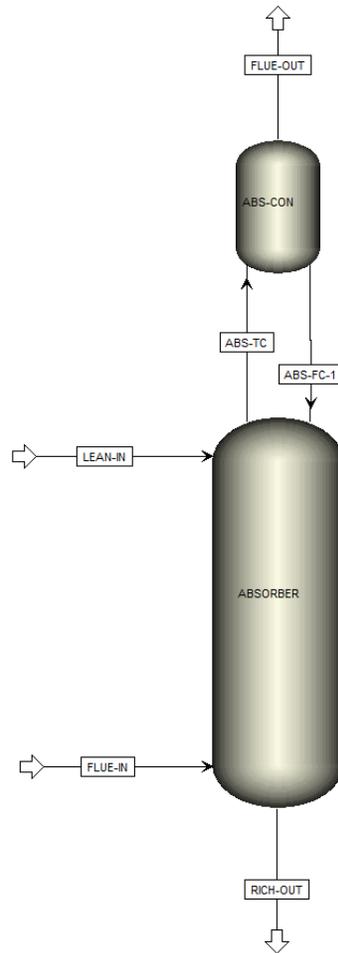


Figure 3.28. Conceptual absorption tower with the feedback from the condenser to the packing,
ABS-FC-1.

The Wegstein method will act on relatively many tear variables rather than just the single variable the prior secant method operated on. This is, rather than varying a single parameter like temperature, we will be vary one mole flow rate for each of the 21 chemical species in the simulation.

To make our code shorter and easier, we add some helper functions to our project. This will require a bit of coding. First, add a new Math folder along with the sub-folders and files shown in Figure 3.29.

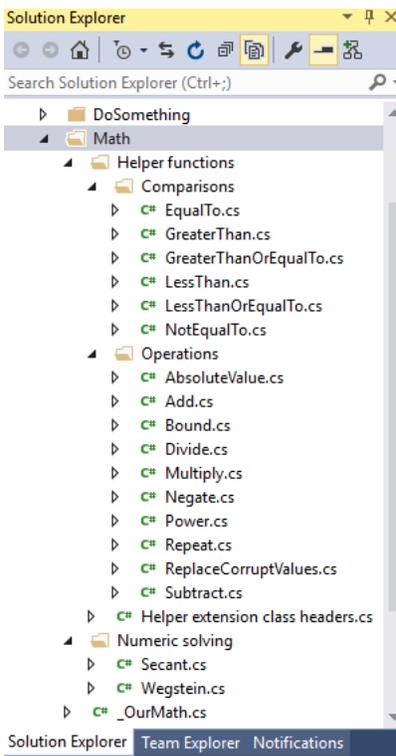


Figure 3.29. The new Math folder and its contents.

The helper functions largely help us operate easily on many variables at the same time so we do not need to write many loops for each math operation on our 21 tear variables. The numeric solving functions provide implementations of the secant method, which we have already used, and of the Wegstein method for our current use.

Now we define `DoSomething004 ()` as shown in Scheme 3.5. Also amend

```
//#define REINITIALIZE_SIMULATION_EACH_RUN
```

just after

```
#define INCLUDE
```

in the same file. This code should open the Aspen Plus simulation drawn in Figure 3.27, using our Wegstein implementation to tear the feedback loop to simulate the flowsheet shown in Figure 3.28.

```
const string PATH_FOR_DOSOMETHING004 =
    @"C:\[...insert your file path and name here...]";
```

```

public static void DoSomething004()
{
    try
    {
        double tolerance = 1e-9;
        double m_min = 0.0;
        double m_max = 25.0;

#if DEBUG
        Utilities.Report("Creating a new instance of Aspen Plus.");
#endif

        var aspenPlusInstance = new HappLS();
#if DEBUG
        Utilities.Report("Opening the basic simulation.");
#endif
        aspenPlusInstance.InitFromFile2(PATH_FOR_DOSOMETHING004);
        aspenPlusInstance.SuppressDialogs = 1;

#if DEBUG
        Utilities.Report("Making Aspen Plus visible.");
#endif
        aspenPlusInstance.Visible = true;

#if DEBUG
        Utilities.Report("Starting the convergence method.");
#endif

        Box<int> evaluationCounter = Box<int>.New(0);

        Func<double[], double[]> streamTearF = (double[] moleFlows) =>
        {
#if REINITIALIZE_SIMULATION_EACH_RUN
            aspenPlusInstance.Reinit();
#endif

            var moleFlowSetNode = aspenPlusInstance.Tree.FindNode(
                @"Data\Streams\ABS-FC-2\Input\FLOW\MIXED"

```

```

    );

    int index = 0;
    foreach (var nodeObject in moleFlowSetNode.Elements)
    {
        var node = nodeObject as IHNode;
        if (node != null) { node.Value = moleFlows[index]; }
        ++index;
    }
#if DEBUG
    Utilities.Report(
        "Simulation Run #"
        + evaluationCounter.Value
        + " starting.  "
        + DateTime.Now
        + ")";
#endif
    aspenPlusInstance.Run2();
#if DEBUG
    Utilities.Report(
        "Simulation Run #"
        + evaluationCounter.Value
        + " complete.  ("
        + DateTime.Now
        + ")";
#endif
    ++evaluationCounter.Value;

    var moleFlowOutputNode = aspenPlusInstance.Tree.FindNode(
        @"Data\Streams\ABS-FC-1\Output\MOLEFLOW\MIXED"
    );
    List<double> moleFlowList = new List<double>();
    foreach (var nodeObject in moleFlowOutputNode.Elements)
    {
        var node = nodeObject as IHNode;
        if (node != null)

```

```

        {
            moleFlowList.Add(node.Value);
        }
    }
    double[] results = moleFlowList.ToArray();

#if DEBUG
    Utilities.Report(results);
#endif

    return results;
};

double[] initialFlows = new double[21];
initialFlows[2] = 1.8;

var tearConvergenceResult = OurMath.Wegstein(
    streamTearF
    , tolerance.Repeat(21)
    , 25
    , m_min.Repeat(21)
    , m_max.Repeat(21)
    , initialFlows
    , -10
    , 10
);

#if DEBUG
    Utilities.Report(tearConvergenceResult);
#endif
}
catch (Exception ex)
{
    Utilities.Report("Exception ended simulation convergence.
Message:");
    Utilities.Report("\"" + ex.Message + "\".");
}

```

```
}
```

Scheme 3.5. Source code for `DoSomething004 ()` . This method will converge our PZ+MDEA absorber by tearing the feedback stream from the condenser to the packing.

This code also uses `Box<T>`. Add a new source code file, `Box.cs`, under the Utilities folder. Give it the source shown in Scheme 3.6.

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace Heat_Pump
{
    public class Box<T>
    {
        public T Value { get; set; }
        protected Box() { }
        public static Box<T> New(T initialValue)
        {
            var toReturn = new Box<T>();

            toReturn.Value = initialValue;

            return toReturn;
        }
    }
}
```

Scheme 3.6. Code for `Box<T>`.

Finally configure `DoSomething ()` to perform only `DoSomething004 ()` and run our program. Aspen Plus will open, and our program will instruct it to converge the absorption packing with its condenser, providing the feedback listed in Scheme 3.7.

```
Starting DoSomething.Run () .
```

```
Creating a new instance of Aspen Plus.
Opening the basic simulation.
Making Aspen Plus visible.
Starting the convergence method.
Simulation Run #0 starting. (2/9/2016 1:53:17 AM)
Simulation Run #0 complete. (2/9/2016 1:55:30 AM)
{ 4.51611775E-06 1.21757893E-06 0.221144699 9.53578499E-08 0
  1.61270761E-11 3.13974443E-08 1.27917989E-05 2.39469735E-07 0 0
  3.07097842E-06 1.02787368E-05 0 2.9128745E-07 4.68810679E-08
  3.5724811E-10 1.74952703E-06 4.7587548E-07 0 0 }
Simulation Run #1 starting. (2/9/2016 1:55:30 AM)
Simulation Run #1 complete. (2/9/2016 1:55:52 AM)
{ 5.50733453E-06 1.27071152E-06 0.257739288 1.33377599E-07 0
  2.13331851E-11 3.23457814E-08 1.58155544E-05 2.62518646E-07 0 0
  4.25062633E-06 1.21750258E-05 0 3.64855933E-07 5.18942205E-08
  4.20850691E-10 2.03893956E-06 5.54599156E-07 0 0 }
Simulation Run #2 starting. (2/9/2016 1:55:52 AM)
Simulation Run #2 complete. (2/9/2016 1:56:04 AM)
{ 5.48304158E-06 1.26932259E-06 0.256946212 1.32493609E-07 0
  2.12177499E-11 3.2319523E-08 1.5746424E-05 2.61946711E-07 0 0
  4.22196854E-06 1.21332561E-05 0 3.63158563E-07 5.17704269E-08
  4.19271306E-10 2.03266702E-06 5.52893033E-07 0 0 }
Simulation Run #3 starting. (2/9/2016 1:56:04 AM)
Simulation Run #3 complete. (2/9/2016 1:56:08 AM)
{ 5.4830842E-06 1.26939672E-06 0.256942605 1.32482184E-07 0
  2.12160797E-11 3.23211517E-08 1.57460805E-05 2.61957806E-07 0 0
  4.22172815E-06 1.21331796E-05 0 3.63153535E-07 5.17730488E-08
  4.1928915E-10 2.03263862E-06 5.52885293E-07 0 0 }
Simulation Run #4 starting. (2/9/2016 1:56:08 AM)
Simulation Run #4 complete. (2/9/2016 1:56:11 AM)
{ 5.48311775E-06 1.26940482E-06 0.256943543 1.32482062E-07 0
  2.12160812E-11 3.23213836E-08 1.57461216E-05 2.61959398E-07 0 0
  4.22173886E-06 1.21332136E-05 0 3.63154193E-07 5.17733252E-08
  4.19290927E-10 2.03264605E-06 5.52887313E-07 0 0 }
Simulation Run #5 starting. (2/9/2016 1:56:11 AM)
Simulation Run #5 complete. (2/9/2016 1:56:15 AM)
{ 5.48310843E-06 1.26940268E-06 0.25694327 1.32482069E-07 0
  2.12160769E-11 3.23213219E-08 1.5746109E-05 2.61958971E-07 0 0
```

```

4.22173532E-06 1.21332036E-05 0 3.63153984E-07 5.17732516E-08
4.19290448E-10 2.03264389E-06 5.52886726E-07 0 0 }
Simulation Run #6 starting. (2/9/2016 1:56:15 AM)
Simulation Run #6 complete. (2/9/2016 1:56:18 AM)
{ 5.48311115E-06 1.26940332E-06 0.25694335 1.32482066E-07 0 2.1216078E-
11 3.23213401E-08 1.57461126E-05 2.61959097E-07 0 0 4.22173632E-06
1.21332065E-05 0 3.63154044E-07 5.17732734E-08 4.19290589E-10
2.03264452E-06 5.52886897E-07 0 0 }
Simulation Run #7 starting. (2/9/2016 1:56:18 AM)
Simulation Run #7 complete. (2/9/2016 1:56:22 AM)
{ 5.48311034E-06 1.26940313E-06 0.256943326 1.32482066E-07 0
2.12160777E-11 3.23213347E-08 1.57461115E-05 2.61959059E-07 0 0
4.22173602E-06 1.21332056E-05 0 3.63154026E-07 5.17732669E-08
4.19290546E-10 2.03264433E-06 5.52886846E-07 0 0 }
Simulation Run #8 starting. (2/9/2016 1:56:22 AM)
Simulation Run #8 complete. (2/9/2016 1:56:25 AM)
{ 5.48311058E-06 1.26940318E-06 0.256943333 1.32482066E-07 0
2.12160777E-11 3.23213364E-08 1.57461119E-05 2.6195907E-07 0 0
4.2217361E-06 1.21332059E-05 0 3.63154031E-07 5.17732688E-08
4.19290559E-10 2.03264439E-06 5.52886861E-07 0 0 }
Simulation Run #9 starting. (2/9/2016 1:56:25 AM)
Simulation Run #9 complete. (2/9/2016 1:56:29 AM)
{ 5.48311051E-06 1.26940317E-06 0.256943331 1.32482066E-07 0
2.12160777E-11 3.23213359E-08 1.57461118E-05 2.61959067E-07 0 0
4.22173608E-06 1.21332058E-05 0 3.6315403E-07 5.17732682E-08
4.19290555E-10 2.03264437E-06 5.52886857E-07 0 0 }
Simulation Run #10 starting. (2/9/2016 1:56:29 AM)
Simulation Run #10 complete. (2/9/2016 1:56:32 AM)
{ 5.48311046E-06 1.26940317E-06 0.256943329 1.32482064E-07 0
2.12160774E-11 3.23213359E-08 1.57461116E-05 2.61959066E-07 0 0
4.22173601E-06 1.21332057E-05 0 3.63154026E-07 5.17732681E-08
4.19290552E-10 2.03264436E-06 5.52886853E-07 0 0 }
Simulation Run #11 starting. (2/9/2016 1:56:32 AM)
Simulation Run #11 complete. (2/9/2016 1:56:36 AM)
{ 5.48311052E-06 1.26940317E-06 0.256943332 1.32482066E-07 0
2.12160777E-11 3.2321336E-08 1.57461118E-05 2.61959068E-07 0 0
4.22173608E-06 1.21332058E-05 0 3.63154029E-07 5.17732684E-08
4.19290556E-10 2.03264437E-06 5.52886857E-07 0 0 }

```

```

Simulation Run #12 starting. (2/9/2016 1:56:36 AM)
Simulation Run #12 complete. (2/9/2016 1:56:39 AM)
{ 5.48311052E-06 1.26940317E-06 0.256943331 1.32482066E-07 0
  2.12160777E-11 3.2321336E-08 1.57461118E-05 2.61959068E-07 0 0
  4.22173608E-06 1.21332058E-05 0 3.6315403E-07 5.17732683E-08
  4.19290556E-10 2.03264437E-06 5.52886857E-07 0 0 }
{ 5.48311052E-06 1.26940317E-06 0.256943331 1.32482066E-07 0
  2.12160777E-11 3.2321336E-08 1.57461118E-05 2.61959068E-07 0 0
  4.22173608E-06 1.21332058E-05 0 3.6315403E-07 5.17732683E-08
  4.19290556E-10 2.03264437E-06 5.52886857E-07 0 0 }

```

Scheme 3.7. Feedback from **DoSomething004 ()**.

3.4.1.3 Analyzing convergence behavior

From the data in Scheme 3.7, we observe the convergence behavior shown in Figure 3.30 with the results shown in Table 3.3. These results show that we are able to obtain exact matches between the mole flow rates in **ABS-FC-1** and **ABS-FC-2** for most components, with a maximum disparity for water of $10^{-9} \frac{\text{kmol H}_2\text{O}}{\text{hr}}$.

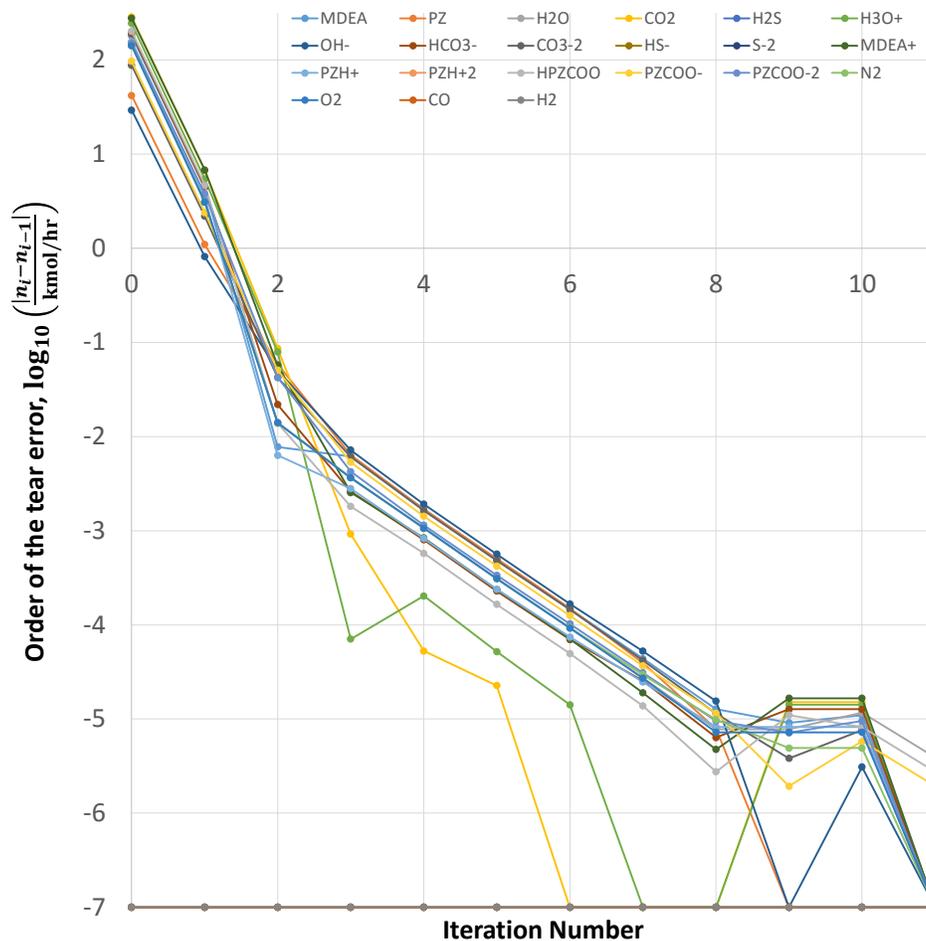


Figure 3.30. Convergence behavior of the 21 mole flow rates involved in the PZ absorber convergence, plotted on a log-10 scale. Some of these errors were exactly zero; for those values, we assigned an order-of-error of -7 .

Table 3.3. Tear stream results from DoSomething004 (). ABS-FC-2 and ABS-FC-1 represent the same stream, so they should have the same values.

	Component mole flow rate $\left(\frac{\text{kmol}}{\text{hr}}\right)$		
	ABS-FC-2	ABS-FC-1	difference
MDEA	5.48E-06	5.48E-06	<i>exact</i>
PZ	1.27E-06	1.27E-06	<i>exact</i>
H₂O	2.57E-01	2.57E-01	-1E-09
CO₂	1.32E-07	1.32E-07	<i>exact</i>
H₂S	0.00E+00	0.00E+00	<i>exact</i>
H₃O⁺	2.12E-11	2.12E-11	<i>exact</i>

OH⁻	3.23E-08	3.23E-08	<i>exact</i>
HCO₃⁻	1.57E-05	1.57E-05	<i>exact</i>
CO₃²⁻	2.62E-07	2.62E-07	<i>exact</i>
HS⁻	0.00E+00	0.00E+00	<i>exact</i>
S²⁻	0.00E+00	0.00E+00	<i>exact</i>
MDEA⁺	4.22E-06	4.22E-06	<i>exact</i>
PZH⁺	1.21E-05	1.21E-05	<i>exact</i>
PZH²⁺	0.00E+00	0.00E+00	<i>exact</i>
HPZCOO	3.63E-07	3.63E-07	1E-15
PZCOO⁻	5.18E-08	5.18E-08	-1E-16
PZCOO²⁻	4.19E-10	4.19E-10	<i>exact</i>
N₂	2.03E-06	2.03E-06	<i>exact</i>
O₂	5.53E-07	5.53E-07	<i>exact</i>
CO	0.00E+00	0.00E+00	<i>exact</i>
H₂	0.00E+00	0.00E+00	<i>exact</i>

We can also reinitialize the simulation before each run, finding the convergence behavior shown in Figure 3.31.

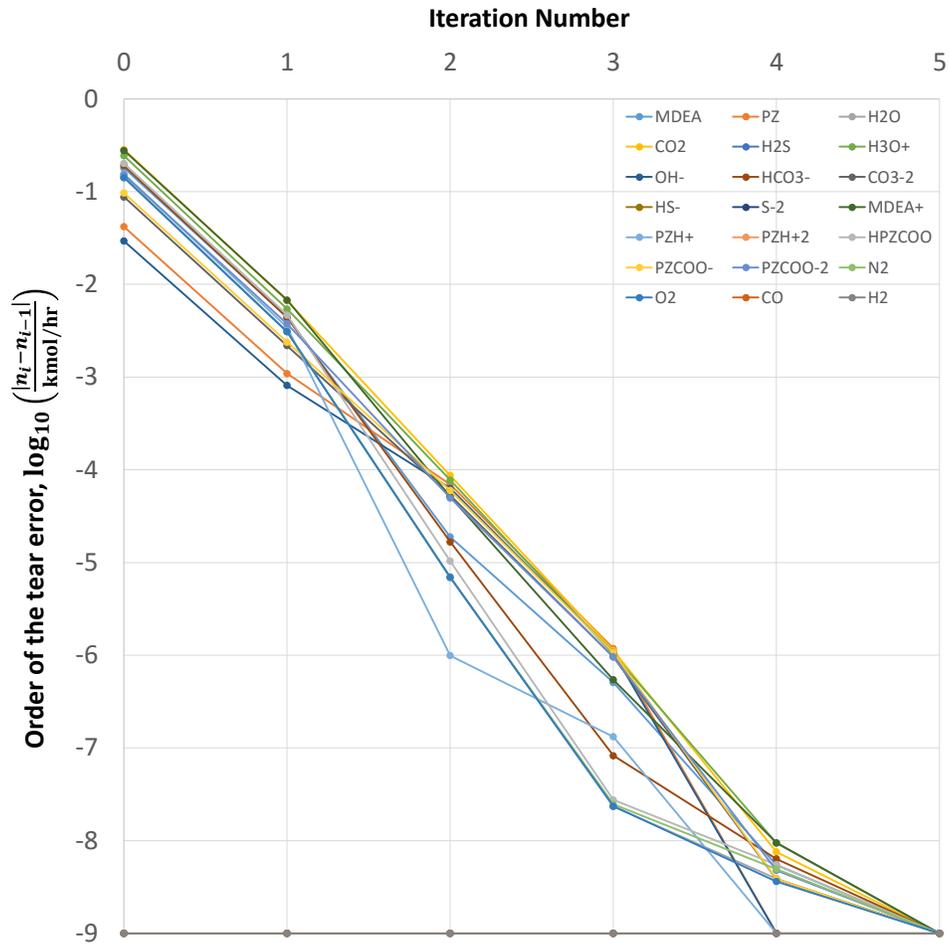


Figure 3.31. Convergence behavior from `DoSomething004 ()` once we enable reinitialization before each run. This simulation required only six evaluations rather than twelve. When there was no error, a value of -9 was assigned. All 21 component mole flow rates were exact on the final iteration.

We can also extract the time stamps from Scheme 3.7 to analyze the simulation run times as shown in Figure 3.32. While using reinitialization halves the number of iterations, it almost doubles the total simulation runtime.

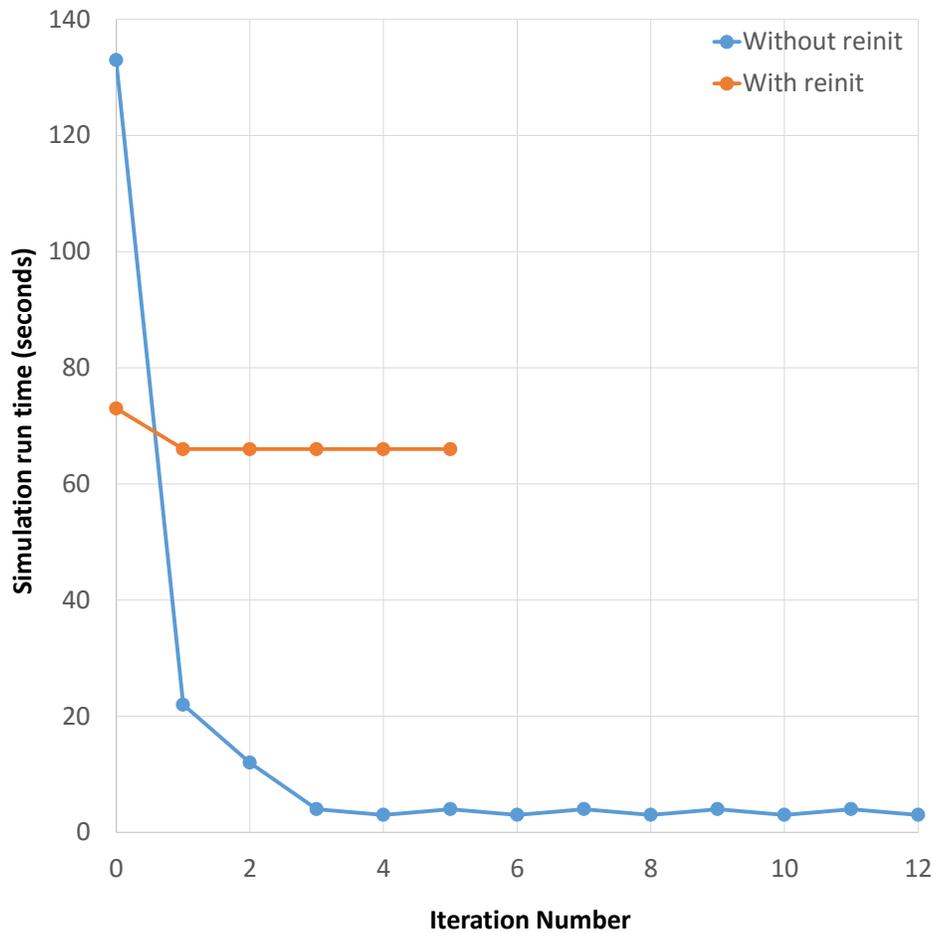


Figure 3.32. Simulation run time for our PZ absorber packing and condenser. The run without reinitializations took 12 iterations, but most of the iterations required less than 4 seconds. The run with reinitializations took half as many iterations, but due to the greatly increased average-time-per-iteration, it took almost twice as long in real time (403 seconds vs. 202 seconds).

Tip

It took an average of 5 minutes for Aspen Plus to do its work while our C# program's own logic took 0 seconds (timer did not include milliseconds). In general, we try to use more robust convergence methods to further reduce the amount of time Aspen Plus has to run.

3.4.1.4 Comparison to Aspen Plus

In our prior simulations, we had Aspen Plus perform the Wegstein method to converge the tear stream feedback. Let us compare our results above to those we would have achieved using only Aspen Plus.

First, save a new copy of our most recent simulation with the PZ absorber, then modify the flowsheet such that it actually looks like Figure 3.28.

Next, we instruct Aspen Plus to use the Wegstein method to converge the feedback from the tear the stream by creating **CV-TEMP** and **SQ-TEMP** as shown in Figure 3.33.

Stream	Tolerance	Trace	State variables	Component group
ABS-FC-1	0.001		None	

Loop-return	Block type	Block
Begin	Convergence	CV-TEMP
	Unit operation	ABSORBER
	Unit operation	ABS-CON
Return to	Convergence	CV-TEMP

Figure 3.33. Definitions for **CV-TEMP** and **SQ-TEMP**. These two blocks work together to specify the Wegstein convergence method and order for **ABS-FC-1**, allowing Aspen Plus to perform the convergence that we had just done in C#.

Run this simulation. We find the **CV-TEMP** results shown in Figure 3.34. These results show some small differences from our C# approach (without reinitialization):

1. Precision differences:
 - a. Most of our C# results are exact (Table 3.3).
 - b. Most of our Aspen Plus results contain some error, except for the components that have zero error due to not being present in the simulation (zero flow rate) (Figure 3.34).

2. Iteration number (Figure 3.32):
 - a. Our C# simulation took 12 evaluations.
 - b. Our Aspen Plus simulation took 10 evaluations.
3. Convergence condition:
 - a. Our C# simulation considered a component converged once its error was less than $10^{-9} \frac{\text{kmol}}{\text{hr}}$, terminating after the most out-of-sync component, H₂O, got down to this value. This value was specified as `tolerance` in `DoSomething004 ()` (Scheme 3.5).
 - b. Our Aspen Plus simulation looks for relative error in non-zero flows such that $\left| \frac{n_i - n_{i-1}}{n_i} \right| \leq 10^{-3}$, as specified in `cv-TEMP` (Figure 3.33).

CV-TEMP (WEGSTEIN) - Results +

Summary | Tear History | Tear Variables | Status

Iteration count
 Total number of iterations: 10
 Number of iterations on last outer loop:

Final values

Status	Variable Number	Variable	Units	Stream	Substream	Component	Attribute	Element	Final value	Previous value	Error / Tolerance
Converged	1	TOTAL MOLEFLOW	KMOL/HR	ABS-FC-1	MIXED				0.256988	0.256869	0.463722
Converged	2	MOLE-FLOW	KMOL/HR	ABS-FC-1	MIXED	MDEA			5.48318e-06	5.48002e-06	0.577648
Converged	3	MOLE-FLOW	KMOL/HR	ABS-FC-1	MIXED	PZ			1.26941e-06	1.26923e-06	0.136001
Converged	4	MOLE-FLOW	KMOL/HR	ABS-FC-1	MIXED	H2O			0.256946	0.256827	0.463703
Converged	5	MOLE-FLOW	KMOL/HR	ABS-FC-1	MIXED	CO2			1.32485e-07	1.32364e-07	0.913973
Converged	6	MOLE-FLOW	KMOL/HR	ABS-FC-1	MIXED	H2S			0	0	0
Converged	7	MOLE-FLOW	KMOL/HR	ABS-FC-1	MIXED	H3O+			2.12165e-11	2.11998e-11	0.789414
Converged	8	MOLE-FLOW	KMOL/HR	ABS-FC-1	MIXED	OH-			3.23214e-08	3.23182e-08	0.0962915
Converged	9	MOLE-FLOW	KMOL/HR	ABS-FC-1	MIXED	HCO3-			1.57463e-05	1.57366e-05	0.617241
Converged	10	MOLE-FLOW	KMOL/HR	ABS-FC-1	MIXED	CO3-2			2.61961e-07	2.61887e-07	0.282904
Converged	11	MOLE-FLOW	KMOL/HR	ABS-FC-1	MIXED	HS-			0	0	0
Converged	12	MOLE-FLOW	KMOL/HR	ABS-FC-1	MIXED	S-2			0	0	0
Converged	13	MOLE-FLOW	KMOL/HR	ABS-FC-1	MIXED	MDEA+			4.22183e-06	4.21809e-06	0.886501
Converged	14	MOLE-FLOW	KMOL/HR	ABS-FC-1	MIXED	PZH+			1.21333e-05	1.21272e-05	0.50644
Converged	15	MOLE-FLOW	KMOL/HR	ABS-FC-1	MIXED	PZH+2			0	0	0
Converged	16	MOLE-FLOW	KMOL/HR	ABS-FC-1	MIXED	HPZCOO			3.6316e-07	3.62924e-07	0.649261
Converged	17	MOLE-FLOW	KMOL/HR	ABS-FC-1	MIXED	PZCOO-			5.17736e-08	5.17576e-08	0.310561
Converged	18	MOLE-FLOW	KMOL/HR	ABS-FC-1	MIXED	PZCOO-2			4.19296e-10	4.19094e-10	0.482348
Converged	19	MOLE-FLOW	KMOL/HR	ABS-FC-1	MIXED	N2			2.03266e-06	2.03172e-06	0.463572
Converged	20	MOLE-FLOW	KMOL/HR	ABS-FC-1	MIXED	O2			5.52892e-07	5.52636e-07	0.46359
Converged	21	MOLE-FLOW	KMOL/HR	ABS-FC-1	MIXED	CO			0	0	0
Converged	22	MOLE-FLOW	KMOL/HR	ABS-FC-1	MIXED	H2			0	0	0

CV-TEMP (WEGSTEIN) - Results +

Summary | Tear History | Tear Variables | Status

Iteration	Maximum error / Tolerance	Variable Number	Variable	Stream ID / *Tear_var	Substream	Component	Attribute	Element
1	-877.142	4	MOLE-FLOW	ABS-FC-1	MIXED	H2O		
2	384.128	13	MOLE-FLOW	ABS-FC-1	MIXED	MDEA+		
3	157.161	13	MOLE-FLOW	ABS-FC-1	MIXED	MDEA+		
4	70.989	13	MOLE-FLOW	ABS-FC-1	MIXED	MDEA+		
5	33.404	13	MOLE-FLOW	ABS-FC-1	MIXED	MDEA+		
6	16.0023	13	MOLE-FLOW	ABS-FC-1	MIXED	MDEA+		
7	7.73056	13	MOLE-FLOW	ABS-FC-1	MIXED	MDEA+		
8	3.75227	13	MOLE-FLOW	ABS-FC-1	MIXED	MDEA+		
9	1.82238	13	MOLE-FLOW	ABS-FC-1	MIXED	MDEA+		
10	0.913973	5	MOLE-FLOW	ABS-FC-1	MIXED	CO2		

Figure 3.34. Summary of results and the tear history for **CV-TEMP**. The “Summary” tab focuses on the final iteration in which the tear variables were all within tolerance. The “Results” tab gives a brief overview of the ten Wegstein iterations, listing the variable with the highest error-over-tolerance for each iteration. Note that the mole flow rate of CO2 during the final iteration appears on both tabs.

3.4.1.5 Adding design specifications to the absorber

As with our Aspen Plus simulation, we will seek a desired carbon capture rate in part by finding a lean solvent flow rate such that the effective capture rate in the absorber, \mathcal{R}_{ABS} as defined in Equation 3.1, is equal to the target carbon capture rate.

$$\mathcal{R}_{\text{ABS}} \equiv 1 - \frac{F_{\text{lost CO}_2}}{F_{\text{flue CO}_2}} \quad 3.1$$

Additionally we would like to keep columns' packing diameters such that each column has a specified maximum approach-to-flooding, e.g. 60%. We note that the maximum approach-to-flooding tends to be inversely proportional to a packed section's cross sectional area, $\mathcal{F}_{\text{column}} \propto \frac{1}{A_{\text{column}}}$, so it is useful to converge on $\mathcal{F}_{\text{column}} \times A_{\text{column}}$ as this value is roughly constant. In practice we generally converge $\mathcal{F}_{\text{column}} \times A_{\text{column}}$ through direct substitution.

We implement both the design specification for effective capture rate \mathcal{R}_{ABS} and maximum approach-to-flooding \mathcal{F}_{ABS} in `DoSomething005()`, given in Scheme 3.8.

```
#define INCLUDE

#if INCLUDE
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using Happ;

namespace Heat_Pump
{
    partial class DoSomething
    {
        public static void DoSomething005()
        {
#if DEBUG
            Utilities.Report("Start: DoSomething005.");
#endif
            try
            {
                string simulationFile =
System.IO.Directory.GetCurrentDirectory()
```

```

+ @"\\Absorber.apwz";

var parameters = new SimulationParameters();
parameters.AminePortion = 0.054997612;
parameters.CaptureRate = 0.8;
parameters.FloodingApproach = 0.6;
parameters.LeanLoading = 0.055456292;
parameters.PZPortion = 0.230779782;
parameters.StripperPressure = 2.0;
parameters.Absorber_FloodingFactorTimesArea = 1.270717863;
parameters.Stripper_FloodingFactorTimesArea = 3.0;
parameters.RegenerationEnergy = 5.37; // GJ/tonne

double tolerance = 1e-9;
double m_min = 0.0;
double m_max = 10000.0;

#if DEBUG
Utilities.Report("Creating a new instance of Aspen Plus.");
#endif

var aspenPlusInstance = new HappLS();

#if DEBUG
Utilities.Report("Opening the absorber simulation.");
#endif

aspenPlusInstance.InitFromFile2(simulationFile);
aspenPlusInstance.SuppressDialogs = 1;

#if DEBUG
Utilities.Report("Making Aspen Plus visible.");
#endif

aspenPlusInstance.Visible = true;

#if DEBUG
Utilities.Report("Starting the convergence method.");
#endif

Box<int> absorberRunCounterBox = Box<int>.New(0);

```

```

Box<double> flowMultiplierBox = Box<double>.New(83.52321);

var flueFlowHandler = ApparentFlows.New();
flueFlowHandler.H2O = 41.80264735;
flueFlowHandler.CO2 = 22.72220602;
flueFlowHandler.N2 = 324.7995852;
flueFlowHandler.O2 = 46.68385665;

var absorberTearStream = ApparentFlows.New();
absorberTearStream.MDEA = 0.001455809;
absorberTearStream.PZ = 0.002057891;
absorberTearStream.H2O = 38.70582925;
absorberTearStream.CO2 = 0.003076571;

var leanFlowHandler = ApparentFlows.New();

Action runAbsorber = () =>
{
    leanFlowHandler.MDEA = flowMultiplierBox.Value
        * (1.0 - parameters.PZPortion);
    leanFlowHandler.PZ = flowMultiplierBox.Value
        * parameters.PZPortion;
    leanFlowHandler.H2O = flowMultiplierBox.Value
        / parameters.AminePortion;
    leanFlowHandler.CO2 = flowMultiplierBox.Value
        * parameters.LeanLoading;

    flueFlowHandler.Assert(aspernPlusInstance, "FLUE-IN");
    absorberTearStream.Assert(aspernPlusInstance, "ABS-FC-2");
    leanFlowHandler.Assert(aspernPlusInstance, "LEAN-IN");

    DateTime startTime = DateTime.Now;
    aspernPlusInstance.Run2();
    DateTime endTime = DateTime.Now;

    var runTime = endTime - startTime;

#if DEBUG
    Utilities.Report(

```

```

        "Absorber\tRun #\t"
        + absorberRunCounterBox.Value
        + "\tRun time (ms):\t"
        + runTime.TotalMilliseconds
    );
#endif

    ++absorberRunCounterBox.Value;
};

Action absorberWithFloodingFactor = () =>
{
    bool loop = true;

    while (loop)
    {
        var setArea =
parameters.Absorber_FloodingFactorTimesArea
        / parameters.FloodingApproach;
        var setDiameter = 2.0 * Math.Sqrt(setArea / Math.PI);
        aspenPlusInstance.Tree.FindNode(
            @"\Data\Blocks\ABSORBER\Input\PR_DIAM\1"
        ).Value = setDiameter;

        runAbsorber();

        var usedDiameter = aspenPlusInstance.Tree.FindNode(
            @"\Data\Blocks\ABSORBER\Input\PR_DIAM\1").Value;
        var usedArea = Math.PI * Math.Pow(usedDiameter / 2.0,
2.0);

        var calculatedFlooding =
aspenPlusInstance.Tree.FindNode(
            @"\Data\Blocks\ABSORBER\Output\FLOOD_FAC2\1").Value;
        parameters.Absorber_FloodingFactorTimesArea = usedArea *
            calculatedFlooding;

        var floodingError = calculatedFlooding
            - parameters.FloodingApproach;

```

```

        loop = Math.Abs(floodingError) >
parameters.FloodingSoftTolerance;
    }
};

Func<double[], double[]> absorberTearStreamFlows
= (double[] inputFlows) =>
{
    absorberTearStream.SetFromArray(inputFlows);

    absorberWithFloodingFactor();

    absorberTearStream.Read(aspensPlusInstance, @"ABS-FC-1");
    return absorberTearStream.ToArray();
};

Action tearAbsorber = () =>
{
    OurMath.Wegstein_Damped(
        absorberTearStreamFlows
        , (tolerance).Repeat(ApparentFlows.APPARENT_COUNT)
        , 30
        , (m_min).Repeat(ApparentFlows.APPARENT_COUNT)
        , (m_max).Repeat(ApparentFlows.APPARENT_COUNT)
        , absorberTearStream.ToArray()
        , -2.5
        , 2.5
        , 2.0
        , 1.0
    );
};

Func<double, double> absorberEffectiveCaptureRateFunction
= (double flowMultiplier) =>
{
    flowMultiplierBox.Value = flowMultiplier;

    tearAbsorber();
};

```

```

        double mCO2Lost = aspenPlusInstance.Tree.FindNode(
            @"Data\Streams\FLUE-
OUT\Output\MASSFLOW\MIXED\CO2").Value;
        double mCO2Flue = aspenPlusInstance.Tree.FindNode(
            @"Data\Streams\FLUE-
IN\Output\MASSFLOW\MIXED\CO2").Value;

        double rABS = 1.0 - (mCO2Lost / mCO2Flue);

        return rABS;
    };

    OurMath.Secant002(
        parameters.CaptureRate
        , absorberEffectiveCaptureRateFunction
        , 0.0001
        , 30
        , 10.0
        , 500.0
        , 83.52321
        , (double firstResult) =>
            { return parameters.CaptureRate / firstResult *
83.52321; }
        , 2.0
        , 1.0
    );
}
catch (Exception ex)
{
    Utilities.Report("Exception ended simulation convergence.
Message:");
    Utilities.Report("\"" + ex.Message + "\".");
}

#if DEBUG
    Utilities.Report("End : DoSomething005.");
#endif

```

```

    }
  }
}
#endif

```

Scheme 3.8. C# source code for `DoSomething005 ()`. `DoSomething005 ()`

works on the absorber, performing three duties: (1) converging the tear stream; (2) adjusting the lean solvent flow rate to achieve the target capture rate; (3) adjusting the packing diameter to achieve the target maximum approach-to-flooding.

In our secant method for the lean solvent flow rate, we use `(double firstResult) => { return parameters.CaptureRate / firstResult * 83.52321; }` as our second estimate. This guesses that the solvent flow rate is proportional to capture rate for the second iteration of the secant method.

Being able to provide a second estimate is a major advantage that we gain by writing our own convergence methods. In Aspen Plus, the second estimate for the secant method is selected based on the specified step size as given in Equation 3.2 where the default $\Delta x_{\text{step}} = 0.01$. Usually we can do better than this random step, at least in guessing the correct direction to step in as opposed to always stepping to a higher value.

$$x_2 = x_1 + \Delta x_{\text{step}}(x_{\text{max}} - x_{\text{min}}) \quad 3.2$$

Running `DoSomething005 ()`, we find a total run-time of 5.14 minutes split up over 56 iterations. Figure 3.35 shows that the time-per-iteration forms peaks. The longer-running iterations appear to correspond to the first Aspen Plus run after the secant method guesses a new lean solvent flow rate.

Other results:

1. The tear stream is converged.
2. The target capture rate of 80% is achieved within numerical tolerance at 79.999854%.
3. The target maximum approach-to-flooding of 60% is achieved exactly.

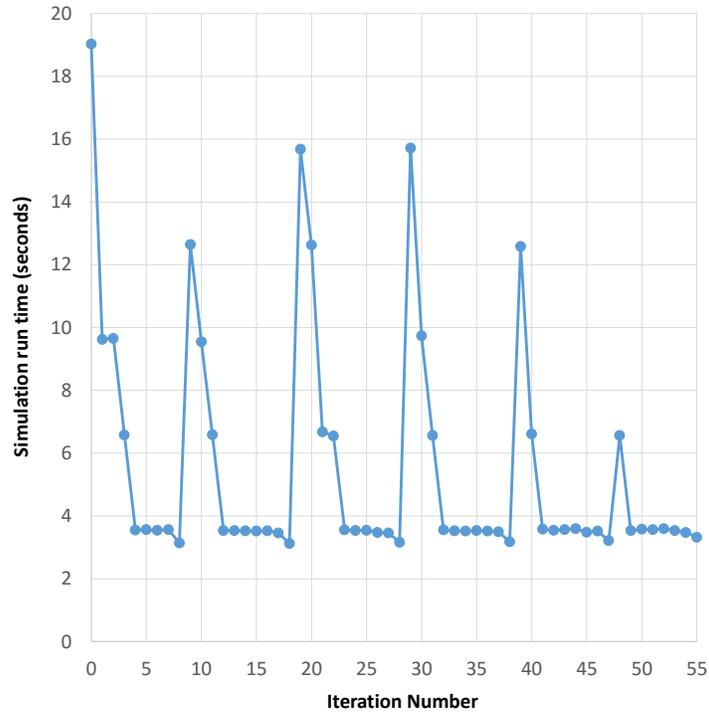


Figure 3.35. Run-time-per-iteration for **DoSomething005 ()**. In total, this simulation took ~308.7 seconds and 56 iterations.

3.4.2 Build the stripper

We have previously described the absorber as the derived object **SQ-ABS-2** and the stripper as a derived object **SQ-STR-2**. Building the stripper will be similar to building the absorber, though we must take into account the reboiler; we need to tear the feedback from the reboiler just as we need to tear the feedback from the condenser.

3.4.2.1 Create the Aspen Plus simulation for the stripper

Open the prior absorber simulation and save it as a new file. Next, delete/modify the flowsheet contents to draw Figure 3.36.

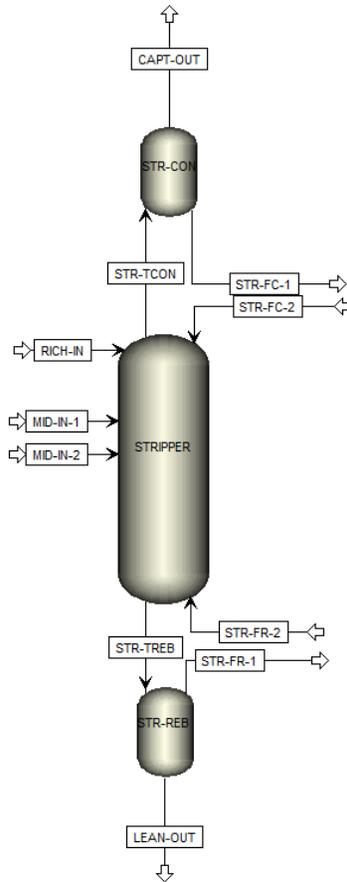


Figure 3.36. Flowsheet for the stripper simulation.

For each input material stream (**RICH-IN**, **MID-IN-1**, **MID-IN-2**, **STR-FC-2**, and **STR-FR-2**), assign:

- temperature $T = 40^{\circ}\text{C}$;
- pressure $P = 1\text{bar}$;
- leave total flow rate blank;
- $1 \frac{\text{kmol}}{\text{hr}}$ of H_2O .

Then, change the specification from $T = 40^{\circ}\text{C}$ to:

- vapor fraction $v = 0$ for **STR-FC-2**;
- vapor fraction $v = 1$ for **STR-FR-2**.

For **STR-CON**, use $T = 40^{\circ}\text{C}$ and $P = 1\text{bar}$. For **STR-REB**, use $P = 1\text{bar}$ and $D = 1\text{MW}$.

Specify **STRIPPER** as shown in Figure 3.37.

STRIPPER (RadFrac) × +

Configuration Streams Pressure Condenser Reboiler 3-Phase Information

Setup options

Calculation type: Rate-Based

Number of stages: 10 [Stage Wizard]

Condenser: None

Reboiler: None

Valid phases: Vapor-Liquid

Convergence: Standard

Operating specifications

Free water reflux ratio: 0 [Feed Basis]

STRIPPER (RadFrac) × +

Configuration Streams Pressure Condenser Reboiler

Feed streams

Name	Stage	Convention
RICH-IN	1	Above-Stage
MID-IN-1	2	Above-Stage
MID-IN-2	3	Above-Stage
STR-FR-2	10	On-Stage
STR-FC-2	1	Above-Stage

Product streams

Name	Stage	Phase	Basis
STR-TREB	10	Liquid	Mole
STR-TCON	1	Vapor	Mole

STRIPPER (RadFrac) × +

Configuration Streams Pressure Condenser Reboiler

View: Top / Bottom

Top stage / Condenser pressure: 1 bar

Stage 1 / Condenser pressure: 1 bar

Stage 2 pressure (optional)

Stage 2 pressure

Condenser pressure drop

Pressure drop for rest of column (optional)

Stage pressure drop

Column pressure drop: 0 bar

STRIPPER Specifications - Reactions × +

Specifications Holdups Residence Times Conversion

Reaction names

Starting stage	Ending stage	Reaction ID	Reaction user	Chemistry II
1	10	P-M-RXN		

STRIPPER Specifications - Reactions × +

Specifications Holdups Residence Times Conversion

Specify holdups for rate-controlled reactions

Starting stage	Ending stage	Liquid holdup	Vapor holdup
1	10	1e-05	

STRIPPER Sizing and Rating Packing Rating 1 - Rate-based × +

Rate Based Correlations Holdups Design Optional Bravo-Rocha-Fair

Mass transfer coefficient method

Correlation: HanleyStruc (2010)

Heat transfer coefficient method

Correlation: Chilton and Colburn

Interfacial area method

Correlation: HanleyStruc (2010)

Mass transfer correlation parameter

Critical surface tension: N/m

Billet and Schultes CL: Billet and Schultes CV:

STRIPPER Sizing and Rating Packing Rating 1 - Setup × +

Specifications Design / Pdrop Stichlmair

Packing section

Starting stage: 1 Ending stage: 10 Type: FLEXIPAC

Packing characteristics

Vendor: KOCH Section diameter: 1 meter [Update parameters]

Material: METAL Packing size: meter

Dimension: 250Y Packing factor: 92.19 1/m

Packed height

Packed height per stage

Section packed height: 5 meter

Figure 3.37. Specifications for **STRIPPER**.

3.4.2.2 Converging the stripper

The stripper requires convergence for:

1. two tear stream feedbacks;
2. packing diameter to meet the target maximum approach-to-flooding;
3. reboiler duty to meet the target effective capture rate.

The tear streams and maximum approach-to-flooding both work much like they did in the absorber. The reboiler duty design spec is much like the lean solvent flow rate design spec, though with the effective capture rate for the stripper \mathcal{R}_{STR} in Equation 3.3.

$$\mathcal{R}_{STR} \equiv \frac{F_{\text{captured CO}_2}}{F_{\text{flue CO}_2}} \quad 3.3$$

We implement this logic in `DoSomething006 ()` using the source code given in Scheme 3.9.

```
#define INCLUDE

#if INCLUDE
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
```

```

using System.Threading.Tasks;
using Happ;

namespace Heat_Pump
{
    partial class DoSomething
    {

        public static void DoSomething006()
        {
            string simulationFile = System.IO.Directory.GetCurrentDirectory()
                + @"\Stripper005.apwz";
#if DEBUG
            Utilities.Report("Start: DoSomething006.");
#endif
            try
            {

                var parameters = new SimulationParameters();
                parameters.AminePortion = 0.054997612;
                parameters.CaptureRate = 0.8;
                parameters.FloodingApproach = 0.6;
                parameters.LeanLoading = 0.055456292;
                parameters.PZPortion = 0.230779782;
                parameters.StripperPressure = 2.0;
                parameters.Absorber_FloodingFactorTimesArea = 1.270717863;
                parameters.Stripper_FloodingFactorTimesArea = 3.0;
                parameters.RegenerationEnergy = 5.37; // GJ/tonne

                double tolerance = 1e-9;
                double m_min = 0.0;
                double m_max = 10000.0;

#if DEBUG
                Utilities.Report("Creating a new instance of Aspen Plus.");
#endif

                var aspenPlusInstance = new HappLS();

```

```

#if DEBUG
    Utilities.Report("Opening the stripper simulation.");
#endif

    aspenPlusInstance.InitFromFile2(simulationFile);
    aspenPlusInstance.SuppressDialogs = 1;

#if DEBUG
    Utilities.Report("Making Aspen Plus visible.");
#endif

    aspenPlusInstance.Visible = true;

#if DEBUG
    Utilities.Report("Starting the convergence method.");
#endif

    Box<int> stripperRunCounter = Box<int>.New(0);

    var richInHandler = ApparentFlows.New();
    richInHandler.MDEA = 61.76699888;
    richInHandler.PZ = 18.5306811;
    richInHandler.H2O = 1464.223819;
    richInHandler.CO2 = 22.63649965;

    var midIn1Handler = ApparentFlows.New();
    midIn1Handler.MDEA = 0.0;
    midIn1Handler.PZ = 0.0;
    midIn1Handler.H2O = 1e-9;
    midIn1Handler.CO2 = 0.0;

    var midIn2Handler = ApparentFlows.New();
    midIn2Handler.MDEA = 0.0;
    midIn2Handler.PZ = 0.0;
    midIn2Handler.H2O = 1e-9;
    midIn2Handler.CO2 = 0.0;

    var stripperCondenserTearStream = ApparentFlows.New();
    stripperCondenserTearStream.MDEA = 0.0;
    stripperCondenserTearStream.PZ = 0.0;

```

```

stripperCondenserTearStream.H2O = 0.001;
stripperCondenserTearStream.CO2 = 0.0;

var stripperReboilerTearStream = ApparentFlows.New();
stripperReboilerTearStream.MDEA = 77.24393507;
stripperReboilerTearStream.PZ = 23.18207732;
stripperReboilerTearStream.H2O = 1956.754119;
stripperReboilerTearStream.CO2 = 5.422902021;

Action runStripper = () =>
{
    richInHandler.Assert(aspernPlusInstance, "RICH-IN");
    midIn1Handler.Assert(aspernPlusInstance, "MID-IN-1");
    midIn2Handler.Assert(aspernPlusInstance, "MID-IN-2");

    aspernPlusInstance.Tree.FindNode(
        @"\Data\Streams\RICH-IN\Input\PRES\MIXED").Value
        = parameters.StripperPressure;
    aspernPlusInstance.Tree.FindNode(
        @"\Data\Streams\MID-IN-1\Input\PRES\MIXED").Value
        = parameters.StripperPressure;
    aspernPlusInstance.Tree.FindNode(
        @"\Data\Streams\MID-IN-2\Input\PRES\MIXED").Value
        = parameters.StripperPressure;
    aspernPlusInstance.Tree.FindNode(
        @"\Data\Streams\STR-FC-2\Input\PRES\MIXED").Value
        = parameters.StripperPressure;
    aspernPlusInstance.Tree.FindNode(
        @"\Data\Streams\STR-FR-2\Input\PRES\MIXED").Value
        = parameters.StripperPressure;
    aspernPlusInstance.Tree.FindNode(
        @"\Data\Blocks\STRIPPER\Input\PRES1").Value
        = parameters.StripperPressure;
    aspernPlusInstance.Tree.FindNode(
        @"\Data\Blocks\STR-CON\Input\PRES").Value
        = parameters.StripperPressure;
    aspernPlusInstance.Tree.FindNode(
        @"\Data\Blocks\STR-REB\Input\PRES").Value

```

```

        = parameters.StripperPressure;

    DateTime startTime = DateTime.Now;
    aspenPlusInstance.Run2();
    DateTime endTime = DateTime.Now;

    var runTime = endTime - startTime;
#if DEBUG
    Utilities.Report(
        "Stripper\tRun #\t"
        + stripperRunCounter.Value
        + "\tRun time (ms):\t"
        + runTime.TotalMilliseconds
    );
#endif

    ++stripperRunCounter.Value;
};

Action stripperWithFloodingFactor = () =>
{
    bool loop = true;

    while (loop)
    {
        var setArea =
parameters.Stripper_FloodingFactorTimesArea
        / parameters.FloodingApproach;
        var setDiameter = 2.0 * Math.Sqrt(setArea / Math.PI);
        aspenPlusInstance.Tree.FindNode(
            @"\Data\Blocks\STRIPPER\Input\PR_DIAM\1").Value
            = setDiameter;

        runStripper();

        var usedDiameter = aspenPlusInstance.Tree.FindNode(
            @"\Data\Blocks\STRIPPER\Input\PR_DIAM\1").Value;

```

```

        var usedArea = Math.PI * Math.Pow(usedDiameter / 2.0,
2.0);

        var calculatedFlooding =
aspenPlusInstance.Tree.FindNode(
            @"\"Data\Blocks\STRIPPER\Output\FLOOD_FAC2\1").Value;
        parameters.Stripper_FloodingFactorTimesArea = usedArea
            * calculatedFlooding;

        var floodingError = calculatedFlooding
            - parameters.FloodingApproach;
        loop = Math.Abs(floodingError) >
parameters.FloodingSoftTolerance;
    }
};

Func<double[], double[]>
stripperTearJustReboilerFeedbackFunction
    = (double[] inputFlows) =>
    {
        var recoveredCondenserArray = new
double[ApparentFlows.APPARENT_COUNT];
        var recoveredReboilerArray = new
double[ApparentFlows.APPARENT_COUNT];
        for (int i = 0; i < ApparentFlows.APPARENT_COUNT; ++i)
        {
            recoveredCondenserArray[i] = inputFlows[i];
            recoveredReboilerArray[i] = inputFlows[i
                + ApparentFlows.APPARENT_COUNT];
        }

        var recoveredCondenserHandler =
            ApparentFlows.New(recoveredCondenserArray);
        var recoveredReboilerHandler =
            ApparentFlows.New(recoveredReboilerArray);
        recoveredCondenserHandler.Assert(aspenPlusInstance, @"STR-
FC-2");
        recoveredReboilerHandler.Assert(aspenPlusInstance, @"STR-
FR-2");
    }

```

```

#if DEBUG
    Utilities.Report(recoveredCondenserArray);
    Utilities.Report(recoveredReboilerArray);
#endif

    stripperWithFloodingFactor();

    var condenserOutputHandler =
ApparentFlows.New(aspPlusInstance, @"STR-FC-1");
    var reboilerOutputHandler =
ApparentFlows.New(aspPlusInstance
        , @"STR-FR-1");
    var readCondenserArray = condenserOutputHandler.ToArray();
    var readReboilerArray = reboilerOutputHandler.ToArray();
    var combinedResults = new double[2 *
ApparentFlows.APPARENT_COUNT];
    for (int i = 0; i < ApparentFlows.APPARENT_COUNT; ++i)
    {
        combinedResults[i] = readCondenserArray[i];
        combinedResults[i + ApparentFlows.APPARENT_COUNT] =
            readReboilerArray[i];
    }
    return combinedResults;
};

Action tearStripper = () =>
{
    var stripperCondenserArray =
stripperCondenserTearStream.ToArray();
    var stripperReboilerArray =
stripperReboilerTearStream.ToArray();
    double[] totalArray = new double[2 *
ApparentFlows.APPARENT_COUNT];
    for (int i = 0; i < ApparentFlows.APPARENT_COUNT; ++i)
    {
        totalArray[i] = stripperCondenserArray[i];    //!

```

```

        stripperReboilerArray[i];
    }

    var tearResults = OurMath.Wegstein_Damped(
        stripperTearJustReboilerFeedbackFunction
        , (tolerance).Repeat(2 *
ApparentFlows.APPARENT_COUNT)
        , 30
        , (m_min).Repeat(2 * ApparentFlows.APPARENT_COUNT)
        , (m_max).Repeat(2 * ApparentFlows.APPARENT_COUNT)
        , totalArray
        , -1.0
        , 1.0
        , 2.0
        , 1.0
    );

    var recoveredCondenserArray = new
double[ApparentFlows.APPARENT_COUNT];
    var recoveredReboilerArray = new
double[ApparentFlows.APPARENT_COUNT];
    for (int i = 0; i < ApparentFlows.APPARENT_COUNT; ++i)
    {
        recoveredCondenserArray[i] = tearResults[i];
        recoveredReboilerArray[i] = tearResults[i
            + ApparentFlows.APPARENT_COUNT];
    }

    stripperCondenserTearStream.SetFromArray(recoveredCondenserArray);

    stripperReboilerTearStream.SetFromArray(recoveredReboilerArray);
};

Func<double, double> stripperEffectiveCaptureRateFunction =
    (double reboilerDuty) =>
    {
        aspenPlusInstance.Tree.FindNode(

```

```

        @"Data\Blocks\STR-REB\Input\DUTY").Value =
reboilerDuty;

        tearStripper();

        var mCO2Capt = aspenPlusInstance.Tree.FindNode(
            @"Data\Streams\CAPT-
OUT\Output\MASSFLOW\MIXED\CO2").Value;
        var mCO2Flue = 1.0;

        var rSTR = mCO2Capt / mCO2Flue;

#if DEBUG
        Utilities.Report(
            "Found effective capture rate:\t"
            + rSTR
            + "\tat reboiler duty (MW):\t"
            + reboilerDuty
        );
#endif

        return rSTR;
    };

    OurMath.Secant002(
        parameters.CaptureRate
        , stripperEffectiveCaptureRateFunction
        , 0.0001
        , 30
        , 0.25
        , 5.0
        , parameters.CaptureRate * parameters.RegenerationEnergy
/ 3.6
        , (double captureRate) =>
            { return parameters.CaptureRate / captureRate *
                parameters.CaptureRate *
parameters.RegenerationEnergy
                / 3.6; }
        , 1.25
        , 0.05

```

```

        );
    }
    catch (Exception ex)
    {
        Utilities.Report("Exception ended simulation convergence.
Message:");
        Utilities.Report("\\" + ex.Message + "\".");
    }

#if DEBUG
    Utilities.Report("End : DoSomething006.");
#endif
    }
}
#endif

```

Scheme 3.9. C# source code for **DoSomething006 ()**. This method (1) converges the stripper's feedback tear streams; (2) sets the stripper's packed diameter to reach the target maximum approach-to-flooding; and (3) sets the stripper's reboiler duty to reach the target effective carbon capture rate.

3.4.2.3 Stripper run results

After running **DoSomething006 ()**, we find that:

1. We have successfully converged both tear-stream feedbacks.
2. We have achieved the target capture rate within numerical precision, finding $\mathcal{R}_{STR} \approx 79.99986\%$ at $D_{STR} = 0.982082578$ MW.
3. We have achieved exactly the target maximum approach-to-flooding for the stripper, $\mathcal{F}_{STR} = 60\%$.

The simulation time was about 10.05 minutes with 198 flowsheet evaluations. Figure 3.38 shows the time per iteration.

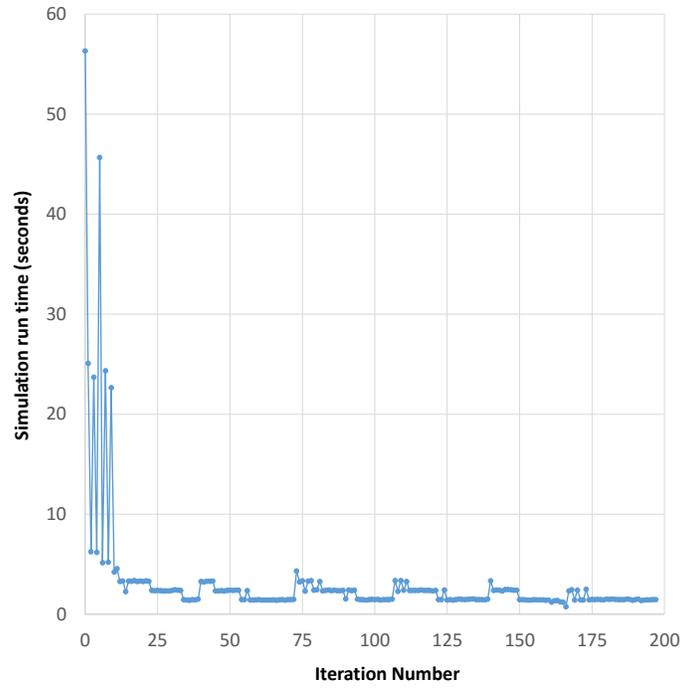


Figure 3.38. Simulation time for each iteration during the stripper's convergence in `DoSomething006()`. The total run time was about 603.21 seconds over 198 iterations.

3.4.3 Build the distributed cross heat exchanger

The baseline acid-gas-capture unit design includes a central cross heat exchanger that could ordinarily be adequately described by a **HeatX** block. In Section 2.4: “Distributed cross heat exchanger”, we discussed the distributed cross heat exchanger figuration that was effectively an upgraded version of the central cross heat exchanger.

We will build this simulation with the distributed cross heat exchanger configuration incorporated from the start.

3.4.3.1 Create the Aspen Plus simulation for the distributed cross heat exchanger

We have previously drawn a flowsheet for the distributed cross heat exchanger in Figure 2.36 as the derived object **SQ-DHEX1**. Now, we will start with a copy of either the absorber or stripper, delete those elements off, and draw the flowsheet as shown in Figure 3.39.

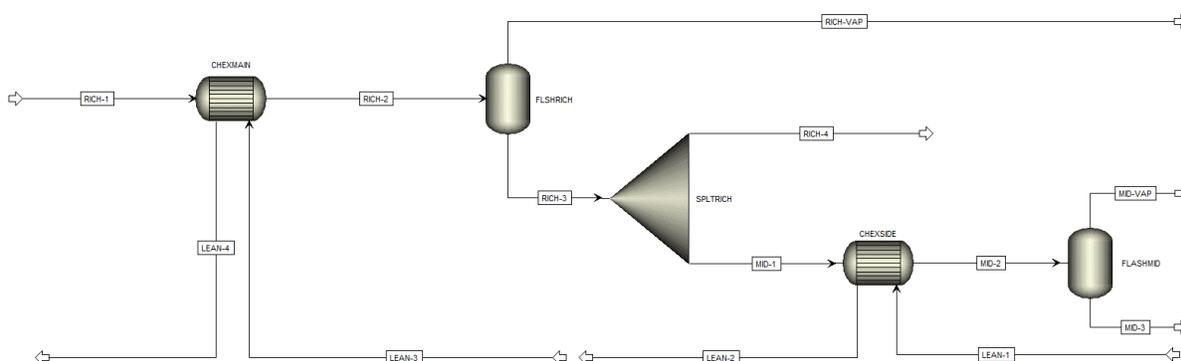


Figure 3.39. Flowsheet for the distributed cross heat exchanger section of the PZ+MDEA simulation.

The feedback stream **LEAN-6** in Figure 2.36 is now broken up into two streams, **LEAN-2** and **LEAN-3**, which we again converge in **C#** rather than by using a **Convergence** block.

For each input material stream (**RICH-1**, **LEAN-1**, **LEAN-3**), assign:

- temperature $T = 40^{\circ}\text{C}$;
- pressure $P = 1\text{bar}$;

- leave total flow rate blank;
- $1 \frac{\text{kmol}}{\text{hr}}$ of H₂O.

For each **Flash2** block (**FLSHRICH** and **FLASHMID**), assign:

- pressure drop $\Delta P = 0$;
- duty $D = 0$.

For the **Fsplit** block **SPLTRICH**, assign a split of zero to **MID-1**.

Specify the two **HeatX** blocks **CHEXMAIN** and **CHEXSIDE** as shown in Figure 3.40.

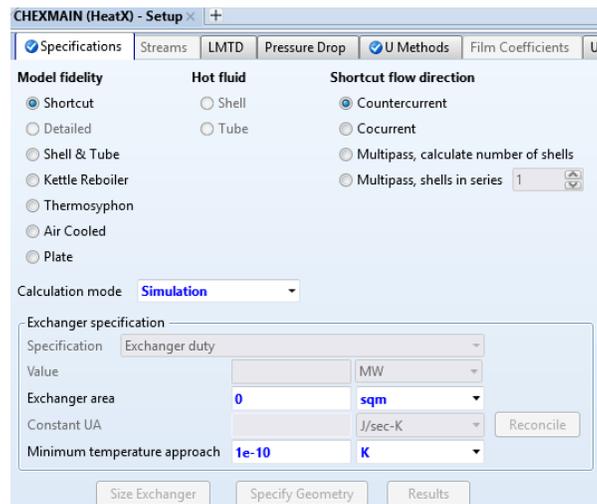


Figure 3.40. Specification for **CHEXMAIN**. Use the same for **CHEXSIDE**.

3.4.3.2 Converging the distributed cross heat exchanger

We need to converge the stream torn into **LEAN-2** and **LEAN-3**. We know the apparent chemical composition of this stream because it will always be exactly as given in **LEAN-1**; any deviation from **LEAN-1**'s apparent composition is a numerical error that may actually be beneficial to ignore.

Additionally, since we assume no pressure drop, the convergence calculation does not involve tearing a pressure variable. This leaves us with tearing just temperature.

We reason that, physically, the temperature of the torn stream must be between the inlet cold stream's temperature and the inlet hot stream's temperature. The cold inlet stream is **RICH-1**, and the hot inlet stream is **LEAN-1**. So we tear t_{DHEX} as described in Equation 3.4.a and defined in Equation 3.4.b.

$$T_{\text{torn stream}} = T_{\text{cold}} + t_{\text{DHEX}}(T_{\text{hot}} - T_{\text{cold}}) \quad 3.4.a$$

$$t_{\text{DHEX}} \equiv \frac{T_{\text{torn stream}} - T_{\text{cold}}}{T_{\text{hot}} - T_{\text{cold}}} \quad 3.4.b$$

The Wegstein method is a good choice for this approach. To minimize the code presented in this dissertation, we do not provide a version of it that works on a single variable; rather, we will just use the multi-variable version that takes a `double[]` and just use a single-element array. Interested readers are encouraged to modify the Wegstein code for single-variable convergence as would be appropriate for cases like this.

Use the source code in Scheme 3.10 for `DoSomething007()`.

```
#define INCLUDE

#if INCLUDE
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using Happ;

namespace Heat_Pump
{
    partial class DoSomething
    {
        public static void DoSomething007()
        {
#if DEBUG
```

```

        Utilities.Report("Start: DoSomething007.");
#endif
    try
    {
        string simulationFile =
System.IO.Directory.GetCurrentDirectory()
        + @"\DHEX.apwz";

        var parameters = new SimulationParameters();
        parameters.AminePortion = 0.054997612;
        parameters.CaptureRate = 0.8;
        parameters.FloodingApproach = 0.6;
        parameters.LeanLoading = 0.055456292;
        parameters.PZPortion = 0.230779782;
        parameters.StripperPressure = 2.0;
        parameters.Absorber_FloodingFactorTimesArea = 1.270717863;
        parameters.Stripper_FloodingFactorTimesArea = 3.0;
        parameters.RegenerationEnergy = 5.37; // GJ/tonne
        parameters.CHEX_TotalArea = 100.0;
        parameters.DHEX_DistributedPortion = 0.5;
        parameters.Threshold_Area = 1e-7;
        parameters.DHEX_SplitPortion = 0.5;

        double toleranceForHeatXTemp = 1e-5;
        double m_min = 0.0;
        double m_max = 10000.0;

#if DEBUG
        Utilities.Report("Creating a new instance of Aspen Plus.");
#endif

        var aspenPlusInstance = new HappLS();
#if DEBUG
        Utilities.Report("Opening the DHEX simulation.");
#endif

        aspenPlusInstance.InitFromFile2(simulationFile);
        aspenPlusInstance.SuppressDialogs = 1;

```

```

#if DEBUG
    Utilities.Report("Making Aspen Plus visible.");
#endif

    aspenPlusInstance.Visible = true;

#if DEBUG
    Utilities.Report("Starting the convergence method.");
#endif

    var value = aspenPlusInstance.Tree.FindNode(
        @"Data\Blocks\CHEXMAIN\Input\BYPASS").Value;

    Box<int> simulationRunCounter = Box<int>.New(0);
    Box<double> leanInTemperature = Box<double>.New(119.789);
    Box<double> richInTemperature = Box<double>.New(40.0);
    Box<double> leanTearTemperaturePortion = Box<double>.New(0.5);

    var richInHandler = ApparentFlows.New();
    richInHandler.MDEA = 61.76699888;
    richInHandler.PZ = 18.5306811;
    richInHandler.H2O = 1464.223819;
    richInHandler.CO2 = 22.63649965;

    var leanInHandler = ApparentFlows.New();
    leanInHandler.MDEA = 61.767;
    leanInHandler.PZ = 18.5302736;
    leanInHandler.H2O = 1463.501423;
    leanInHandler.CO2 = 4.4587616;

    Action runDHEX = () =>
    {
        richInHandler.Assert(aspenPlusInstance, "RICH-1");
        leanInHandler.Assert(aspenPlusInstance, "LEAN-1");
        leanInHandler.Assert(aspenPlusInstance, "LEAN-3");

        aspenPlusInstance.Tree.FindNode(
            @"Data\Streams\RICH-1\Input\PRES\MIXED").Value =
            parameters.StripperPressure;
    }

```

```

    aspenPlusInstance.Tree.FindNode (
        @"Data\Streams\LEAN-1\Input\PRES\MIXED").Value =
        parameters.StripperPressure;
    aspenPlusInstance.Tree.FindNode (
        @"Data\Streams\LEAN-3\Input\PRES\MIXED").Value =
        parameters.StripperPressure;

    aspenPlusInstance.Tree.FindNode (
        @"Data\Streams\RICH-1\Input\TEMP\MIXED").Value =
        richInTemperature.Value;
    aspenPlusInstance.Tree.FindNode (
        @"Data\Streams\LEAN-1\Input\TEMP\MIXED").Value =
        leanInTemperature.Value;
    aspenPlusInstance.Tree.FindNode (
        @"Data\Streams\LEAN-3\Input\TEMP\MIXED").Value =
        richInTemperature.Value
        + leanTearTemperaturePortion.Value
        * (leanInTemperature.Value -
richInTemperature.Value);

    aspenPlusInstance.Tree.FindNode (
        @"Data\Blocks\SPLTRICH\Input\FRAC\MID-1").Value =
        parameters.DHEX_SplitPortion;

    var chexArea = parameters.CHEX_TotalArea
        * (1.0 - parameters.DHEX_DistributedPortion);
    aspenPlusInstance.Tree.FindNode (
        @"Data\Blocks\CHEXMAIN\Input\AREA").Value =
        chexArea;
    aspenPlusInstance.Tree.FindNode (
        @"Data\Blocks\CHEXMAIN\Input\BYPASS").Value =
        chexArea > parameters.Threshold_Area ? "NO" : "YES";

    var dhexArea = parameters.CHEX_TotalArea
        * parameters.DHEX_DistributedPortion;
    aspenPlusInstance.Tree.FindNode (
        @"Data\Blocks\CHEXSIDE\Input\AREA").Value = dhexArea;
    aspenPlusInstance.Tree.FindNode (

```

```

        @"Data\Blocks\CHEXSIDE\Input\BYPASS").Value =
            dhexArea > parameters.Threshold_Area
        & parameters.DHEX_SplitPortion > 0.0
        ? "NO" : "YES";

DateTime startTime = DateTime.Now;
aspenPlusInstance.Run2();
DateTime endTime = DateTime.Now;

var runTime = endTime - startTime;
#if DEBUG
    Utilities.Report(
        "DHEX\tRun #\t"
        + simulationRunCounter.Value
        + "\tRun time (ms):\t"
        + runTime.TotalMilliseconds
        + "\twith temperature portion:\t"
        + leanTearTemperaturePortion.Value
    );
#endif

    ++simulationRunCounter.Value;
};

Func<double[], double[]> dhexTemperaturePortionFunction =
    (double[] temperaturePortion) =>
    {
        leanTearTemperaturePortion.Value =
temperaturePortion[0];

        runDHEX();

        var tempLow = aspenPlusInstance.Tree.FindNode(
            @"Data\Streams\RICH-1\Output\TEMP_OUT\MIXED").Value;
        var tempMid = aspenPlusInstance.Tree.FindNode(
            @"Data\Streams\LEAN-2\Output\TEMP_OUT\MIXED").Value;
        var tempHigh = aspenPlusInstance.Tree.FindNode(
            @"Data\Streams\LEAN-1\Output\TEMP_OUT\MIXED").Value;

```

```

        var calculatedTemperaturePortion =
            (tempMid - tempLow) / (tempHigh - tempLow);

        return new double[] { calculatedTemperaturePortion };
    };

    OurMath.Wegstein_Damped(
        dhexTemperaturePortionFunction
        , toleranceForHeatXTemp.Repeat(1)
        , 1000
        , (0.0).Repeat(1)
        , (1.0).Repeat(1)
        , (0.5).Repeat(1)
        , -2.5
        , 2.5
        , 10.0
        , 1.0
    );
}
catch (Exception ex)
{
    Utilities.Report("Exception ended simulation convergence.
Message:");
    Utilities.Report("\\"" + ex.Message + "\".");
}

#if DEBUG
    Utilities.Report("End : DoSomething007.");
#endif
}
}
#endif

```

Scheme 3.10. C# source code for **DoSomething007 ()**. This method runs the distributed cross heat exchanger simulation and converges the enthalpy (temperature) feedback using the Wegstein method.

3.4.3.3 Distributed cross heat exchanger run results

The distributed cross heat exchanger in `DoSomething007 ()` converges in four iterations. We used a tolerance of 10^{-5} in the code, though we can also experiment with a higher tolerance such as 10^{-9} . Figure 3.41 illustrates the convergence behavior.

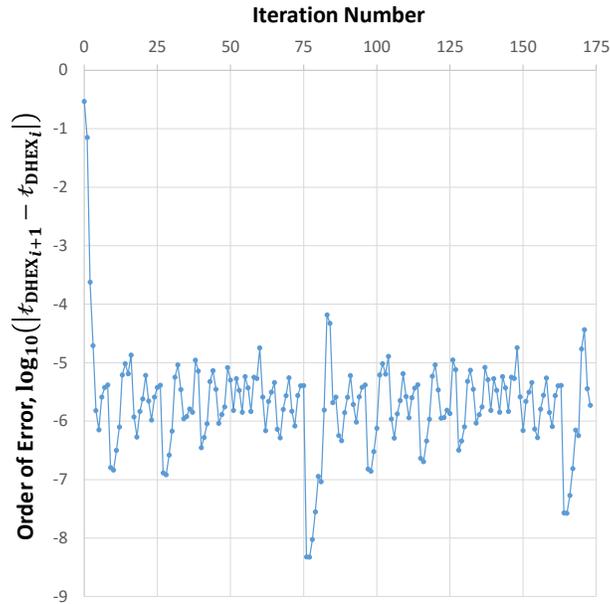


Figure 3.41. Convergence behavior for the distributed cross heat exchanger. We use an error tolerance of 10^{-5} in part to avoid the noise below the precision limit.

We can analyze the tried tear values t_{DHEX_i} against their resulting calculated values $t_{DHEX_{i+1}}$ to attempt to guess a more precise value despite the noise. Figure 3.42 shows a scatter plot, suggesting that $t_{DHEX} \cong 0.760632354290263$ in this case.

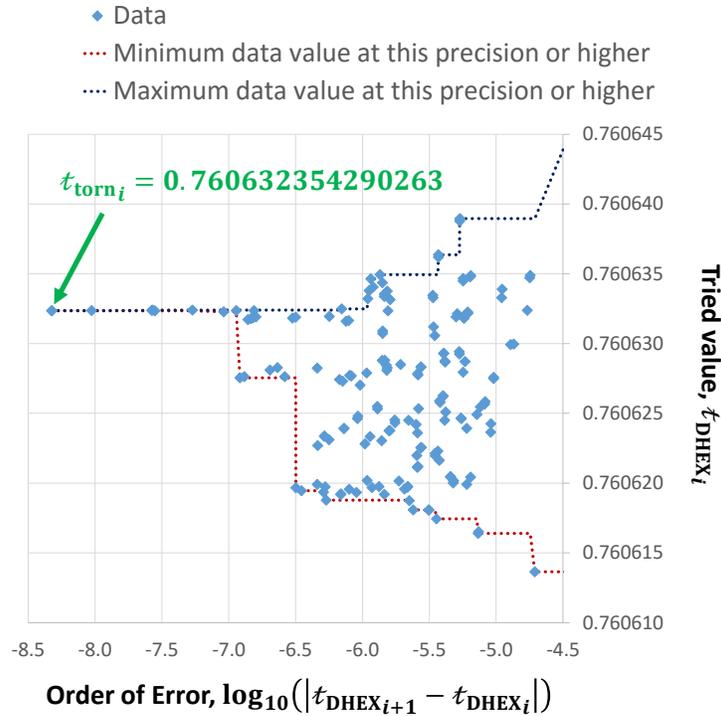


Figure 3.42. Order of error for each tried tear value, t_{torn_i} . We can see that the values with the lowest errors are concentrated around a particular peak, suggesting a more precise value despite the noise.

In practice, we will simply use the Wegstein method with an error tolerance of $|t_{\text{DHEX}_{i+1}} - t_{\text{DHEX}_i}| \leq 10^{-5}$ to keep our simulation simple, converging after the fourth iteration with $t_{\text{DHEX}} \cong 0.760628458395189$ at a relative error of $\sim 5.12 \cdot 10^{-6}$ compared to our best-guess from Figure 3.42.

3.4.4 Build the acid-gas capture unit

We have now created the three primary flowsheet elements for the acid-gas capture unit:

1. the absorber complex;
2. the stripper complex;
3. the cross heat exchanger complex.

These units roughly correspond to the derived objects in the prior chapter, **SQ-ABS-2**, **SQ-STR-2**, and **SQ-MID-2**, respectively.

We do **not** draw these units together in a single flowsheet. Rather, we will control our overall simulation in C#, transferring data when appropriate. For example, our C# application will read

information about the rich solvent coming out of the bottom of the absorber and feed it into our DHEX simulation before running it.

Figure 3.43 shows our overall flowsheet logic. We consider the absorber to be computationally upstream of the other units such that we simply run the absorber first and are then done with it. The DHEX unit does provide the lean solvent to the absorber, though we ignore this information flow for the purpose of convergence. This is extremely convenient, and we can justify it by noting that:

1. the lean stream cooler (LSC) cools the lean solvent to a specified temperature, so the enthalpy result from the DHEX is moot;
2. the make-up unit provides the missing solvent, so we consider the solvent flow rates as process specifications to be optimized;
3. the selected capture rate fully determines the acid gas flow rate in the solvent;
4. the other gases in the process, e.g. N_2 , O_2 , and H_2 , are heavily dampened by their low solubilities such that their feedback in the lean solvent stream is largely irrelevant to the simulation.

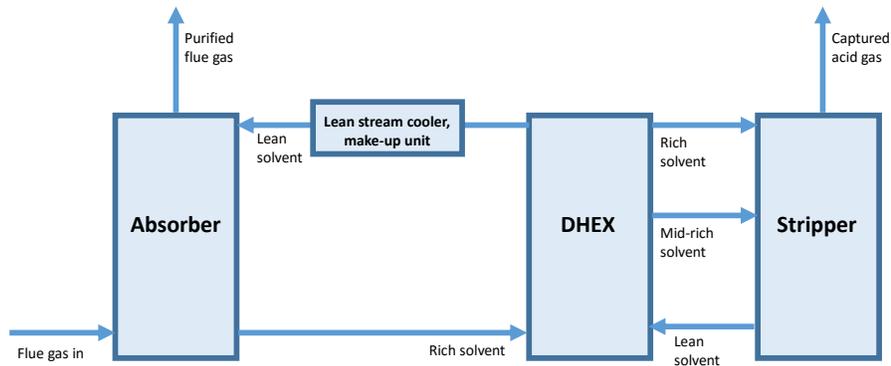


Figure 3.43. Flowsheet logic merging our individual flowsheets together under object-oriented logic.

However, we do not have similarly convenient arguments for the interaction between the DHEX and stripper, especially after implementing the energy-saving schemes. We decide to tear the lean solvent flow from the stripper to the DHEX using the Wegstein method.

3.4.4.1 Addressing licensing issues

Before proceeding, it is worth considering the practical limitations imposed by commercial licensing requirements. At the time of writing, Aspen Tech generally provides client organizations with a limited number of licenses, each permitting a single concurrent session of Aspen Plus. For example, an organization with 150 licenses can have up to 150 instances of Aspen Plus running simultaneously.

Our current acid-gas-capture simulation employs three instances of Aspen Plus to simulate what we previously had in a single flowsheet. Since each instance requires a license, this suggests that we now need three times as many licenses to run the basic simulation. Further, we still have missing flowsheet components such as the rich solvent pump that would, in principle, call for their own flowsheet and thus additional licenses.

We can work around this licensing issue in several ways. The ideal solution depends on your particular licensing situation.

3.4.4.1.1 Opening and closing simulations when needed

It does not require much work to add in C# code that opens and closes each instance of Aspen Plus as it is needed. For example, as we use the Wegstein method to converge the DHEX and stripper, we could open and close both the DHEX and stripper simulations on each iteration.

It is easy to open and close simulations, however we try to avoid excessive opening and closing to minimize licensing delays.

If your licensing situation is very constricting, you may wish to open-and-close simulations on each run as standard procedure. This will reduce your license usage from zero-to-one, depending on if you are actively running a simulation, which is actually better than always using one as a normal Aspen Plus user does. Unfortunately, you will have to settle for longer-running simulations due to the overhead from waiting for Aspen Plus to open.

We will not employ this method by default in this text because our licensing situation is not significantly constraining.

3.4.4.1.2 Adding minor units into nearby derived objects

The rich solvent comes out of the bottom of the absorber, pumped up to the stripper's pressure, and sent to the distributed cross heat exchanger. In principle, each of these three units would have its own Aspen Plus simulation:

1. the absorber complex;
2. the rich solvent pump;
3. the distributed cross heat exchanger (DHEX).

These three simulations would be logically connected as shown in Figure 3.44. Running this part of the flowsheet would mean:

1. Open all three simulations: Absorber.apwz, Pump.apwz, and DHEX.apwz.
2. Run Absorber.apwz.
3. Transfer the results from **RICH-OUT** in Absorber.apwz to input for **RICH-OUT** in Pump.apwz.
4. Run Pump.apwz.
5. Transfer the results from **RICH-1** in Pump.apwz to **RICH-1** in DHEX.apwz.
6. Run DHEX.apwz.

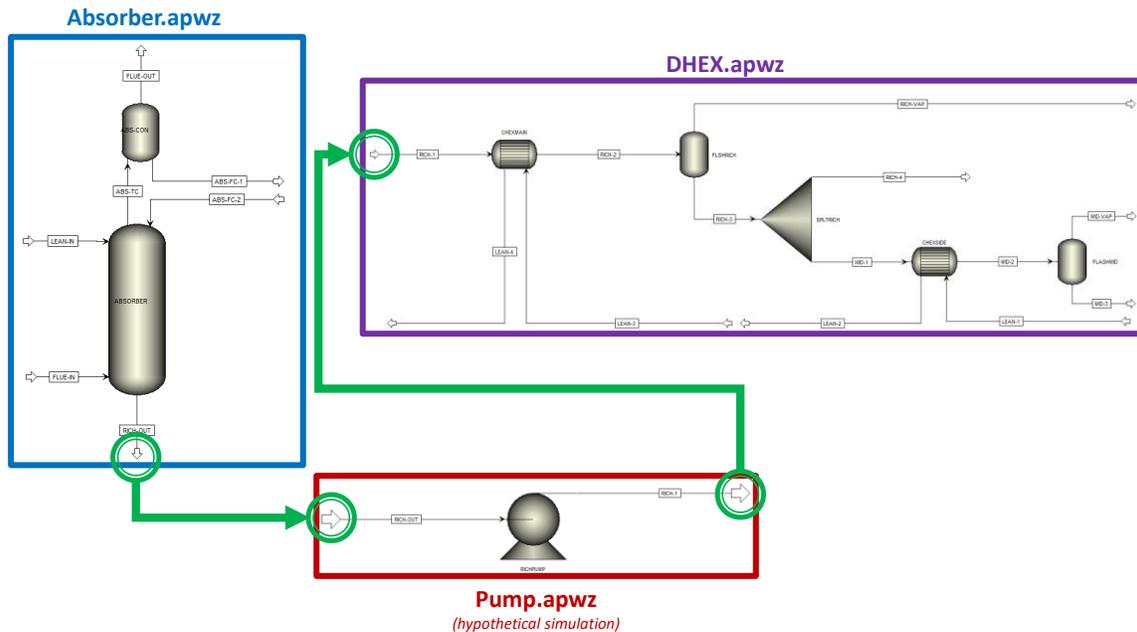


Figure 3.44. The absorber and DHEX are separated by the rich solvent pump. In principle, the rich solvent pump would exist in its own simulation connected to both the absorber simulation

(Absorber.apwz) and the DHEX simulation (DHEX.apwz). This configuration would require either (1) an additional license to keep Pump.apwz open or (2) for us to open-and-close Pump.apwz each time we want to run the rich solvent pump.

In theory, if we contained Aspen Tech's code in our own C# project, this would be the ideal approach. It would mean that the rich solvent pump, **RICHPUMP**, runs only when the program transfers the rich solvent from the absorber to the DHEX.

In practice, we would have to run Pump.apwz as its own Aspen Plus instance, costing us:

- an additional Aspen Plus license;
- the RAM overhead for another instance of Aspen Plus;
- additional simulation time to open/license Pump.apwz;
- greater exposure to Aspen Plus crashes due to relying on another instance of Aspen Plus.

To avoid these additional penalties, we will add **RICHPUMP** to either Absorber.apwz or DHEX.apwz. The main drawback to this approach is that **RICHPUMP** will run unnecessarily as Absorber.apwz or DHEX.apwz converge.

In our most recent runs, the absorber complex took 55 iterations to converge and the DHEX took 5 iterations to converge. So, adding **RICHPUMP** to Absorber.apwz implies 54 unnecessary runs per absorber run while adding it to DHEX implies 4 unnecessary runs per DHEX run. The DHEX.apwz may seem like the obvious winner, though in general DHEX.apwz will run many more times than Absorber.apwz since the DHEX must converge with the stripper complex.

We make a judgement call in adding **RICHPUMP** to the absorber complex in Absorber.apwz, modifying it as shown in Figure 3.45.

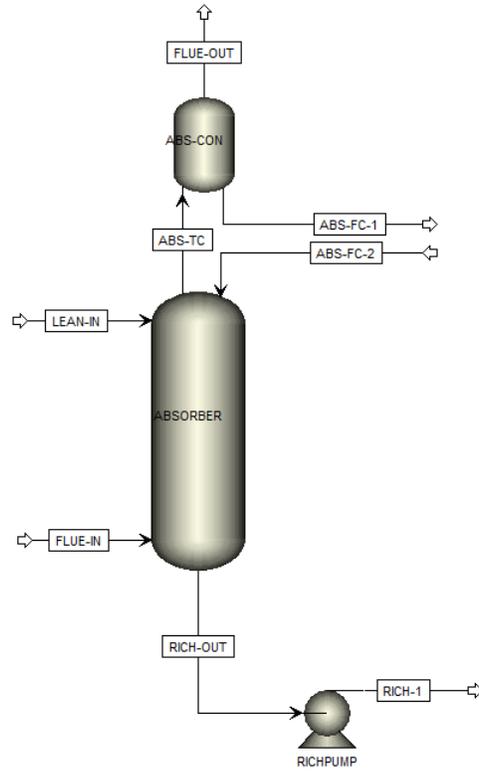


Figure 3.45. Our new absorber complex flowsheet in Absorber.apwz. This flowsheet adds **RICHPUMP** to avoid having to put **RICHPUMP** in its own simulation.

Figure 3.46 shows the resulting simulation-level logic.

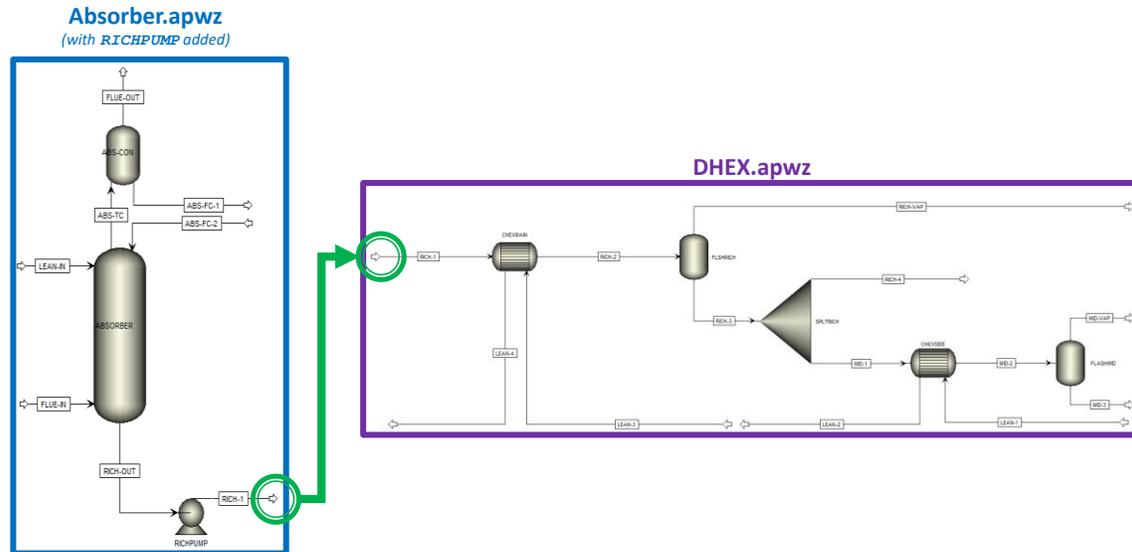


Figure 3.46. The absorber complex and the DHEX are still separated by **RICHPUMP**, though we avoid the need for an additional rich solvent pump simulation, e.g. Pump.apwz. This comes at the cost of having to run **RICHPUMP** every time Absorber.apwz is run, e.g. about 54 unnecessary times in our last test in which we used the Wegstein method to converge the absorber's condenser feedback (**ABS-FC-1** and **ABS-FC-2**) and the secant method to set the appropriate effective carbon capture rate in the absorber, \mathcal{R}_{ABS} , by adjusting the lean solvent flow rate.

In principle, we could perform this trick for all of our flowsheeting elements, combining the absorber complex, DHEX, stripper complex, and any other unit into a single Aspen Plus simulation containing multiple, disconnected components. We do not cover this approach in the present chapter because that flowsheet would have a significantly longer run time.

Adventurous readers might try implementing a single flowsheet using this technique and attempting to use affected block logic to avoid unnecessary unit runs. We do not cover this technique in the present dissertation.

3.4.4.1.3 Performing simple operations without calling Aspen Plus

Some operations do not require Aspen Plus at all. For example, in the last chapter we described using **FSplit** blocks to feed streams into non-integer stages on a column using interpolation and weighted

splitting (see Section 1.6.1: “Interpolating non-integer values”). This operation is very simple, so we do not need to make an `FSplit` block at all.

In general, the more you can do in C#, the less you need Aspen Plus and other commercial tools. For example, if you can implement flash calculations in C#, then you presumably could avoid simulations that just contain flash-based units, e.g. `Flash2`'s (such as used for the condensers and reboiler), `Heater`'s, `Pump`'s, `Valve`'s, etc.

In this text we will not make use of any significant property model library outside of Aspen Tech's products, so we will rely on Aspen Plus to perform all major physical calculations. However there are quite a few blocks that we can forego using only simple C# code:

- `Calculator`;
- `FSplit`;
- `Dupe`;
- `Mult`;
- `Selector`;
- `Design Spec`;
- `Transfer`;
- `Sequence`;
- `Convergence`;
- `Sensitivity Analysis`;
- `Balance`;
- `Measurement`;
- `Hierarchy`.

3.4.4.2 [Modifying our simulations](#)

We will modify some of our simulations for the reasons discussed above.

First, modify Absorber.apwz to reflect the new design in Figure 3.45 by adding a Pump, **RICHPUMP**. Set **RICHPUMP** to have a discharge pressure of 1bar. As with other simulation elements, this is a dummy value that will always be overwritten.

Next, modify Stripper.apwz to have the additional feed streams into **STRIPPER** shown in Figure 3.47. These new streams are **RVAP-IN1**, **RVAP-IN2**, **MVAP-IN1**, and **MVAP-IN2**. Have all feed into Stage 1 with the On-Stage convention. Specify each has having a temperature of 40°C, a pressure of 2 bar, and a component mole flow rate of $1 \frac{\text{kmol H}_2\text{O}}{\text{hr}}$, just like the other feed streams.

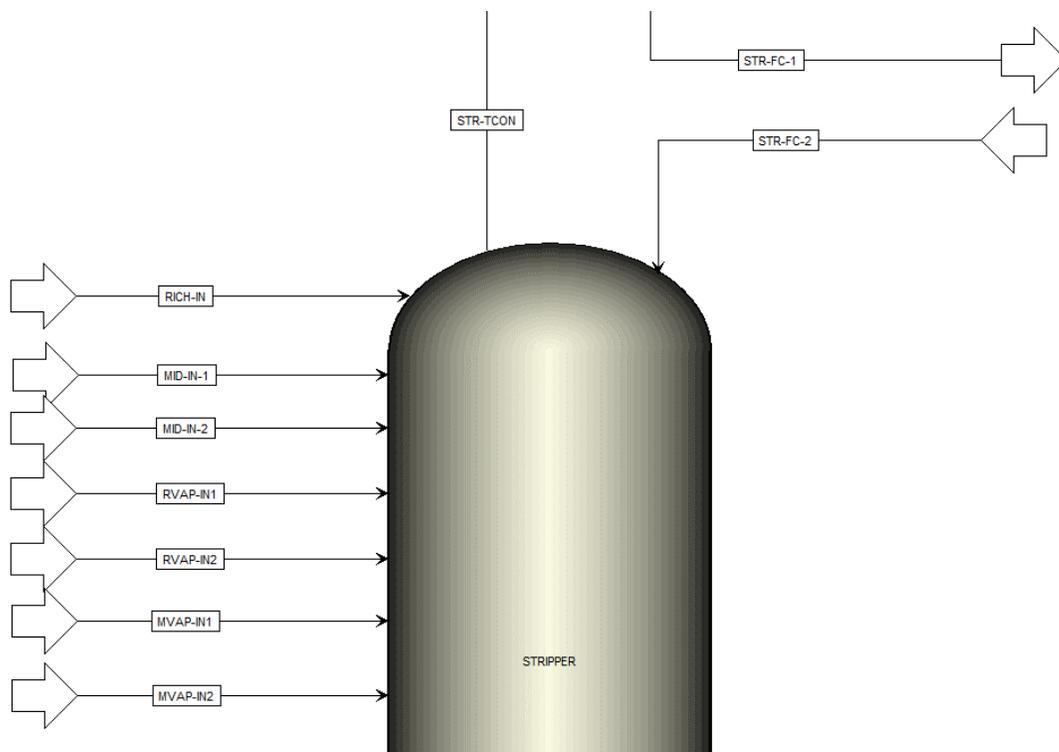


Figure 3.47. Additional streams to add to Stripper.apwz, feeding into **STRIPPER**: **RVAP-IN1**, **RVAP-IN2**, **MVAP-IN1**, and **MVAP-IN2**.

3.4.4.3 Converging the acid-gas-capture unit

Converging the acid-gas-capture unit shown in Figure 3.43 requires:

1. tearing the feedback between the DHEX and stripper;
2. tearing the feedback between the absorber and DHEX/stripper;
3. adjusting the lean solvent flow rate to meet the capture rate in the absorber complex;

4. adjusting the reboiler duty to meet the capture rate in the stripper complex.

We can perform these methods using the code shown in Scheme 6.1.

3.5 Wrapping Aspen Plus

In prior sections, we interacted with Aspen Plus through its **Happ** interface. We loaded simulation files, set/read values, started simulations, etc. using the methods provided by Aspen Tech on the **HappLS** object.

While this approach is demonstrably adequate, we can greatly improve our situation by making our own class that “wraps” **HappLS**. Wrappers can include many useful features such as:

1. Provide methods for commonly used methods.
2. Implement exception handling or/and error handling.
3. Provide lazy loading for simulations, where simulations are opened only when needed.
4. To limit license usage, a wrapper can open simulations and then immediately close them after reading off the results.
5. Wrappers can attempt to run the same simulation in multiple threads with different parameters or/and convergence methods to perform sensitivity analyses or speed convergence.

In this section, we will look at implementing a basic wrapper with common features. Regular Aspen Plus users will likely want to customize their own wrapper specific to their style and use needs.

Scheme 6.2 in the Appendix shows the source code for the basic wrapper. This wrapper:

1. Provides simple methods for opening/running/showing/hiding simulations and getting/setting simulation values.
2. Catches errors and automatically repeats actions that failed.
3. Loads Aspen Plus lazily. This is, the simulation won't load until used.
4. Automatically closes Aspen Plus once the wrapper is no longer in use.
5. Automatically restarts Aspen Plus if Aspen Plus crashes, fails due to a licensing error, etc.

3.5.1 Automatic crash recovery

Our wrapper provides automatic crash recovery. Whenever Aspen Plus fails, the wrapper:

- i. detects the failure;
- ii. kills the remaining Aspen Plus process, if any, to ensure that a frozen Aspen Plus instance does not continue to hog a license and computer resources;
- iii. reopens the Aspen Plus simulation;
- iv. writes the most recent set of run values to the Aspen Plus simulation;
- v. resumes the Aspen Plus run.

To demonstrate this feature, we intentionally crash Aspen Plus by using **Ctrl+Alt+Del** to force it closed in the middle of a run. Figure 3.48 shows this forced crash.

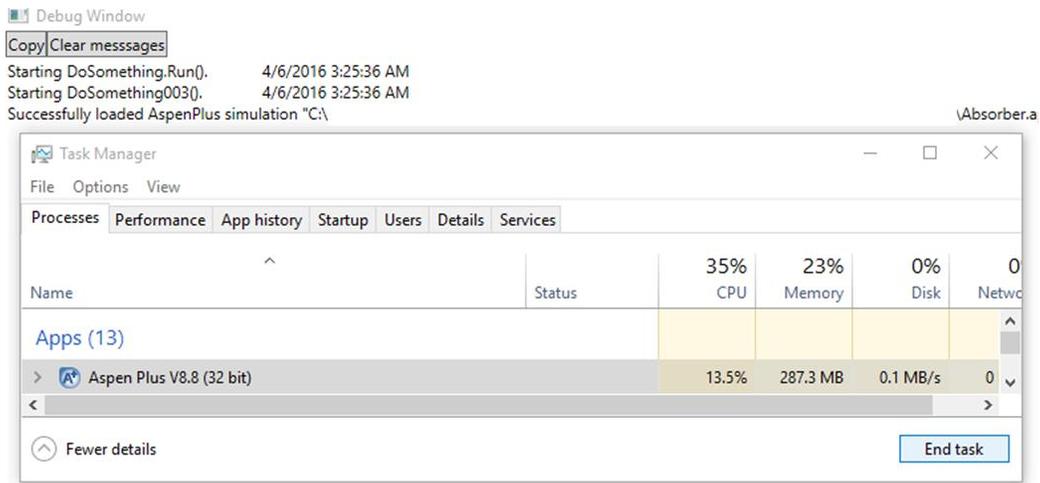


Figure 3.48. We use Task Manager, accessible by **Ctrl+Alt+Del** in Windows, to intentionally crash Aspen Plus while it is running.

This crash causes the **HappLS** object in our Aspen Plus wrapper to throw an exception which is caught by the **try{}catch{}** logic. Scheme 3.11 shows an abridged version of the **Run()** method in which this happens.

```
public bool Run()
{
    bool runSuccess = false;
    for (int i=0; i < MAX_RUN_ATTEMPTS; ++i)
```

```

{
    try
    {
        instance.Run2 ();
        runSuccess = true;
        break;
    }
    catch (Exception exception)
    {
        this.Internal_ClearDueToError ();
    }
}
return runSuccess;
}

```

Scheme 3.11. Abridged **Run ()** method in our Aspen Plus wrapper. This method wraps the **Run2 ()** method provided by **HappLS**.

Figure 3.49 shows the resulting recovery. The run attempt limit keeps track of the number of times that a specific execution attempt has failed; the attempt count is reset on every run attempt.

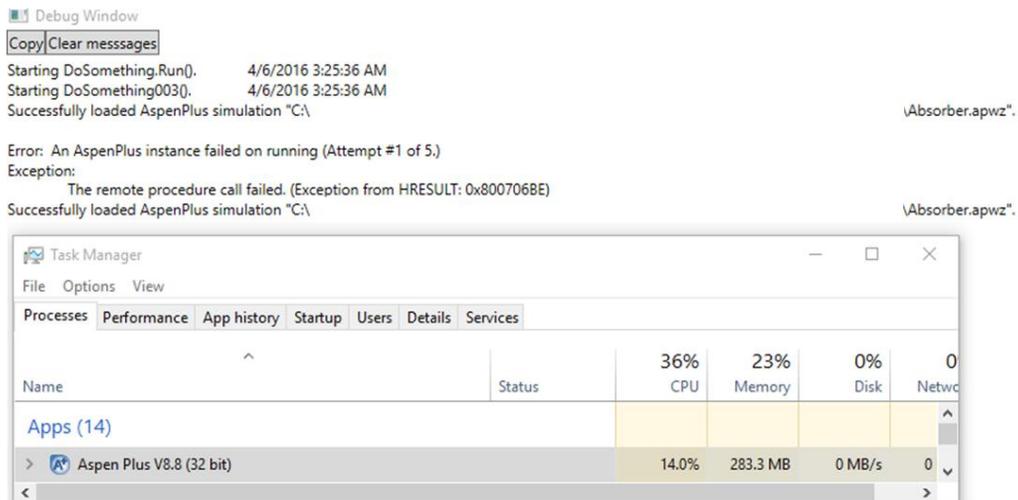


Figure 3.49. The forced crash caused an exception shown above. This exception is immediately received by the wrapper, and the wrapper recovers by ensuring that the crashed Aspen Plus instance is fully closed, then opens up a fresh instance.

We can easily change the current maximum number of attempts by modifying the line of code

```
protected const int MAX_RUN_ATTEMPTS = 5;
```

We selected a maximum of five attempts under the reasoning that, if a single run fails 5 consecutive times, the problem is unlikely to go away on subsequent attempts.

We can further improve and expand the basic wrapper presented in this section. However, the implementation given in Scheme 6.2 can be useful if you wish to start your own simulation framework from scratch rather than use one given in the electronic media associated with this dissertation.

3.6 Developing our simulation tools

In Section 3.4: “Creating an acid-gas-capture simulation in C#”, we managed to create an acid-gas-capture simulation using a minimally developed code base. We did not use any premade class types or even a wrapper for Aspen Plus. While our code worked, it was relatively unorganized, lacked a user interface, had no error-recovery mechanisms, could not pause or save runs, was single-threaded, etc.

In this section, we will develop a basic framework to implement the acid-gas-capture unit. We will attempt to add the missing features such as error recovery, a user interface, run controls, etc.

3.6.1 Framework structure

At the flowsheet level, Aspen Plus has a recursive tree structure composed of blocks. For example, in the prior chapter, we focused on deriving new objects composed of a **Sequence** of component blocks.

3.6.1.1 FlowsheetBlock's

In this framework, we will use the same recursive tree structure composed of blocks of the **abstract** type **FlowsheetBlock** as defined in Scheme 3.12. While our actual implementation will contain far more source code, the most important feature that all **FlowsheetBlock**'s must contain is the ability to run. Many **FlowsheetBlock**'s will tell others to **Run ()** as part of their own execution, forming a recursive tree evaluation like Aspen Plus employs. All simulations will ultimately be composed of a single top-level **FlowsheetBlock** which, when called, will lead to all others being executed as appropriate.

```
public abstract class FlowsheetBlock
{
```

```

public void Run(SP parameters) { this.InternalRun(parameters); }
protected abstract void InternalRun(SP parameters);
}

```

Scheme 3.12. Main conceptual definition for a **FlowsheetBlock**. Our actual implementation will contain far more code.

Aspen Plus simulations are terminal nodes in the recursive block tree, wrapped up as **AspenPlusSimulationBlock**'s. For example, we will use the three-simulation structure described in Section 3.4: "Creating an acid-gas-capture simulation in C#", so we will have three **AspenPlusSimulationBlock**'s in the recursive tree structure.

3.6.1.2 **InternalElement**'s

We want our Aspen Plus wrapper should restart Aspen Plus whenever Aspen Plus crashes. However, if we use our prior approach of setting simulation values at random locations in the code, then when the Aspen Plus simulation is restarted, it would be missing those values. Additionally we could not use lazy loading since all Aspen Plus instances would need to be open at all times for reading and writing values during the overall run process.

To get around these limitations, we define sub-wrappers for individual Aspen Plus blocks. Whenever an Aspen Plus wrapper is **Run ()**, it:

1. Asserts the values from all the sub-wrappers onto the corresponding blocks in Aspen Plus.
2. Runs Aspen Plus.
3. Recovers values from Aspen Plus, setting them to the sub-wrappers.

We implement these sub-wrappers as **InternalElement**'s, as conceptually defined in Scheme 3.13.

Each **InternalElement** has:

1. an **Assert ()** method, called before Aspen Plus is run to assert its values;
2. an **Update ()** method, called after Aspen Plus is run to recover its values.

```

public abstract partial class InternalElement
{
    public abstract void Assert(SP parameters, AspenSimulationBlock
    aspenWrapper

```

```

        , string internalName);
    public abstract void Update(SP parameters, AspenSimulationBlock
    aspenWrapper
        , string internalName);
}

```

Scheme 3.13. Source code for the conceptual implementation of **InternalElement**.

3.6.1.3 SimulationParameters

When constructing our acid-gas-capture unit in Aspen Plus, we made heavy use of global parameters to move information around the simulation. In our C# framework, we will use **SimulationParameters** to cover that need, though **SimulationParameters** are significantly more flexible and easy-to-use.

If you inspect the definitions for **FlowsheetBlock's**, **InternalElement's**, etc., you can see that **SP parameters** are passed down the recursive tree during all types of evaluations. These are the **SimulationParameters**, though we shorten it to **SP** for convenience since this type is heavily used.

For the moment, we will focus on single-threaded operation. However, one of the great advantages **SimulationParameters** provide is the ability to fork: any time we want a branch to be split into multiple different evaluations, such as for multi-threaded operation, we can duplicate the **SimulationParameters** into multiple copies and pass a different copy to each. This same approach also enables distributed computation across arbitrary networks as well as robust sensitivity analyses.

Scheme 3.14 shows a heavily abridged implementation of **SimulationParameters**. The full implementation is largely similar until we start adding in advanced features for simulation control, forking, etc.

```

public partial class SP // "SP" is short for "SimulationParameters"
{
    public P<double> CaptureRate { get; private set; } = 0.8;

    #region Flue gas
    public P<double> Flue_Temperature { get; private set; } = 40;

```

```

public P<double> Flue_Pressure { get; private set; } = 1.01;
public P<double> Flue_CO2MoleFlowRate { get; private set; }
    = 1000.0 / MolecularMass.CO2;
public P<double> Flue_InertPortionOfDryFlue { get; private set; } =
0.942359127;
    public P<double> Flue_NitrogenPortionOfInertDryFlue { get; private set;
}
    = 0.874331258;
public P<double> Flue_WaterAsMultipleOfDryFlue { get; private set; } =
0.106042698;
#endregion

#region Lean solvent
public P<double> LeanSolvent_AmineMoleFlowRate { get; private set; } =
83.52321;
    public P<double> LeanSolvent_AminePortion { get; private set; } =
0.054997612;
    public P<double> LeanSolvent_PZPortion { get; private set; } =
0.230779782;
    public P<double> LeanSolvent_Loading { get; private set; } =
0.055456292;
#endregion

public P<double> LSC_OutputTemperature { get; private set; } = 40;

public P<double> Absorber_Pressure { get; private set; } = 1.0;
public P<double> Absorber_PackingHeight { get; private set; } = 15.0;
public P<double> Absorber_Condenser_Temperature { get; private set; } =
40.0;

public P<double> Target_ApproachToFloodingFactor { get; private set; }
= 0.6;

public P<double> Stripper_Pressure { get; private set; } = 2.0;
public P<double> Stripper_PackingHeight { get; private set; } = 2.6;
public P<double> Stripper_Condenser_Temperature { get; private set; } =
40.0;

```

```

public P<double> CHEX_TotalArea { get; private set; } = 100.0;

public P<double> DHEX_DistributedPortion { get; private set; } = 0.5;

public P<double> Threshold_Area { get; private set; } = 1e-7;

public P<double> DHEX_SplitPortion { get; private set; } = 0.25;

public P<double> DHEX_HeightPortion_Distributed_Liquid { get; private
set; } = 0.5;
    public P<double> DHEX_HeightPortion_Main_Vapor { get; private set; } =
1.0;
    public P<double> DHEX_HeightPortion_Distributed_Vapor { get; private
set; } = 1.0;

    public P<double> DHEX_Tear_TemperaturePortion { get; private set; } =
0.5;

    public P<int> Absorber_StageCount { get; private set; } = 25;

    public P<int> Stripper_StageCount { get; private set; } = 10;

    public P<double> FloodingSoftTolerance { get; private set; } = 0.01;
}

```

Scheme 3.14. A heavily abridged version of the basic data-storing parameters in **SimulationParameters**.

3.6.2 Using the new framework

Previously we have constructed the acid-gas-capture unit in Aspen Plus and then again in unstructured C#. Now we will construct the acid-gas-capture unit in the new, structured framework.

3.6.2.1 Defining the simulation

Scheme 6.3 shows a specification of the simulation; readers who have used Aspen Plus input files might see this code as similar to an input file. Much of the early code simply defines the names for blocks and streams in our Aspen Plus simulations so that the framework knows how to connect

InternalElement's with internal simulation blocks. Alternatively, some users might prefer to have their own framework merely assume that the same elements in different simulations, e.g. the **RICH-OUT** stream connecting the absorber and DHEX, also have the same internal names. We avoid this assumption to maintain flexibility.

3.6.2.2 Running the simulation

Running the new acid-gas-capture simulation causes the window in Figure 3.50 to appear. This new window shows the recursive tree structure of the simulation, including the run times, run counts, start times of active runs, and status of which blocks are currently active.

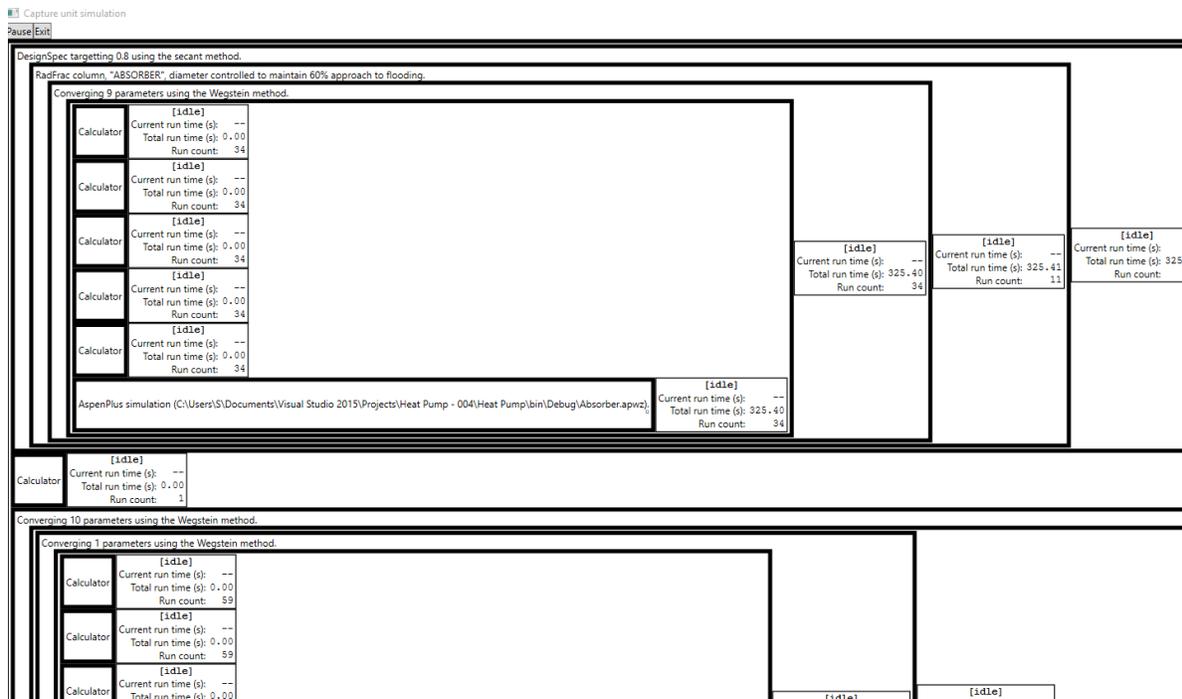


Figure 3.50. Part of the simulation interface automatically generated by our simulation framework.

This particular screenshot was taken after the run completed. We can see the recursive tree structure of blocks, including the number of times each block was executed and how much time that block spent running. Additionally we added a Pause button and an Exit button for basic simulation control.

This entire simulation took about 1 hour, 46 minutes, and 15.35 seconds. Of that time, all of our C# calculations – including the user interface – consumed only 0.09 seconds. The simulation time breakdown is as follows:

- 0.001% of the time for our C# framework;
- 5.104% of the time for the absorber simulation;
- 19.623% of the time for the DHEX simulation;
- 75.272% of the time for the stripper simulation.

Since the stripper simulation took over 75% of the overall simulation time alone, we might focus on reducing its runtime to increase simulation speed.

3.7 Summary

The Aspen Plus COM interface is a powerful tool for building developing our simulations. When we worked entirely within Aspen Plus, we were limited to methods that were easily expressed using the built-in tools.

The COM interface allows us to take advantage of tools like Windows Presentation Foundation (WPF) for a GUI, exception handling and crash recovery for reliability, multi-threading to run multiple simulations at once, and custom convergence methods for quicker, more reliable simulations.

New programmers may experience a learning curve as they become acclimated to C#, though we feel that the benefits are well worth the effort.

4 Conclusions

The primary academic conclusions from those research include:

- i. a methodology for process design and optimization;
- ii. specific energy-efficient designs for CO₂-capture units;
- iii. a method for effectively recovering waste heat using an aqueous lithium bromide heat pump;
- iv. software techniques for modeling, simulating, and optimizing processes.

4.1 Accomplishments

4.1.1 Industrial implementation

We presented these research findings at the Sinopec headquarters in Beijing on January 14, 2014 to the senior management of Sinopec Science and Technology Department and SINOPEC Petroleum Engineering Design Company as part of the “Development and Application Demonstration of Carbon Capture, EOR and Storage Technology at Large-scale Coal-fired Power Plant” project. This project focuses on building a unit to capture one million tonnes of CO₂ per year at the Shengli Oil Field Power Plant. Captured CO₂ is to be pumped underground for enhanced oil recovery (EOR). The initial plant design was completed in December 2013, and operations are to expected to begin in 2017.

This research was well received. Former Senior Vice President and Chief Technology Officer of Sinopec, Cao Xianghong, who is a member of the Chinese Academy of Engineering and an elected foreign member of the U.S. National Academy of Engineering, commented:

My colleagues and I were so pleased with the lowest solvent regeneration energy requirement of the new process that is leading competing processes internationally, and have recommended its actual implementation in our Shengli Oil Field Power Plant to capture 1 million tonnes of CO₂ per year for enhanced oil recovery.

4.1.2 Patent

We have patented our design and methods as part of “Energy-Efficient Extraction of Acid Gas from Flue Gases”. Our US patent application is No. 62/008,587, and our international application is PCT/US15/34584.^{34,35} The Virginia Tech Intellectual Properties office (VTIP) has noted interest from software companies in the patented algorithms and from power companies in the patented capture unit design.

4.1.3 Publication

This research appears in the American Chemical Society’s *Industrial & Engineering Chemistry Research* (I&EC) in 2015 as “CO2 Capture Modeling, Energy Savings, and Heat Pump Integration”.

Additionally, a large portion of this dissertation will appear as a textbook on the subject.

4.1.4 Tools

We began this research with inadequate tools to solve the problem; building those tools has been a major accomplishment of the project. The COM-based approaches discussed in Chapter 3: “Using Aspen Plus through its COM interface using C#” are a major development.

Additionally, we have developed some relatively minor tools, such as our computer lab, Excel add-ins, and small helper applications.

4.2 Recommendations for future research

Future work may consider further process generalizations and improvements such as:

- applying the heat pump to other areas of the process rather than just the reboiler;
- using alternative solvent systems;
- more rigorous physical modeling.

We strongly recommend that future researchers consider using the COM interface for interacting with Aspen Plus. The standard user interface has proven to be convenient for smaller projects, though automation can greatly benefit long-term projects. Future work should focus on designing an appropriate automation framework rather than relying on manual user control.

5 References

1. NASA. Mars Climate Orbiter Fact Sheet. 2016; <http://mars.jpl.nasa.gov/msp98/orbiter/fact.html>.
2. IPCC. Carbon Dioxide Capture and Storage. 2005. http://www.ipcc.ch/pdf/special-reports/srccs/srccs_wholereport.pdf.
3. Grossmann IE, Westerberg AW. Research Challenges in Process Systems Engineering. *AIChE Journal*. September 2000 2000;46(9):4.
4. *Climate Change 2014 Synthesis Report Summary for Policymakers*: Intergovernmental Panel on Climate Change (IPCC);2014.
5. Bottoms RR, Inventor. Process for Separating Acidic Gases. US patent US 1,783,9011930.
6. Higgins SJ, Liu YA. CO₂ Capture Modeling, Energy Savings, and Heat Pump Integration. *Industrial & Engineering Chemistry Research*. 2015;54(9):2526-2553.
7. Gaus W, Hochschwender K, Schunck W, Inventors; I. G. Farbenindustrie Aktiengesellschaft, assignee. Process for Extracting Carbon Dioxide from Gaseous Mixtures and Forming Alkaline Carbonates. US patent US Patent 1,897,725. May 16, 1927.
8. Gall J. *Systemantics: How Systems Work and Especially How They Fail*: Pocket; 1978.
9. (EPA) USEPA. Clean Power Plan. 2016; <https://www.epa.gov/cleanpowerplan>, 2016.
10. Rubin ES, Rao AB, Chen C. Comparative Assessments of Fossil Fuel Power Plants with CO₂ Capture and Storage. *Proceedings of the 7th International Conference on Greenhouse Gas Control Technologies (GHGT-7)*. 2004:9.
11. Oyenekan BA, Rochelle GT. Alternative strippre configurations to minimize energy for CO₂ capture. *8th International Conference on Greenhouse Gas Control Technologies (GHGT-8)*. Trondheim, Norway2006:5.
12. Oyenekan BA, Rochelle GT. Alternative stripper configurations for CO₂ capture by aqueous amines. *AIChE Journal*. 2007;53(12):3144-3154.
13. Oyenekan BA, Rochelle GT. Alternative Stripper Configurations for CO₂ Capture by Aqueous Amines. *Sixth Annual Conference on Carbon Capture and Sequestration*. Pittsburgh, PA2007:18.
14. Oyenekan BA. *Modeling of Strippers for CO₂ Capture by Aqueous Amines* [Ph.D. Dissertation], The University of Texas at Austin; 2007.
15. Aspen Technology I. Energy Analysis Configuration Sheet. 2015(Aspen Plus V8.8 Help).
16. Ragatz EG, Inventor. Method for the Absorption of Gases. US patent US19872671935.

17. William B. Borst J, Inventor. Absorption Process. US patent US19872671967.
18. Fluor's Econamine FG Plus Technology. *NETL Carbon Sequestration Conference*2003.
19. Scherffius JR, Reddy S, Klumpyan JP, Armpriester A. Large-Scale CO₂ Capture Demonstration Plant using Fluor's Econamine FG Plus Technology. *Energy Procedia*. 2013;37:9.
20. Freguia S. *Modeling of CO₂ Removal from Flue Gases with Monoethanolamine* [M.S. Thesis]: Engineering, The University of Texas at Austin; 2002.
21. Karimi M, Hillestad M, Svendsen HF. Investigation of intercooling effect in CO₂ capture energy consumption. *Energy Procedia*. 2011;4:1601-1607.
22. Baburao B, Schubert C, Inventors. Advanced Intercooling and Recycling in CO₂ Absorption. US patent US 8,460,436 B22013.
23. Cabanaw EJ, Mann JW, Inventors. Alkylation Process. US patent US4404419 A. September 13, 1983.
24. *The Handbook of the Nitrogen Engineer*. Russia: Chimia; 1986.
25. Bresler SA, Francis AW, Inventors; Chemical Construction Corporation, assignee. Process for Removal of Acid Gas from Gas Streams. US patent US Patent 3,101,996. March 29, 1961.
26. Carré F, Inventor. Improvement in Apparatus for Freezing Liquids. US patent US 30201. 1860, 1860.
27. Kasley AT, Inventor; Westinghouse Electric & Mfg Co, assignee. Refrigerator. US patent US Patent 1,506,530. February 4, 1924.
28. Erickson DC, Inventor; Donald C. Erickson, assignee. Absorption Heat Pump Augmented Thermal Separation Process. US patent US Patent 4,350,571. September 21, 1982.
29. Wilinson WH, Hanna WT, Inventors; Battelle Development Corp., assignee. Process and System for Boosting the Temperature of Sensible Waste Heat Sources. US patent US Patent 4,333,5151982.
30. Rockenfeller U, Sarkisian P, Inventors; Rocky Research, assignee. Triple Effect Absorption Cycle Apparatus. US patent US Patent 5,335,515. August 9, 1994.
31. Zhang K, Liu Z, Li Y, Li Q, Zhang J, Liu H. The improved CO₂ capture system with heat recovery based on absorption heat transformer and flash evaporator. *Applied Thermal Engineering*. 2014;62:7.
32. Delfort B, Carrette P-L, Bonnard L. MEA 40% with improved oxidative stability for CO₂ capture in post-combustion. *Energy Procedia*. 2011;4:9-14.

- 33.** Lin Y-J, Madan T, Rochelle GT. Regeneration with Rich Bypass of Aqueous Piperazine and Monoethanolamine for CO₂Capture. *Industrial & Engineering Chemistry Research*. 2014;53(10):4067-4074.
- 34.** Higgins S, Liu YA, Yu Y, Inventors; Virginia Tech Intellectual Properties, Inc., assignee. Energy-Efficient Extraction of Acid Gas from Flue Gases. June 6, 2015, 2015.
- 35.** Higgins S, Liu YA, Yu Y, Inventors; Virginia Tech Intellectual Properties, Inc., assignee. Energy-Efficient Extraction of Acid Gas from Flue Gases. June 5, 2015, 2015.

6 Appendices

Appendix A. Code

Appendix A.1. CO₂-capture unit simulation without a framework

The code in this appendix executes a basic CO₂-capture unit simulation. This code does not make use of a developed framework, but rather relies on relatively direct access to Aspen Plus through a wrapper.

```
#define INCLUDE
//#define REINITIALIZE_SIMULATION_EACH_RUN
//#define OPEN_SIMULATIONS_IN_PARALLEL
#if INCLUDE
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using Happ;

namespace Heat_Pump
{
    partial class DoSomething
    {
        public static void DoSomething009()
        {
#if DEBUG
            Utilities.Report("Start: DoSomething009.");
#endif

            try
            {
                string absorberSimulationFile =
System.IO.Directory.GetCurrentDirectory() + @"Absorber.apwz";
```

```

        string stripperSimulationFile =
System.IO.Directory.GetCurrentDirectory() + @"\Stripper005.apwz";
        string dhexSimulationFile =
System.IO.Directory.GetCurrentDirectory() + @"\DHEX003.apwz";

#if DEBUG
        Utilities.Report("Beginning to open the Aspen Plus
simulations...");
#endif

#if OPEN_SIMULATIONS_IN_PARALLEL
        HappLS absorberInstance = null;
        HappLS stripperInstance = null;
        HappLS dhexInstance = null;

        Task openAbsorberSimulationTask = new Task(() => {
#if DEBUG
            Utilities.Report("Loading the Aspen Plus simulation for the
absorber.");
#endif
            absorberInstance = new HappLS();
            absorberInstance.InitFromFile2(absorberSimulationFile);
            absorberInstance.SuppressDialogs = 1;
            absorberInstance.Visible = true;
#if DEBUG
            Utilities.Report("Absorber simulation loaded.");
#endif
        });
        openAbsorberSimulationTask.Start();

        Task openStripperSimulationTask = new Task(() =>
        {
#if DEBUG
            Utilities.Report("Loading the Aspen Plus simulation for the
stripper.");
#endif
            stripperInstance = new HappLS();

```

```

        stripperInstance.InitFromFile2(stripperSimulationFile);
        stripperInstance.SuppressDialogs = 1;
        stripperInstance.Visible = true;
#if DEBUG
        Utilities.Report("Stripper simulation loaded.");
#endif
    });
    openStripperSimulationTask.Start();

    Task openDHEXSimulationTask = new Task(() => {
#if DEBUG
        Utilities.Report("Loading the Aspen Plus simulation for the
DHEX.");
#endif
        dhexInstance = new HappLS();
        dhexInstance.InitFromFile2(dhexSimulationFile);
        dhexInstance.SuppressDialogs = 1;
        dhexInstance.Visible = true;
#if DEBUG
        Utilities.Report("DHEX simulation loaded.");
#endif
    });
    openDHEXSimulationTask.Start();

    Task.WhenAll(
        openAbsorberSimulationTask
        , openStripperSimulationTask
        , openDHEXSimulationTask
    ).Wait();
#else
#if DEBUG
        Utilities.Report("Loading the Aspen Plus simulation for the
absorber.");
#endif
        var absorberInstance = new HappLS();
        absorberInstance.InitFromFile2(absorberSimulationFile);

```

```

        absorberInstance.SuppressDialogs = 1;
        absorberInstance.Visible = true;

#if DEBUG
        Utilities.Report("Loading the Aspen Plus simulation for the
stripper.");
#endif

        var stripperInstance = new HappLS();
        stripperInstance.InitFromFile2(stripperSimulationFile);
        stripperInstance.SuppressDialogs = 1;
        stripperInstance.Visible = true;

#if DEBUG
        Utilities.Report("Loading the Aspen Plus simulation for the
DHEX.");
#endif

        var dhexInstance = new HappLS();
        dhexInstance.InitFromFile2(dhexSimulationFile);
        dhexInstance.SuppressDialogs = 1;
        dhexInstance.Visible = true;

#endif

#if DEBUG
        Utilities.Report("All Aspen Plus simulations successfully
loaded.");
#endif

        var parameters = new SimulationParameters_OLD001();
        parameters.AminePortion = 0.054997612;
        parameters.CaptureRate = 0.8;
        parameters.FloodingApproach = 0.6;
        parameters.LeanLoading = 0.055456292;
        parameters.PZPortion = 0.230779782;
        parameters.StripperPressure = 2.0;
        parameters.Absorber_FloodingFactorTimesArea = 1.270717863;
        parameters.Stripper_FloodingFactorTimesArea = 3.0;
        parameters.RegenerationEnergy = 5.37; // GJ/tonne

```

```

parameters.CHEX_TotalArea = 100.0;
parameters.DHEX_DistributedPortion = 0.5;
parameters.Threshold_Area = 1e-7;
parameters.DHEX_SplitPortion = 0.5;
parameters.Absorber_StageCount = 25;
parameters.Stripper_StageCount = 10;
parameters.DHEX_MidFeedHeightPortion = 0.5;

double tolerance = 1e-9;
double toleranceForHeatXTemp = 1e-5;
double temperatureTolerance = 1e-5;
double m_min = 0.0;
double m_max = 10000.0;

Box<int> absorberRunCounterBox = Box<int>.New(0);
Box<int> stripperRunCounterBox = Box<int>.New(0);
Box<int> dhexRunCounterBox = Box<int>.New(0);
Box<double> leanStreamFlowMultiplierBox =
Box<double>.New(83.52321);

#region Flow handlers
#region Flow handlers for the absorber
var flueFlowHandler = ApparentFlows.New();
flueFlowHandler.H2O = 41.80264735;
flueFlowHandler.CO2 = 22.72220602;
flueFlowHandler.N2 = 324.7995852;
flueFlowHandler.O2 = 46.68385665;

var absorberTearStream = ApparentFlows.New();
absorberTearStream.MDEA = 0.001455809;
absorberTearStream.PZ = 0.002057891;
absorberTearStream.H2O = 38.70582925;
absorberTearStream.CO2 = 0.003076571;

var leanFlowHandler = ApparentFlows.New();
#endregion

#region Flow handlers for the stripper

```

```

        Box<double> stripperRichInTemperatureHandler =
Box<double>.New(110.0);
        Box<double> stripperMidInTemperatureHandler =
Box<double>.New(110.0);

var stripperRichInHandler = ApparentFlows.New();
stripperRichInHandler.MDEA = 61.76699888;
stripperRichInHandler.PZ = 18.5306811;
stripperRichInHandler.H2O = 1464.223819;
stripperRichInHandler.CO2 = 22.63649965;

var midIn1Handler = ApparentFlows.New();
midIn1Handler.MDEA = 0.0;
midIn1Handler.PZ = 0.0;
midIn1Handler.H2O = 1e-9;
midIn1Handler.CO2 = 0.0;

var midIn2Handler = ApparentFlows.New();
midIn2Handler.MDEA = 0.0;
midIn2Handler.PZ = 0.0;
midIn2Handler.H2O = 1e-9;
midIn2Handler.CO2 = 0.0;

var stripperRichVaporIn1Handler = ApparentFlows.New();
stripperRichVaporIn1Handler.MDEA = 0.0;
stripperRichVaporIn1Handler.PZ = 0.0;
stripperRichVaporIn1Handler.H2O = 1e-9;
stripperRichVaporIn1Handler.CO2 = 0.0;

var stripperRichVaporIn2Handler = ApparentFlows.New();
stripperRichVaporIn2Handler.MDEA = 0.0;
stripperRichVaporIn2Handler.PZ = 0.0;
stripperRichVaporIn2Handler.H2O = 1e-9;
stripperRichVaporIn2Handler.CO2 = 0.0;

var stripperMidVaporIn1Handler = ApparentFlows.New();
stripperMidVaporIn1Handler.MDEA = 0.0;

```

```

stripperMidVaporIn1Handler.PZ = 0.0;
stripperMidVaporIn1Handler.H2O = 1e-9;
stripperMidVaporIn1Handler.CO2 = 0.0;

var stripperMidVaporIn2Handler = ApparentFlows.New();
stripperMidVaporIn2Handler.MDEA = 0.0;
stripperMidVaporIn2Handler.PZ = 0.0;
stripperMidVaporIn2Handler.H2O = 1e-9;
stripperMidVaporIn2Handler.CO2 = 0.0;

var stripperCondenserTearStream = ApparentFlows.New();
stripperCondenserTearStream.MDEA = 0.0;
stripperCondenserTearStream.PZ = 0.0;
stripperCondenserTearStream.H2O = 0.001;
stripperCondenserTearStream.CO2 = 0.0;

var stripperReboilerTearStream = ApparentFlows.New();
stripperReboilerTearStream.MDEA = 77.24393507;
stripperReboilerTearStream.PZ = 23.18207732;
stripperReboilerTearStream.H2O = 1956.754119;
stripperReboilerTearStream.CO2 = 5.422902021;

#endregion

#region Flow handlers for the DHEX

Box<double> dhexLeanInTemperature = Box<double>.New(119.789);
Box<double> dhexRichInTemperature = Box<double>.New(40.0);
Box<double> dhexLeanTearTemperaturePortion =
Box<double>.New(0.5);

var dhexRichInHandler = ApparentFlows.New();

var dhexLeanInHandler = ApparentFlows.New();
dhexLeanInHandler.MDEA = 61.767;
dhexLeanInHandler.PZ = 18.5302736;
dhexLeanInHandler.H2O = 1463.501423;
dhexLeanInHandler.CO2 = 4.4587616;

```

```

        #endregion

#endregion

#region Absorber

Action runAbsorber = () =>
{
    leanFlowHandler.MDEA = leanStreamFlowMultiplierBox.Value *
(1.0 - parameters.PZPortion);
    leanFlowHandler.PZ = leanStreamFlowMultiplierBox.Value *
parameters.PZPortion;
    leanFlowHandler.H2O = leanStreamFlowMultiplierBox.Value /
parameters.AminePortion;
    leanFlowHandler.CO2 = leanStreamFlowMultiplierBox.Value *
parameters.LeanLoading;

    flueFlowHandler.Assert(absorberInstance, "FLUE-IN");
    absorberTearStream.Assert(absorberInstance, "ABS-FC-2");
    leanFlowHandler.Assert(absorberInstance, "LEAN-IN");

    absorberInstance.Tree.FindNode(@"\Data\Blocks\RICHPUMP\Input\PRES").Value = parameters.StripperPressure;

    DateTime startTime = DateTime.Now;
    absorberInstance.Run2();
    DateTime endTime = DateTime.Now;

    var runTime = endTime - startTime;
#if DEBUG
    Utilities.Report("Absorber\tRun #\t" +
absorberRunCounterBox.Value + "\tRun time (ms):\t" +
runTime.TotalMilliseconds);
#endif
    ++absorberRunCounterBox.Value;

```

```

};

Action absorberWithFloodingFactor = () =>
{
    bool loop = true;

    while (loop)
    {
        var setArea =
parameters.Absorber_FloodingFactorTimesArea / parameters.FloodingApproach;
        var setDiameter = 2.0 * Math.Sqrt(setArea / Math.PI);

        absorberInstance.Tree.FindNode(@"\Data\Blocks\ABSORBER\Input\PR_DIAM\1"
).Value = setDiameter;

        runAbsorber();

        var usedDiameter =
absorberInstance.Tree.FindNode(@"\Data\Blocks\ABSORBER\Input\PR_DIAM\1").Va
lue;
        var usedArea = Math.PI * Math.Pow(usedDiameter / 2.0,
2.0);
        var calculatedFlooding =
absorberInstance.Tree.FindNode(@"\Data\Blocks\ABSORBER\Output\FLOOD_FAC2\1"
).Value;
        parameters.Absorber_FloodingFactorTimesArea = usedArea *
calculatedFlooding;

        var floodingError = calculatedFlooding -
parameters.FloodingApproach;
        loop = Math.Abs(floodingError) >
parameters.FloodingSoftTolerance;
    }
};

Func<double[], double[]> absorberTearStreamFlows = (double[]
inputFlows) =>
{

```

```

        absorberTearStream.SetFromArray(inputFlows);

        absorberWithFloodingFactor();

        absorberTearStream.Read(absorberInstance, @"ABS-FC-1");
        return absorberTearStream.ToArray();
    };

    Action tearAbsorber = () =>
    {
        OurMath.Wegstein_Damped(
            absorberTearStreamFlows
            , (tolerance).Repeat(ApparentFlows.APPARENT_COUNT)
            , 30
            , (m_min).Repeat(ApparentFlows.APPARENT_COUNT)
            , (m_max).Repeat(ApparentFlows.APPARENT_COUNT)
            , absorberTearStream.ToArray()
            , -2.5
            , 2.5
            , 2.0
            , 1.0
        );
    };

    Func<double, double> absorberEffectiveCaptureRateFunction =
(double flowMultiplier) =>
    {
        leanStreamFlowMultiplierBox.Value = flowMultiplier;

        tearAbsorber();

        double mCO2Lost =
absorberInstance.Tree.FindNode(@"\Data\Streams\FLUE-
OUT\Output\MASSFLOW\MIXED\CO2").Value;
        double mCO2Flue =
absorberInstance.Tree.FindNode(@"\Data\Streams\FLUE-
IN\Output\MASSFLOW\MIXED\CO2").Value;

```

```

        double rABS = 1.0 - (mCO2Lost / mCO2Flue);

        return rABS;
    };

    Action runAbsorber_SpecOnCaptureRate = () =>
    {
        OurMath.Secant002(
            parameters.CaptureRate
            , absorberEffectiveCaptureRateFunction
            , 0.0001
            , 30
            , 10.0
            , 500.0
            , 83.52321
            , (double firstResult) => { return
parameters.CaptureRate / firstResult * 83.52321; }
            , 2.0
            , 1.0
        );
    };

#endregion

#region Stripper
Action runStripper = () =>
{
    stripperRichInHandler.Assert(stripperInstance, "RICH-IN");
    midIn1Handler.Assert(stripperInstance, "MID-IN-1");
    midIn2Handler.Assert(stripperInstance, "MID-IN-2");
    stripperRichVaporIn1Handler.Assert(stripperInstance, "RVAP-
IN1");
    stripperRichVaporIn2Handler.Assert(stripperInstance, "RVAP-
IN2");
    stripperMidVaporIn1Handler.Assert(stripperInstance, "MVAP-
IN1");

```

```

        stripperMidVaporIn2Handler.Assert(stripperInstance, "MVAP-
IN2");

        stripperInstance.Tree.FindNode(@"\Data\Streams\RICH-
IN\Input\PRES\MIXED").Value = parameters.StripperPressure;
        stripperInstance.Tree.FindNode(@"\Data\Streams\MID-IN-
1\Input\PRES\MIXED").Value = parameters.StripperPressure;
        stripperInstance.Tree.FindNode(@"\Data\Streams\MID-IN-
2\Input\PRES\MIXED").Value = parameters.StripperPressure;
        stripperInstance.Tree.FindNode(@"\Data\Streams\RVAP-
IN1\Input\PRES\MIXED").Value = parameters.StripperPressure;
        stripperInstance.Tree.FindNode(@"\Data\Streams\RVAP-
IN2\Input\PRES\MIXED").Value = parameters.StripperPressure;
        stripperInstance.Tree.FindNode(@"\Data\Streams\MVAP-
IN1\Input\PRES\MIXED").Value = parameters.StripperPressure;
        stripperInstance.Tree.FindNode(@"\Data\Streams\MVAP-
IN2\Input\PRES\MIXED").Value = parameters.StripperPressure;
        stripperInstance.Tree.FindNode(@"\Data\Streams\STR-FC-
2\Input\PRES\MIXED").Value = parameters.StripperPressure;
        stripperInstance.Tree.FindNode(@"\Data\Streams\STR-FR-
2\Input\PRES\MIXED").Value = parameters.StripperPressure;

        stripperInstance.Tree.FindNode(@"\Data\Blocks\STRIPPER\Input\PRES1").Va
lue = parameters.StripperPressure;
        stripperInstance.Tree.FindNode(@"\Data\Blocks\STR-
CON\Input\PRES").Value = parameters.StripperPressure;
        stripperInstance.Tree.FindNode(@"\Data\Blocks\STR-
REB\Input\PRES").Value = parameters.StripperPressure;

        stripperInstance.Tree.FindNode(@"\Data\Streams\RICH-
IN\Input\TEMP\MIXED").Value = stripperRichInTemperatureHandler.Value;
        stripperInstance.Tree.FindNode(@"\Data\Streams\RVAP-
IN1\Input\TEMP\MIXED").Value = stripperRichInTemperatureHandler.Value;
        stripperInstance.Tree.FindNode(@"\Data\Streams\RVAP-
IN2\Input\TEMP\MIXED").Value = stripperRichInTemperatureHandler.Value;

        stripperInstance.Tree.FindNode(@"\Data\Streams\MID-IN-
1\Input\TEMP\MIXED").Value = stripperMidInTemperatureHandler.Value;

```

```

        stripperInstance.Tree.FindNode(@"\Data\Streams\MID-IN-
2\Input\TEMP\MIXED").Value = stripperMidInTemperatureHandler.Value;
        stripperInstance.Tree.FindNode(@"\Data\Streams\MVAP-
IN1\Input\TEMP\MIXED").Value = stripperMidInTemperatureHandler.Value;
        stripperInstance.Tree.FindNode(@"\Data\Streams\MVAP-
IN2\Input\TEMP\MIXED").Value = stripperMidInTemperatureHandler.Value;

        DateTime startTime = DateTime.Now;
        stripperInstance.Run2();
        DateTime endTime = DateTime.Now;

        var runTime = endTime - startTime;

        var capturedCO2MassFlowRate =
stripperInstance.Tree.FindNode(@"\Data\Streams\CAPT-
OUT\Output\MASSFLOW\MIXED\CO2").Value; // in tonne/hr; between 0 and 1,
ideally at-or-near the capture rate (e.g. 0.8) since the flue gas
specification is such that there's 1 tonne/hr of CO2
        var reboilerDuty_MW =
stripperInstance.Tree.FindNode(@"\Data\Blocks\STR-REB\Output\QCALC").Value;
// in MW; usually on-the-order-of 1.5 MW
        var calculatedRegenerationEnergy_GJ_per_tonne_captured_CO2
= 3.6 * reboilerDuty_MW / capturedCO2MassFlowRate;
        parameters.RegenerationEnergy =
calculatedRegenerationEnergy_GJ_per_tonne_captured_CO2;
#ifdef DEBUG
        Utilities.Report("Stripper\tRun #\t" +
stripperRunCounterBox.Value + "\tRun time (ms):\t" +
runTime.TotalMilliseconds + "\tcapturing\t" + capturedCO2MassFlowRate +
"\ttonne_CO2/hr using\t" + reboilerDuty_MW + "\tMW reboiler duty, so
E_regen/(GJ/tonne)=\t" +
calculatedRegenerationEnergy_GJ_per_tonne_captured_CO2);
#endif

        ++stripperRunCounterBox.Value;
    };

    Action stripperWithFloodingFactor = () =>

```

```

        {
            bool loop = true;

            while (loop)
            {
                var setArea =
parameters.Stripper_FloodingFactorTimesArea / parameters.FloodingApproach;
                var setDiameter = 2.0 * Math.Sqrt(setArea / Math.PI);

                stripperInstance.Tree.FindNode(@"\Data\Blocks\STRIPPER\Input\PR_DIAM\1"
).Value = setDiameter;

                runStripper();

                var usedDiameter =
stripperInstance.Tree.FindNode(@"\Data\Blocks\STRIPPER\Input\PR_DIAM\1").Va
lue;

                var usedArea = Math.PI * Math.Pow(usedDiameter / 2.0,
2.0);

                var calculatedFlooding =
stripperInstance.Tree.FindNode(@"\Data\Blocks\STRIPPER\Output\FLOOD_FAC2\1"
).Value;

                parameters.Stripper_FloodingFactorTimesArea = usedArea *
calculatedFlooding;

                var floodingError = calculatedFlooding -
parameters.FloodingApproach;

                loop = Math.Abs(floodingError) >
parameters.FloodingSoftTolerance;
            }
        };

        Func<double[], double[]>
stripperTearJustReboilerFeedbackFunction = (double[] inputFlows) =>
        {
            var recoveredCondenserArray = new
double[ApparentFlows.APPARENT_COUNT];

```

```

        var recoveredReboilerArray = new
double[ApparentFlows.APPARENT_COUNT];
        for (int i = 0; i < ApparentFlows.APPARENT_COUNT; ++i)
        {
            recoveredCondenserArray[i] = inputFlows[i];
            recoveredReboilerArray[i] = inputFlows[i +
ApparentFlows.APPARENT_COUNT];
        }

        var recoveredCondenserHandler =
ApparentFlows.New(recoveredCondenserArray);
        var recoveredReboilerHandler =
ApparentFlows.New(recoveredReboilerArray);
        recoveredCondenserHandler.Assert(stripperInstance, @"STR-
FC-2");
        recoveredReboilerHandler.Assert(stripperInstance, @"STR-FR-
2");

#if DEBUG
        //Utilities.Report(recoveredCondenserArray);
        //Utilities.Report(recoveredReboilerArray);
#endif

        stripperWithFloodingFactor();

        var condenserOutputHandler =
ApparentFlows.New(stripperInstance, @"STR-FC-1");
        var reboilerOutputHandler =
ApparentFlows.New(stripperInstance, @"STR-FR-1");
        var readCondenserArray = condenserOutputHandler.ToArray();
        var readReboilerArray = reboilerOutputHandler.ToArray();
        var combinedResults = new double[2 *
ApparentFlows.APPARENT_COUNT];
        for (int i = 0; i < ApparentFlows.APPARENT_COUNT; ++i)
        {
            combinedResults[i] = readCondenserArray[i];
            combinedResults[i + ApparentFlows.APPARENT_COUNT] =
readReboilerArray[i];

```

```

    }
    return combinedResults;
};

Action tearStripper = () =>
{
    var stripperCondenserArray =
stripperCondenserTearStream.ToArray();
    var stripperReboilerArray =
stripperReboilerTearStream.ToArray();
    double[] totalArray = new double[2 *
ApparentFlows.APPARENT_COUNT];
    for (int i = 0; i < ApparentFlows.APPARENT_COUNT; ++i)
    {
        totalArray[i] = stripperCondenserArray[i];
        totalArray[i + ApparentFlows.APPARENT_COUNT] =
stripperReboilerArray[i];
    }

    var tearResults = OurMath.Wegstein_Damped(
        stripperTearJustReboilerFeedbackFunction
        , (tolerance).Repeat(2 *
ApparentFlows.APPARENT_COUNT)
        , 30
        , (m_min).Repeat(2 * ApparentFlows.APPARENT_COUNT)
        , (m_max).Repeat(2 * ApparentFlows.APPARENT_COUNT)
        , totalArray
        , -1.0
        , 1.0
        , 1.5
        , 0.75
    );

    var recoveredCondenserArray = new
double[ApparentFlows.APPARENT_COUNT];
    var recoveredReboilerArray = new
double[ApparentFlows.APPARENT_COUNT];
    for (int i = 0; i < ApparentFlows.APPARENT_COUNT; ++i)

```

```

        {
            recoveredCondenserArray[i] = tearResults[i];
            recoveredReboilerArray[i] = tearResults[i +
ApparentFlows.APPARENT_COUNT];
        }

        stripperCondenserTearStream.SetFromArray(recoveredCondenserArray);

        stripperReboilerTearStream.SetFromArray(recoveredReboilerArray);
    };

#if false

        Func<double, double> stripperEffectiveCaptureRateFunction =
(double reboilerDuty) =>
    {
        stripperInstance.Tree.FindNode(@"\Data\Blocks\STR-
REB\Input\DUTY").Value = reboilerDuty;

        tearStripper();

        var mCO2Capt =
stripperInstance.Tree.FindNode(@"\Data\Streams\CAPT-
OUT\Output\MASSFLOW\MIXED\CO2").Value;
        var mCO2Flue =
absorberInstance.Tree.FindNode(@"\Data\Streams\FLUE-
IN\Output\MASSFLOW\MIXED\CO2").Value;

        var rSTR = mCO2Capt / mCO2Flue;
#if DEBUG
        Utilities.Report("Found effective capture rate:\t" + rSTR +
"\tat reboiler duty (MW):\t" + reboilerDuty);
#endif

        return rSTR;
    };

```

```

OurMath.Secant002(
    parameters.CaptureRate
    , stripperEffectiveCaptureRateFunction
    , 0.0001
    , 30
    , 0.25
    , 5.0
    , parameters.CaptureRate * parameters.RegenerationEnergy
/ 3.6
    , (double captureRate) => { return
parameters.CaptureRate / captureRate * parameters.CaptureRate *
parameters.RegenerationEnergy / 3.6; }
    , 2.0
    , 0.05
);
#endif

#endregion

#region DHEX
Action runDHEX = () =>
{
    dhexRichInHandler.Assert(dhexInstance, "RICH-1");
    dhexLeanInHandler.Assert(dhexInstance, "LEAN-1");
    dhexLeanInHandler.Assert(dhexInstance, "LEAN-3");

    dhexInstance.Tree.FindNode(@"\Data\Streams\RICH-
1\Input\PRES\MIXED").Value = parameters.StripperPressure;
    dhexInstance.Tree.FindNode(@"\Data\Streams\LEAN-
1\Input\PRES\MIXED").Value = parameters.StripperPressure;
    dhexInstance.Tree.FindNode(@"\Data\Streams\LEAN-
3\Input\PRES\MIXED").Value = parameters.StripperPressure;

    dhexInstance.Tree.FindNode(@"\Data\Streams\RICH-
1\Input\TEMP\MIXED").Value = dhexRichInTemperature.Value;
    //dhexInstance.Tree.FindNode(@"\Data\Streams\LEAN-
1\Input\TEMP\MIXED").Value = dhexLeanInTemperature.Value; // Switching to
vapor fraction = 0.

```

```

        dhexInstance.Tree.FindNode(@"\Data\Streams\LEAN-
3\Input\TEMP\MIXED").Value = dhexRichInTemperature.Value +
dhexLeanTearTemperaturePortion.Value * (dhexLeanInTemperature.Value -
dhexRichInTemperature.Value);

        dhexInstance.Tree.FindNode(@"\Data\Blocks\SPLTRICH\Input\FRAC\MID-
1").Value = parameters.DHEX_SplitPortion;

        var chexArea = parameters.CHEX_TotalArea * (1.0 -
parameters.DHEX_DistributedPortion);

        dhexInstance.Tree.FindNode(@"\Data\Blocks\CHEXMAIN\Input\AREA").Value =
chexArea;

        dhexInstance.Tree.FindNode(@"\Data\Blocks\CHEXMAIN\Input\BYPASS").Value
= chexArea > parameters.Threshold_Area ? "NO" : "YES";

        var dhexArea = parameters.CHEX_TotalArea *
parameters.DHEX_DistributedPortion;

        dhexInstance.Tree.FindNode(@"\Data\Blocks\CHEXSIDE\Input\AREA").Value =
dhexArea;

        dhexInstance.Tree.FindNode(@"\Data\Blocks\CHEXSIDE\Input\BYPASS").Value
= dhexArea > parameters.Threshold_Area & parameters.DHEX_SplitPortion > 0.0
? "NO" : "YES";

        DateTime startTime = DateTime.Now;
        dhexInstance.Run2();
        DateTime endTime = DateTime.Now;

        var runTime = endTime - startTime;
#if DEBUG
        Utilities.Report("DHEX\tRun #\t" + dhexRunCounterBox.Value
+ "\tRun time (ms):\t" + runTime.TotalMilliseconds + "\twith temperature
portion:\t" + dhexLeanTearTemperaturePortion.Value);
#endif

```

```

        ++dhexRunCounterBox.Value;
    };

    Func<double[], double[]> dhexTemperaturePortionFunction =
(double[] temperaturePortion) =>
    {
        dhexLeanTearTemperaturePortion.Value =
temperaturePortion[0];

        runDHEX();

        var tempLow =
dhexInstance.Tree.FindNode(@"\Data\Streams\RICH-
1\Output\TEMP_OUT\MIXED").Value;
        var tempMid =
dhexInstance.Tree.FindNode(@"\Data\Streams\LEAN-
2\Output\TEMP_OUT\MIXED").Value;
        var tempHigh =
dhexInstance.Tree.FindNode(@"\Data\Streams\LEAN-
1\Output\TEMP_OUT\MIXED").Value;

        var calculatedTemperaturePortion = (tempMid - tempLow) /
(tempHigh - tempLow);

        return new double[] { calculatedTemperaturePortion };
    };

    Action runDHEXComplex = () =>
    {
        OurMath.Wegstein_Damped(
            dhexTemperaturePortionFunction
            , toleranceForHeatXTemp.Repeat(1)
            , 1000
            , (0.0).Repeat(1)
            , (1.0).Repeat(1)
            , (dhexLeanTearTemperaturePortion.Value).Repeat(1)
            , -2.5

```

```

        , 2.5
        , 10.0
        , 1.0
    );
};

#endregion

#region Stripper-DHEX complex

Func<double[], double[]> dhexStripperFeedbackFunction =
(double[] inputFlows) =>
{
    double[] leanOutFlowRates = new
double[ApparentFlows.APPARENT_COUNT];
    for (int i = 0; i < ApparentFlows.APPARENT_COUNT; ++i) {
leanOutFlowRates[i] = inputFlows[i]; }
    dhexLeanInHandler.SetFromArray(leanOutFlowRates);
    double leanOutTemperature =
inputFlows[ApparentFlows.APPARENT_COUNT];
    dhexLeanInTemperature.Value = leanOutTemperature;

    runDHEXComplex();

    stripperRichInTemperatureHandler.Value =
dhexInstance.Tree.FindNode(@"\Data\Streams\RICH-
4\Output\TEMP_OUT\MIXED").Value;
    stripperMidInTemperatureHandler.Value =
dhexInstance.Tree.FindNode(@"\Data\Streams\MID-
2\Output\TEMP_OUT\MIXED").Value;

    var midFeedStage = 1.0 +
(parameters.DHEX_MidFeedHeightPortion *
(double)(parameters.Stripper_StageCount - 1));

    int midFeedStage1;
    int midFeedStage2;
    double midFeedPortion1;

```

```

double midFeedPortion2 = midFeedStage % 1;
if (midFeedPortion2 == 0)
{
    int commonMidFeedStage = (int)midFeedStage;
    midFeedStage1 = commonMidFeedStage;
    midFeedStage2 = commonMidFeedStage;
    midFeedPortion1 = 0.5;
    midFeedPortion2 = 1.0 - midFeedPortion1;
}
else
{
    midFeedStage1 = (int)Math.Floor(midFeedStage);
    midFeedStage2 = midFeedStage1 + 1;

    midFeedPortion1 = 1.0 - midFeedPortion2;
}

stripperRichInHandler.Read(dhexInstance, "RICH-4");

midIn1Handler.Read(dhexInstance, "MID-3");
midIn2Handler.SetFromArray(midIn1Handler.ToArray());
midIn1Handler.Multiply(midFeedPortion1);
midIn2Handler.Multiply(midFeedPortion2);

stripperInstance.Tree.FindNode(@"\Data\Blocks\STRIPPER\Input\FEED_STAGE\MID-IN-1").Value = midFeedStage1;

stripperInstance.Tree.FindNode(@"\Data\Blocks\STRIPPER\Input\FEED_STAGE\MID-IN-2").Value = midFeedStage2;

stripperInstance.Tree.FindNode(@"\Data\Blocks\STRIPPER\Input\FEED_STAGE\RVAP-IN1").Value = 1;

stripperInstance.Tree.FindNode(@"\Data\Blocks\STRIPPER\Input\FEED_STAGE\RVAP-IN2").Value = 1;

stripperInstance.Tree.FindNode(@"\Data\Blocks\STRIPPER\Input\FEED_STAGE\MVAP-IN1").Value = 1;

```

```

        stripperInstance.Tree.FindNode(@"\Data\Blocks\STRIPPER\Input\FEED_STAGE
\MVAP-IN2").Value = 1;

        stripperMidVaporIn1Handler.Read(dhexInstance, "MID-VAP");

        stripperMidVaporIn2Handler.SetFromArray(stripperMidVaporIn1Handler.ToAr
ray());

        stripperMidVaporIn1Handler.Multiply(0.5);
        stripperMidVaporIn2Handler.Multiply(0.5);

        stripperRichVaporIn1Handler.Read(dhexInstance, "RICH-VAP");

        stripperRichVaporIn2Handler.SetFromArray(stripperRichVaporIn1Handler.To
Array());

        stripperRichVaporIn1Handler.Multiply(0.5);
        stripperRichVaporIn2Handler.Multiply(0.5);

        tearStripper();

        dhexLeanInHandler.Read(stripperInstance, "LEAN-OUT");
        //dhexLeanInTemperature.Value =
stripperInstance.Tree.FindNode(@"\Data\Streams\LEAN-
OUT\Output\TEMP_OUT\MIXED").Value; // Switching to vapor fraction = 0.

        var leanInHandlerArray = dhexLeanInHandler.ToArray();
        double[] resultsArray = new
double[ApparentFlows.APPARENT_COUNT + 1];
        for (int i = 0; i < ApparentFlows.APPARENT_COUNT; ++i) {
resultsArray[i] = leanInHandlerArray[i]; }
        resultsArray[ApparentFlows.APPARENT_COUNT] =
dhexLeanInTemperature.Value;

        return resultsArray;
    };

    Action tearDHEXAndStripperComplex = () =>
    {

```

```

        double[] leanOutTolerance = new
double[ApparentFlows.APPARENT_COUNT + 1];
        for (int i = 0; i < ApparentFlows.APPARENT_COUNT; ++i) {
leanOutTolerance[i] = tolerance; }
        leanOutTolerance[ApparentFlows.APPARENT_COUNT] =
temperatureTolerance;

        double[] leanOutLowerBound = new
double[ApparentFlows.APPARENT_COUNT + 1];
        for (int i = 0; i < ApparentFlows.APPARENT_COUNT; ++i) {
leanOutLowerBound[i] = m_min; }
        leanOutLowerBound[ApparentFlows.APPARENT_COUNT] = 70.0;

        double[] leanOutUpperBound = new
double[ApparentFlows.APPARENT_COUNT + 1];
        for (int i = 0; i < ApparentFlows.APPARENT_COUNT; ++i) {
leanOutUpperBound[i] = m_max; }
        leanOutUpperBound[ApparentFlows.APPARENT_COUNT] = 250.0;

        double[] leanOutInitialEstimate = new
double[ApparentFlows.APPARENT_COUNT + 1];
        var leanOutFromApparent = dhexLeanInHandler.ToArray();
        for (int i = 0; i < ApparentFlows.APPARENT_COUNT; ++i) {
leanOutInitialEstimate[i] = leanOutFromApparent[i]; }
        leanOutInitialEstimate[ApparentFlows.APPARENT_COUNT] =
dhexLeanInTemperature.Value;

        OurMath.Wegstein_Damped(
            dhexStripperFeedbackFunction
            , leanOutTolerance
            , 30
            , leanOutLowerBound
            , leanOutUpperBound
            , leanOutInitialEstimate
            , -2.5
            , 2.5
            , 2.0
            , 5.0

```

```

        );
    };

    Func<double, double> stripperEffectiveCaptureRateFunction =
(double reboilerDuty) =>
    {
        stripperInstance.Tree.FindNode(@"\Data\Blocks\STR-
REB\Input\DUTY").Value = reboilerDuty;

        tearDHEXAndStripperComplex();

        var mCO2Capt =
stripperInstance.Tree.FindNode(@"\Data\Streams\CAPT-
OUT\Output\MASSFLOW\MIXED\CO2").Value;
        var mCO2Flue =
absorberInstance.Tree.FindNode(@"\Data\Streams\FLUE-
IN\Output\MASSFLOW\MIXED\CO2").Value;

        var rSTR = mCO2Capt / mCO2Flue;
#if DEBUG
        Utilities.Report("Found effective capture rate:\t" + rSTR +
"\tat reboiler duty (MW):\t" + reboilerDuty);
#endif

        return rSTR;
    };

    Action runDHEXAndStripperComplex_SpecOnCaptureRate = () =>
    {
        OurMath.Secant002(
            parameters.CaptureRate
            , stripperEffectiveCaptureRateFunction
            , 0.0001
            , 30
            , 0.25
            , 5.0

```

```

        , parameters.CaptureRate * parameters.RegenerationEnergy
/ 3.6
        , (double captureRate) => { return
parameters.CaptureRate / captureRate * parameters.CaptureRate *
parameters.RegenerationEnergy / 3.6; }
        , 2.0
        , 0.05
    );
};

#endregion

#region Flowsheet
#if DEBUG
    Utilities.Report("Starting flowsheet convergence.");
#endif
#if DEBUG
    Utilities.Report("Starting to converge the absorber
complex.");
#endif
    runAbsorber_SpecOnCaptureRate();
#if DEBUG
    Utilities.Report("Absorber convergence complete.");
#endif

#if DEBUG
    Utilities.Report("Transferring absorber result for RICH-1 to
DHEX.");
#endif
    dhexRichInHandler.Read(absorberInstance, "RICH-1");
    dhexRichInTemperature.Value =
absorberInstance.Tree.FindNode(@"\Data\Streams\RICH-
1\Output\TEMP_OUT\MIXED").Value;
#if DEBUG
    Utilities.Report("RICH-OUT temp (degC):\t" +
dhexRichInTemperature.Value);
    Utilities.Report(dhexRichInHandler.ToArray());

```

```

#endif
#if DEBUG
        Utilities.Report("Transfer complete.");
#endif

#if DEBUG
        Utilities.Report("Starting to converge the DHEX-and-stripper
complex.");
#endif
        runDHEXAndStripperComplex_SpecOnCaptureRate();
#if DEBUG
        Utilities.Report("DHEX-and-stripper complex convergence
complete.");
#endif

#if DEBUG
        Utilities.Report("Flowsheet convergence complete.");
#endif
        #endregion
    }
    catch (Exception ex)
    {
        Utilities.Report("Exception ended simulation convergence.
Message:");
        Utilities.Report("\\"" + ex.Message + "\".");
    }
#if DEBUG
        Utilities.Report("End : DoSomething009.");
#endif
    }
}
#endif

```

Scheme 6.1. Method for simulating the acid-gas-capture unit. This model runs the absorber, stripper, and DHEX; tears these units together; and adjusts the lean solvent flowrate and stripper reboiler duty to meet the target capture rate.

Appendix A.2. Aspen Plus wrapper

This section contains C# source code for an Aspen Plus wrapper.

```
#define CLOSE_SIMULATIONS_ON_GARBAGE_COLLECT
#define DEBUG__REPORT_SIMULATION_FAILS
#define DEBUG__REPORT_SIMULATION_LOADS
#define DEBUG__REPORT_VALUE_SET_FAILURES
#define DEBUG__REPORT_VALUE_GET_FAILURES

using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using Happ;
using System.Windows;

namespace Heat_Pump
{
    public class AspenSimulationBlock
    {
        protected const bool DEFAULT_VISIBILITY = true;
        protected const int MAX_RUN_ATTEMPTS = 5;
        protected const int MAX_OPEN_ATTEMPTS = 5;
        protected const int MAX_CLOSE_ATTEMPTS = 5;
        protected const int MAX_KILL_ATTEMPTS = 2;
        protected const int MAX_SET_ATTEMPTS = 2;
        protected const int MAX_GET_ATTEMPTS = 2;

        public string FileName {get ; private set; }

        protected HappLS Instance
        {
            get
            {
```

```

        if (this._cachedInstance == null)
        {
            this._cachedInstance = Open(this.FileName);
        }
        if (this.Visible) { this._cachedInstance.Visible = true; }
        return this._cachedInstance;
    }
}
private HappLS _cachedInstance = null;
protected void Internal_ClearDueToError()
{
    Close(this._cachedInstance);
    this._cachedInstance = null;
}

[System.Xml.Serialization.XmlIgnore]
public P<bool> VisibleParameter { get; private set; }
    = P<bool>.New(DEFAULT_VISIBILITY);
private object _visibilityLockObject = new object();
public bool Visible
{
    get
    {
        lock (this._visibilityLockObject)
        {
            return this.VisibleParameter.Value;
        }
    }
    set
    {
        lock (this._visibilityLockObject)
        {
            bool priorValue = VisibleParameter.Value;
            if (value == priorValue) { return; }
            this.VisibleParameter.Value = value;

            var cachedInstance = this._cachedInstance;
            if (cachedInstance == null) { return; }

```

```

        for (int i = 0; i < MAX_SET_ATTEMPTS; ++i)
        {
            try
            {
                cachedInstance.Visible = value;
                return;
            }
            catch
            {
#if DEBUG
#if DEBUG__REPORT_VALUE_SET_FAILURES
                Utilities.Report("Error: Failed to make AspenPlus "
+ (value ? "visible" : "invisible") + " on Attempt #" + i.ToString() + " of
" + MAX_SET_ATTEMPTS.ToString() + ".");
#endif
#endif
            }
        }

        throw new Exception("Error: Failed all " +
MAX_SET_ATTEMPTS.ToString() + " attempts to make AspenPlus " + (value ?
"visible" : "invisible") + ".");
    }
}

    public AspenSimulationBlock() { }
    ~AspenSimulationBlock()
    {
#if CLOSE_SIMULATIONS_ON_GARBAGE_COLLECT
        Close(this._cachedInstance);
#endif
    }

    public void SetValue<T>(string path, T value)
    {
        for (int i=0; i < MAX_SET_ATTEMPTS; ++i)
        {

```

```

        try
        {
            this.Instance.Tree.FindNode(path).Value = value;
            return;
        }
        catch (Exception exception)
        {
            #if DEBUG
            #if DEBUG___REPORT_VALUE_SET_FAILURES
                Utilities.Report(
                    "Error: Failed to set AspenPlus value (" +
                    (typeof(T).ToString()) + "\\\"" + value.ToString()
                    + "\" to path \"" + path + "\" on Attempt #" +
                    i.ToString() + " of " + MAX_SET_ATTEMPTS.ToString() + "."
                    + Environment.NewLine + "\tException message:"
                    + Environment.NewLine + "\t\t\"" + exception.Message
                    + "\"."
                    );
            #endif
            #endif
        }
    }

    throw new Exception("Error: Failed to set Node(\"" + path + "\"
to " + value + ".");
}
public T GetValue<T>(string path)
{
    for (int i = 0; i < MAX_GET_ATTEMPTS; ++i)
    {
        try
        {
            var result = this.Instance.Tree.FindNode(path).Value;

            if (result == null)
            {
                var returnType = typeof(T);

```

```

        if (returnType.Equals(typeof(double)))
        {
            result = double.NaN;
        }
        else if (returnType.Equals(typeof(string)))
        {
            result = "";
        }
        else
        {
            result = default(T);
        }
    }

    return result;
}
catch (Exception exception)
{
#if DEBUG
#if DEBUG__REPORT_VALUE_GET_FAILURES
        Utilities.Report(
            "Error: Failed to get AspenPlus value of Type \"" +
            (typeof(T).ToString()) + "\"
            + " from path \"" + path + "\" on Attempt #" +
            i.ToString() + " of " + MAX_SET_ATTEMPTS.ToString() + "."
            + Environment.NewLine + "\tException message:"
            + Environment.NewLine + "\t\t\"" + exception.Message
            + "\".");
        );
#endif
#endif
    }

    throw new Exception("Error: Failed to get value for Node(\"" +
    path + "\").");
}

```

```

protected static void Close(HappLS instance)
{
    if (instance == null) { return; }

    for (int i=0; i < MAX_CLOSE_ATTEMPTS; ++i)
    {
        try
        {
            instance.Close();
            return;
        }
        catch { }
    }

    // Failing to close the simulation, so just kill it instead.
    KillProcess(instance);
}

protected static HappLS Open(string fullFileName)
{
    bool success = false;
    HappLS newInstance = null;

    for (int i=0; !success & i < MAX_OPEN_ATTEMPTS; ++i)
    {
        try
        {
            newInstance = new HappLS();
            newInstance.InitFromFile2(fullFileName);
            newInstance.SuppressDialogs = 1;
            success = true;
        }
        catch
        {
            KillProcess(newInstance);
        }
    }
}

```

```

        if (!success) { throw new Exception(
            "Error: Failed to open the simulation file \""
            + fullFileName + "\".");
        }; }

#if DEBUG
#if DEBUG__REPORT_SIMULATION_LOADS
    Utilities.Report("Successfully loaded AspenPlus simulation \"" +
fullFileName + "\".");
#endif
#endif

    return newInstance;
}

protected static void KillProcess(HappLS instance)
{
    if (instance == null) { return; }

    bool success = false;

    for (int i = 0; !success & i < MAX_KILL_ATTEMPTS; ++i)
    {
        try
        {
            var process = System.Diagnostics.Process.GetProcessById(
                instance.ProcessId);
            process.Kill();
            return;
        }
        catch { }

        try
        {
            var potentiallyExitedProcess =

System.Diagnostics.Process.GetProcessById(instance.ProcessId);
            if (potentiallyExitedProcess.HasExited) { return; }

```

```

        }
        catch { }
    }
}

public static AspenSimulationBlock New(string fullFileName)
{
    var toReturn = new AspenSimulationBlock();

    toReturn.FileName = fullFileName;
    //toReturn.Open(fullFileName);

    return toReturn;
}

public static AspenSimulationBlock NewLocalSimulation(string
fileName)
{
    return New(System.IO.Directory.GetCurrentDirectory() + @"\" +
fileName);
}

public bool Run()
{
    bool runSuccess = false;

    for (int i=0; i < MAX_RUN_ATTEMPTS; ++i)
    {

        try
        {
            instance.Run2();
            runSuccess = true;
            break;
        }
        catch (Exception exception)
        {
            #if DEBUG
            #if DEBUG__REPORT_SIMULATION_FAILS
                Utilities.Report(

```

```

        "Error:  An AspenPlus instance failed on running
(Attempt #"
        + (runIndex + 1) + " of " + MAX_RUN_ATTEMPTS + ".)"
        + Environment.NewLine + "Exception:\t"
        + Environment.NewLine + "\t" + exception.Message
    );
#endif
#endif
        this.Internal_ClearDueToError();
    }
}
return runSuccess;
}
}
}
}
}

```

Scheme 6.2. Source code for a basic Aspen Plus wrapper.

Appendix A.3. CO₂-capture simulation using the framework

This section contains C# source code for the CO₂-capture simulation using the framework discussed in [?].

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using Happ;

namespace Heat_Pump
{
    partial class DoSomething
    {
        public static void DoSomething003()
        {

#if DEBUG
            DateTime runStartTime = DateTime.Now;
            Utilities.Report("Starting DoSomething003().\t" +
runStartTime.ToString());
#endif

            var p = SP.New();

            p.DeclareInternalElements(InternalElement.Types.MaterialStream
                // Absorber:
                , "ABSORBER-FLUE-IN"
                , "ABSORBER-FLUE-OUT"
                , "ABSORBER-LEAN-IN"
                , "ABSORBER-FROM-CONDENSER"
                , "ABSORBER-RICH-OUT"
```

```

// Rich pump:
, "RICH-PUMP-OUT"

// DHEX:
, "DHEX-TEAR"

//,   "DHEX-RICH-MAIN-LIQUID"
, "DHEX-RICH-MAIN-VAPOR"

, "DHEX-RICH-DISTRIBUTED-LIQUID"
, "DHEX-RICH-DISTRIBUTED-VAPOR"

// Stripper:
, "STRIPPER-CAPTURE-STREAM"
, "STRIPPER-LEAN-OUT"

, "STRIPPER-FROM-CONDENSER"
, "STRIPPER-TO-CONDENSER"

, "STRIPPER-FROM-REBOILER"
, "STRIPPER-TO-REBOILER"

, "STRIPPER-RICH-MAIN-LIQUID-IN"
, "STRIPPER-RICH-MAIN-VAPOR-IN-1"
, "STRIPPER-RICH-MAIN-VAPOR-IN-2"

, "STRIPPER-RICH-DISTRIBUTED-LIQUID-IN-1"
, "STRIPPER-RICH-DISTRIBUTED-LIQUID-IN-2"
, "STRIPPER-RICH-DISTRIBUTED-VAPOR-IN-1"
, "STRIPPER-RICH-DISTRIBUTED-VAPOR-IN-2"
);
p.DeclareInternalElements (InternalElement.Types.RadFrac
, "ABSORBER"
, "STRIPPER"
);
p.DeclareInternalElements (InternalElement.Types.Flash2
, "ABSORBER-CONDENSER"

```

```

        , "STRIPPER-CONDENSER"
        , "STRIPPER-REBOILER"
        , "DHEX-FLASH-MAIN"
        , "DHEX-FLASH-DISTRIBUTED"
    );
    p.DeclareInternalElements (InternalElement.Types.FSplit
        , "CHEX-DISTRIBUTOR"
    );
    p.DeclareInternalElements (InternalElement.Types.Heater
    );
    p.DeclareInternalElements (InternalElement.Types.HeatX
        , "CHEX-MAIN"
        , "CHEX-DISTRIBUTED"
    );

    p.DeclareInternalElements (InternalElement.Types.Pump
        , "RICH-PUMP"
    );

    var absorberSimulation =
    AspenSimulationBlock.NewLocalSimulation ("Absorber.apwz");
    {
        absorberSimulation.Visible = true;

        // Streams
        {
            absorberSimulation.Define ("ABSORBER-FLUE-IN", "FLUE-IN");
            absorberSimulation.Define ("ABSORBER-FLUE-OUT", null, "FLUE-
OUT");
            absorberSimulation.Define ("ABSORBER-LEAN-IN", "LEAN-IN");
            absorberSimulation.Define ("ABSORBER-FROM-CONDENSER", "ABS-
FC-2", "ABS-FC-1");
            absorberSimulation.Define ("ABSORBER-RICH-OUT", null, "RICH-
OUT");
            absorberSimulation.Define ("RICH-PUMP-OUT", null, "RICH-1");
        }
    }

```

```

        // Blocks
        {
            absorberSimulation.Define("ABSORBER");
            absorberSimulation.Define("ABSORBER-CONDENSER", "ABS-CON");
            absorberSimulation.Define("RICH-PUMP", "RICHPUMP");

        }
    }

    var dHEXSimulation =
AspenSimulationBlock.NewLocalSimulation("DHEX.apwz");
    {
        dHEXSimulation.Visible = true;

        // Streams
        {
            dHEXSimulation.Define("RICH-PUMP-OUT", "RICH-1");
            dHEXSimulation.Define("STRIPPER-LEAN-OUT", "LEAN-1");
            dHEXSimulation.Define("DHEX-RICH-MAIN-VAPOR", null, "RICH-
VAP");
            dHEXSimulation.Define("DHEX-RICH-DISTRIBUTED-LIQUID", null,
"MID-3");
            dHEXSimulation.Define("DHEX-RICH-DISTRIBUTED-VAPOR", null,
"MID-VAP");
            dHEXSimulation.Define("STRIPPER-RICH-MAIN-LIQUID-IN", null,
"RICH-4");
            dHEXSimulation.Define("DHEX-TEAR", "LEAN-3", "LEAN-2");
        }

        // Blocks
        {
            dHEXSimulation.Define("CHEX-MAIN", "CHEXMAIN");
            dHEXSimulation.Define("CHEX-DISTRIBUTED", "CHEXSIDE");
            dHEXSimulation.Define("CHEX-DISTRIBUTOR", "SPLTRICH");
            dHEXSimulation.Define("DHEX-FLASH-MAIN", "FLSHRICH");
            dHEXSimulation.Define("DHEX-FLASH-DISTRIBUTED",
"FLASHMID");
        }
    }

```

```

    }

    var stripperSimulation =
AspenSimulationBlock.NewLocalSimulation("Stripper.apwz");
    {
        stripperSimulation.Visible = true;

        // Streams
        {
            stripperSimulation.Define("STRIPPER-CAPTURE-STREAM", null,
"CAPT-OUT");
            stripperSimulation.Define("STRIPPER-LEAN-OUT", null, "LEAN-
OUT");

            stripperSimulation.Define("STRIPPER-RICH-MAIN-LIQUID-IN",
"RICH-IN");
            stripperSimulation.Define("STRIPPER-RICH-MAIN-VAPOR-IN-1",
"RVAP-IN1");
            stripperSimulation.Define("STRIPPER-RICH-MAIN-VAPOR-IN-2",
"RVAP-IN2");

            stripperSimulation.Define("STRIPPER-RICH-DISTRIBUTED-
LIQUID-IN-1", "MID-IN-1");
            stripperSimulation.Define("STRIPPER-RICH-DISTRIBUTED-
LIQUID-IN-2", "MID-IN-2");
            stripperSimulation.Define("STRIPPER-RICH-DISTRIBUTED-VAPOR-
IN-1", "MVAP-IN1");
            stripperSimulation.Define("STRIPPER-RICH-DISTRIBUTED-VAPOR-
IN-2", "MVAP-IN2");

            stripperSimulation.Define("STRIPPER-FROM-CONDENSER", "STR-
FC-2", "STR-FC-1");
            stripperSimulation.Define("STRIPPER-TO-CONDENSER", null,
"STR-TCON");

            stripperSimulation.Define("STRIPPER-FROM-REBOILER", "STR-
FR-2", "STR-FR-1");

```

```

        stripperSimulation.Define("STRIPPER-TO-REBOILER", null,
"STR-TREB");
    }

    // Blocks
    {
        stripperSimulation.Define("STRIPPER");
        stripperSimulation.Define("STRIPPER-CONDENSER", "STR-CON");
        stripperSimulation.Define("STRIPPER-REBOILER", "STR-REB");
    }
}

FlowsheetBlock absorber = absorberSimulation;
FlowsheetBlock stripper = stripperSimulation;
FlowsheetBlock dhex = dHEXSimulation;

absorber = FlowsheetBlock.Sequence(
    CalculatorBlock.New((SP parameters) =>
    {
        var absorberRadFrac =
parameters.ExceptGetInternalElement<RadFracInternalElement>("ABSORBER");

        absorberRadFrac.PressureParameter.DefineAs(parameters.Absorber_Pressure
[parameters]);

        absorberRadFrac.PackingHeightParameter.DefineAs(parameters.Absorber_Pac
kingHeight[parameters]);

        absorberRadFrac.StageCountParameter.DefineAs(parameters.Absorber_StageC
ount[parameters]);

    })
    , CalculatorBlock.New((SP parameters) =>
    {

```

```

        var absorberCondenser =
parameters.ExceptGetInternalElement<Flash2InternalElement>("ABSORBER-
CONDENSER");

        absorberCondenser.TemperatureParameter.DefineAs(parameters.Absorber_Con
denser_Temperature[parameters]);

        absorberCondenser.PressureParameter.DefineAs(parameters.Absorber_Pressu
re[parameters]);
    })
    , CalculatorBlock.New((SP parameters) =>
    {
        var richSolventPump =
parameters.ExceptGetInternalElement<PumpInternalElement>("RICH-PUMP");

        richSolventPump.OutletPressureParameter.DefineAs(parameters.Stripper_Pr
essure[parameters]);
    })
    , CalculatorBlock.New((SP parameters) =>
    {
        var flueInStream =
parameters.ExceptGetInternalElement<MaterialStream>("ABSORBER-FLUE-IN");

        flueInStream.Temperature.DefineAs(parameters.Flue_Temperature[parameter
s]);

        flueInStream.Pressure.DefineAs(parameters.Flue_Pressure[parameters]);

        var flueCO2MoleFlowRate =
parameters.Flue_CO2MoleFlowRate[parameters];
        var flueInertPortion =
parameters.Flue_InertPortionOfDryFlue[parameters];
        var flueNitrogenPortionOfInert =
parameters.Flue_NitrogenPortionOfInertDryFlue[parameters];

```

```

        var flueWaterMultipleOfDryFlue =
parameters.Flue_WaterAsMultipleOfDryFlue[parameters];

        var inertMoleFlowRate = flueInertPortion *
flueCO2MoleFlowRate / (1.0 - flueInertPortion);

        var n_CO2 = flueCO2MoleFlowRate;
        var n_N2 = flueNitrogenPortionOfInert *
inertMoleFlowRate;
        var n_O2 = inertMoleFlowRate - n_N2;
        var n_H2O = flueWaterMultipleOfDryFlue * (n_CO2 + n_N2 +
n_O2);

        flueInStream.CO2.DefineAs(n_CO2);
        flueInStream.H2O.DefineAs(n_H2O);
        flueInStream.N2.DefineAs(n_N2);
        flueInStream.O2.DefineAs(n_O2);
    })
, CalculatorBlock.New((SP parameters) =>
{
    var leanInStream =
parameters.ExceptGetInternalElement<MaterialStream>("ABSORBER-LEAN-IN");

    leanInStream.Temperature.DefineAs(parameters.LSC_OutputTemperature[para
meters]);

    leanInStream.Pressure.DefineAs(parameters.Absorber_Pressure[parameters]
);

    var amineMoleFlowRate =
parameters.LeanSolvent_AmineMoleFlowRate[parameters];
    var aminePortionOfUnloadedLeanSolvent =
parameters.LeanSolvent_AminePortion[parameters];
    var pzPortionOfAmine =
parameters.LeanSolvent_PZPortion[parameters];

```

```

        var leanSolventLoading =
parameters.LeanSolvent_Loading[parameters];
        leanInStream.PZ.DefineAs(amineMoleFlowRate *
pzPortionOfAmine);
        leanInStream.MDEA.DefineAs(amineMoleFlowRate * (1.0 -
pzPortionOfAmine));
        leanInStream.H2O.DefineAs((1.0 -
aminePortionOfUnloadedLeanSolvent) * amineMoleFlowRate /
aminePortionOfUnloadedLeanSolvent);
        leanInStream.CO2.DefineAs(amineMoleFlowRate *
leanSolventLoading);
    })
    , absorber

);

#if RELAX_TOLERANCES_FOR_QUICK_RUNS
#else
    absorber = WegsteinConvergenceBlock.New(
        absorber
        , P<ConvergenceParameter[]>.New((SP parameters) => {
            var absorberFromCondenserStream =
parameters.ExceptGetInternalElement<MaterialStream>("ABSORBER-FROM-
CONDENSER");
            return
absorberFromCondenserStream.GetConvergenceParameters(parameters, false,
false, true);
        })
    );
#endif

#if RELAX_TOLERANCES_FOR_QUICK_RUNS
#else
    absorber = RadFracWithFloodingControlBlock.New(
        absorber
        , P<double>.New(1.25)
        , P<string>.New("ABSORBER")
    );

```

```

#endif

    var absorberTransform = OurMath.Transform.Square;
    absorber = DesignSpecBlock.New(
        absorber
        , ConvergenceParameter.New(
            p.LeanSolvent_AmineMoleFlowRate
#if RELAX_TOLERANCES_FOR_QUICK_RUNS
            , 0.25
#else
            ,
p.Convergence_Tolerance_Absolute_Default_CaptureRate
#endif
            ,
p.Convergence_Tolerance_Relative_Default_CaptureRate
            , P<double>.New((SP parameters) => { return
parameters.Flue_CO2MoleFlowRate[parameters] *
parameters.CaptureRate[parameters] /
(Limitations.Chemistry.MaximumPossibleLoadingInMolesCO2PerMoleAmine -
parameters.LeanSolvent_Loading[parameters]); })
            , P<double>.New((SP parameters) => { return
parameters.Flue_CO2MoleFlowRate[parameters] *
parameters.CaptureRate[parameters] /
Limitations.Chemistry.MinimumPossibleDeltaLoading; })
            ,
p.Convergence_DampingThreshold_Default_MoleFlowRate
            , p.Convergence_DampingFactor_Default_MoleFlowRate
        )
        , P<double>.New((SP parameters) => {
            var flueOutStream =
parameters.ExceptGetInternalElement<MaterialStream>("ABSORBER-FLUE-OUT");
            var r_ABS = 1.0 - (flueOutStream.CO2[parameters] /
parameters.Flue_CO2MoleFlowRate[parameters]);
#if DEBUG

            Utilities.Report(p.LeanSolvent_AmineMoleFlowRate[parameters] + "\t" +
r_ABS);
#endif
        })
    )
#endif

```

```

        return r_ABS;
    })
    , P<double>.New((SP parameters) => { return
parameters.CaptureRate[parameters]; })
    , (SP parameters, double x_0, double f_0) => { return x_0 *
absorberTransform[parameters.CaptureRate[parameters]] / f_0; } // !!!
Modify to use the transform at zero. Not an issue for linear, square, etc.
transforms where the transform of zero is zero.
    , P<OurMath.Transform>.New(absorberTransform)
);

stripper = FlowsheetBlock.Sequence(
    CalculatorBlock.New((SP parameters) =>
    {
        var stripperRadFrac =
parameters.ExceptGetInternalElement<RadFracInternalElement>("STRIPPER");

        stripperRadFrac.PressureParameter.DefineAs(parameters.Stripper_Pressure
[parameters]);

        stripperRadFrac.PackingHeightParameter.DefineAs(parameters.Stripper_Pac
kingHeight[parameters]);

        stripperRadFrac.StageCountParameter.DefineAs(parameters.Stripper_StageC
ount[parameters]);
    })
    , CalculatorBlock.New((SP parameters) =>
    {
        var stripperCondenser =
parameters.ExceptGetInternalElement<Flash2InternalElement>("STRIPPER-
CONDENSER");

        stripperCondenser.TemperatureParameter.DefineAs(parameters.Stripper_Con
denser_Temperature[parameters]);
    })
);

```

```

        stripperCondenser.PressureParameter.DefineAs(parameters.Stripper_Pressure[parameters]);
    })
    , CalculatorBlock.New((SP parameters) =>
    {
        var stripperReboiler =
parameters.ExceptGetInternalElement<Flash2InternalElement>("STRIPPER-
REBOILER");

        stripperReboiler.PressureParameter.DefineAs(parameters.Stripper_Pressure[parameters]);
    })
    , MaterialSplitterBlock.New(
        "DHEX-RICH-MAIN-VAPOR"
        , "STRIPPER-RICH-MAIN-VAPOR-IN-1"
        , "STRIPPER-RICH-MAIN-VAPOR-IN-2"
        , P<double>.New((SP parameters) => { var split =
(parameters.DHEX_HeightPortion_Main_Vapor[parameters] *
(double)(parameters.Stripper_StageCount[parameters] - 1)) % 1; return split
== 0 ? 0.5 : split; })
        )
    , MaterialSplitterBlock.New(
        "DHEX-RICH-DISTRIBUTED-VAPOR"
        , "STRIPPER-RICH-DISTRIBUTED-VAPOR-IN-1"
        , "STRIPPER-RICH-DISTRIBUTED-VAPOR-IN-2"
        , P<double>.New((SP parameters) => { var split =
(parameters.DHEX_HeightPortion_Distributed_Vapor[parameters] *
(double)(parameters.Stripper_StageCount[parameters] - 1)) % 1; return split
== 0 ? 0.5 : split; })
        )
    , MaterialSplitterBlock.New(
        "DHEX-RICH-DISTRIBUTED-LIQUID"
        , "STRIPPER-RICH-DISTRIBUTED-LIQUID-IN-1"
        , "STRIPPER-RICH-DISTRIBUTED-LIQUID-IN-2"
        , P<double>.New((SP parameters) => { var split =
(parameters.DHEX_HeightPortion_Distributed_Liquid[parameters] *

```

```

(double)(parameters.Stripper_StageCount[parameters] - 1)) % 1; return split
== 0 ? 0.5 : split; })
    )
#if false
    , CalculatorBlock.New((SP parameters) =>
    {
        var stripperStageCount =
parameters.Stripper_StageCount[parameters];

        var mainVaporStage = 1 +
(parameters.DHEX_HeightPortion_Main_Vapor[parameters] *
(double)(stripperStageCount - 1));
        var distVaporStage = 1 +
(parameters.DHEX_HeightPortion_Distributed_Vapor[parameters] *
(double)(stripperStageCount - 1));
        var distLiquidStage = 1 +
(parameters.DHEX_HeightPortion_Distributed_Liquid[parameters] *
(double)(stripperStageCount - 1));

        var mainVapor =
parameters.ExceptGetInternalElement<MaterialStream>("DHEX-RICH-MAIN-
VAPOR");
        var mainVaporSplit1 =
parameters.ExceptGetInternalElement<MaterialStream>("STRIPPER-RICH-MAIN-
VAPOR-IN-1");
        var mainVaporSplit2 =
parameters.ExceptGetInternalElement<MaterialStream>("STRIPPER-RICH-MAIN-
VAPOR-IN-2");

        mainVaporSplit1.Temperature.DefineAs(mainVapor.Temperature[parameters])
;

        mainVaporSplit2.Temperature.DefineAs(mainVapor.Temperature[parameters])
;

        mainVaporSplit1.Pressure.DefineAs(mainVapor.Pressure[parameters]);

```

```

mainVaporSplit2.Pressure.DefineAs (mainVapor.Pressure [parameters]);

mainVaporSplit1.VaporFraction.DefineAs (mainVapor.VaporFraction [parameters]);

mainVaporSplit2.VaporFraction.DefineAs (mainVapor.VaporFraction [parameters]);

var distVapor =
parameters.ExceptGetInternalElement<MaterialStream> ("DHEX-RICH-DISTRIBUTED-
VAPOR");

var distVaporSplit1 =
parameters.ExceptGetInternalElement<MaterialStream> ("STRIPPER-RICH-
DISTRIBUTED-VAPOR-IN-1");

var distVaporSplit2 =
parameters.ExceptGetInternalElement<MaterialStream> ("STRIPPER-RICH-
DISTRIBUTED-VAPOR-IN-2");

distVaporSplit1.Temperature.DefineAs (distVapor.Temperature [parameters])
;

distVaporSplit2.Temperature.DefineAs (distVapor.Temperature [parameters])
;

distVaporSplit1.Pressure.DefineAs (distVapor.Pressure [parameters]);

distVaporSplit2.Pressure.DefineAs (distVapor.Pressure [parameters]);
    })
#endif

, stripper
);

stripper = RadFracWithFloodingControlBlock.New(
    stripper
, 3.0

```

```

        , "STRIPPER"
    );

    dhex = FlowsheetBlock.Sequence(
        CalculatorBlock.New((SP parameters) =>
            {
                var areaSplitPortion =
parameters.DHEX_DistributedPortion[parameters];
                var totalArea = parameters.CHEX_TotalArea[parameters];

                var chexMain =
parameters.ExceptGetInternalElement<HeatXInternalElement>("CHEX-MAIN");
                chexMain.AreaParameter.DefineAs((1.0 - areaSplitPortion)
* totalArea);

                var chexDist =
parameters.ExceptGetInternalElement<HeatXInternalElement>("CHEX-
DISTRIBUTED");
                chexDist.AreaParameter.DefineAs(areaSplitPortion *
totalArea);
            })
        , CalculatorBlock.New((SP parameters) =>
            {
                var dhexSplitter =
parameters.ExceptGetInternalElement<FSplitInternalElement>("CHEX-
DISTRIBUTOR");
                dhexSplitter.SplitFractionsParameter.DefineAs((SP
parameters2) =>
                    {
                        return new Tuple<string, double>[] { new
Tuple<string, double>("MID-1", parameters2.DHEX_SplitPortion[parameters2])
};
                    })
            })
        , CalculatorBlock.New((SP parameters) =>
            {
                var stripperPressure =
parameters.Stripper_Pressure[parameters];

```

```

        var mainFlash2 =
parameters.ExceptGetInternalElement<Flash2InternalElement>("DHEX-FLASH-
MAIN");

        var distFlash2 =
parameters.ExceptGetInternalElement<Flash2InternalElement>("DHEX-FLASH-
DISTRIBUTED");

        mainFlash2.DutyParameter.DefineAs(0);
        distFlash2.DutyParameter.DefineAs(0);

        mainFlash2.PressureParameter.DefineAs(stripperPressure);
        distFlash2.PressureParameter.DefineAs(stripperPressure);
    })
, CalculatorBlock.New((SP parameters) =>
{
    var tearStream =
parameters.ExceptGetInternalElement<MaterialStream>("DHEX-TEAR");
    var temperaturePortion =
parameters.DHEX_Tear_TemperaturePortion[parameters];
    var t_COLD_IN =
parameters.ExceptGetInternalElement<MaterialStream>("RICH-PUMP-
OUT").Temperature[parameters];
    var t_HOT_IN =
parameters.ExceptGetInternalElement<MaterialStream>("STRIPPER-LEAN-
OUT").Temperature[parameters];

    tearStream.Temperature.DefineAs(t_COLD_IN +
(parametersPortion * (t_HOT_IN - t_COLD_IN)));
    })
, dhex
, CalculatorBlock.New((SP parameters) =>
{
    var t_TEAR =
parameters.ExceptGetInternalElement<MaterialStream>("DHEX-
TEAR").Temperature[parameters];

```

```

        var t_COLD_IN =
parameters.ExceptGetInternalElement<MaterialStream>("RICH-PUMP-
OUT").Temperature[parameters];
        var t_HOT_IN =
parameters.ExceptGetInternalElement<MaterialStream>("STRIPPER-LEAN-
OUT").Temperature[parameters];

        parameters.DHEX_Tear_TemperaturePortion.DefineAs((t_TEAR
- t_COLD_IN) / (t_HOT_IN - t_COLD_IN));
        })
    );
    dhex = WegsteinConvergenceBlock.New(
        dhex
        , ConvergenceParameter.New(
            p.DHEX_Tear_TemperaturePortion
            , 1e-5
            , 1e-5
            , 0.01
            , (1.0 - 0.01)
            , 0.1
            , 2.5
        )
    );

    stripper = WegsteinConvergenceBlock.New(
        stripper
        , P<ConvergenceParameter[]>.New((SP parameters) => {
            var stripperFromCondenserStream =
parameters.ExceptGetInternalElement<MaterialStream>("STRIPPER-FROM-
CONDENSER");
            var stripperFromReboilerStream =
parameters.ExceptGetInternalElement<MaterialStream>("STRIPPER-FROM-
REBOILER");

            var combinedList = new List<ConvergenceParameter>();

            combinedList.AddRange(stripperFromCondenserStream.GetConvergenceParamet
ers(parameters, false, false, true));

```

```

        combinedList.AddRange(stripperFromReboilerStream.GetConvergenceParameters(parameters, false, false, true));

        return combinedList.ToArray();
    })
);

stripper = SequenceBlock.New(
    CalculatorBlock.New((SP parameters) =>
    {
        var stripperReboiler =
parameters.ExceptGetInternalElement<Flash2InternalElement>("STRIPPER-
REBOILER");

        var n_flueCO2 =
parameters.Flue_CO2MoleFlowRate[parameters];
        var R_target = parameters.CaptureRate[parameters];
        var n_captCO2 = R_target * n_flueCO2;
        var m_captCO2 = n_captCO2 * MolecularMass.CO2 / 1000.0;
// ! Unit conversions in this line.
        var E_regen = parameters.RegenerationEnergy[parameters];
        var D_reboiler = E_regen * m_captCO2 / 3.6; // ! Unit
conversions in this line.

        stripperReboiler.DutyParameter.DefineAs(D_reboiler);
    })
, stripper
, CalculatorBlock.New((SP parameters) =>
{
    var stripperReboiler =
parameters.ExceptGetInternalElement<Flash2InternalElement>("STRIPPER-
REBOILER");

    var captureStream =
parameters.ExceptGetInternalElement<MaterialStream>("STRIPPER-CAPTURE-
STREAM");

```

```

        var D_reboiler =
stripperReboiler.DutyParameter[parameters];
        var n_captCO2 = captureStream.CO2[parameters];
        var m_captCO2 = n_captCO2 * MolecularMass.CO2 / 1000.0;
// ! Unit conversions in this line.
        var E_regen = 3.6 * D_reboiler / m_captCO2; // ! Unit
conversions in this line.

        parameters.RegenerationEnergy.DefineAs(E_regen);
    })
);

stripper = SequenceBlock.New(
    dhex
    , stripper
);

stripper = WegsteinConvergenceBlock.New(
    stripper
    , P<ConvergenceParameter[]>.New((SP parameters) => {
        var leanSolventStream =
parameters.ExceptGetInternalElement<MaterialStream>("STRIPPER-LEAN-OUT");

        // !! Experimenting with combining the DHEX-STR
tear and the R_STR design spec.
        var combinedList = new List<ConvergenceParameter>();

        combinedList.AddRange(leanSolventStream.GetConvergenceParameters(parame
ters, false, false, true));
        combinedList.Add(ConvergenceParameter.New(
            parameters.RegenerationEnergy
            , 1e-5
            ,
parameters.Convergence_Tolerance_Relative_Default_CaptureRate
            , 1
            , 10
            , 0.1
            , 1.1

```

```

        )
    );

    return combinedList.ToArray();
})
);

var captureUnit = SequenceBlock.New(
    absorber
    , CalculatorBlock.New((SP parameters) =>
    {
        // Estimate the lean solvent feed into the DHEX based
on the absorber's solvent.
        var pumpedRichStream =
parameters.ExceptGetInternalElement<MaterialStream>("RICH-PUMP-OUT");
        var leanInStream =
parameters.ExceptGetInternalElement<MaterialStream>("STRIPPER-LEAN-OUT");
        var dhexTearStream =
parameters.ExceptGetInternalElement<MaterialStream>("DHEX-TEAR");
        pumpedRichStream.TransferTo(parameters, leanInStream);

        leanInStream.Pressure.DefineAs(parameters.Stripper_Pressure[parameters]
);

        var t_RICH_IN =
pumpedRichStream.Temperature[parameters];
        var t_LEAN_IN = Math.Max(t_RICH_IN + 5.0, 101.7777777);
// Guess a lean solvent temperature of 100degC, with a precaution for hot
rich solvent streams.
        leanInStream.Temperature.DefineAs(t_LEAN_IN);

        var n_LEAN_IN_CO2 =
parameters.LeanSolvent_AmineMoleFlowRate[parameters] *
parameters.LeanSolvent_Loading[parameters];
        leanInStream.CO2.DefineAs(n_LEAN_IN_CO2); // Define
lean CO2 concentration.
    }
);

```

```

leanInStream.N2.DefineAs(0); // Remove
gas, to avoid messing up the zero-vapor-fraction stream spec. Also the
stripper likely removes much of these.

leanInStream.O2.DefineAs(0); // Remove
gas, to avoid messing up the zero-vapor-fraction stream spec. Also the
stripper likely removes much of these.

leanInStream.H2.DefineAs(0); // Remove
gas, to avoid messing up the zero-vapor-fraction stream spec. Also the
stripper likely removes much of these.

leanInStream.TransferTo(parameters, dhexTearStream);
})
, stripper
);

#if false
Utilities.Show(
    new Func<System.Windows.UIElement>(() => { return
captureUnit.GetGUIRepresentation(p); })
    , new Func<string>(() => { return "Capture unit
simulation"; })
);

Utilities.Show(
    new Func<System.Windows.UIElement>(() => {
        System.Windows.Controls.Button button = new
System.Windows.Controls.Button();
        button.Content = "Pause";
        button.Click += new
System.Windows.RoutedEventHandler((object sender,
System.Windows.RoutedEventArgs e) => {
            p.Control_IsPaused = !p.Control_IsPaused;
        });
        return button;
    })
    , new Func<string>(() => { return "Pause control"; })
);
#endif

```

```

Utilities.Show(
    new Func<System.Windows.UIElement>(() => {
        var stackPanel = new
System.Windows.Controls.StackPanel();
        stackPanel.Orientation =
System.Windows.Controls.Orientation.Vertical;

        var controlStackPanel = new
System.Windows.Controls.StackPanel();
        controlStackPanel.Orientation =
System.Windows.Controls.Orientation.Horizontal;
        stackPanel.Children.Add(controlStackPanel);

        System.Windows.Controls.Button pauseButton = new
System.Windows.Controls.Button();
        pauseButton.Content = "Pause";
        pauseButton.Click += new
System.Windows.RoutedEventHandler((object sender,
System.Windows.RoutedEventArgs e) => {
            if (p.Control_IsPaused)
            {
                p.Control_IsPaused = false;
                pauseButton.Content = "Pause";
            }
            else
            {
                p.Control_IsPaused = true;
                pauseButton.Content = "Unpause";
            }
        });
        controlStackPanel.Children.Add(pauseButton);

        System.Windows.Controls.Button exitButton = new
System.Windows.Controls.Button();
        exitButton.Content = "Exit";
        exitButton.Click += new
System.Windows.RoutedEventHandler((object sender,
System.Windows.RoutedEventArgs e) => {

```

```

        p.Control_IsPaused = true;
        App.Current.Shutdown(0);
    });
    controlStackPanel.Children.Add(exitButton);

    stackPanel.Children.Add(captureUnit.GetGUIRepresentation(p));
    return stackPanel;
    })
    , new Func<string>(() => { return "Capture unit
simulation"; })
    );
#endif

    captureUnit.Run(p);

#if DEBUG
    Utilities.Report("DoSomething003() simulations complete. Stream
table:");
    Utilities.Report(p.StreamTable());
#endif

#if DEBUG
    DateTime runEndTime = DateTime.Now;
    Utilities.Report("Completed DoSomethign003()." +
runEndTime.ToString()
    + Environment.NewLine + "Total run time:\t" + (runEndTime -
runStartTime).ToString());
#endif
    }
}
}

```

Scheme 6.3. Source code for the acid-gas-capture unit using the new framework.