COLLOCATION METHODS FOR FLEXIBLE DISTILLATION DESIGN

A THESIS SUBMITTED TO THE GRADUATE SCHOOL IN
PARTIAL FULFILLMENT OF THE REQUIREMENTS

for the degree

DOCTOR OF PHILOSOPHY

in

CHEMICAL ENGINEERING

by

Robert S. Huss

Department of Chemical Engineering
Carnegie Mellon University
5000 Forbes Ave.
Pittsburgh, PA 15213

May 22, 1995

Abstract

In this thesis we present a collocation method originally developed for minimum reflux calculations which is applicable to the flexible distillation design problem. Solvent recovery plants need to separate a wide range of mixed solvent streams into their pure components. To have a flexible distillation system, single columns must be able to handle multiple separation tasks. We address the design problem of a single flexible column model, successfully designing a column for an azeotropic system with different separation tasks for different feed streams.

The collocation model incorporates two variable transformations which enable it to successfully model columns near pinch and columns with very high purities. With this model, the number of trays in each column section is a continuous variable, allowing us to optimize the column size and feed location. We developed the models in the ASCEND system, which we have found to be an excellent environment for model and design algorithm development.

We also discuss the application of the collocation methods to the minimum reflux problem. We can accurately model nonsharp minimum reflux problems, and we have explored in detail the trends when approaching a saddle pinch. The model experiences numerical instability when very close to a saddle pinch, but by detecting the error we can approach the saddle pinch to approximate a sharp split minimum reflux.

Acknowledgment

This thesis is the result of five years of graduate study, throughout which I have received significant aid. In the first two years, I learned the ropes from Peter Piela and Joe Zaher. I am grateful for the time they put in to help me with ASCEND. I also received some nicely summarized papers from Mark Thomas which have been well used. I'd like to thank him also for being my general computer reference.

Throughout my later years here, I've had many bull sessions with Boyd Safrit as we explored modeling distillation in ASCEND. I thank him for his help as well as the opportunity to consider all the new problems he encountered. I also thank Kirk Abbott and Ben Allan. If they had not put the effort into converting ASCEND into C and continually improving it, my job would have been much harder. I am very proud of the teamwork we've put into the ASCEND system.

I give a special thank you to my advisor, Art Westerberg. He has always been able to find key directions and problems in my work. He has also helped me to organize my thoughts more clearly and approach new problems confidently. Last, but not least, he has forced me to be a better writer.

Although she does not help me with research, my most significant supporter is my wife, Cindi. Thanks to her I have satisfaction beyond my work. Her sure confidence in my ability to succeed has provided a solid foundation for this thesis.

Thank you all.

Support from Eastman Chemicals and DOE contract number DE FG02-85ER13396 provide support for this research. Facilities support is from NSF grant number EEC-8942146 which supports the EDRC, an NSF funded Engineering Research Center.

Table of Contents

List of Figures

List of Tables

CHAPTER 1

INTRODUCTION AND OVERVIEW

1.1 Introduction

Chemical plants use a wide variety of solvents which must be recovered for reuse. However, the feed to a solvent recovery plant will vary as the chemical plant demands change. Any number of plants will send their waste mixed solvents to the recovery plant. Therefore, the feed to the recovery plant will change as the operation of the plants change. Azeotropic systems are particularly difficult because a change in the feed composition can move the system into a different distillation region. A simple, but expensive, solution to this problem is to require a constant feed composition to the recovery plant, using mixing to maintain the composition. A flexible solvent recovery plant, capable of separating a range of feeds with the same equipment, would be very useful to the chemical industry.

It is our goal to create the capability for designing flexible solvent recovery plants. However, this thesis only addresses a small part of that task, concentrating mainly on the tools we have developed toward that goal. We have developed an advanced collocation method for distillation design. Our collocation method incorporates two variable transformations which allow us to model large columns reaching high purities. We have tested this model against rigorous stage-by-stage calculations, used it to explore minimum reflux conditions, applied it to distillation column design, and applied it to the flexible distillation design problem.

We developed the model within the ASCEND system, an equation-based modeling environment. Within this environment we can create hierarchical model structures and through the user interface access any part of the model for manipulation or solving. Within this system we have learned how to solve column models effectively and developed complex design algorithms. We have created a user interface to the application we built to design columns.

In this thesis we use three completed papers for Chapters 2-4. The numbering in these chapters has been made consistent with the rest of the thesis, but the text is not changed from what we plan to publish. Therefore each paper has its own nomenclature and references sections. We also include complete nomenclature and references for the entire thesis at the end of the thesis. The papers are “Collocation Methods for Distillation Design I: Model Description and Testing,” “Collocation Methods for Distillation Design II: Minimum Reflux,” and “Collocation Methods for Distillation Design III: Flexible Column Design.” We plan to publish them in *Industrial & Engineering Chemistry Research*.

1.2 Overview of Chapter 2

In the first paper we describe the collocation model in detail. We begin with our original motivation for looking at collocation, minimum reflux determination. We proposed that modeling infinite column sections with

collocation models would allow us to develop a minimum reflux calculation method that did not rely on approximate geometric criteria. After developing the collocation models, however, we saw that it would be very applicable to design.

We then give a background of collocation, covering the work of Cho and Joseph [1983a and b], Stewart et al. [1984], Swartz and Stewart [1986], Srivastava and Joseph [1987], and Seferlis and Hrymak [1994a and b]. We also present a detailed degrees of freedom analysis for collocation, which demonstrates how we formulated our model.

We discuss how placement of the collocation points is a non-trivial problem, demonstrating that Stewart's improvement of using Hahn polynomials rather than Jacobi polynomials to place the points was due to the fact that the Hahn placement spreads the points out towards the ends of the collocation section more than the default Jacobi placement. Stewart et al. [1984] noted that the superior Hahn placement did not require any adjusting of parameters, which the Jacobi did. However, for most systems, the optimal placement is even further out than the Hahn placement.

We describe the variable transformations we incorporated into our model, one an exponential transformation of stage location, mapping infinite stages to a finite transform variable, and the second a hyperbolic tangent transformation on mole fraction, transforming asymptotic approach to zero or one to an increase or decrease towards positive or negative infinity. These variable transformations improve accuracy when modeling large columns or columns with high purities. The improvements from the variable transformations are more significant than the effects of optimal point placement.

We describe our standard collocation model, which has 11 tray models and as many equations as an 18 tray stage-by-stage model. We present several tests of the model against stage-by-stage rigorous models, including an azeotropic

system and a five component nonideal system.

1.3 Overview of Chapter 3

In the second paper we concentrate on the minimum reflux problem. We describe what pinch points are and how they can be exploited for simplified models of minimum reflux conditions. We describe how nonsharp splits require finite column sections and cannot be modeled by the existing methods which use approximate geometric arguments. We also describe sharp splits, which are characterized by saddle pinches where an infinite number of trays are required to remove a component completely.

We use the collocation model to explore the trends of large column sections, which approach a saddle pinch condition. We show how the sensitivity of reflux ratio on the amount of trace components in the products decreases sharply as less sharp splits are required between the key components. We also show how the collocation model can become inaccurate when going to large numbers of trays, even with the variable transformations.

We present techniques for detecting the error in the collocation at large trays, and propose a minimum reflux calculation for sharp splits that approaches the saddle pinch as closely as possible before numerical instabilities occur. With these techniques we can approximate minimum reflux for sharp splits. For nonsharp splits we see no numerical problems with calculating minimum reflux.

We describe in detail all the difficulties we have encountered with the collocation model to open discussion on these problems.

1.4 Overview of Chapter 4

In the third paper we present our algorithms for column design and flexible column design. We discuss how we use the ASCEND system to develop models and with its interactive solving environment learn how to solve these

problems. We describe in detail the algorithm for designing a column to meet a specific separation task. Once we find a nominal design, we can optimize that column design for cost, using a simple gradient based algorithm.

We present two examples of simple column design, one an ideal system which we optimized with MINOS, and the other a fully rigorous system which we optimized with the gradient based algorithm. With the application we developed in ASCEND for design, we designed a fully thermodynamic, heat balanced column for a propanol, isobutanol, butanol system in 20 minutes on an HP700. This solution represents 39 column solutions.

We then present background on the flexible design problem. Very little work has been done on the specific problem of flexible distillation design, but we discuss some of the more general work on flexible design by Grossmann and coworkers. Unfortunately, those techniques require significant manipulation of the equations, and complex thermodynamics do not lend themselves to those methods.

We present our flexible design algorithm for a single column. A single flexible column represents the subproblem of a single column within a flexible solvent recovery plant which must be able to handle a set of different feeds with different separation tasks. We overcome the problem of having multiple column models within an optimization or formulating the Kuhn-Tucker conditions for 2000 equations including UNIFAC liquid mixture models by approximating the operation of the column with each feed. We use the column model to create a quadratic approximation of the reflux ratio as a function of the number of trays and the feed tray location. With an approximation of each column and overall cost equations, we used MINOS to optimize the column locally within the range of the approximation. When MINOS returned a solution on the border of the approximation region, we reapproximated around the current point, continuing until the minimum cost was within the current approximation region.

We designed flexible columns for two systems, one with four possible feeds of propanol, isobutanol, and butanol, which took two hours to optimize. The other was an azeotropic system where the separation tasks for the different feeds were in different distillation regions, demonstrating how a single column can be designed for quite different tasks.

1.5 Overview of Chapter 5

Chapter 5 is not a paper to be published. In Chapter 5, we describe the ASCEND models included in the appendix. We include this lengthy ASCEND code to present the equations of our models for anyone who wants to reproduce them. Since our formulation of the models is strongly dependent on the ASCEND system, we include the code rather than just the equations. This allows us to demonstrate the hierarchical nature of ASCEND. For brevity, we do not include some of the standard procedures which are functionally identical for each model. We also do not include the models we created to facilitate plotting of the columns.

1.6 Overview of Chapter 6

Chapter 6 may be published as a technical report within the Engineering Design Research Center. It is a manual for using our column design application. We do not go into detail of the algorithm code.

1.7 Overview of Chapter 7

In Chapter 7 we present conclusions as well as directions for future work.

CHAPTER 2

COLLOCATION METHODS FOR DISTILLATION DESIGN I: MODEL DESCRIPTION AND TESTING

Robert S. Huss and Arthur W. Westerberg

Department of Chemical Engineering
and Engineering Design Research Center
Carnegie Mellon University
5000 Forbes Ave.
Pittsburgh, PA 15213

To be submitted to *Industrial & Engineering Chemistry Research*

Abstract

Fast and accurate distillation design requires a model that significantly reduces the problem size while accurately approximating a full order distillation column model. Variable number of trays and variable feed tray location make optimization possible.

This collocation model builds on the concepts of past collocation models for design of complex real-world separation systems. Two variable transformations make this method unique. Polynomials cannot accurately fit trajectories which flatten out. In columns, flat sections occur in the middle of large column sections, or where concentrations go to zero or one. With an exponential transformation of the tray number which maps zero to an infinite number of trays onto the range zero to one, two collocation trays can accurately simulate a large column section. With a hyperbolic tangent transformation of the mole fractions, the model can simulate columns which reach high purities. Furthermore, this model uses multiple collocation elements for a column section, which is more accurate than a single high order collocation section.

In this paper, we describe the collocation method and present some testing of the method. In two companion papers we will discuss applications for this model and use of the model in the ASCEND system.

2.1 Introduction

Several researchers have explored and developed collocation for distillation column modeling. In this paper, we present a collocation model which expands on prior models, addressing the problems specific to steady-state, continuous columns. We describe the formulation of the model in detail and compare it to prior models, using rigorous column simulations as a benchmark.

2.2 Motivation

The desire for a method to perform minimum reflux calculations for highly nonideal systems was the original motivation for this collocation method. For a specific separation, the required reflux ratio decreases as the number of trays in the column increases. As one increases the number of trays in a column section to a very large number, a region of constant composition occurs, referred to as a pinch. At a pinch point, the vapor and liquid passing each other are in equilibrium. They are also in material balance with the compositions entering and leaving at the end of that column section.

Consider a separation of a ternary mixture A, B, and C, where we want to separate A from B and C (see Figure 2.1). Given the products and a specified reflux ratio, Levy et al. [1985] solved this type of problem by starting from the ends of the column and calculating the trajectories inward. When the concentration changes in each section fall below some minimum value, they stop calculating and assume this is the pinch point. If the trajectories intersect, then they have a feasible column profile. If one trajectory just pinches on the other, they have a minimum reflux profile.

For a larger number of species, one cannot completely specify a product for a column. Typically one can ask that component splits satisfy inequalities -- e.g., we want at least 99.9% of A and 99% of B and no more than 1% of C and less than 0.01% of D to exit in the distillate. Solving such a problem requires one to discover

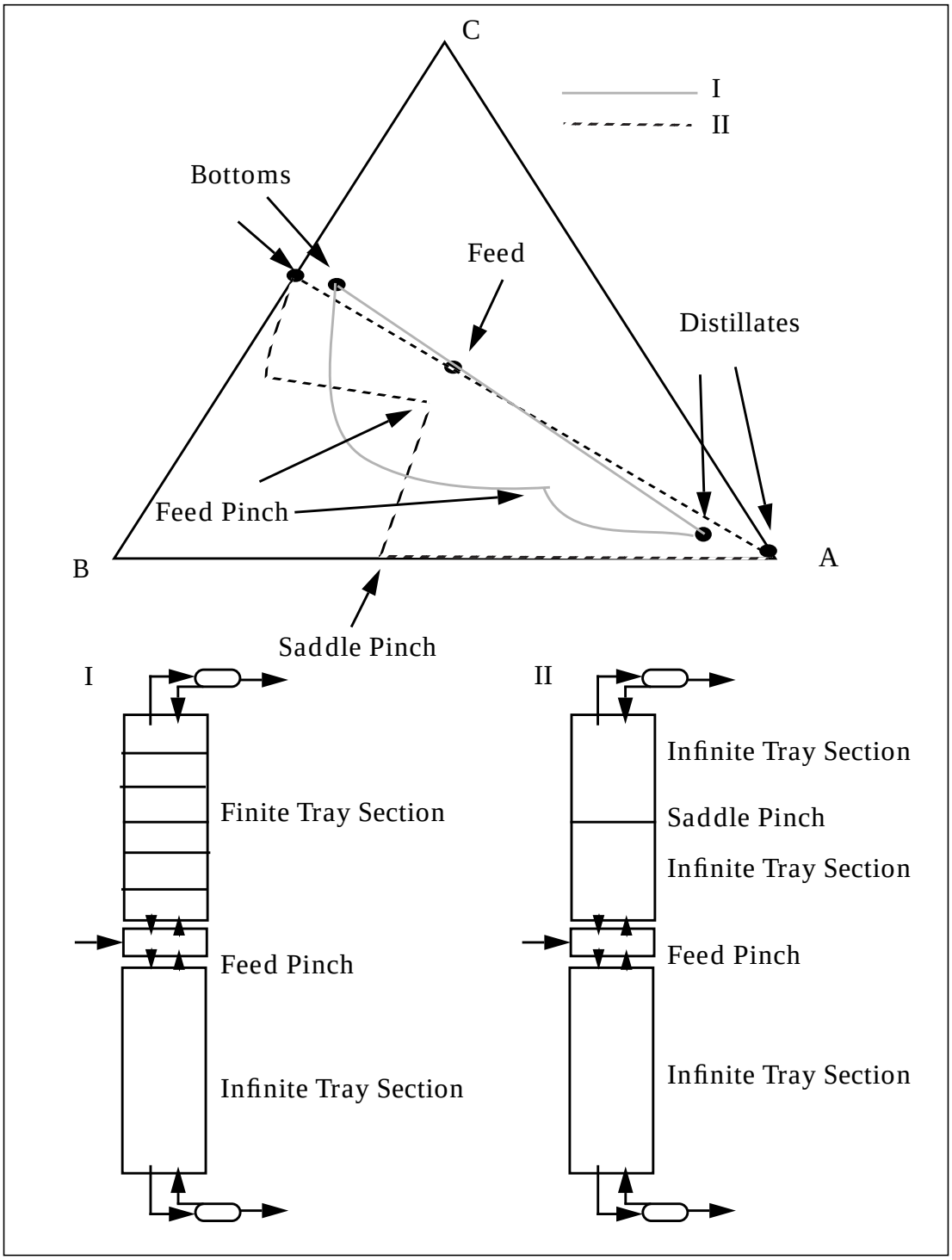

Figure 2.1. Minimum reflux trajectories and column configurations

which of these constraints are active. We saw that one can still use a pinch strategically located, but we also discovered that one needs a tray by tray column

section model in a form which can compute the number of trays it contains. Conventional column models require one to specify the trays as an integer input.

This thinking led us to consider using a collocation model for the parts of the column where Levy et al. [1985] used a tray by tray computation. For such a column section, the number of trays is a continuous variable which the model can compute.

Returning to the three component example, Levy et al. [1985] showed that, as they decreased the concentration of C in the distillate, the minimum reflux decreased. Therefore, for a specific separation between the key components, A and B, the minimum reflux occurs when they allow no C in the distillate. In this case the top section has a saddle pinch point along the A-B edge where the concentration of C goes to zero. Levy et al. [1985] noticed that, as their computations approached this saddle pinch, it, the feed composition, and the feed pinch became colinear. Requiring colinearity of these points gave them a method to determine minimum reflux without the needed tray by tray computation. For a constant relative volatility, constant molar overflow system, they proved this method becomes exactly Underwood's method. They proposed using this colinearity even for nonideal systems. In a similar manner and for multicomponent nonideal systems, Koehler et al. [1991] used a minimum angle criterion between the three points, and Julka and Doherty[1990] revised the colinearity condition to one of having a set of pinch points and the feed composition lie inside a minimum volume in composition space. All these methods assume sharp splits as they are based on computing a saddle pinch point. These geometric techniques do not guarantee a feasible column for nonideal systems, and they do not work for nonsharp splits.

To guarantee the intersection of the column sections for sharp splits, one must prove the existence of a tray by tray calculation linking a saddle pinch point and the feed tray composition, a calculation that passes through an infinite

number of trays. This requirement led us to examine extending the collocation model to handle an infinite number of trays. We looked at different transformations to map tray number going from zero to infinity onto a variable z that goes from zero to one. With such a model, we concluded we should be able to compute minimum reflux for any column by computing pinch points and properly located column sections that can have a finite or an infinite number of trays. For case I shown in Figure 2.1, a finite collocation section simulates the unknown number of trays. For case II, the infinite tray section between the saddle pinch point and the feed tray is modeled by an infinite collocation section.

We began developing a collocation method capable of simulating infinite column sections and discovered several other advantages and uses for collocation. Distillation design requires an adaptable column model, with the ability to compute the number of trays in each distillation section, a computation that discovers how many trays one needs and where to place the feed. One can simulate complex column configurations. Optimal design of distillation sequences requires small robust models for each distillation column. Collocation both reduces the size of a column model and provides a continuous variable for the number of trays. Furthermore, the variable transformation required for modeling an infinite tray section improved the accuracy whenever such separation problem requires a relatively large number of trays.

2.3 Background of Collocation

Cho and Joseph [1983] developed a reduced-order method for modeling staged separation processes. They used orthogonal collocation to obtain accurate solutions of significantly reduced-order. Their model had a single collocation section for each section of the column. They tested by modeling a simple absorber system, and binary and three component distillation, and used the Antoine equation for the equilibrium relationship. In later papers Srivastava and Joseph [1984, 1987a] developed methods for handling multiple feeds and side draws.

They also developed a complex method for handling steep and flat composition profiles by fitting the composition profiles with different polynomials for each component. They developed a complicated approach using two sets of collocation points, global and local, to fit both the key components and non key components [Srivastava and Joseph, 1987b]. They tested these later ideas using constant relative volatility systems.

Stewart, Levien, and Morari [1984] developed a collocation method that stresses selecting gridpoints based on the stagewise nature of distillation. Their method became stage-by-stage at full order, and had errors at least an order of magnitude smaller than Cho and Joseph's default choice of collocation points. They tested for binary and six-component distillation columns with constant relative volatility and for a ternary system using UNIQUAC for equilibrium. Swartz and Stewart [1986] applied the method to design, iteratively passing from the model to an SQP optimization algorithm. Swartz and Stewart [1987] also developed a finite-element method for handling multiphase distillation problems.

Recently, Seferlis and Hrymak [1994] adapted the model of Stewart et al. [1984] by using collocation elements to track irregularities in column profiles for existing columns. They investigated optimal placing of the collocation elements, based on comparison with the actual column. They obtained higher accuracy with multiple collocation sections of lower order than with a single collocation section of higher order.

Table 2.1 lists the characteristics of each of these collocation methods as well as the characteristics of the model presented in this paper.

2.4 Description of Model

Collocation is generally thought of as a method for numerically solving differential equations. The use of collocation for simulation of a distillation column is an extension of this technique. Given a differential equation,

Table 2.1: Comparison of Collocation Methods

	Elements/ Section	Placement of points	Thermo-dynamics tested	Variable transformations
Cho and Joseph 1983	Single	Continuous orthogonal (Jacobi)	Antoine	none
Srivastava and Joseph 1987	Single(global), Multiple(local)	Continuous orthogonal (Jacobi)	Constant relative volatility	none
Stewart et al 1984	Single	Discrete orthogonal (Hahn)	UNIQUAC	none
Swartz and Stewart 1986	Single	Discrete orthogonal (Hahn)	Ideal	none
Swartz and Stewart 1987	Multiple (breakpoints at phase changes)	Discrete orthogonal (Hahn)	Nonideal three phase	none
Seferlis and Hrymak 1994	Multiple	Discrete orthogonal (Hahn)	Regression of data	none
This work	Multiple	Continuous (orthogonal for 2 or 3 points)	UNIFAC/ Pitzer	Transform tray number and mole fractions

$$\frac{dy}{dx} = f(x, y)\ ,\ y(0) = y_0 \qquad (2.1)$$

we want to find y as a function of x. We can approximate y as a polynomial in x,

$$\hat{y} = y_0 + a_1 x + a_2 x^2 + \ldots + a_n x^n \qquad (2.2)$$

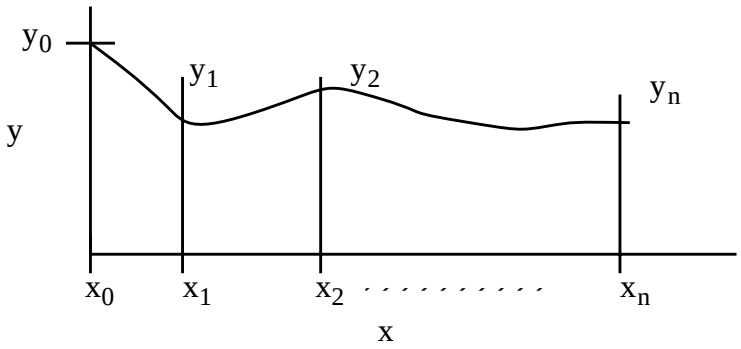

Figure 2.2. Collocation of a differential equation

We can then approximate *y, dy/dx*, and *f* as functions of *x*. At any point in *x*, we can define an error for this approximation,

$$\boldsymbol{error}(x) = \boldsymbol{f}(\boldsymbol{x}, \hat{\boldsymbol{y}}) - \left(\boldsymbol{a_1} + 2\boldsymbol{a_2}\boldsymbol{x}^{\mathbf{1}} + \ldots + \boldsymbol{n}\boldsymbol{a_n}\boldsymbol{x}^{\boldsymbol{n}-\mathbf{1}} \right) \qquad \textbf{(2.3)}$$

At *n* collocation points, $x..x_n$, we say that the error =0 and get *n* equations to solve for *n* coefficients to the polynomial. See Figure 2.2.

Collocation of a distillation column uses the same concepts. A set of equations defines a distillation column tray, where *x* would be the tray location in the column and *y* would be mole fractions. Polynomials defined by tray location approximate the mole fractions and, at each collocation point, the set of equations for a distillation column must be satisfied.

There are many equations in a distillation column model, even a reduced order model. Rather than just list the basic equations, we are going to provide a detailed degrees of freedom analysis to demonstrate the reason for using a certain set of equations.

We begin the degrees of freedom analysis with a stream model. Gibbs phase rule gives the number of degrees of freedom for a system in equilibrium.

$$F = 2 + n_c - n_p \quad (2.4)$$

F is the number of degrees of freedom for a given number of components, n_c, and a given number of phases, n_p. For a single phase, F is $n_c + 1$. For a stream we also need a flowrate which adds one more variable, giving $n_c + 2$ degrees of freedom. The set of variables could be the molar flowrates, temperature, and pressure. Once we know $n_c + 2$ of these variables and assume the phase, we can compute all other molar properties. We shall assume a stream introduces a net of $n_c + 2$ new variables and shall assume all other properties are available.

A standard distillation tray has two input streams and two output streams, as shown in Figure 2.3. The four streams introduce a net of $4(n_c + 2)$ new variables, which is the first entry on Table 2.2. By keeping track of the number of variables and equations introduced by each new element of the model, we can determine the degrees of freedom, and how many variables must be fixed to obtain a system with the same number of equations and free variables. We can write the following equations for a single tray.

Component Material Balances:

$$L(out)x_i(out) - L(in)x_i(in) = V(in)y_i(in) - V(out)y_i(out) \quad (2.5)$$

Equilibrium:

$$y_i(out) = \frac{\alpha_i}{\bar{\alpha}} x_i(out) \quad , T_L(out) = T_V(out),\ P_L(out) = P_V(out) \quad (2.6)$$

Heat balance:

$$L(in)h(in) - L(out)h(out) = V(in)H(in) - V(out)H(out) \quad (2.7)$$

L, V, x, y, h, H are liquid flowrate, vapor flowrate, liquid mole fraction, vapor mole fraction, liquid molar enthalpy, and vapor molar enthalpy, respectively. α_i is the relative volatility of species i, and $\bar{\alpha}$ is the mole fraction average relative volatility. The liquid and vapor molar enthalpies and the relative volatilities are functions of

composition and temperature. We write equations 2.5 and 2.6 for each component

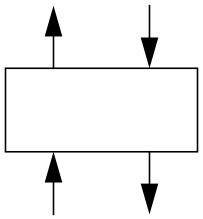

Figure 2.3. Diagram of tray

and equation 2.7 once. As shown in Table 2.2, the component material balances therefore introduce n_c equations. There are $n_c + 2$ equilibrium equations. The equilibrium equation also introduces one new variable, $\bar{\alpha}$. The heat balance, which could be replaced by a constant molar overflow assumption, introduces one equation. Therefore, the degrees of freedom for an isolated tray are $2(n_c+2)+2$. Specifying two input streams, the pressure and $\bar{\alpha}$ of the tray would be sufficient to solve the tray model.

Table 2.2: Degrees of Freedom for an Isolated Tray

	New Variables	New Equations
4 streams	$4(n_c+2)$	
CMB		n_c
Equilibrium	1	$n_c + 2$
HB		1
totals	$4(n_c +2) +1$	$2n_c +3$
DOF for isolated tray	$2(n_c+2) + 2$	

Figure 2.4 shows a diagram of a single collocation section. A collocation section has a liquid input and vapor output at the top and a vapor input and liquid output at the bottom. The example shown has two collocation trays which are not connected. In this example, we number from the top of the collocation

section downward. Liquid and vapor streams passing each other have the same

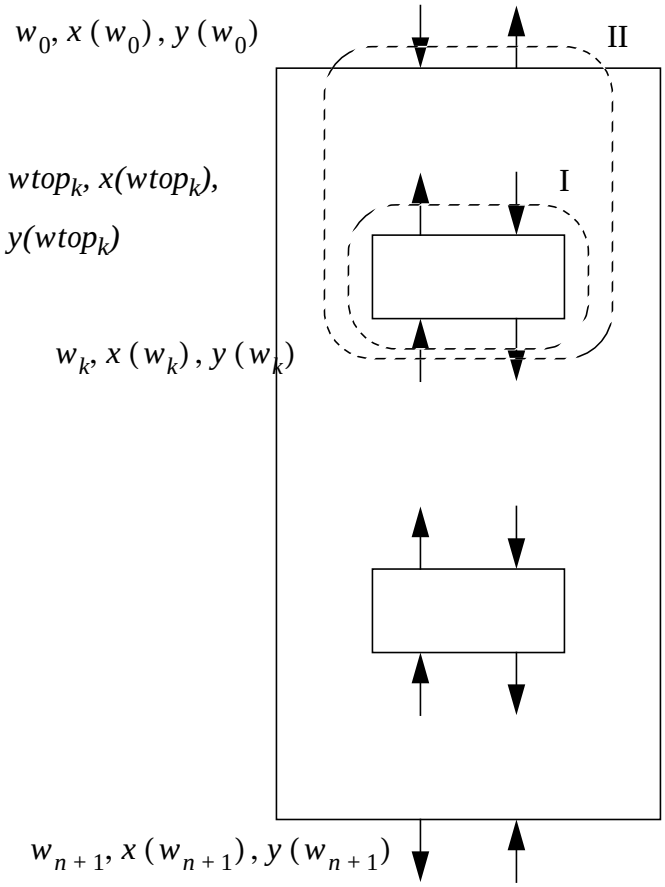

Figure 2.4. Diagram of collocation section, I and II are material and energy balance envelopes for a tray.

index. We use a general position index, w, which will be either the tray number or a transformation of the tray number which we shall describe later. The index denotes the distance from the top of the collocation section. We index each tray *k* by w_k, the position of the bottom of the tray. We index the liquid entering and the vapor leaving the top of the tray by $wtop_k$, which in tray location is w_k-1, but with a transformation on the tray location is more complicated. Equations 5, 6, and 7 become the following

$$L(w_k)x_i(w_k) - L(wtop_k)x_i(wtop_k) = V(w_k)y_i(w_k) - V(wtop_k)y_i(wtop_k) \qquad (2.8)$$

$$y_i(wtop_k) = \frac{\alpha_i}{\bar{\alpha}} x_i(w_k) \quad , T_L(w_k) = T_V(wtop_k),\ P_L(w_k) = P_V(wtop_k) \qquad (2.9)$$

$$L(w_k)h(w_k) - L(wtop_k)h(wtop_k) = V(w_k)H(w_k) - V(wtop_k)H(wtop_k) \qquad (2.10)$$

Starting with an isolated tray, Table 2.3 aids the degrees of freedom analysis for a collocation tray. Collocation of the liquid mole fractions requires the liquid input and output compositions for the tray to be on the polynomial approximations of the liquid mole fraction. Collocating the liquid and vapor mole fractions for n_c-1 components creates 4(n_c-1) equations. The number of variables introduced (the polynomial coefficients) will depend on the order of the polynomial, which we shall decide later.

Using a Lagrange polynomial, the following equations:

$$x_i(w) = \sum_{k=0}^{n+1} W_k(w)\, x_i(w_k) \quad i = 1 \ldots nc-1 \qquad (2.11)$$

$$y_i(w) = \sum_{k=0}^{n+1} W_k(w)\, y_i(w_k) \quad i = 1 \ldots nc-1 \qquad (2.12)$$

are the polynomial approximations of order n for the liquid and vapor mole fractions at position w. We use a Lagrange form because the coefficients of the polynomials, $x_i(w_k)$ and $y_i(w_k)$ are also the liquid and vapor mole fraction at position w_k, the location of the k^{th} collocation point. The k^{th} term of a Lagrange polynomial, W_k, is defined by the following equation:

$$W_k(w) = \prod_{\substack{j=0 \\ j \neq k}}^{n+1} \frac{w - w_j}{w_k - w_j} \qquad (2.13)$$

Table 2.3: Degrees of Freedom for a Single Tray

	New Variables	New Equations
tray model	$2(n_c+2)+2$	
collocate x, y, for n_c -1	?	$4(n_c-1)$
collocate $\bar{h}$ for liquid and vapor	?	4
CMB around end		n_c
HB around end		1
Fix Pressures		3
Fix $\bar{\alpha}$ at 1		1
totals	$2n_c+6+?$	$5n_c+5$
Net equations	??	$3n_c-1$

Collocating the enthalpies of the liquid and vapor entering and leaving introduces 4 equations. Again, the number of polynomial coefficients introduced as new variables will depend on the order of the polynomial. The following equations define the enthalpy polynomials.

$$\boldsymbol{h}(\boldsymbol{w}) = \sum_{\boldsymbol{k}=\boldsymbol{0}}^{\boldsymbol{n}+\boldsymbol{1}} \boldsymbol{W_k}(\boldsymbol{w})\,\boldsymbol{h}(\boldsymbol{w_k}) \qquad \textbf{(2.14)}$$

$$\boldsymbol{H}(\boldsymbol{w}) = \sum_{\boldsymbol{k}=\boldsymbol{0}}^{\boldsymbol{n}+\boldsymbol{1}} \boldsymbol{W_k}(\boldsymbol{w})\,\boldsymbol{H}(\boldsymbol{w_k}) \qquad \textbf{(2.15)}$$

Since a single tray is isolated, we can add some overall balance equations between the tray and the end of the collocation section:

Component Mass Balances:

$$\begin{aligned} \boldsymbol{L}(\boldsymbol{w_k})\,\boldsymbol{x_i}(\boldsymbol{w_k}) - \boldsymbol{L}\left(\boldsymbol{w_{n_s+1}}\right)x_i\left(\boldsymbol{w_{n_s+1}}\right) &= \\ \boldsymbol{V}(\boldsymbol{w_k})\,\boldsymbol{y_i}(\boldsymbol{w_k}) - \boldsymbol{V}\left(\boldsymbol{w_{n_s+1}}\right)\boldsymbol{y_i}\left(\boldsymbol{w_{n_s+1}}\right) & \end{aligned} \tag{2.16}$$

Heat balance:

$$\begin{aligned} \boldsymbol{L}(\boldsymbol{w_k})\,h(\boldsymbol{w_k}) - \boldsymbol{L}\left(\boldsymbol{w_{n_s+1}}\right)h\left(\boldsymbol{w_{n_s+1}}\right) &= \\ \boldsymbol{V}(\boldsymbol{w_k})\,H(\boldsymbol{w_k}) - \boldsymbol{V}\left(\boldsymbol{w_{n_s+1}}\right)H\left(\boldsymbol{w_{n_s+1}}\right) & \end{aligned} \tag{2.17}$$

The component mass balance adds n_c equations, and the heat balance adds 1 equation. Since we are assuming a constant pressure column, we need to fix the pressures of both input streams and one output stream, adding 3 equations. Finally, we specify that $\overline{\alpha}$ for the tray is fixed at 1.0. This leaves a total of $3n_c$-1 excess equations for each collocation tray. The question marks indicate that we have not yet accounted for the polynomial coefficients.

Table 2.4 aids the degrees of freedom analysis for the entire section. The section has n_s trays, introducing $n_s(3n_c - 1)$ net new equations. It also has two input and two output streams, creating $4(n_c+2)$ variables. Specifying the two input streams creates $2(n_c+2)$ equations. We can write the following balances over the entire section:

Component Mass Balances:

$$\begin{aligned} \boldsymbol{L}(0)\,\boldsymbol{x_i}(0) - \boldsymbol{L}\left(\boldsymbol{w_{n_s+1}}\right)x_i\left(\boldsymbol{w_{n_s+1}}\right) &= \\ \boldsymbol{V}(0)\,\boldsymbol{y_i}(0) - \boldsymbol{V}\left(\boldsymbol{w_{n_s+1}}\right)\boldsymbol{y_i}\left(\boldsymbol{w_{n_s+1}}\right) & \end{aligned} \tag{2.18}$$

Table 2.4: Degrees of Freedom for Section

	New Variables	New Equations
ns trays		$n_s(3n_c-1)$
4 streams	$4(n_c+2)$	
2 stream specs		$2(n_c+2)$
CMB		n_c
HB		1
Fix Pressures		2
collocate x, y for n_c - 1	?	$4(n_c-1)$
collocate $\bar{h}$ for liquid and vapor	?	4
totals	$4n_c+8$ + ?	$7n_c+7+n_s(3n_c-1)$
Net equations		$(n_s+1)(3n_c-1)$

Heat balance:

$$\boldsymbol{L}(0)\,h(0) - \boldsymbol{L}\left(\boldsymbol{w}_{\boldsymbol{n_s+1}}\right)h\left(\boldsymbol{w}_{\boldsymbol{n_s+1}}\right) = \boldsymbol{V}(0)\,H(0) - \boldsymbol{V}\left(\boldsymbol{w}_{\boldsymbol{n_s+1}}\right)H\left(\boldsymbol{w}_{\boldsymbol{n_s+1}}\right) \tag{2.19}$$

The component mass balance creates n_c equations, and the heat balance creates 1 equation. Fixing the pressures of the two output streams introduces 2 equations. Collocating the liquid and vapor mole fractions introduces $4(n_c-1)$ equations. Again, the number of variables introduced depends on the order of the polynomials. Collocating the enthalpies creates 4 equations. The net equations for a collocation section is $(n_s+1)(3n_c-1)$.

With n_c-1 liquid and vapor polynomials, and 1 polynomial for the liquid and vapor enthalpies, there are a total of $2n_c$ polynomials, so the number of variables introduced by the polynomials will be $2n_c(n+1)$, where n is the order of the polynomial. In principle we want to choose n so the number of equations and

variables are equal for the section model:

$$2n_c(n + 1) = (n_s + 1)(3n_c - 1) \qquad (2.20)$$

For three components and two stages, n is 3, while for four components and two stages n is 3.25. In most cases, n is not an integer, so something is not right. Previous papers select $n = n_s$. This leaves $(n_s + 1)(n_c - 1)$ excess equations. We expect the number of trays and the order of the polynomials to be linked, but we must remove the excess equations.

Cho and Joseph[1983] showed that when they assumed constant molar overflow the component material balances between the trays and the end of the collocation section given by equation 2.16 were held even when not enforced. We have found that the error in the component mass balances is negligible even for heat balanced columns. Rather than enforce the component mass balances between each tray and the end of the collocation section, we can enforce only the overall mass balance for each tray:

$$L(w_k) - L(w_{ns+1}) = V(w_k) - V(w_{ns+1}) \qquad (2.21)$$

This removes $(n_c - 1)$ equations per tray. Furthermore, if we enforce the component mass balance over the entire section given by equation 18, we do not need to calculate both output streams from the polynomials. Therefore, we can ignore the polynomial equations for the compositions for one output stream, removing the remaining n_c-1 equations for the entire collocation section. This removes the $(n_s + 1)(n_c - 1)$ extra equations, giving us zero degrees of freedom if we set $n = n_s$. Therefore, for a two tray collocation section, we get a second degree polynomial with three coefficients for each fitted component.

Another option is to add slacks to the ignored equations in the first example. If the slacks are too large, one can add more trays. When the component material balances between the individual trays and the top of the collocation section are ignored, the components can be out of balance on individual trays

even though they will be in mass balance over the entire section. Our tests have shown that even with nonideal systems with constant molar overflow, the component mass balances are satisfied. Also the polynomial equations for the one output stream that we ignored in the first case is satisfied. When we use heat balances rather than constant molar overflow, the component mass balances have very small errors, and the polynomial equations have slightly more significant errors. However, even when the trajectories of the collocation section are inaccurate, the residuals of these equations are not good indicators of the error.

A third alternative is to minimize the residuals of all the equations and solve for a best set of n_s+1 coefficients for each polynomial:

$$\begin{aligned} &\boldsymbol{min}\,(\|\varepsilon\|) \\ &\boldsymbol{eqn_j} = \varepsilon_{\boldsymbol{j}} \end{aligned} \qquad \textbf{(2.22)}$$

However, the arguments just given show that the additional equations did not have significant error terms. Therefore, the optimization would probably only yield a minor improvement on the first option.

We have used the first option in this work. The order of the polynomial will be the same as the number of stages used as collocation points.

2.5 Point Placement

The most difficult decision in collocation is the placement of the collocation points. Carnahan et al [69] showed that, for integration of differential equations, collocation points placed at the zeros of an orthogonal polynomial were best. However, this is not necessarily true for collocation of a distillation column. Cho and Joseph [1983] placed their points at the zeros of Jacobi polynomials defined by,

$$\int_0^1 z^{\beta} (1-z) z^{\alpha} P_n^{(\alpha,\beta)} (z)\, dz = 0 \qquad \textbf{(2.23)}$$

$$j = 0, 1, \ldots n-1$$

where α and β are parameters. Their default choice of the parameters for the Jacobi polynomial ($\alpha = 1, \beta = 1$) resulted in evenly spaced points. They could have moved the points toward either end of the collocation section by adjusting the parameters and still have been using an orthogonal polynomial. Stewart et al. [1984] showed that placing the collocation points at the zeros of the Hahn polynomial created smaller errors than placing the points by the Jacobi polynomial, using the default values for α and β. They argued that the Hahn polynomial was a better choice because it maintained the stagewise nature of the column and did not require manipulating parameters for best placement of the collocation points. For full order, the collocation points would be placed exactly at the tray locations.

However, the benefit of the Hahn polynomial appears to be due largely to the fact that it spreads out the collocation points more than the default Jacobi selection. As one spreads the collocation points out from being evenly spaced to being all at the ends of the collocation section, the error will go through a minimum. The Hahn placement is closer to this minimum than the default Jacobi, but it is not the optimum. The following experiment demonstrates this. Three different collocation models were used to approximate a three component, constant relative volatility, constant molar overflow column with 15 trays above and below the feed. Figure 2.5 shows the column trajectories for the three different reflux ratios, generated by a tray-by-tray model. Two collocation sections were used to model the column, one above and one below the feed. We did several simulations with each collocation model with different spreads of the collocation points. For any number of collocation trays, the following equation defined the midpoint of the trays.

$$w_{mid} = w_0 + f_{mid}(w_{ns+1} - w_0) \quad (2.24)$$

where f_{mid} is 0.5 to have w_{mid} at the actual center of the collocation section. For the two tray model, the placement of the two collocation trays is defined by,

$$w_1 = w_{mid} + f_{int}(w_{mid} - w_0) \quad (2.25)$$

$$w_2 = w_{mid} - f_{int}(w_{ns+1} - w_{mid}) \quad (2.26)$$

where f_{int} is 0.3333 for evenly spaced points.

For the three tray model, w_2 is at w_{mid}, and we define w_1 and w_3 as we defined w_1 and w_2 for a two tray model. For the three tray model, an f_{int} of 0.5 gives evenly spaced points.

For the four tray model, we define w_1 and w_4 as we defined w_1 and w_2 for a two tray model. We place the interior points, w_2 and w_3, one third of the distance

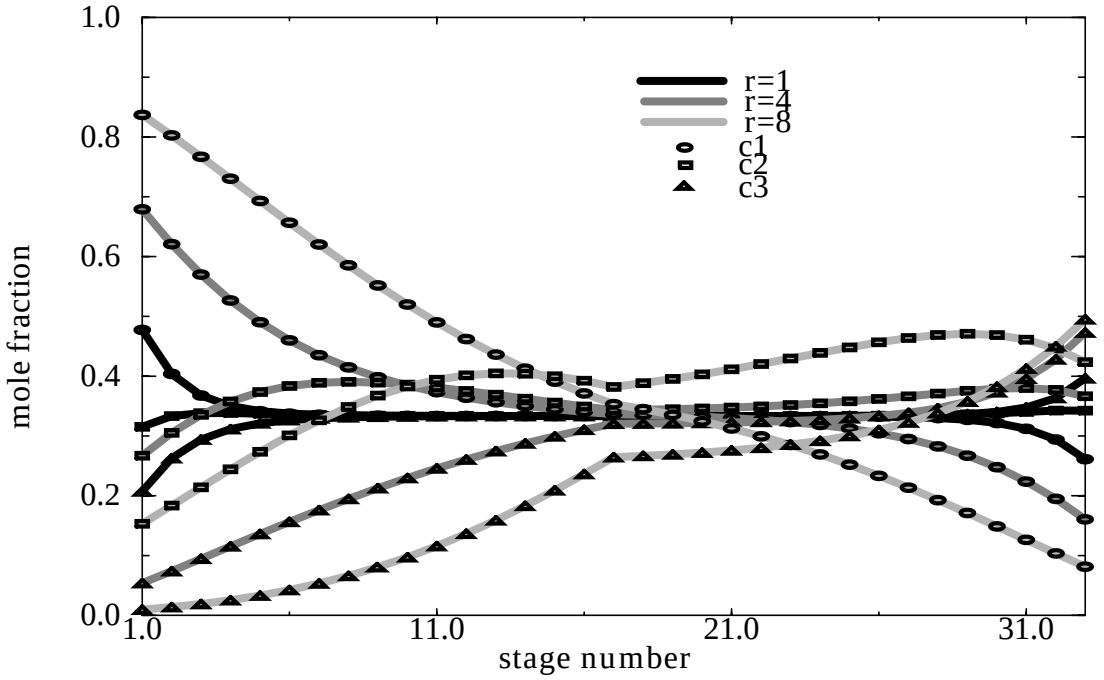

Figure 2.5. Column trajectories for components c1, c2, and c3 for a range of reflux ratios, r.

from w_{mid} to w_1 and w_4 respectively. For the four tray model, an f_{int} of 0.6 gives evenly spaced points. For the two and three tray models, the Hahn placement is at a spread factor of 0.53748 and 0.71864.

For the four tray model, the Hahn placement of the outer points corresponds to a spread factor of 0.79244, but our placement of the inner points does not correspond to the Hahn placement. Figure 2.6 shows the average error in the mole fractions of the distillate over varying spread factors for each model at the three reflux ratios. Each plot is on a different scale, shown by the labels on the x axis, and the maximum error on the y axis. The Hahn placement points are shown with a larger data point. The standard Jacobi placement is evenly spaced points, which is at f_{int} of 0.333, 0.5, and 0.6 for the two, three, and four tray collocations respectively. So, for each case, the Hahn placement has a smaller error than the default Jacobi placement, but not the minimum error possible. This figure also shows that the optimal spread of the collocation points is different for different reflux ratios.

For a low reflux ratio, the trajectories are very flat, and the minimum error occurs with a very wide spread of the collocation points, to get the nonlinear polynomial as flat as possible over the collocation section. As the reflux increases, the trajectories become less flat and then even linear with a fairly large slope. For these cases, there is an optimal spread of the collocation points. In the next section, we will show how variable transformations are more significant than point placement for increasing accuracy.

2.6 Variable Transformations

We use two variable transformations in this model to alleviate the problem of flattened trajectories. When the mole fraction of a component is not changing over part of a tray section, we call that a flattened trajectory. Flattened trajectories

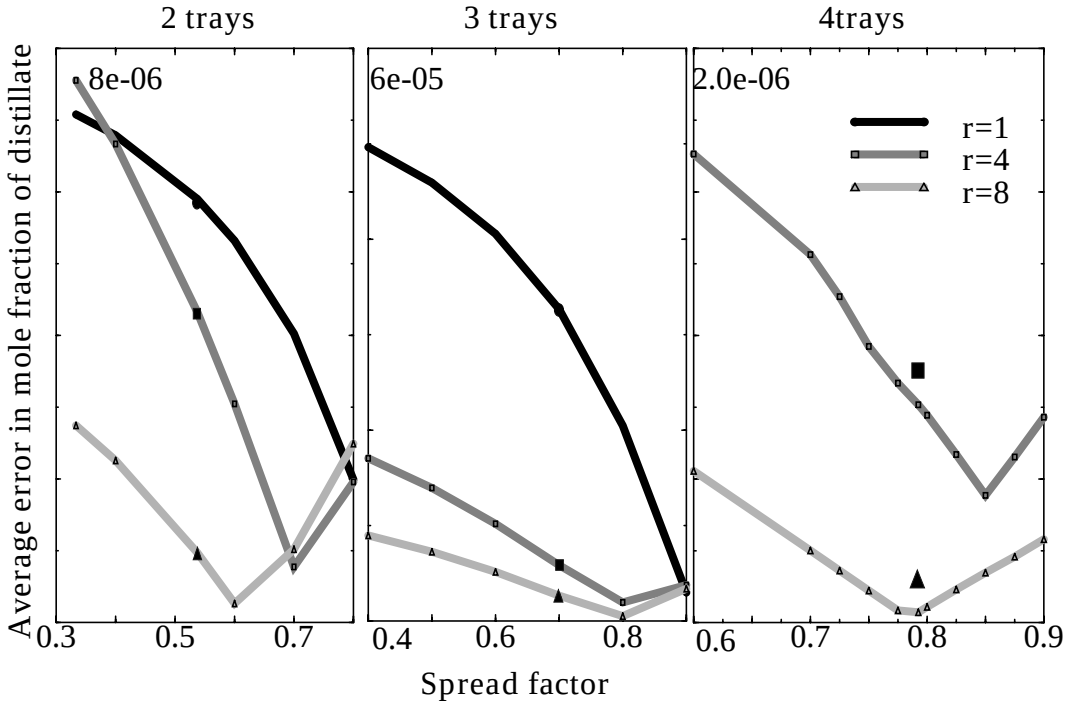

Figure 2.6. Effect of point placement on error

cannot be fit well with polynomials, so we perform variable transformations to alter the shape.

The first is a transformation of the tray number. To simulate a large number of trays, or an infinite number of trays, we want an index that goes to a finite value as the tray number goes to infinity. Even for finite columns with a large number of trays, the trajectories flatten out as the number of trays increases. Some possible transformations are:

$$z = 1 - e^{(-as)} \tag{2.27}$$

$$z = \frac{s}{s + a} \tag{2.28}$$

In both these equations, *s* is stage number, *z* is the transform variable, and *a* is a parameter. In both cases, *z*=0 when *s*=0, and *z* tends to 1 as *s* tends to infinity. To discover the better form of the transformation, we first investigated fitting results

from the Kremser approximation:

$$y_s = \frac{1 - A^s}{1 - A} y_1 - \frac{A - A^s}{1 - A} \hat{y}_0 \qquad \textbf{(2.29)}$$

$$A = \frac{L}{KV}, \hat{y}_0 = Kx_0 \qquad \textbf{(2.30)}$$

Figure 2.7 shows that the variable transformation given by equation 2.27 did the best at straightening out the trajectory. In the Appendix, we show that, with the exponential variable transformation, the Kremser approximation can be exactly straightened out for the correct choice of *a*. Figure 2.8 shows how the choice of *a* affects the shape of the trajectories. At the proper selection of *a*, the data can be fitwith a linear function. Therefore, we use the variable transformation in equation 2.27.

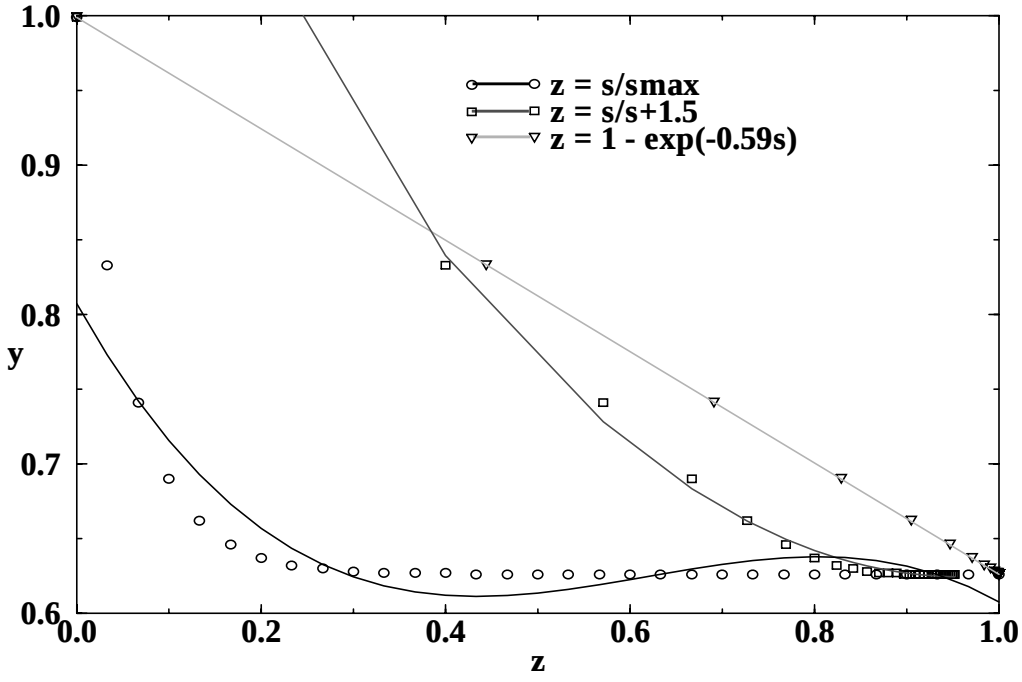

Figure 2.7. Comparison of variable transformations on *s*

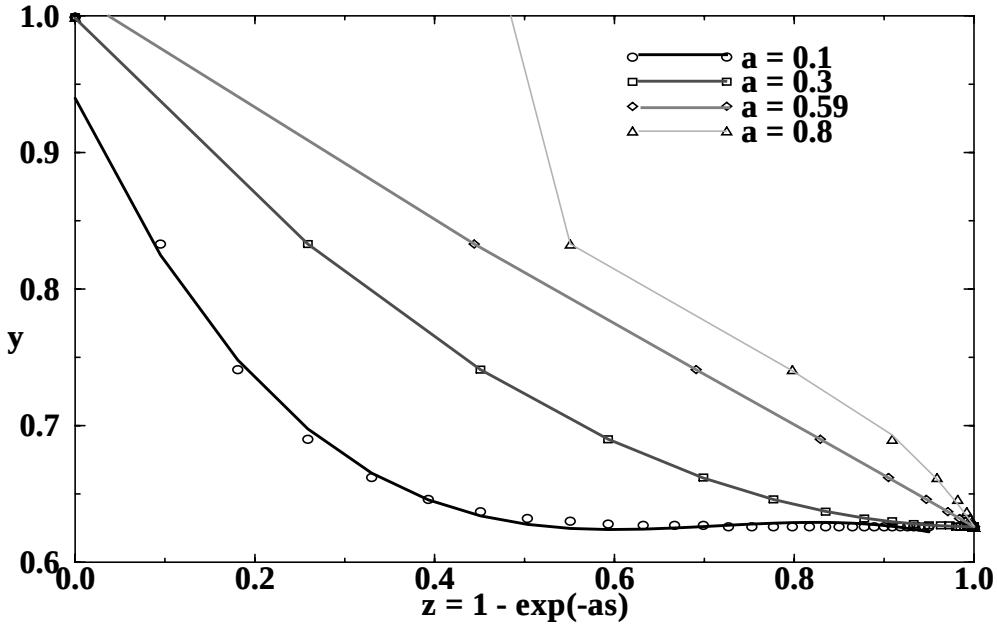

Figure 2.8. Effect of choice of *a*

Figure 2.9 shows column trajectories for a three component, constant relative volatility, constant molar overflow column with fifty trays above and below the feed, for three different reflux ratios, generated by a tray-by-tray model. Over the range of reflux ratios, *r*, we tested the collocation model, comparing the composition of the distillate product to the tray-by-tray calculation. The collocation model used had two collocation sections per column section, with two trays in each collocation section. We compared the *s* based and *z* based collocations over a range of point placements, using the point placements described in the last section. Figure 2.10 shows the average errors of both cases over the same range of point spreads for different reflux ratios. The *z* based collocation had lower average errors for every reflux ratio. For all but the lowest reflux ratio, the best solution was achieved with the z based collocation.

The second variable transformation is one on the mole fractions. When distilling to high purity, mole fractions go to one or zero, again flattening out the

trajectory. We need to transform the asymptotic approach to zero and one into a decreasing and increasing function that can be fitted by the polynomial used in the collocation. We use the following transformation:

$$2x_i - 1 = \tanh(\hat{x}_i) \qquad \textbf{(2.31)}$$

As the mole fraction goes to one or zero, the transformation variable, $\hat{x}_i$, goes to negative infinity and plus infinity. Figure 2.11 shows the effect of this transformation. For exponential approach to one and zero, the transformation straightens the trajectory out, so the slope never goes to zero.

Without this transformation, modeling sharp splits is very difficult. As the mole fractions of some of the components approach zero or one, the polynomial will create a curved trajectory, "bouncing" off the boundary. It becomes impossible to model a column with a component going to a mole fraction of 10^{-6}

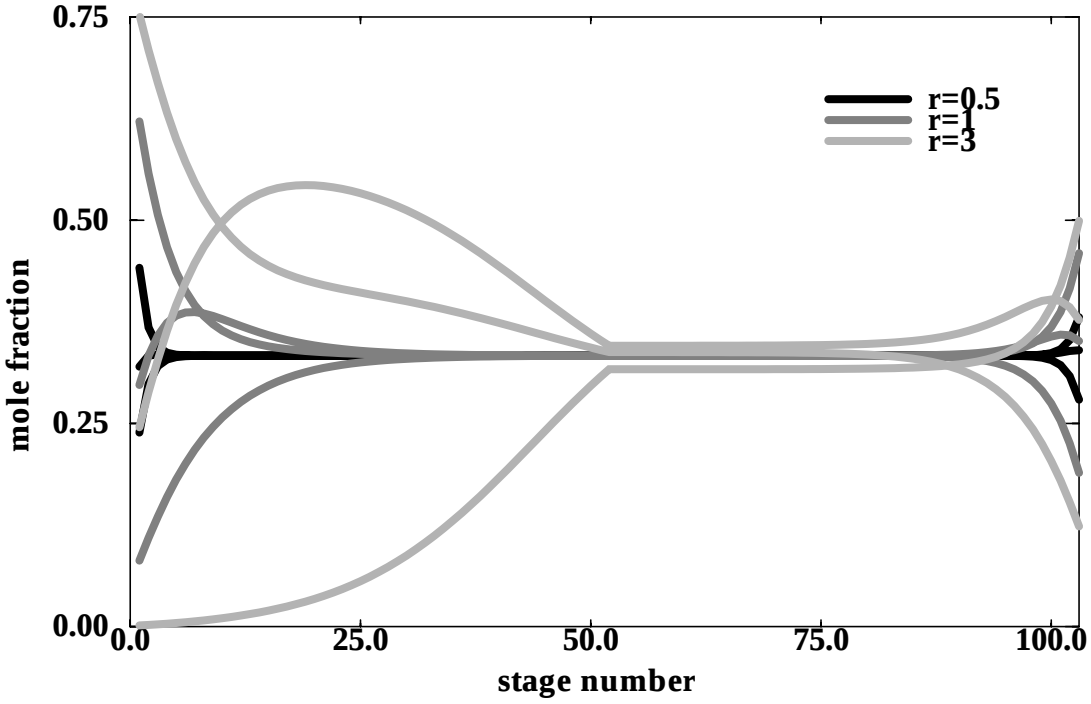

Figure 2.9. Column trajectories for a large column over a range of reflux ratios

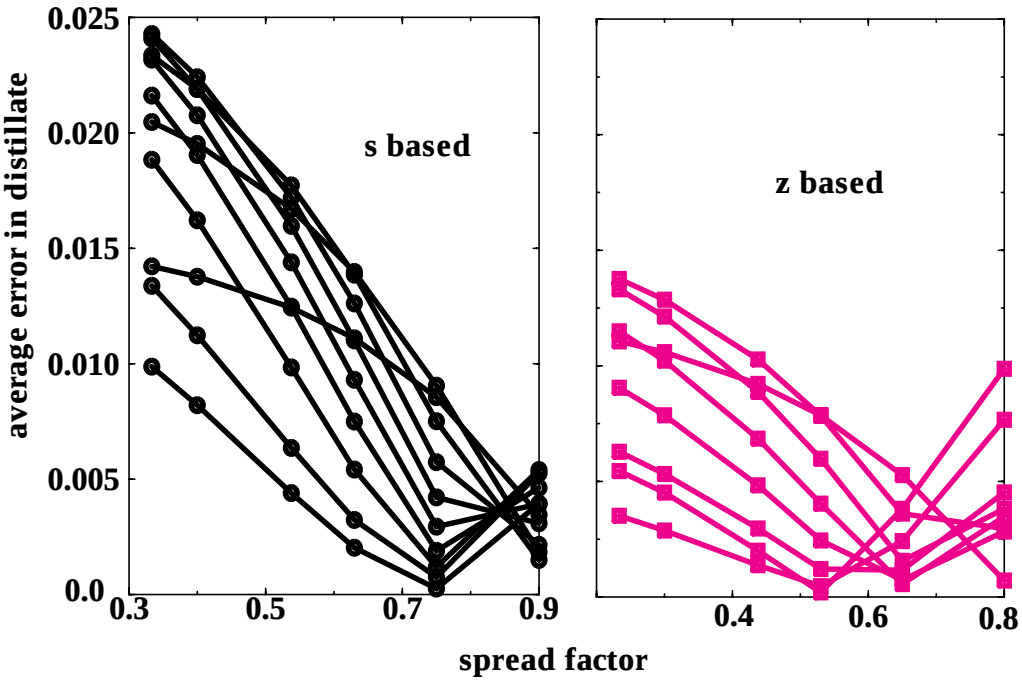

Figure 2.10. Error curves for *s* and *z* based collocation

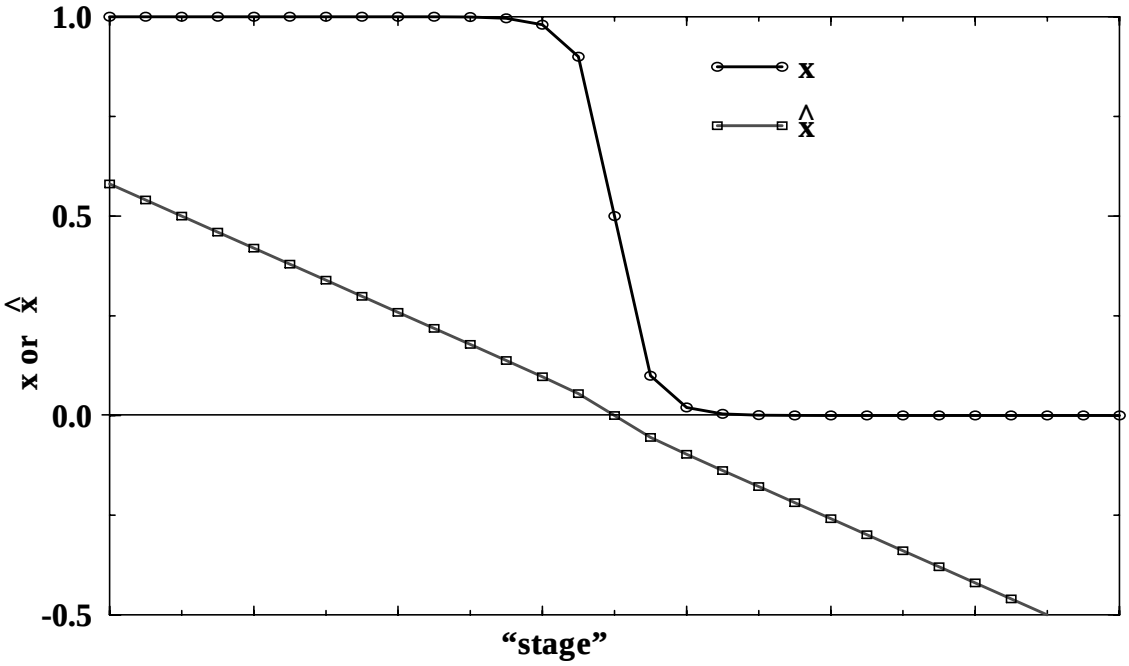

Figure 2.11. Effect of transformation on mole fraction, *x*

or smaller. However, the above transformation will give the polynomial room to move and will allow an asymptotic approach to the boundary. Figure 2.12 shows the four possible models of a 63 tray column which is removing all of the heavy component from the distillate. Figure 2.13 is a blowup of the trajectories for the heavy component near the top of the column. The two simulations without the transformation on mole fraction are curved and “bounce” up. The two solutions using the transformation smoothly approach the top of the column.

Figure 2.14 shows the combined benefits of the two variable transformation, showing two collocation models of a 103 tray column with high purities. Both models used the transformation on mole fraction, since this problem will not converge without it. For one, the polynomial is based on stage number and, for the other, the polynomial is based on the transformed stage number. The s based solution has high curvature in the bottom half of the column. This also demonstrates why it is beneficial in an s based collocation for the

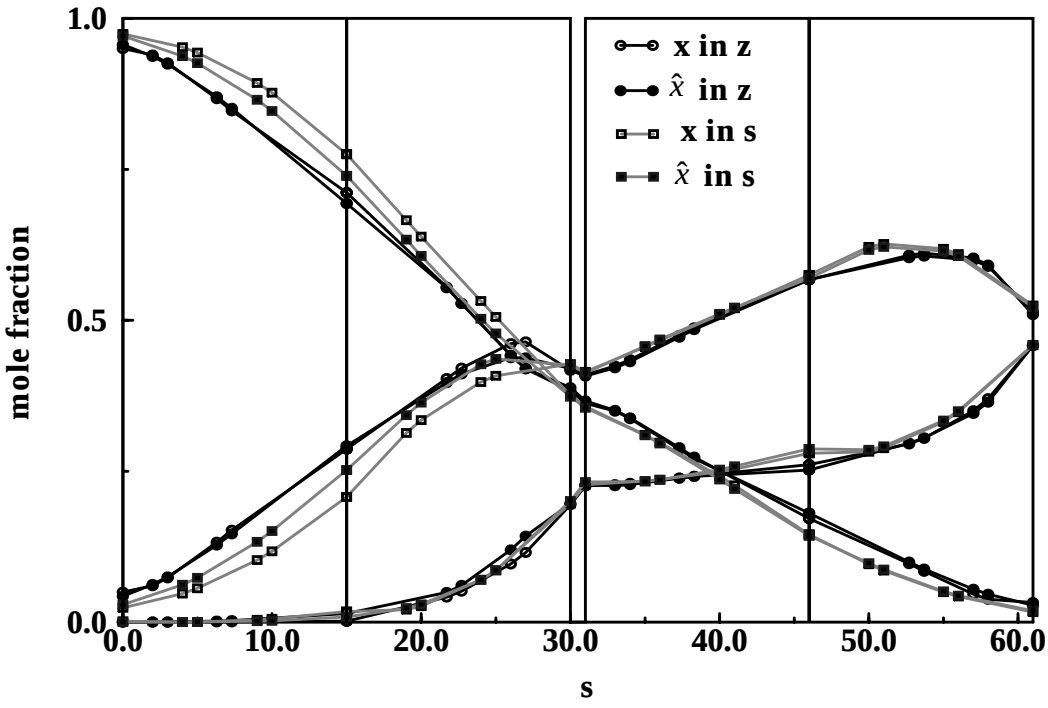

Figure 2.12. Effect of *x* transformation in a column simulation

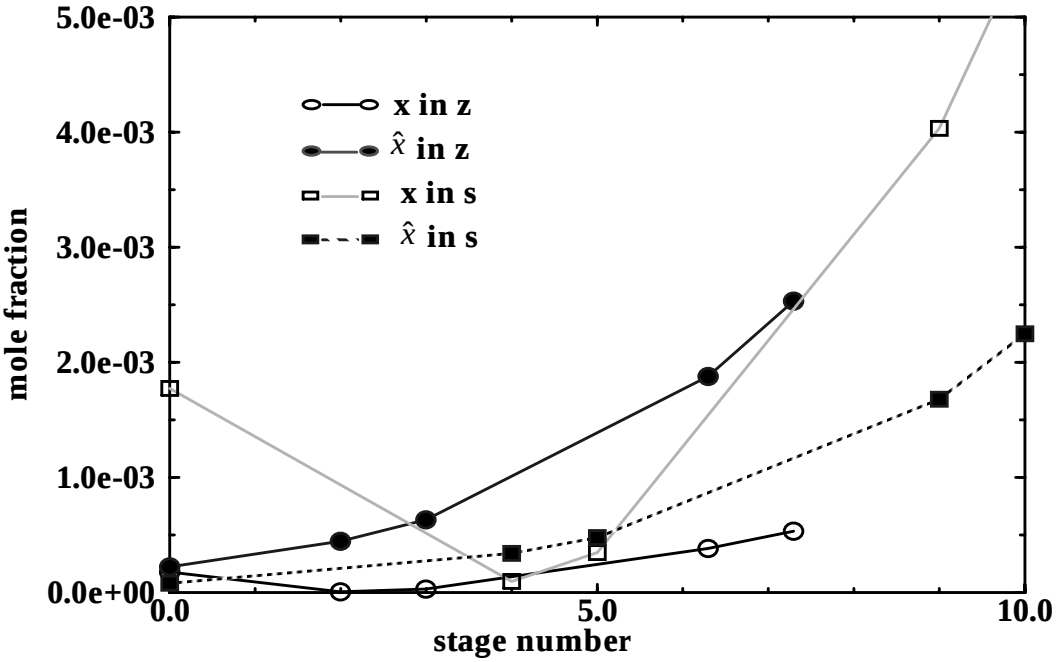

Figure 2.13. Blowup of lower left corner

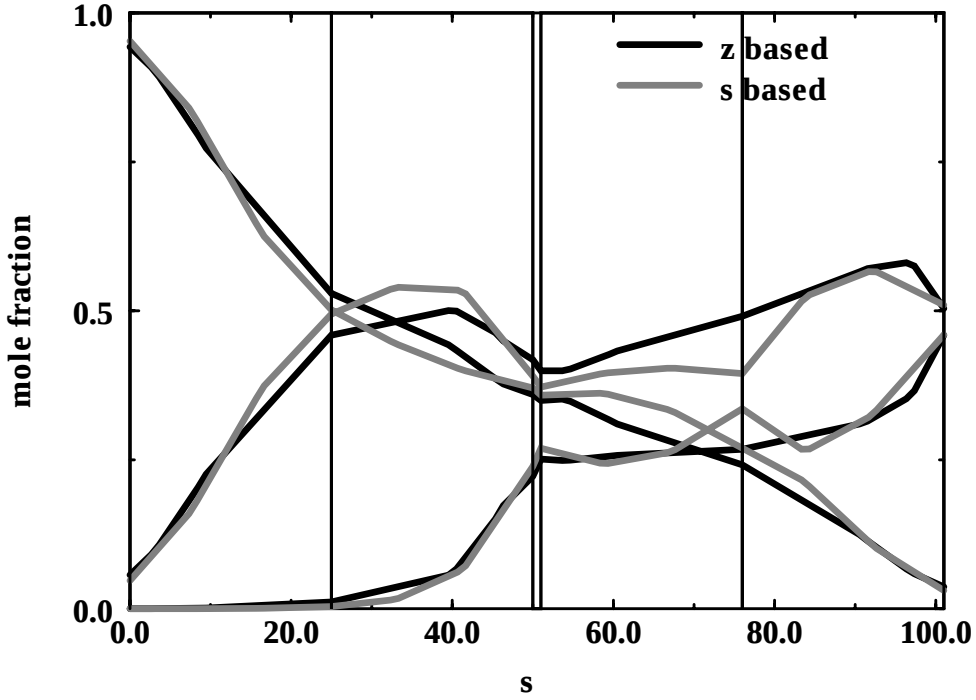

Figure 2.14. Effect of s transformation on a column with many trays

collocation points to be spread towards the ends of the collocation section to keep the curvature to a minimum.

2.7 Formulation of Collocation Column Model

Our standard formulation of the collocation column model is shown by Figure 2.15. Each column section is divided into two collocation sections with two trays each. Sererflis and Hrymak[1994a] used multiple collocation sections so they could use more collocation points in specific areas of the column where the temperature and composition profiles changed rapidly. We observe that the areas of activity are at the ends of column sections. By breaking each column section into two parts, and using the transformation on stage number, we de-emphasize the center of the column section by numbering the top collocation section downward and the bottom collocation section upward. With this transformation, the points at the beginning of a collocation section are stressed, and those at the end are less important. Therefore, for large columns with relatively low reflux ratios, the collocation points will be located where the compositions are changing, and the area of no activity will join the two collocation sections, but no collocation points need to be located there.

This standard formulation is sufficient for modeling large columns with reasonable accuracy and is small enough to model small columns without overkill. The model has four collocation sections with two trays each, a feed tray, a condenser, and a reboiler. This is eleven tray calculations. Since the collocation trays are not connected the way they would be for a tray-by-tray model, and since there are polynomial equations, there are more equations than there would be for an eleven tray column model. For a three component system, the collocation model has 1811 equations and variables, including all thermodynamic equations. A tray-by-tray model with 19 trays has 1856 equations and variables. For a four component system, the collocation model has 2215 equations and variables compared to 2205 for a tray-by-tray model with 18 trays. For a set number of trays

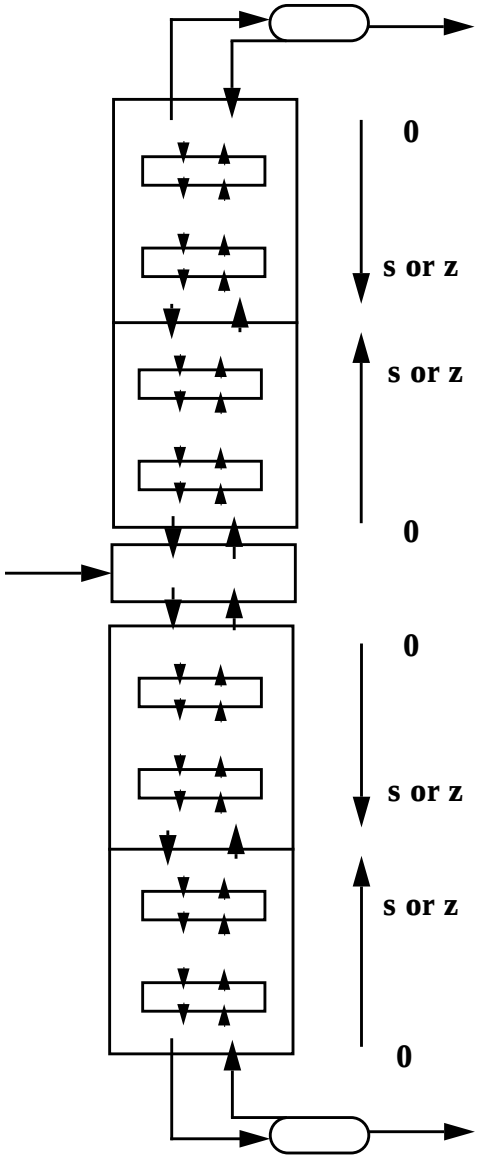

Figure 2.15. Column configuration

below 18, a tray-by-tray model might be more efficient, or a nonstandard collocation model can be used with fewer collocation sections.

We space the collocation points in each section using two parameters as described in the previous section.The parameter f_{mid} sets where the center of the collocation points is relative to the actual center of the collocation section, and f_{int} sets how spread out the points will be. For two and three tray collocation sections,

this formulation can emulate Jacobi or Hahn placement, and is easy to use.

2.8 Testing the Collocation Model

We have performed several tests of the collocation model using nonideal thermodynamics. In each of the following examples, the standard collocation model was used, with four collocation sections of two trays each. We used UNIFAC liquid mixture and Pitzer vapor mixture models for the thermodynamics and equilibrium, and assumed constant molar overflow. We would like to note that performing tests like these is a nontrivial task. The process of obtaining a full thermodynamic model is complex, but once a tray-by-tray collocation model has been successfully refined and converged, it is relatively easy to perform many sequential incremental changes to obtain a wealth of data. The two examples below where we performed a series of calculations to determine the binary separations over a range of operations required 50 solutions of the tray-by-tray and collocation models. Most of the work was done in getting that first useful solution. Then the models could be resolved repeatedly as the distillate to feed ratio was increased incrementally. The collocation has many parameters that can be adjusted, but it is much more robust than a tray-by-tray model. The process of solving these models will be discussed further in a two follow-up papers.

The first example is the separation of a 50/50 mixture of methanol and water. The column has 46 trays, and a reflux of 1.0. The purity of each product is 99.6%. Figure 2.16 shows a comparision between the tray-by-tray solution and the collocation model. The curves are the tray-by-tray, and the points are the collocation. The figure shows an excellent fit. Including all the thermodynamic calculations, the tray-by-tray and collocation models had 3255 and 1407 equations respectively. The error in the distillate composition is 0.02 percent.

Using the acetone, chloroform, benzene system, we performed many tests of the collocation model. The feed was a 36/24/40 mixture of acetone,

chloroform, and benzene. A 33 tray column with a reflux ratio of 4.0 was used. We performed a search over the range of distillate to feed ratios to find the maximum binary separations, as done by Wahnschafft [1992]. Figure 2.17 shows the comparison of the binary separation range plots with those for a tray-by-tray calculation. The chloroform benzene binary separation factor is not meaningful before a D:F of 0.3, since practically nothing of either component is coming out of the distillate at low D:F. For the acetone-benzene binary separation factor curve the average error was 1%, and for the acetone-chloroform separation factor curve the average error was 3%. The collocation shows very good agreement with tray-by-tray calculations. The error in the acetone concentration in the distillate was less than 2% over the range of D:F ratios, with an average error of 1%. Figure 2.18 shows comparisons of three different column simulations on a ternary diagram. Including all the thermodynamic calculations, the tray-by-tray and collocation models had 3144 and 1811 equations respectively.

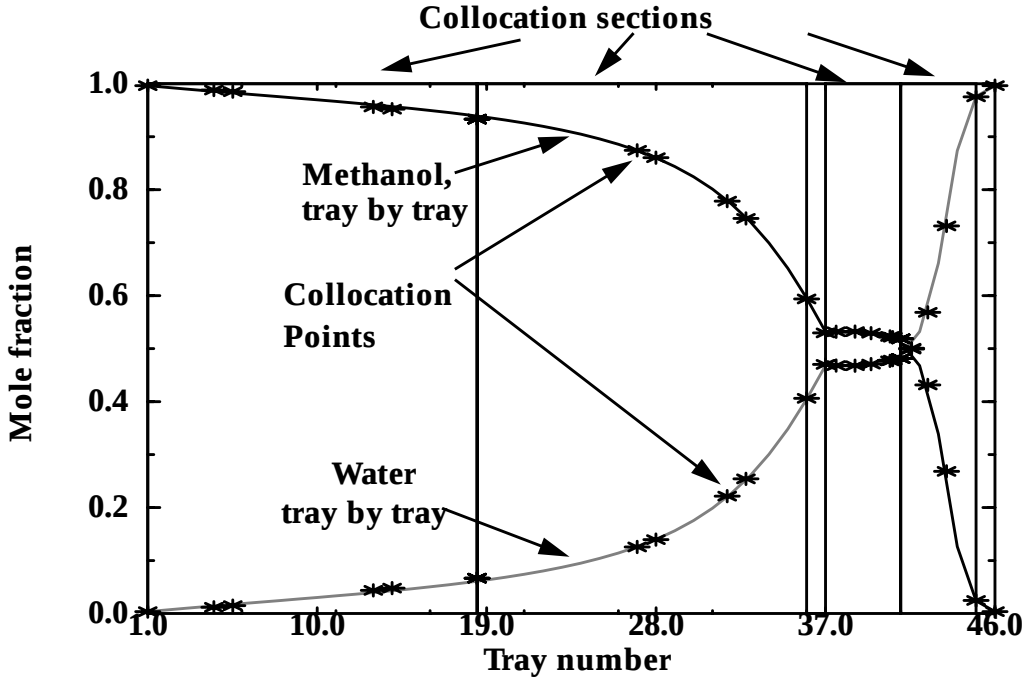

Figure 2.16. Comparison of collocation to rigorous model for methanol-water column

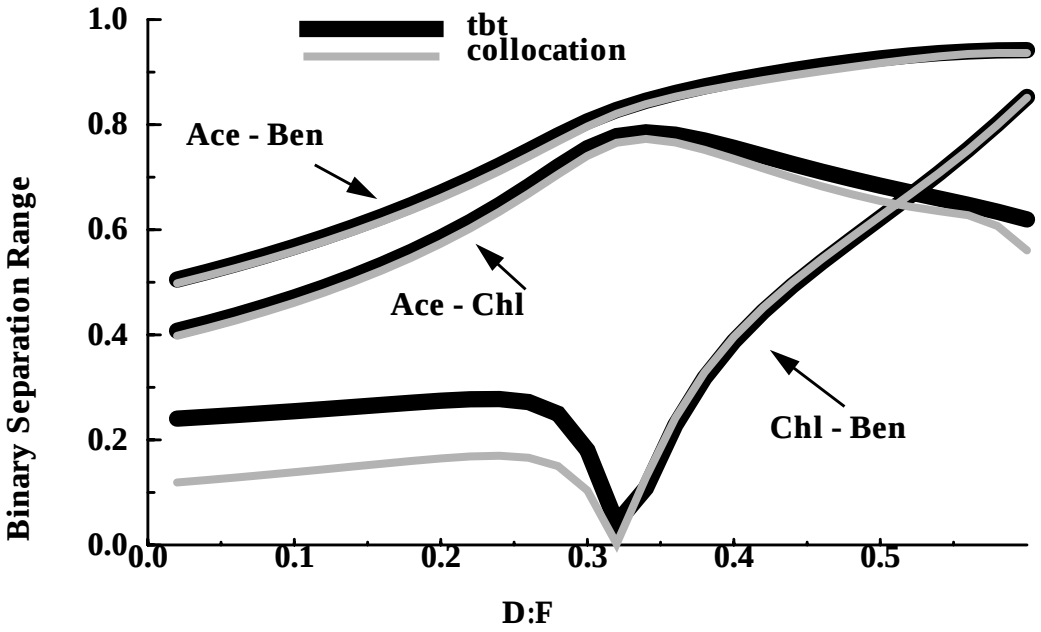

Figure 2.17. Comparison of collocation to rigorous model of separation range over D:F ratio for acetone-benzene-chloroform system

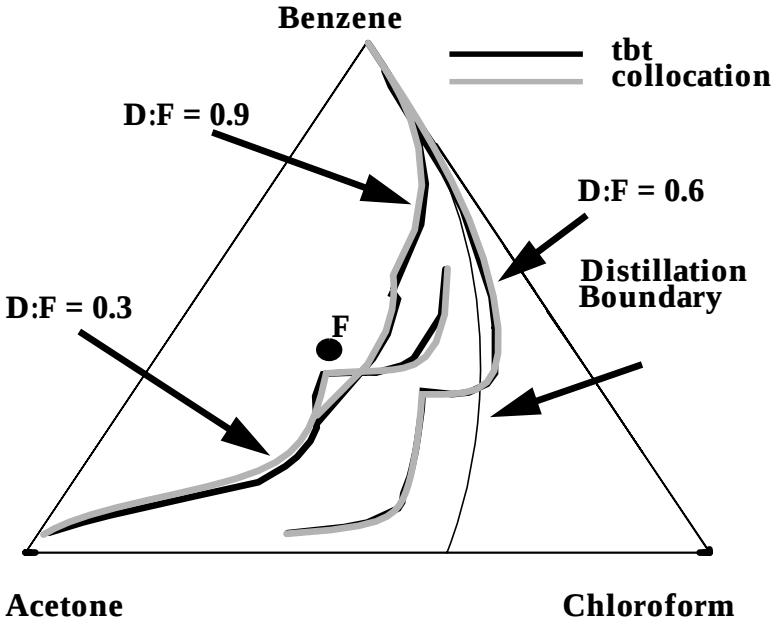

Figure 2.18. Comparison of collocation to rigorous model for acetone-benzene-chloroform column, for three different D:F ratios

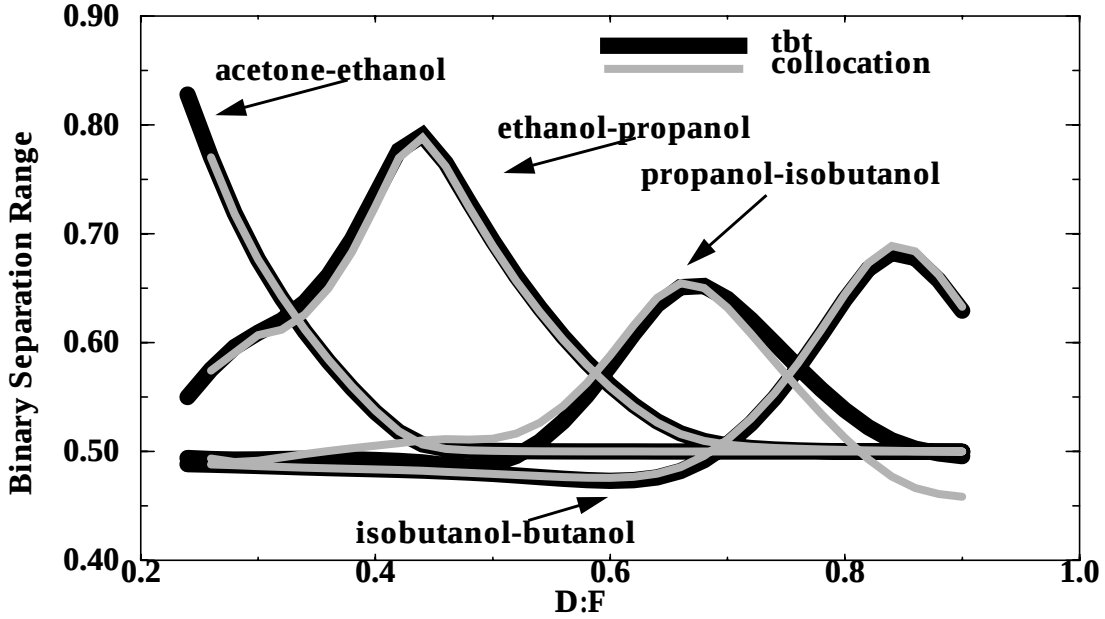

Figure 2.19. Comparison of collocation to rigorous model of separation range over D:F ratio for acetone-ethanol-propanol-isobutanol-butanol system

Finally, we performed a set of tests on an equimolar mixture of acetone, ethanol, propanol, isobutanol, and normal butanol, using a 23 tray column with a reflux ratio of 0.8. Figure 2.19 shows the comparison of the binary separation range plots for the components which are adjacent in the order of relative volatility. The other binary separation ranges compare equally well but would clutter the figure.

2.9 Conclusions

In this paper, we have demonstrated that this new collocation method can accurately reduce the order of column models. The two variable transformations greatly expand the capabilities of standard collocation methods. We have found that the degrees of freedom selection is important, and demonstrated what equations can be ignored. The choice of point placement is non-trivial, and no particular polynomial will give optimal point placement. Variable transformations more significantly reduce errors than proper point placement.

In two companion papers, we will discuss how collocation provides the missing link for simulation of minimum reflux conditions. We will also discuss a design algorithm for designing arbitrary columns using the collocation model.

Acknowledgment

Support from Eastman Chemicals and DOE contract number DE FG02-85ER13396 provide support for this research. Facilities support is from NSF grant number EEC-8942146 which supports the EDRC, an NSF funded Engineering Research Center.

Nomenclature

A	Simplification variable for Kremser approximation (L / KV)
α	Mole fraction average relative volatility
a	Parameter for exponential transformation of stage location
α	Parameter of Jacobi polynomial
α_i	Relative volatility of species i
β	Parameter of Jacobi polynomial
f_{int}	Factor for selection of spread of collocation points
f_{mid}	Factor for selection of the midpoint of collocation points
h	Liquid molar enthalpy
H	Vapor molar enthalpy
K	Equilibrium constant used in Kremser approximation
L	Liquid molar flowrate
n	Order of polynomial
n_c	Number of components
n_p	Number of phases
n_s	Number of collocation points
P	Pressure
P_n	Jacobi polynomial of order n
s	Stage location
T	Temperature
V	Vapor molar flowrate
W_k	kth term of Lagrange polynomial
w_k	Position of bottom of tray k
w_{mid}	Midpoint for placement of collocation points
$wtop_k$	Position of top of tray k
x_i	Liquid mole fraction of component i
$\hat{x}_i$	Transformed mole fraction
y_i	Vapor mole fraction of component i
z	Transformed stage location

References

Carnahan B., Luther H. A., Wilkes J. O., 1969, Applied Numerical Methods, Wiley, New York.

Cho Y.S. and B. Joseph., 1983. "Reduced-Order Steady-State and Dynamic Models for Separation Processes." *AIChE J.* **29**, 261-269, 270-276

Julka V., and M. F. Doherty, 1990. Geometric Behavior and Minimum Flows for Nonideal Multicomponent Distillation. *Chemical Engineering Science*, **45**, p. 1801-1822.

Koehler J., P Aguirre, and E. Blass., 1991. "Minimum Reflux Calculations for Nonideal Mixtures Using the Reversible Distillation Model." *Chem. Eng. Sci.*, **46**. 3007-3021.

Levy S. G., D.B. Van Dongen, and M. G. Doherty., 1985"Design and Synthesis of Homogeneous Azeotropic Distillations II: Minimum Reflux Calculations for Nonideal an d Azeotropic Columns." *Ind. and Eng. Chem. Fun.*, **24** 463-474

Seferlis P and A. N. Hrymak, 1994, 'Optimization of Distillation units using collocation models', *AIChe J.* **40**, 813-825

Seferlis P and A. N. Hrymak, 1994, "Adaptive Collocation of Finite Elements Models for the Optimization of Multi-Stage Distillation Unit," *Chem. Eng. Sci.*. **49**, 1369-1382

Stewart W. E., K. L. Levien, and M. Morari. 1984. "Collocation Methods in Distillation." in Westerberg, A.W., and H.H. Chien (eds), Proc. 2nd Intn'l Conf. Foundations Computer-aided Process Design (FOCAPD'83), CACHE Corp, Ann Arbor, MI, 535-569.

Swartz C. L. E. and Stewart W. E. 1986, "A Collocation Approach to Distillation Column Design," *AIChE J.* **32**, 1832-1838

Swartz C. L.E. and Stewart W. E. 1987, “Finite-Element Steady State Simulation of Multiphase Distillation,” *AIChE J.* **33**, 1977-1985

Underwood, A. J. V. *J. Inst. Pet.* 1945, **31**, 111

Underwood, A. J. V., *J. Inst. Pet.* 1946, **32**, 598

Underwood, A. J. V., *Chem. Eng. Prog.* 1948, **44**, 603

Wahnschafft O. M. 1992, “Synthesis of Separation Systems for Azeotropic Mixtures with and Emphasis on Distillation-Based Methods,” Ph.D. Thesis, Dept of Chem. Eng., Technical University of Munich.

Appendix

Rewriting the Kremser approximation to gather terms with *s* produces the following equation.

$$y_s = \frac{y_1 - A\hat{y}_0}{1-A} - \frac{y_1 - \hat{y}_0}{1-A}A^s \tag{2.32}$$

Rewriting the variable transformation

$$z = 1 - e^{(-as)} \tag{2.33}$$

for *s* in terms of *z*,

$$s = ln\left((1-z)^{-a} \right) \tag{2.34}$$

and placing it into the Kremser approximation produces the following equation.

$$y(z)) = \frac{y_1 - A\hat{y}_0}{1-A} - \frac{y_1 - \hat{y}_0}{1-A}A^{ln\left((1-z)^{-a} \right)} \tag{2.35}$$

If we define $A = exp(B)$, then we can take the following steps

$$\begin{aligned} A^{ln\left((1-z)^{-a} \right)} &= (exp(B))^{ln\left((1-z)^{-a} \right)} \\ &= exp\left(ln\left((exp(B))^{ln\left((1-z)^{-a} \right)} \right) \right) \\ &= exp\left(B\left(ln\left((1-z)^{-a} \right) \right) \right) \\ &= exp\left(ln\left((1-z)^{-aB} \right) \right) \\ &= (1-z)^{-aB} \end{aligned} \tag{2.36}$$

Now equation 2.35 becomes the following.

$$y(z) = \frac{y_1 - A\hat{y}_0}{1 - A} - \frac{y_1 - \hat{y}_0}{1 - A}(1 - z)^{-aB} \quad \textbf{(2.37)}$$

Therefore, when $-aB = 1$, the equation is linear in z.

CHAPTER 3

COLLOCATION METHODS FOR DISTILLATION DESIGN II: MINIMUM REFLUX

Robert S. Huss and Arthur W. Westerberg

Department of Chemical Engineering
and Engineering Design Research Center
Carnegie Mellon University
5000 Forbes Ave.
Pittsburgh, PA 15213

To be submitted to *Industrial & Engineering Chemistry Research*

Abstract

In this second paper on collocation methods for distillation design, we address the minimum reflux problem. Existing minimum reflux techniques do not work for nonsharp splits, as they use approximate geometric criteria to simplify modeling of minimum reflux conditions to a set of pinch point calculations for sharp splits. Collocation allows exact calculation of nonsharp minimum reflux and provides the missing link that geometric criteria replace.

We examine the behavior of distillation near minimum reflux and get results consistent with prior work. We also show that sensitivity of trace species drops as less sharp splits for the key components are sought.

We have found that collocation can be applied to sharp split minimum reflux problems, but we cannot exactly reproduce Underwood's values for constant relative volatility systems due to breakdown of the model near saddle pinches. We discuss the modeling of minimum reflux conditions with collocation and discuss the difficulties we have encountered.

We can accurately calculate minimum reflux for nonsharp splits for nonideal systems, even when approaching a sharp split. We overestimate the minimum reflux for the sharp split case, using nonsharp calculations at very low impurities. We encourage discussion of the problems we have encountered.

3.1 Introduction

Several researchers have explored and developed collocation for distillation column modeling. In the first paper in this series, we presented a collocation model which expands on prior models, addressing the problems specific to steady-state, continuous columns. In this paper we discuss use of the collocation method for minimum reflux determinations.

Minimum reflux determination is useful during the early stages of distillation design and for approximate design methods. Several methods of varying complexity and accuracy exist for determining minimum reflux. Underwood's [1945, 1946, 1948] method is very quick but is accurate only for ideal systems and only for sharp splits. As described in the motivation section of paper I in this series, Doherty and coworkers,[1990,1991] have done a great deal of work on distillation design and minimum reflux. Their methods rely on approximate geometrical conditions to remove the need for infinite column section calculations. The development of their techniques demonstrates how nonsharp splits should be handled, but their minimum reflux algorithms were developed only for sharp splits.

For sharp splits, Koehler et al. [1991] developed a minimum reflux determination method that uses a reversible distillation model. They have shown that for nonideal and azeotropic systems their method provides accurate results, based on comparisons to rigorous column models with a large number of trays. However, their method requires the nonkey components to be nondistributing and is based on the same geometric considerations in Doherty and coworkers.

We present an approach for minimum reflux calculations that does not require approximate geometric constraints. We discuss some interesting insights as one moves from nonsharp to sharp splits. We also discuss computational difficulties we found in realizing the theory with collocation models. By presenting both the strong and weak points of collocation for distillation

modeling we hope to initiate discussions leading to better methods.

3.2 Pinch Points

When a distillation section has a very large number of trays in it, a region of constant composition occurs. The liquid and vapor passing each other approach being in equilibrium with each other, i.e., they approach being pinched. Minimum reflux occurs when at least one column section is pinched. When modeling minimum reflux conditions, “pinch point” calculations can sometimes remove the need for tray-by-tray models of column sections. A pinch point is calculated by requiring that the liquid and vapor passing each other are in equilibrium and in mass and heat balance with the other end of the distillation section. Therefore, if one half of a column has a large number of trays and is pinched adjacent to the feed tray, the whole section can be modeled with only a pinch point calculation. This simplification can be exploited for calculation of minimum reflux.

3.3 Nonsharp Splits

A separation where all the components are distributed in some amount to both products is nonsharp. In the case of a nonsharp split, there is a feasible intersection region where the tray by tray trajectories from the top and bottom product can intersect [Wahnschafft 1992]. Figure 3.1 shows a ternary diagram for a nonsharp minimum reflux problem. The regions between the infinite reflux (distillation) and minimum reflux (pinch point) curves for both the top and bottom products are the only feasible regions where the trajectories can go. The darkened region that shows the intersection of the feasible space for the top and bottom trajectories shows the only feasible region for the feed tray. At minimum reflux, the bottom section will pinch in this region. Figure 3.2 shows a blowup of this region. Any trajectory from the bottom of the column will terminate on the pinch point curve as the number of trays increases. As the trajectory from the bottom of the column moves to the right, the reboil ratio ($\overline{R}$) increases. However,

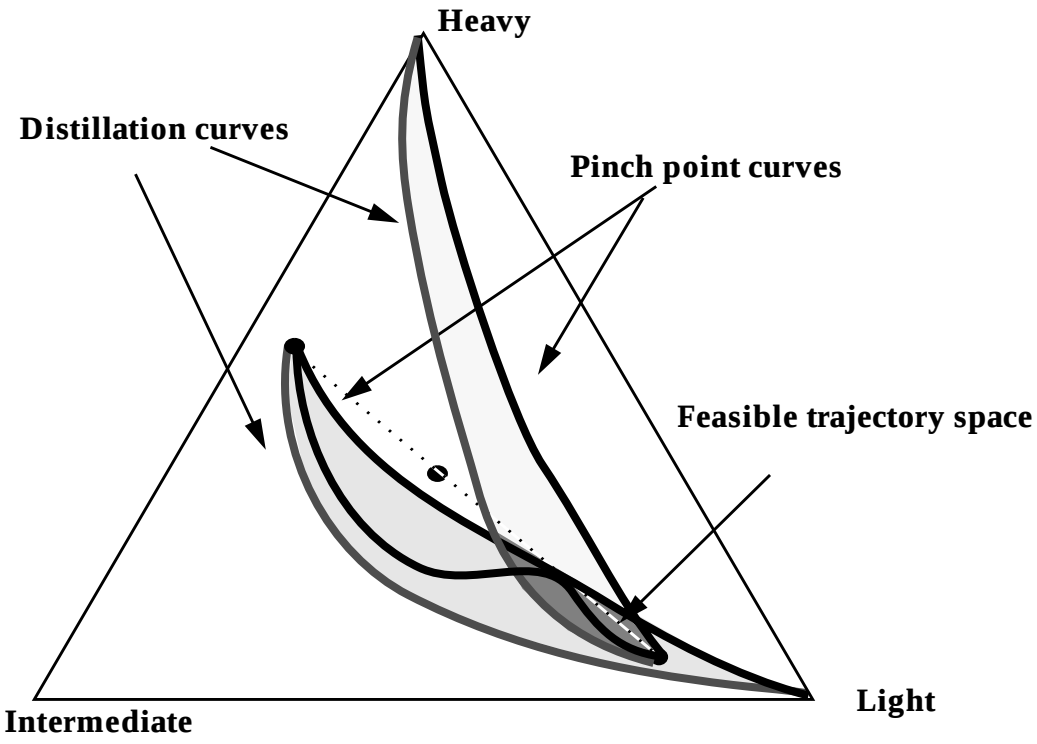

Figure 3.1. Pinch behavior for ternary nonsharp split

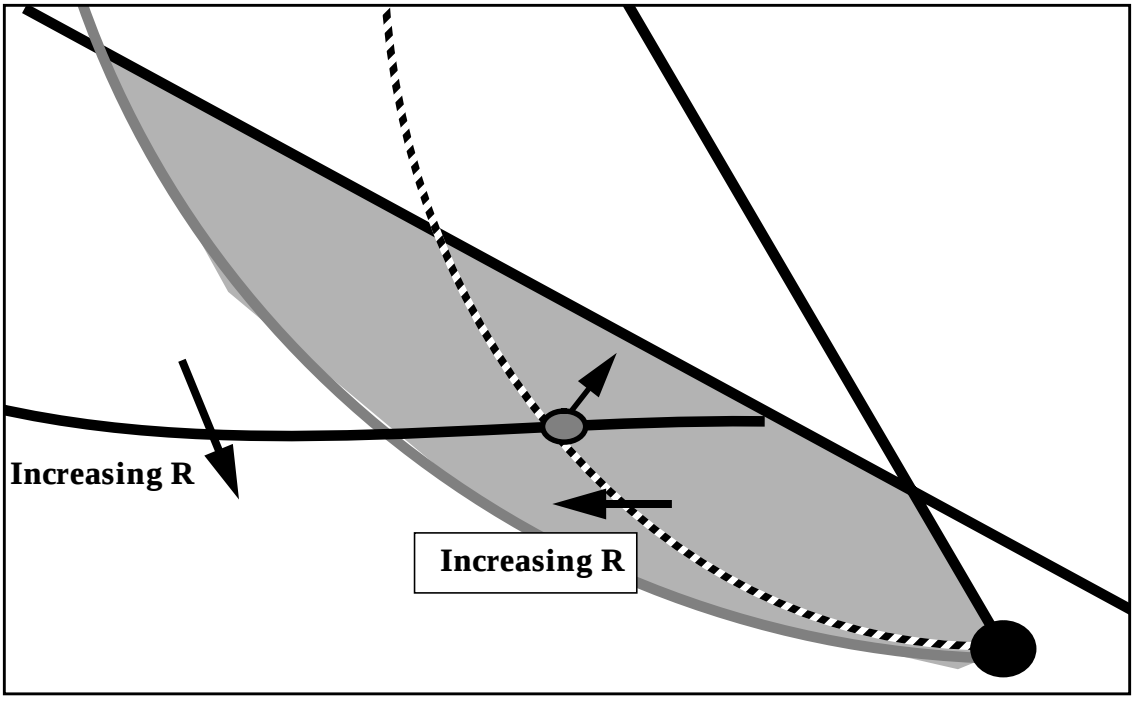

Figure 3.2. Blowup of Figure 3.1

for the top section, the pinch point curve, representing the lowest reflux ratio (R) is on the right, and the infinite reflux curve is on the left. Therefore along the pinch point curve for the top half, the left most point corresponds to minimum $\overline{R}$, but maximum R, and the right most point corresponds to maximum $\overline{R}$ and minimum R. However, R and $\overline{R}$ are linked by the mass balances and quality of the feed.

$$V = \overline{V} + (1-q)F \tag{3.1}$$

$$(R+1)D = \overline{R}B + (1-q)F \tag{3.2}$$

For any given feasible column, R and $\overline{R}$ will decrease or increase together. Assume that the grey point is a feasible intersection point. To decrease both R and $\overline{R}$ the column must move towards the pinch point curve. Therefore, we know that the nonsharp minimum reflux solution for this type of problem will have a pinched bottom section and a finite top section. At some point along this line, R and $\overline{R}$ will correspond to a feasible trajectory from the top of the column. The minimum requirement to model this system is a pinch calculation from the distillate, then a feed tray model, and then a tray-by-tray or reduced order model for the bottom half of the column. We use the standard collocation model in our first paper [Huss and Westerberg, 1995a] by enforcing a pinch at the junction of the two bottom collocation sections and calculating the number of trays in the top section.

Figure 3.3 shows how this modeling requirement maps onto the standard collocation model. We enforce a pinch between sections 3 and 4 by adding equations forcing the liquid and vapor passing each other at the junction between the two collocation sections to be in equilibrium and removing the requirement that the liquid stream at the pinch must be on the polynomial for section 4.

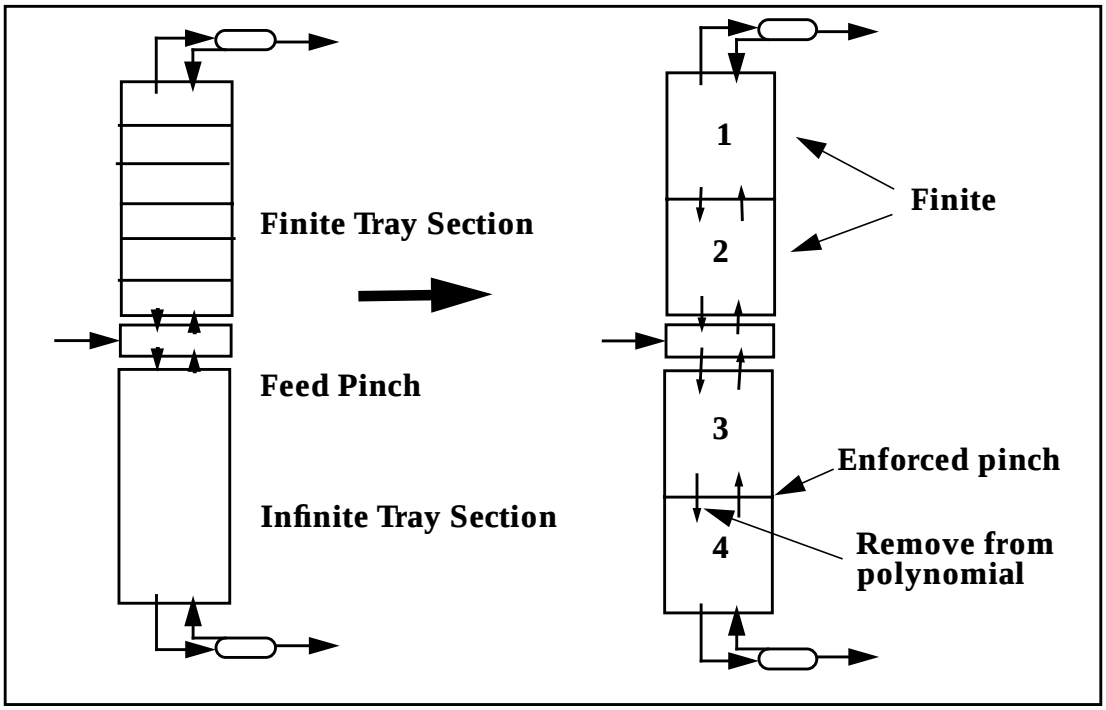

Figure 3.3. Mapping of nonsharp minimum reflux on to standard collocation model

Section 3 will have completely flat trajectories. We evenly split the number of trays needed in the top section between collocation sections 1 and 2. The number of trays in the top section is solved for when we completely specify the distillate. Note that this example is for a direct separation, splitting between the lightest component and the intermediate. For an indirect separation, splitting between the heaviest component and an adjacent intermediate, the column structure would be inverted, but the solution method the same.

3.4 Sharp Splits

When a sharp split is specified, one assumes that the non-key components are nondistributing. This forces a saddle pinch, since it takes an infinite number of trays to completely remove a component. Figure 3.4 shows the trajectory for such a case. When there is a saddle pinch, the pinch point is not adjacent to the feed because it takes an infinite number of trays to remove the nondistributed components from the feed as well as an infinite number of trays to get from the

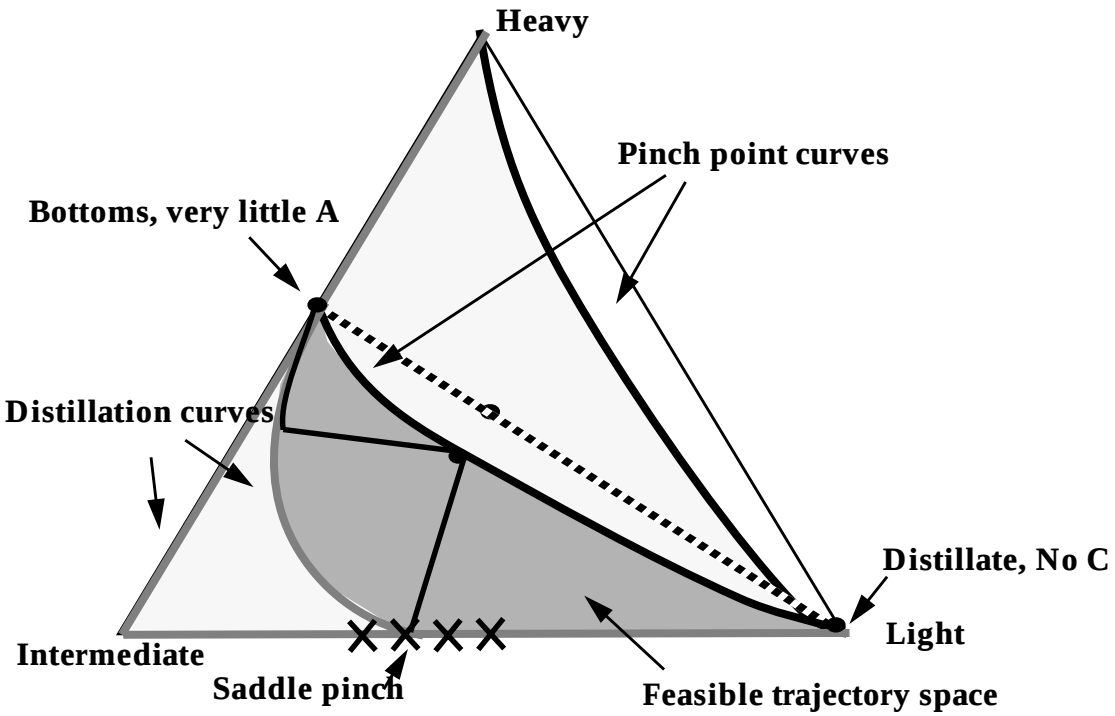

Figure 3.4. Pinch behavior for ternary sharp split

product to the pinch. When the pinch is not adjacent to the feed, a simple pinch point calculation is not sufficient. A pinch point calculation could determine the possible saddle pinch points, but any saddle pinch which is in mass balance with the top of the column will necessarily be in mass balance with the feed tray. The

X's in Figure 3.4 show multiple saddle pinch points. The pinch points move to the left as the reflux ratio increases. These points represent the binary pinch point curve between the light and intermediate. There is some missing connection between the correct saddle pinch and the feed tray. Levy et al. [1985] noticed that for ideal three component systems, the saddle pinch point, the feed tray pinch point and the feed composition are colinear for sharp splits. They used this colinearity to identify the correct saddle pinch point. For more components, Julka and Doherty [1990] required that a set of pinch points, the feed composition, and the feed tray must be in a minimum volume. Kohler et al. [1991] used a minimum angle criterion for multi-component systems to achieve the same effect.

The missing connection is actually an infinite set of stage-by-stage calculations connecting the saddle pinch and the feed tray, i.e., the saddle pinch and the feed tray must lie on the same tray by tray trajectory. We proposed that a collocation model could approximate these calculations. We describe the details of our collocation methods in full detail in the first paper of this series. When defining the polynomials in terms of stage location, it is impossible to model an infinite column section because the polynomials explode as the number of stages increases. However, by performing a variable transformation on stage number, as shown in equation 3.3, we can write the polynomials in terms of a bounded variable. The transformation variable z will go to 1 as s goes to infinity.

$$z = 1 - e^{(-as)} \tag{3.3}$$

We must show that a stage-by-stage trajectory from this pinch point leads to both the top product and the feed tray. Therefore, we need an infinite column section model. This can be achieved by collocating and doing a variable transformation on the tray number so that, as the tray number goes to infinity, the reference variable goes to some finite value.

Shortcut methods that require a sharp split determine a theoretical minimum reflux for the case where the non-key components are non-distributing. However, the actual column will not achieve this theoretical split. The minimum reflux for the actual separation will be different than that for the non-distributing sharp split. Levy et al. [1985] has shown that minimum reflux decreases sharply as the impurity of the heavy nonkey decreases from 10^{-5} to 10^{-11}.

Figure 3.5 shows how the sharp split problem maps onto the standard collocation model. It is very similar to the nonsharp case, but we want sections 1 and 2 to bring us as close to the saddle pinch as they can. We have found that we cannot enforce a saddle pinch in the same manner as the feed pinch. When slack

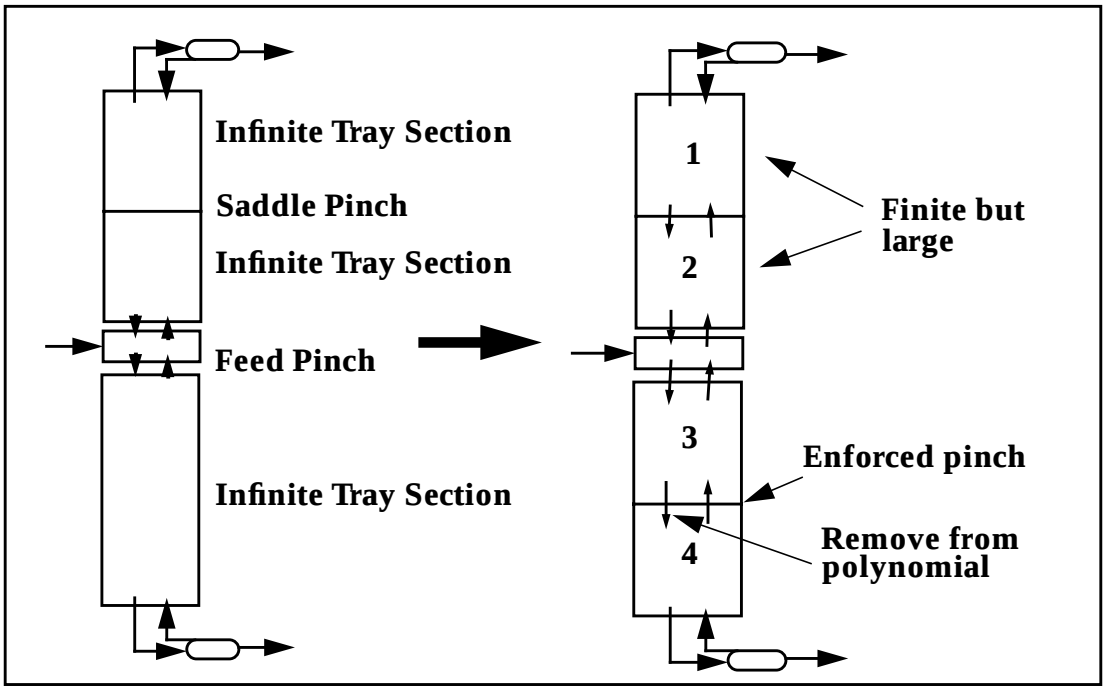

Figure 3.5. Mapping of sharp minimum reflux on to standard collocation model

variables are added to the equilibrium equations for the pinch point, we find that we can approach the pinch, but we cannot force the slack variables completely to zero. Inaccuracies occur that we cannot compensate for, which we will discuss later in the paper. Therefore, we approximate a sharp split minimum reflux by simulating a large column section. The number of trays needed to approximate an

infinite number of trays is dependent on the relative volatilities. For components with large relative volatilities, only a few trays are necessary to approximate an infinite number of trays. Also, for multicomponent problems where a saddle pinch occurs in both column sections, we use the standard collocation model with all four collocation sections as large as possible.

3.5 Trends of Large Column Sections

We did many examples with constant relative volatility to compare with Underwood's method. We tested three ternary systems over a range of separations. We examined systems with relative volatilities of 1.5, 1.2, 1.0;

3.0,2.0,1.0; and 9.0,3.0,1.0.

In each test, we did a direct split, forcing a pinch adjacent to the feed in the bottom of the column, and determining the reflux vs. the number of trays in the top section. For every case, it was possible to get exactly Underwood's value of the reflux ratio with some finite but large number of trays in the top. However, it was also possible to increase the trays and get lower than Underwood's reflux ratio. When increasing the trays to larger numbers, the trajectory became unsmooth, suggesting that the model was no longer accurate.

Figure 3.6 shows the different reflux ratios for various recovery specifications on the light and heavy key obtained by increasing the size of the top section of the column, for the relative volatilities of 3, 2, 1. The large points mark the number of trays and reflux required to get various levels of impurity of the heaviest component in the distillate. These represent nonsharp minimum reflux calculations. Beyond 10^{-9} is below the error tolerance so we cannot determine nonsharp minimum reflux for these points as anything other than the answer determined for 10^{-9}. We can continue to increase the number of trays and get smaller values for the reflux ratio. The straight lines indicate the Underwood value for the various recoveries. For each recovery specification, some number of trays will give us exactly Underwood's value, but for even more trays we go below Underwood.

Figure 3.7 shows how the minimum reflux decreases in increasing amounts as you reduce the impurity. The grey points at zero are the Underwood values. From this plot we can see that a straight line extrapolation from two points with impurities of 10^{-7} and 10^{-8} will overestimate the minimum reflux. The degree of the overestimation depends on the degree of separation between light and heavy key.

For nonideal systems, the techniques are the same as above. However, we

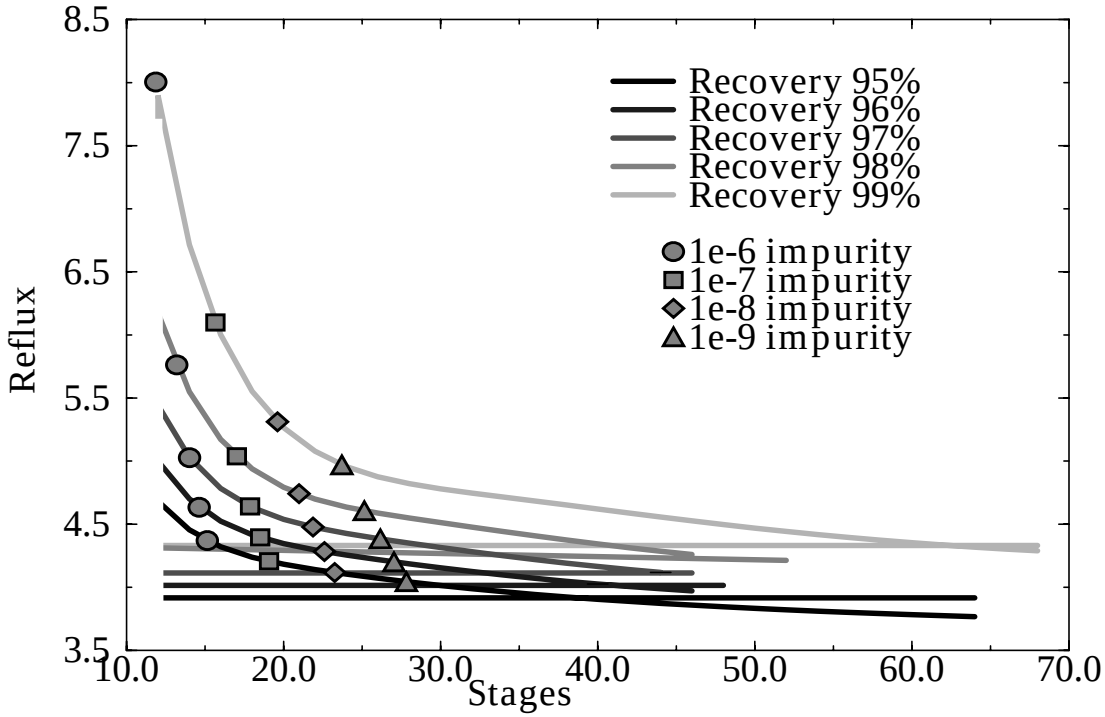

Figure 3.6. Increasing trays and reducing reflux Relative volatilities (3,2,1)

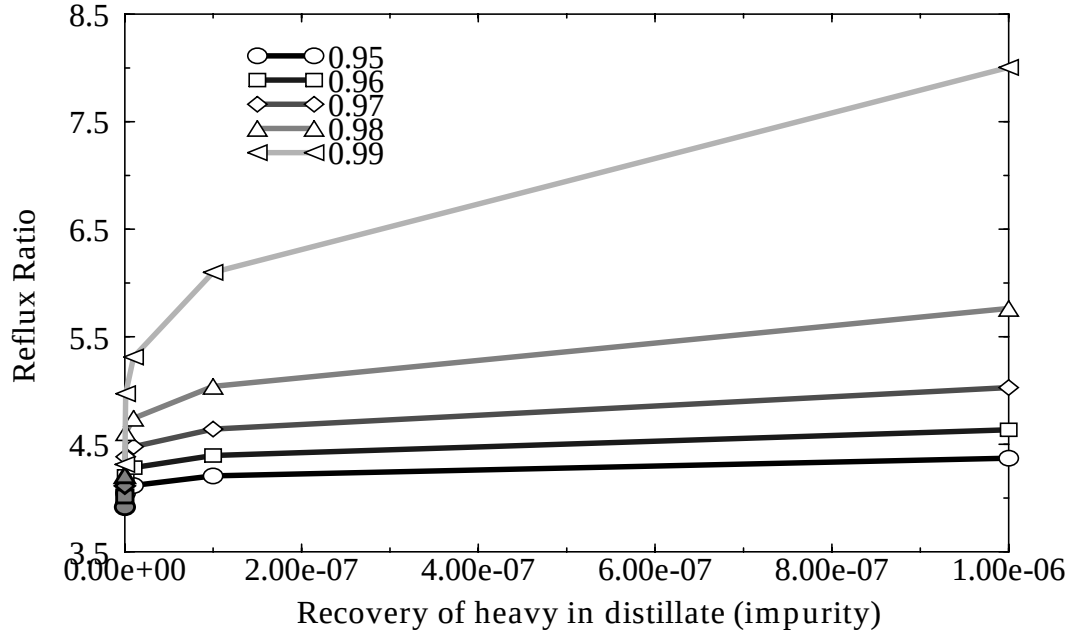

Figure 3.7. Reflux ratio vs. impurity of heavy component.

have no Underwood value to compare to. For nonideal systems we have successfully enforced pinches adjacent to the feed, allowing many nonsharp minimum reflux determinations. For sharp splits, the behavior was similar to the ideal examples, with inaccuracies forming at large numbers of trays. Figure 3.8 shows several column trajectories for nonsharp minimum reflux calculations for a propanol, isobutanol, butanol system, where propanol and isobutanol were the light and heavy key respectively. Recoveries of the light and heavy key in top and bottom respectively were 0.8, 0.9, 0.95, and 0.99 for the four diagrams. For each example, we varied the recovery of the heavy component, solving for the number of trays required in the top half of the column. The bottom half of the column was pinched adjacent to the feed, similar to the example shown in Figure 3.1. Note that the last curves of plots (a) and (b) demonstrate the poor trajectories that can occur when getting close to the saddle pinch.

Figure 3.9 shows the reflux ratio as a function of the impurity of the heavy component for each recovery. For lower recoveries the effect of the amount of impurity on the reflux ratio becomes smaller. For the 80% recovery example, the sharp split minimum reflux could be extrapolated to zero with very little error. However, for the 99% recovery the effect of the impurity is large, so we cannot be sure how much we are overestimating the minimum reflux for a sharp split.

Figures 3.7 and 3.9 demonstrate how nonsharp minimum reflux calculations can be useful for sharp split requirements. As Levy et al. [1985] showed, the reflux ratio can be very sensitive to the impurity of the nondistributing component. However, for lower recoveries, this sensitivity decreases significantly. Even for relatively high recoveries, we can extrapolate nonsharp minimum reflux calculations with low impurities and at worst slightly overestimate the minimum reflux ratio.

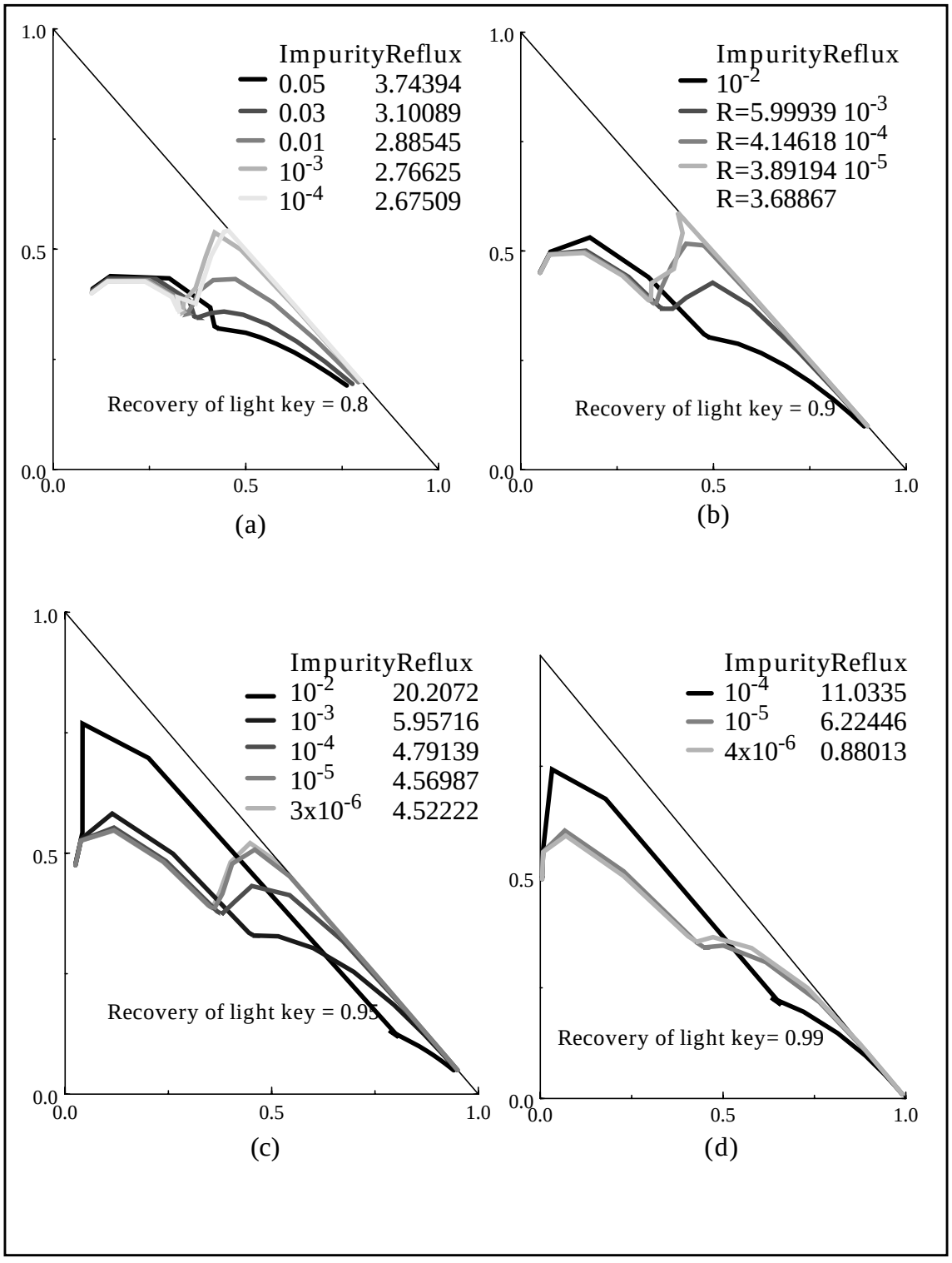

Figure 3.8. Several nonsharp minimum reflux calculations for propanol, isobutanol, butanol system, for varying recoveries and impurities of heavy component.

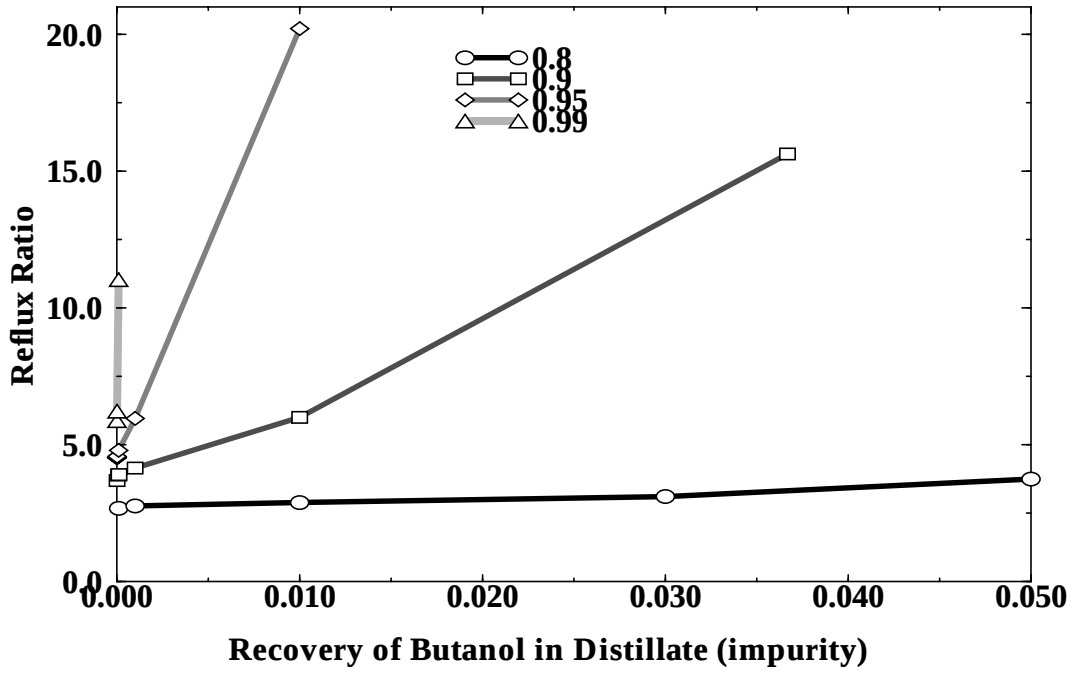

Figure 3.9. Reflux as a function of impurity of butanol for propanol, isobutanol, butanol system.

3.6 Error Detection

Figure 3.10 is an example of the inappropriate trajectories that can occur. It shows the trajectories over the transform variable *z*. The top of the column is on the left. The vertical lines show the boundaries of the collocation sections. The slope of the light component becomes positive at the top of the second collocation section. This should not occur. Once this ragged sort of trajectory occurs, the model can no longer be trusted. The model converges to a nonphysical solution.

Furthermore, even when the mole fraction trajectories appear smooth, the trajectories of the transformed mole fraction may not be smooth. As we describe in the first paper of this series [Huss and Westerberg, 1995a], we perform the following transformation on mole fraction.

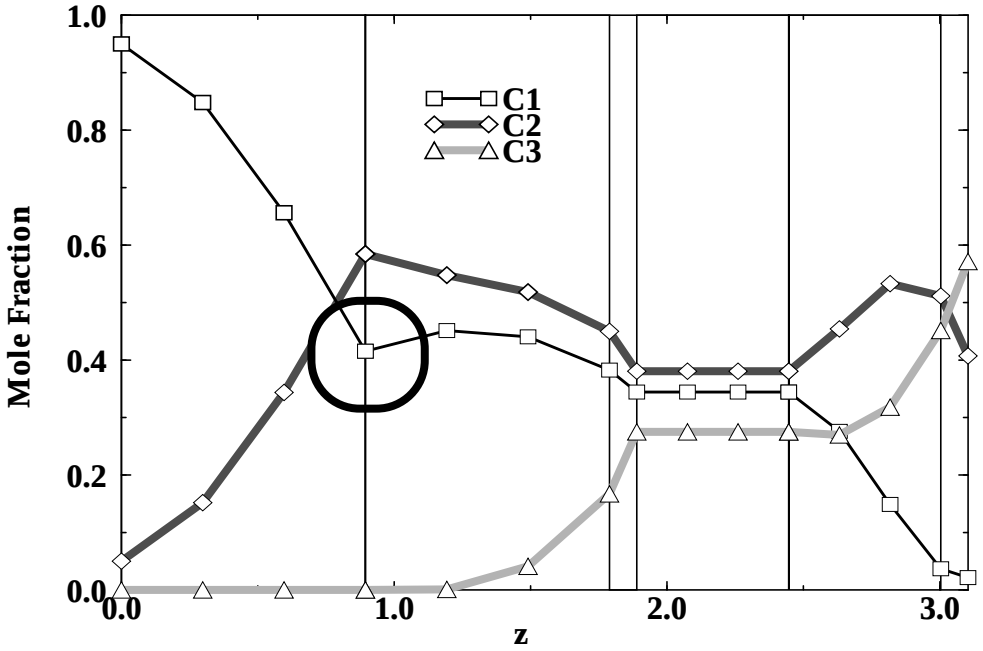

Figure 3.10. Example of inaccurate collocation of large column section. Relative volatilities (3, 2, 1)

$$2x_i - 1 = \tanh(\hat{x}_i) \qquad \textbf{(3.4)}$$

Figure 3.11 shows the mole fraction trajectories (**x**) and transformed mole fraction trajectories ($\hat{x}_i$) for the same column. The top figure looks fine, but the bottom one shows the error occurring in the trace component. One might believe that a mole fraction of less than 10^{-5} is not going to affect the model, but since we perform a variable transformation, re-emphasizing the mole fractions near zero and one, they can be very significant.

Even though we have these clear signs of inaccuracies, some of the constant relative volatility examples went below Underwood's value before the clear signs showed up. We created an extra tray in each section to test the accuracy. The test tray took its input streams from the polynomials, and we compared the output streams to the polynomial. We placed the test tray near the

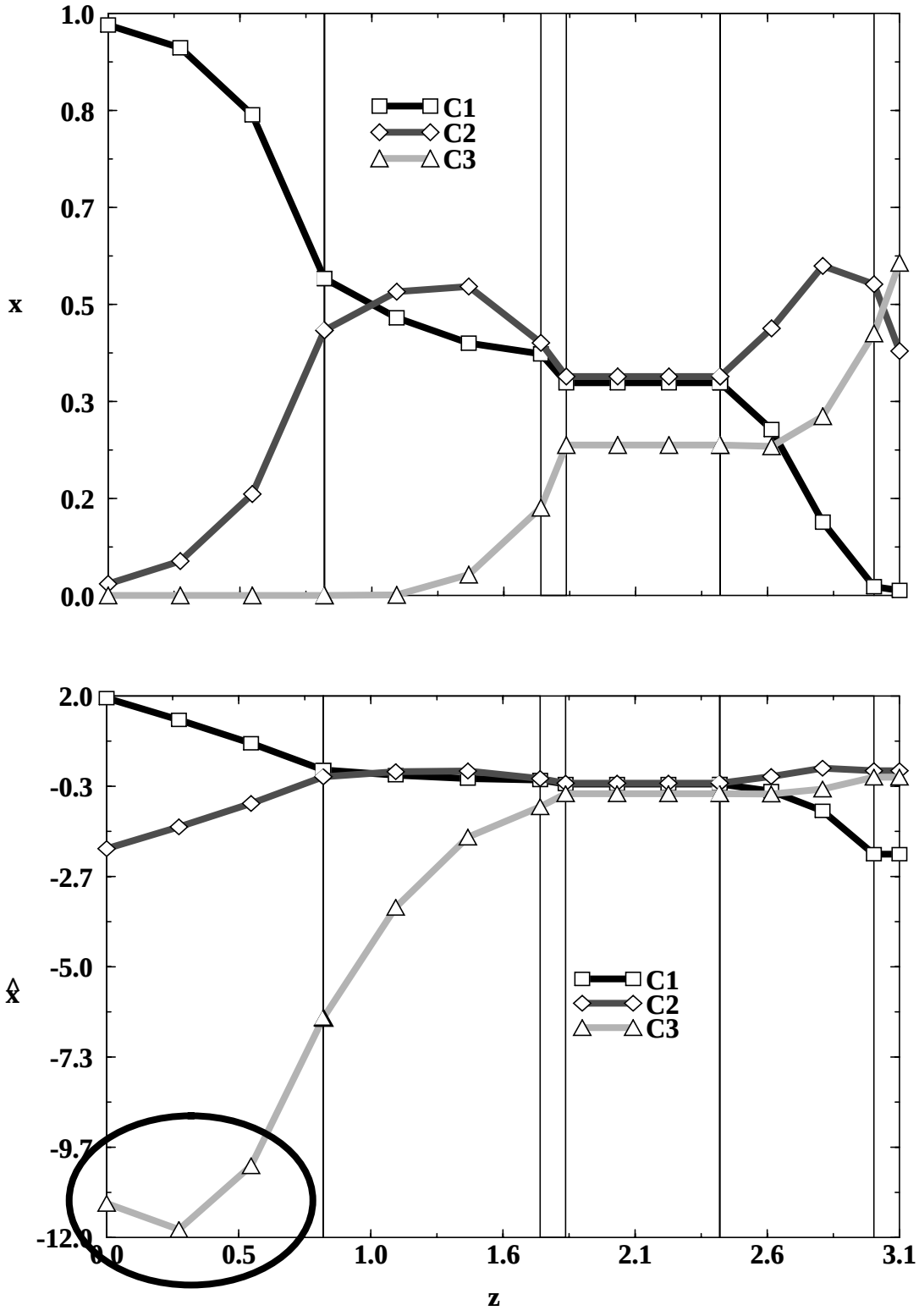

Figure 3.11. Example if inaccuracy in trace component. Volatilities (3, 2, 1)

end of the collocation sections adjacent to the expected saddle pinch point. Figure 3.12 shows the error in four different polynomials, for a column recovering 98% of the light key and 2% of the heavy key in the distillate. C1 and C3 were the active components. The figure shows that C3 in section 1 (the top section in the column) is most sensitive. This error reflects the behavior we see in Figure 3.11. The other errors don't show any erratic behavior as the number of trays increases. Figure 3.13 shows the error in C3 in section 1 for several recoveries. They all display the same behavior. The error in the trace component in top section of the column should be a good indicator of the onset of inaccuracies. Combined with requirements that the light component monotonically decreases going down the column, and that the transformed trace component monotonically increases going down the column, we should detect any problems.

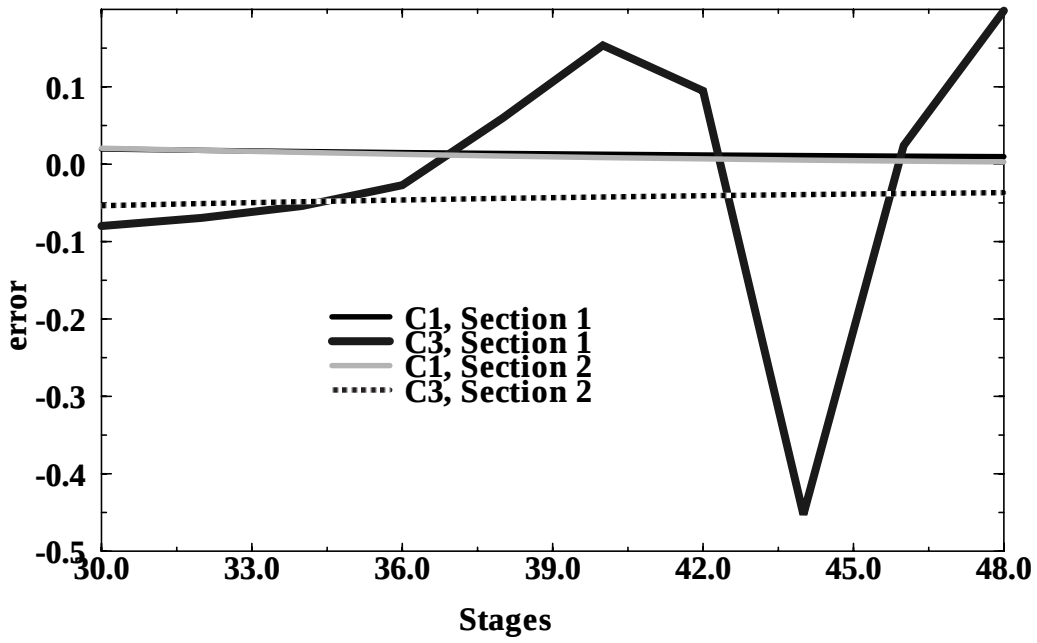

Figure 3.12. Error in polynomial prediction for isolated trays. Recovery of keys 98%, Relative volatilities (3, 2, 1)

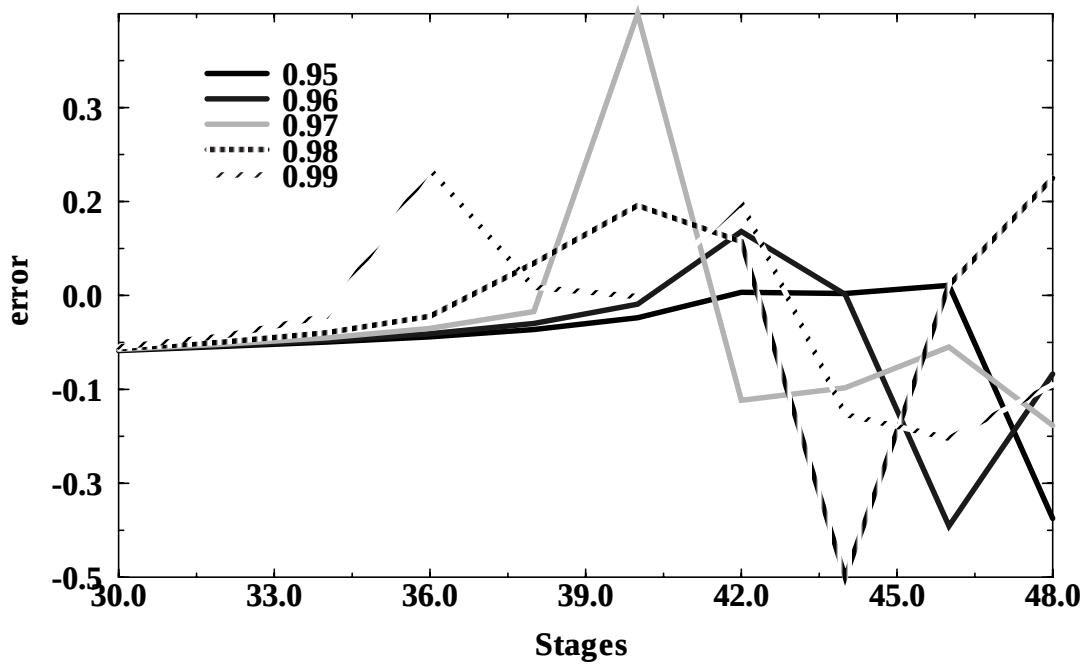

Figure 3.13. Error in C3, section 1 for a range of recoveries. Relative volatilities (3, 2, 1)

3.7 Sharp Split Calculations

To guarantee appropriate trajectories, we added the following equations.

$$x_n^{C1} - x_{n+1}^{C1} = diff_n^{C1},\ n = 1..npoints \qquad (3.5)$$

$$\hat{x}_n^{C3} - \hat{x}_{n+1}^{C3} = diff_n^{C3},\ n = 1..npoints \qquad (3.6)$$

We put a lower bound of zero on $diff_n^{C1}$ and an upper bound of zero on $diff_n^{C3}$. Furthermore, we place a test tray at the bottom of the top collocation section and bound the error on C3 to be between 0.1 and -0.1. Then we try to increase the number of trays as far as possible without crossing those bounds. With this technique, we tested the effect of the order of the collocation on the accuracy of

the model. Figure 3.14 shows the data for the 3, 2, 1 system, using 2, 3, 4, and 5 points in each collocation section in the top of the column. The plot shows reflux ratio as a function of recovery. Underwood's values are shown by the striped line. Figure 3.15 shows the same type of plot for the 9, 3, 1 system. We would expect that all the curves would at least be above minimum reflux, which is reflected in Figure 3.15, but the 4 point collocation in Figure 3.14 is below Underwood for every recovery. It is possible that error occurs in the model before the trajectories reflect this error. Figure 3.16 shows the same type of figure, but for 30 trays, rather than the maximum before crossing the bounds. The columns in Figure 3.14 each reached over 35 trays. Note that none of the models in Figure 3.16 are underestimating the minimum reflux, and as the number of collocation points increases we get closer to the Underwood value. This figure shows more what we would expect, getting better accuracy as the number of collocation points

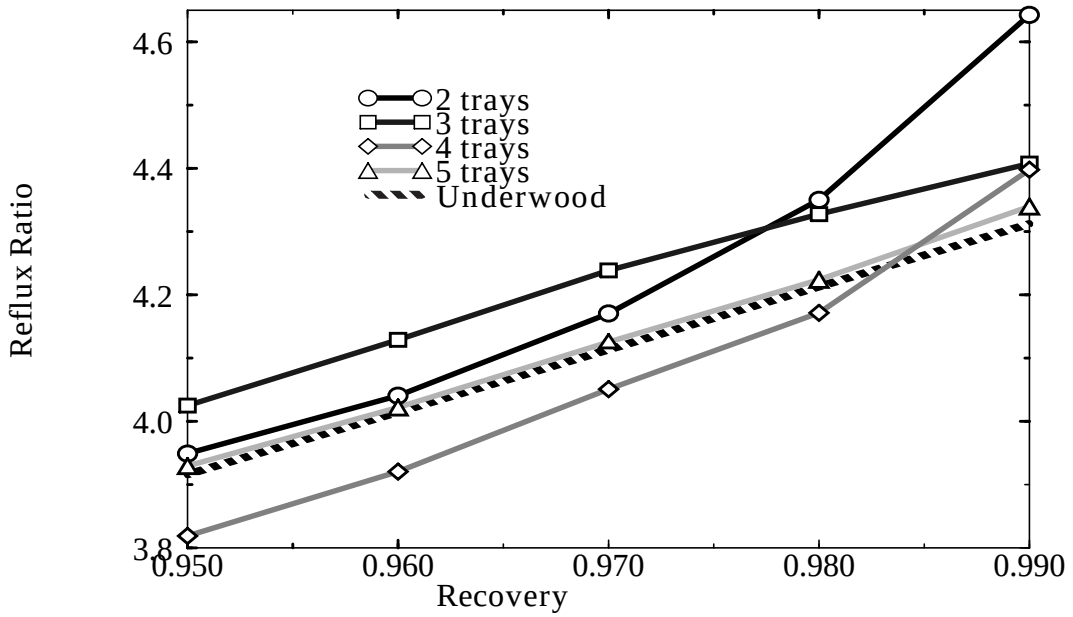

Figure 3.14. Reflux as a function of recovery and order. Volatilities 3, 2, 1

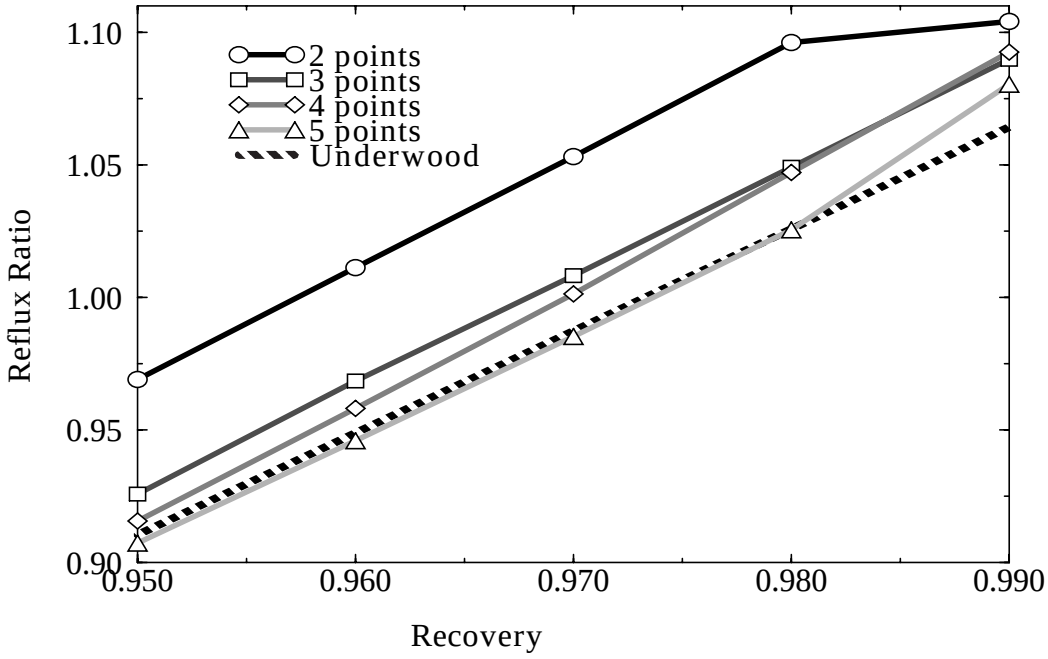

Figure 3.15. Reflux as a function of recovery and order. Volatilities 9, 3, 1

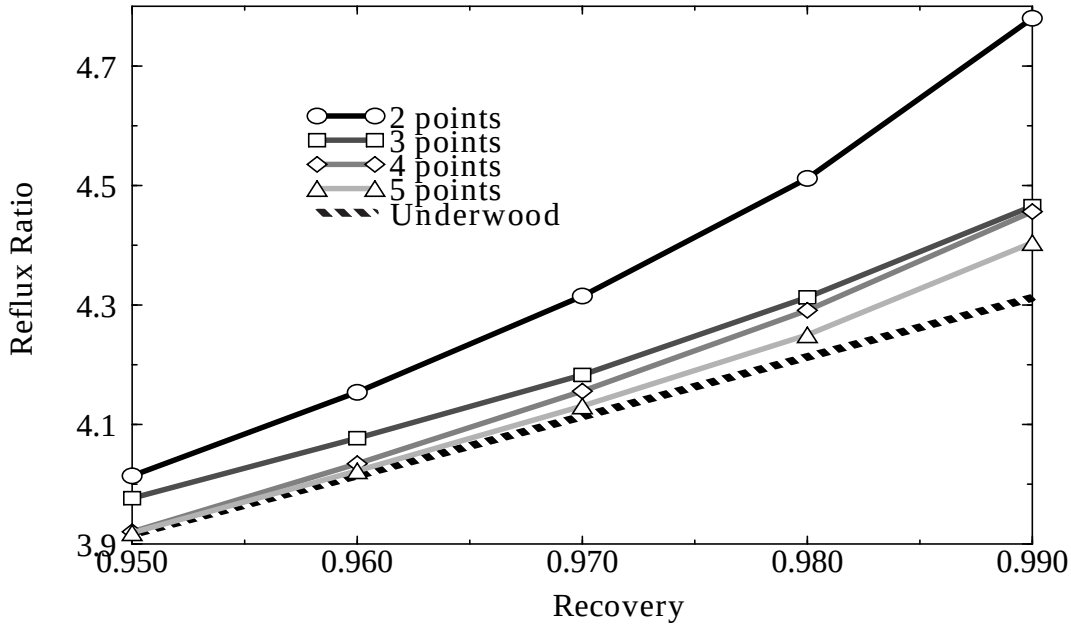

Figure 3.16. Reflux at 30 trays as a function of recovery and order. Relative volatilities 3, 2, 1

increases. It is possible that the 4 point collocation in Figure 3.14 is inaccurate because the fourth order polynomial produces unrealistic trajectories.

When going to large number of trays, the choice of the parameter *a* in equation 3.3 is potentially important. Experience has shown that a value of 0.1 is good for most normal problems. When we do wish to go to an infinite number of trays, (z = 1), it is possible to choose *a* such that the trajectory of one of the components is straightened out. However, our efforts to solve for the parameter *a* have not been successful. We have experimented with different values for *a* with some success. Figures 3.17 and 3.18 show the affect of *a* on the reflux calculations when using 2 point and 5 point collocations respectively. Each figure shows Underwood's values with a striped line and the closest approximation with a thick grey line. These figures demonstrate that 2 and 5 point collocation do not underestimate the minimum reflux when we use error detection. We believe that 2 point collocation sections are best for distillation modeling since higher order

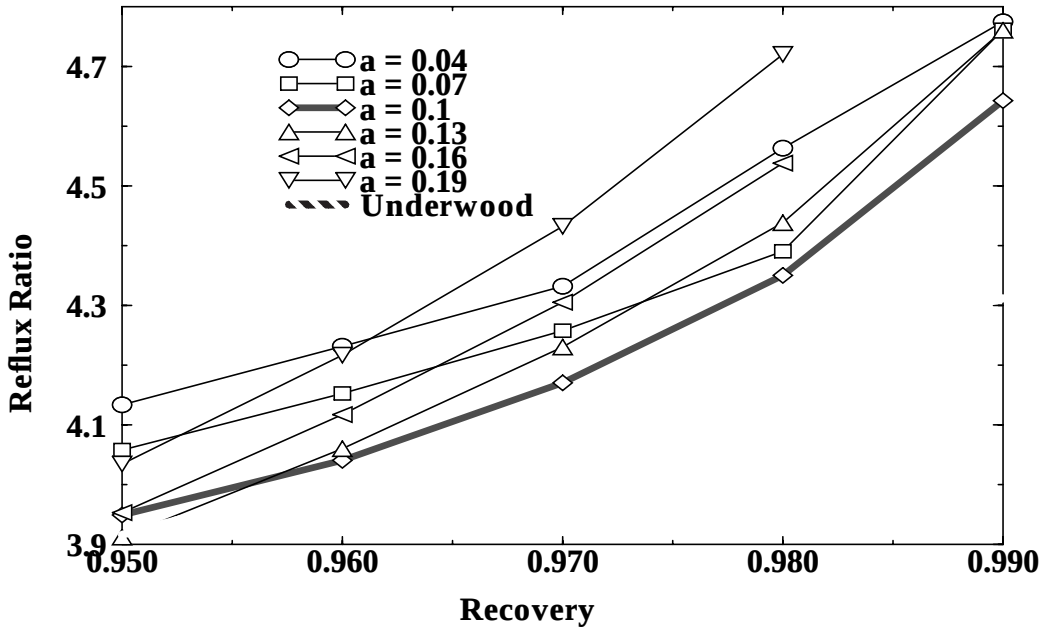

Figure 3.17. Efect of parameter *a* on reflux determination with 2 collocation points.

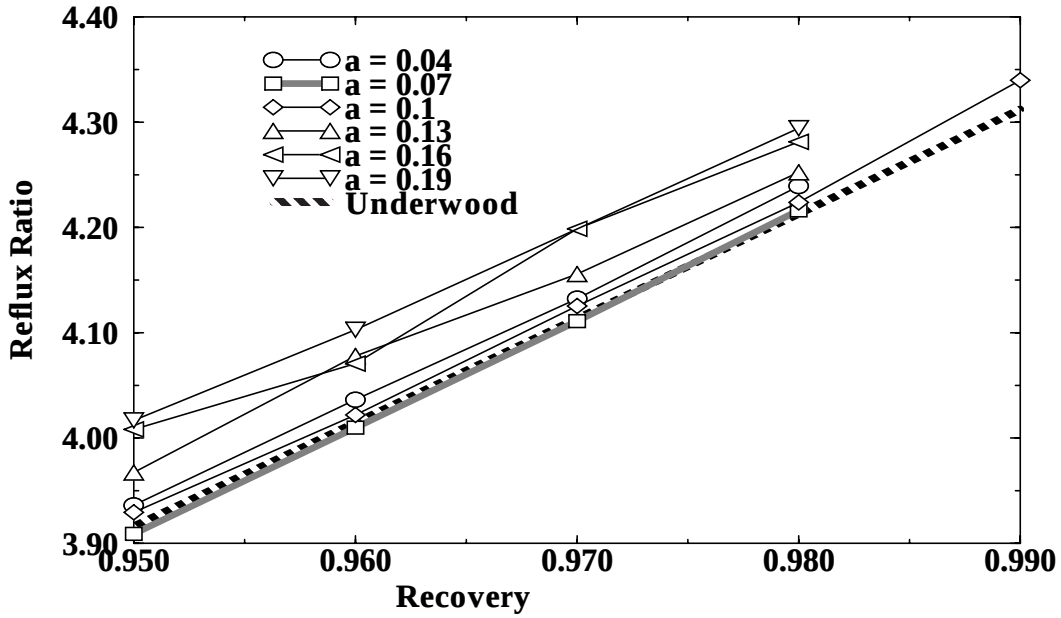

Figure 3.18. Effect of parameter *a* on reflux determination with 5 collocation points

polynomials can have trajectories that are not realistic for distillation. However, the 5 point collocation demonstrates how the accuracy can be improved with very high order, and what the trend of collocation will be if it is accurate.

3.8 Minimum Reflux Algorithm

Figure 3.19 shows the suggested algorithm for minimum reflux calculations using the collocation model. The first step, solving the standard collocation model to full equilibrium and heat balance is described in more detail in the third paper of this series [Huss and Westerberg, 1995b]. Through this step, we get a fully thermodynamic, heat balanced model of a very small distillation column with low reflux. If a pinch adjacent to the feed is expected, we enforce it, short circuiting the collocation section between the pinch point and the end of the column. The number of trays in this half of the column is no longer important, but to get nice looking column profiles, we fix the average slope of one of the components in that section and free up the number of trays. We discuss this

technique in detail in the third paper of this series [Huss and Westerberg, 1995b]. If no pinch adjacent to the feed is expected, we go directly to the next step, meeting the recovery specification. Again, see the third paper of this series for a detailed discussion of this technique. We free up the reflux ratio, distillate flowrate, and number of trays top and bottom while incrementing the recovery of the key components. The average slope technique mentioned above determines the number of trays in each column section.

For nonsharp splits, the only remaining step is to fix the recovery of a third component and solve for the number of trays in the finite column section. For a sharp split, we recommend two steps. First, we increase the number of trays in each section where a saddle pinch is expected, enforcing the bounds described above. This may not work for azeotropic systems. For azeotropic systems, the user would have to look at the plots of the column and decide if kinks occur, or only use the test tray to detect error. The second method is to perform two nonsharp calculations at very low impurities, but not close enough to zero to require enough trays for inaccuracies to occur. Using these two points we can extrapolate the reflux ratio at zero impurity.

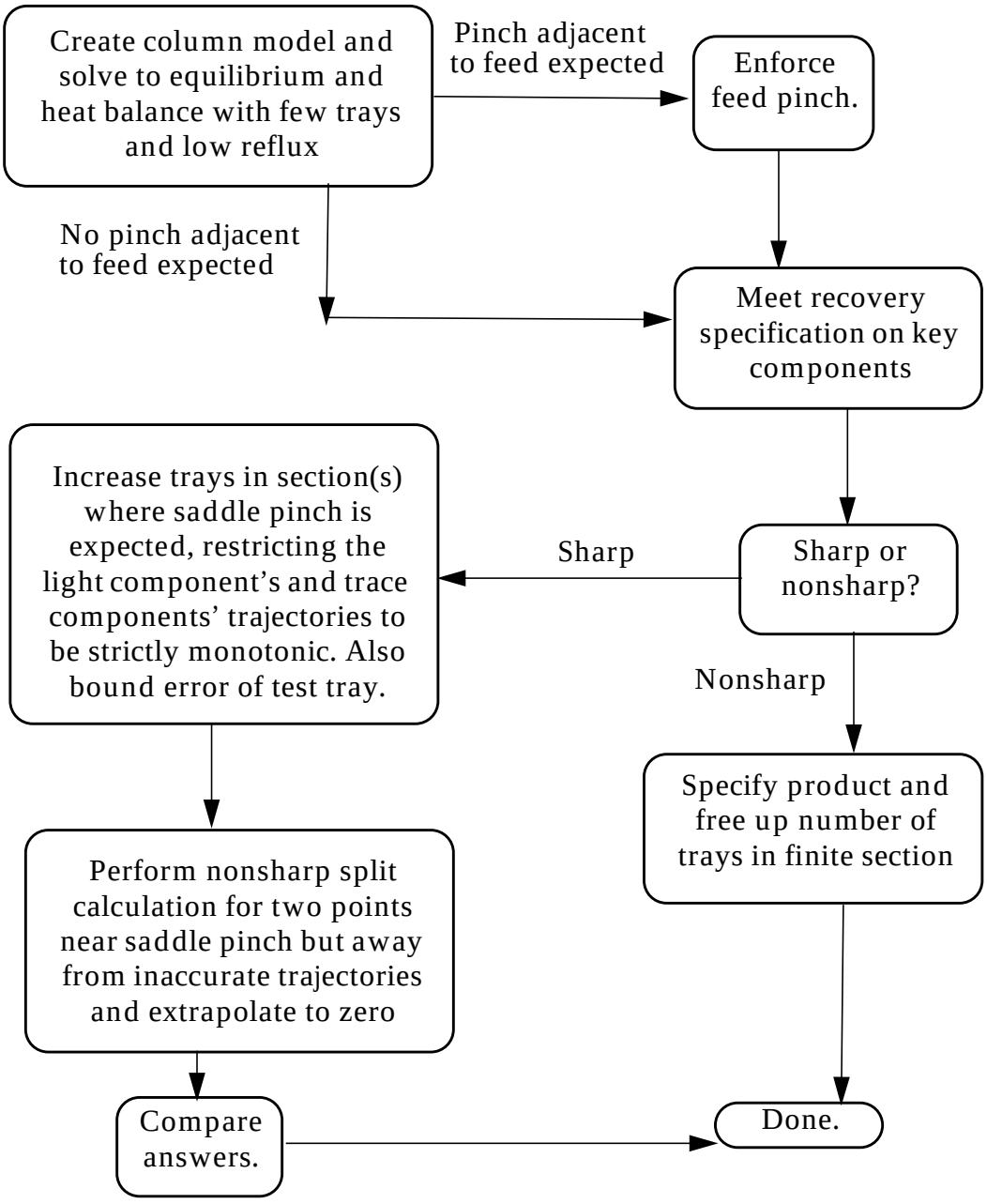

Figure 3.19. Minimum reflux algorithm

3.9 Difficulties

As we demonstrated in the first paper of this series [Huss and Westerberg, 1995a], the variable transformations allow us to model larger columns with

higher purities. However, as we've shown in this paper, difficulties still occur when going towards saddle pinches. In this section we explicitly list the areas where this model breaks down, providing reasons where we can. We provide this description of difficulties to encourage discussion into the problems typically enountered with any modeling technique.

When modeling a large column section in which the trajectories are flat, the collocation may develop kinks. The exponential transformation on trays reduces this effect but does not eliminate it. For an infinite column section it is theoretically possible to select a value for parameter *a* to straighten one of the components. However, in practice it is difficult to solve for the correct value for *a*. Also, when dealing with a finite number of trays, *z* behaves somewhat similarly to *s*. When this problem occurs away from a saddle pinch, it is possible to bypass the difficulty by requiring a pinch point and ignoring a collocation point. But at this point, we no longer have a collocation model for that section of the column.

Further difficulties occur when we approach a saddle pinch. The kinks in the trajectory begin to occur when going to a large number of trays, but the pinch point bypass does not work. In the case of a pinch adjacent to the feed, the collocation section adjacent to the feed will be completely flat, and the forced pinch will be at the end of that collocation section, as demonstrated in the right half of Figure 3.10. In that case, we essentially short-circuit the collocation section at the very bottom of the column by ignoring the polynomial definition for the liquid stream entering that section. The collocation section between the pinch point and the feed tray is capable of remaining constant throughout. But for the saddle pinch case, the collocation sections on either side of the saddle pinch have substantial change. In Figure 3.10, above the saddle pinch, the two remaining components change. Below the pinch, all three components are moving. When we try to enforce the saddle pinch by ignoring some polynomial definition in the top collocation section, the model fails to converge.

3.10 Conclusions

Theoretically, collocation provides the missing link for simulation of minimum reflux conditions. For nonsharp splits, we can exactly calculate the minimum reflux. For sharp splits we can approximate minimum reflux, overestimating the actual value. The minimum reflux problem exposes some weaknesses of this collocation model. Even with variable transformations enabling us to model higher purities and larger columns, the model breaks down when approaching a saddle pinch. We observe that the approach allows us to extrapolate to the saddle pinch.

Acknowledgment

Support from Eastman Chemicals and DOE contract number DE FG02-85ER13396 provide support for this research. Facilities support is from NSF grant number EEC-8942146 which supports the EDRC, an NSF funded Engineering Research Center.

Nomenclature

a	Parameter for exponential transformation of stage location
s	Stage number
z	Transformation variable on stage number

References

Huss, R. S. and A. W. Westerberg. 1995. “Collocation Methods for Distillation Design I: Model Description and Testing” to be submitted.

Huss, R. S. and A. W. Westerberg. 1995. “Collocation Methods for Distillation Design III: Flexible Column Design” to be submitted.

Julka V., and M. F. Doherty, 1990. Geometric Behavior and Minimum Flows for Nonideal Multicomponent Distillation. *Chemical Engineering Science*, **45**, p. 1801-1822.

Koehler J., P Aguirre, and E. Blass., 1991. “Minimum Reflux Calculations for Nonideal Mixtures Using the Reversible Distillation Model.” *Chem. Eng. Sci.*, **46**. 3007-3021.

Levy S. G., D.B. Van Dongen, and M. G. Doherty., 1985“Design and Synthesis of Homogeneous Azeotropic Distillations II: Minimum Reflux Calculations for Nonideal and Azeotropic Columns.” *Ind. and Eng. Chem. Fun.*, **24** 463-474

Underwood, A. J. V. *J. Inst. Pet.* 1945, **31**, 111

Underwood, A. J. V., *J. Inst. Pet.* 1946, **32**, 598

Underwood, A. J. V., *Chem. Eng. Prog.* 1948, **44**, 603

CHAPTER 4

COLLOCATION METHODS FOR DISTILLATION DESIGN III: FLEXIBLE COLUMN DESIGN

Robert S. Huss and Arthur W. Westerberg

Department of Chemical Engineering
and Engineering Design Research Center
Carnegie Mellon University
5000 Forbes Ave.
Pittsburgh, PA 15213

To be submitted to *Industrial & Engineering Chemistry Research*

Abstract

In this third paper on collocation methods for distillation design, we explore the use of the collocation models for design of simple distillation columns as well as flexible columns. Solvent recovery plants must deal with a wide range of feeds and still return pure solvents. The design problem we address is a single flexible column within the overall solvent recovery plant.

We have developed the models and algorithms in the ASCEND system. We discuss the attributes and use of the ASCEND system. With ASCEND we can create complex models with simple building blocks and interactively learn to solve them.

We found the collocation model an excellent tool for distillation design, allowing us to develop new concepts in design strategies. We designed a single column as would exist in a flexible solvent recovery plant for an azeotropic system. It was designed to handle three possible feeds, each with a distinct separation task.

For each possible feed to a column, we approximate the operation of the column for that feed by creating a quadratic approximation of the reflux ratio as a function of the number of trays and feed location. We optimize the cost of the column over the approximation range, and reapproximate if the minimum is on a bound. We move the approximation range until the local optimum occurs within that range.

4.1 Introduction

Several researchers have explored and developed collocation for distillation column modeling. In the first paper in this series, we presented a collocation model which expands on prior models, addressing the problems specific to steady-state, continuous columns [Huss and Westerberg, 1995a]. In the second paper, we discussed the use of the collocation method for minimum reflux determinations [Huss and Westerberg, 1995b]. In this paper, we present algorithms for designing distillations columns with the collocation model, including methods for designing flexible distillation columns. We also discuss the benefits of the ASCEND system for developing and using mathematical models.

4.2 Learning to Solve Models

A mathematical model begins as a set of equations and variables. However, formulation is only a small part of modeling and design. For complex models, solving is much harder than creating. We do not always know what parts of the model will be known and what parts need to be solved for. Also, we need to learn the best path to solution. The ASCEND system supports these needs. ASCEND (Advanced System for Computations in ENgineering Design) is a rapid model development tool, which has a strongly typed declarative modeling language and incorporates object-oriented concepts [Piela et al., 1993]. It also has an interactive model building and solving environment, which creates a great deal of flexibility in model use.

With ASCEND, we can build a simple flash model from stream models and a vapor-liquid equilibrium (VLE) model. We can then model a distillation column with an array of flash models. This hierarchical building of models aids in creating and organizing complex constructs. ASCEND also allows us to refine existing models to include more complexity.

We typically solve a normal stage-by-stage distillation model with the

following steps. The initial model has no thermodynamics, simple stream models with molar flowrates and mole fractions, constant relative volatility for the VLE, and constant molar overflow. This is generally a simple type of model to solve, but if it is difficult, we can solve each tray individually first. Any part of an ASCEND model is accessible from the user interface. Once we solve the simple column, we can "refine" the simple stream models to liquid and vapor stream models and include enthalpy and Gibbs free energy calculations. To refine the model we use the deferred binding capability of ASCEND. We can interactively locate any part of a compiled (and perhaps solved) model and direct the type of the part be changed to a more refined type compatible with its current type. The compiler reinterprets and propagates the changes that result throughout the model. Values of variables already solved for become initial values for the modified model. Holding the temperatures constant and still using constant relative volatility for the equilibrium, we calculate initial values for the thermodynamic properties. Then the VLE can be refined to a thermodynamic equilibrium model, requiring the partial molar Gibbs free energies of the components in the liquid and vapor phase to be equal. The last step is to release the constant molar overflow specification and include heat balances instead. Using this incremental process, we can get a converged solution of a complex thermodynamic distillation model without trying to solve the whole thing from scratch.

By formulating the collocation model in ASCEND, we achieved a great deal of flexibility, as well as the ability to incrementally refine the model to include complex thermodynamics. We describe the details of this model in the first paper in this series.[Huss and Westerberg, 1995a] When creating a test model, we refine a generic collocation model and decide the number of collocation sections, the number of trays in each section, and the number of components. Interactively, we can change the degrees of freedom for the column, the direction for number staged (up or down) of each collocation section, and the spacing of

collocation sections and trays.

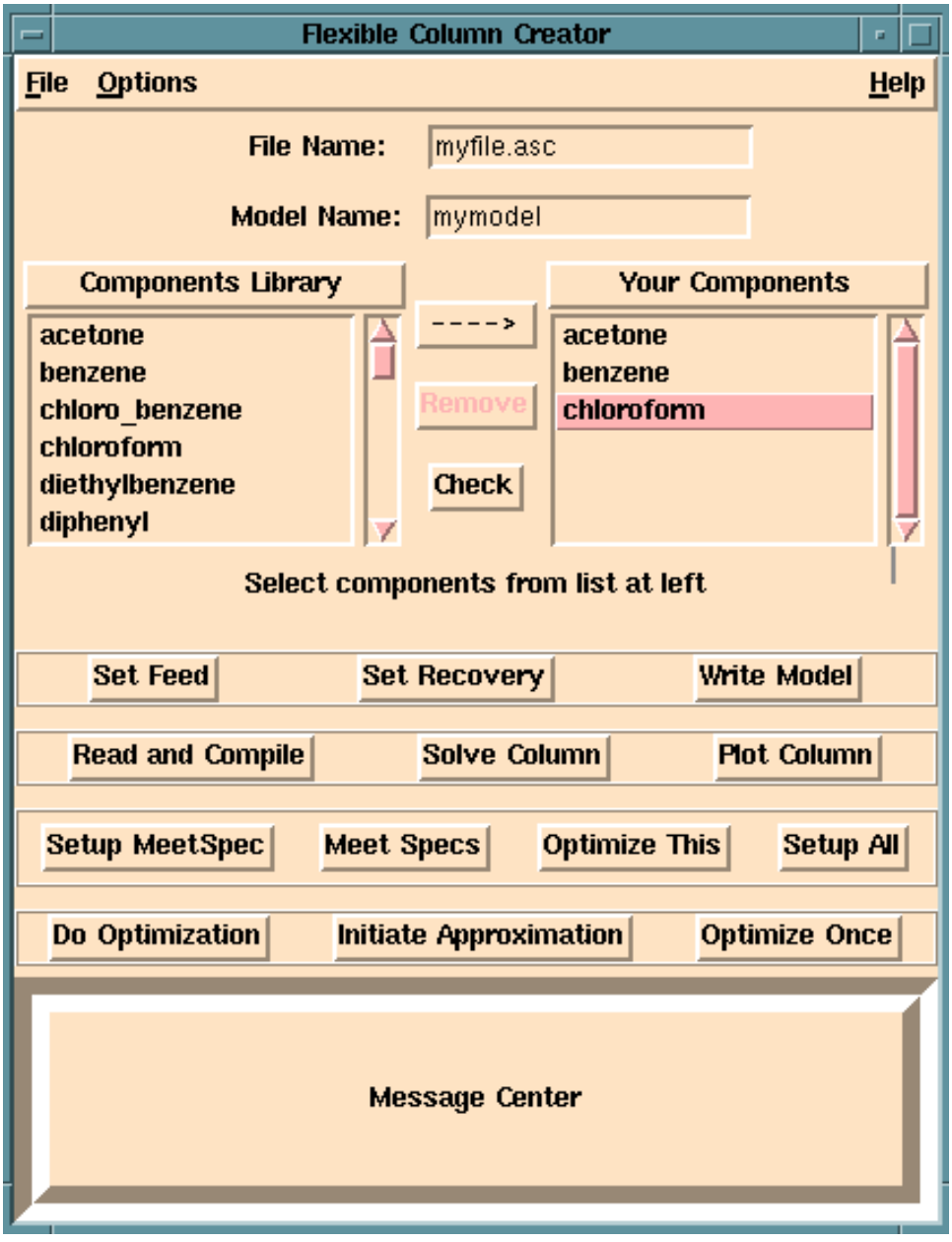

Figure 4.1. Column design application

ASCEND also has a scripting language with simple commands which reproduce interactive actions as well as the capability for developing complex procedures to automate the simulation process. We have used this scripting language to develop solution and design algorithms. This language also allows us

to add to the graphical interface, creating our own applications which can run on top of the ASCEND system. Figure 4.1 is a picture of the interactive window for the application we created to design columns.

4.3 Design Methodology

We learned the following general strategy for designing a column given the feed and a required separation. We present the algorithm in Figures 4.2 and 4.3. Figure 4.2 shows the algorithm for getting to a rigorous column model for a specified feed, making no demands on the separation. Figure 4.3 shows the algorithm for getting from this starting point to a column which meets the design specifications. The steps in this algorithm are executed by the first two rows of buttons on the interactive window shown in Figure 4.1.

The algorithm in Figure 4.2 begins by getting information about the problem. We need to decide what components will be used, what the feed flowrate and composition will be, and what the recovery specifications will be. We then check on the nonideality of the components involved. The check is based on a database of infinite dilution K values for all the components in our library. From these values, we can guess if binary azeotropes are expected and if heterogeneous behavior is expected. If the components in the system will exhibit heterogeneous behavior, we stop because we currently cannot accurately model this behavior. If azeotropes are expected without heterogeneous behavior, we can model the system, but we need to be aware of distillation boundaries. After checking the components, we generate a standard collocation model.

For a single feed column, this model has four collocation sections, two above and two below the feed, with two trays in each collocation section.[Huss and Westerberg, 1995a] We start with only two trays in each half of the column and a reflux ratio of 0.5, so the initial models will solve even for systems with high relative volatilities. We also start with the variable transformations on stage number and mole fraction (described in Huss and Westerberg, 1995a) turned on,

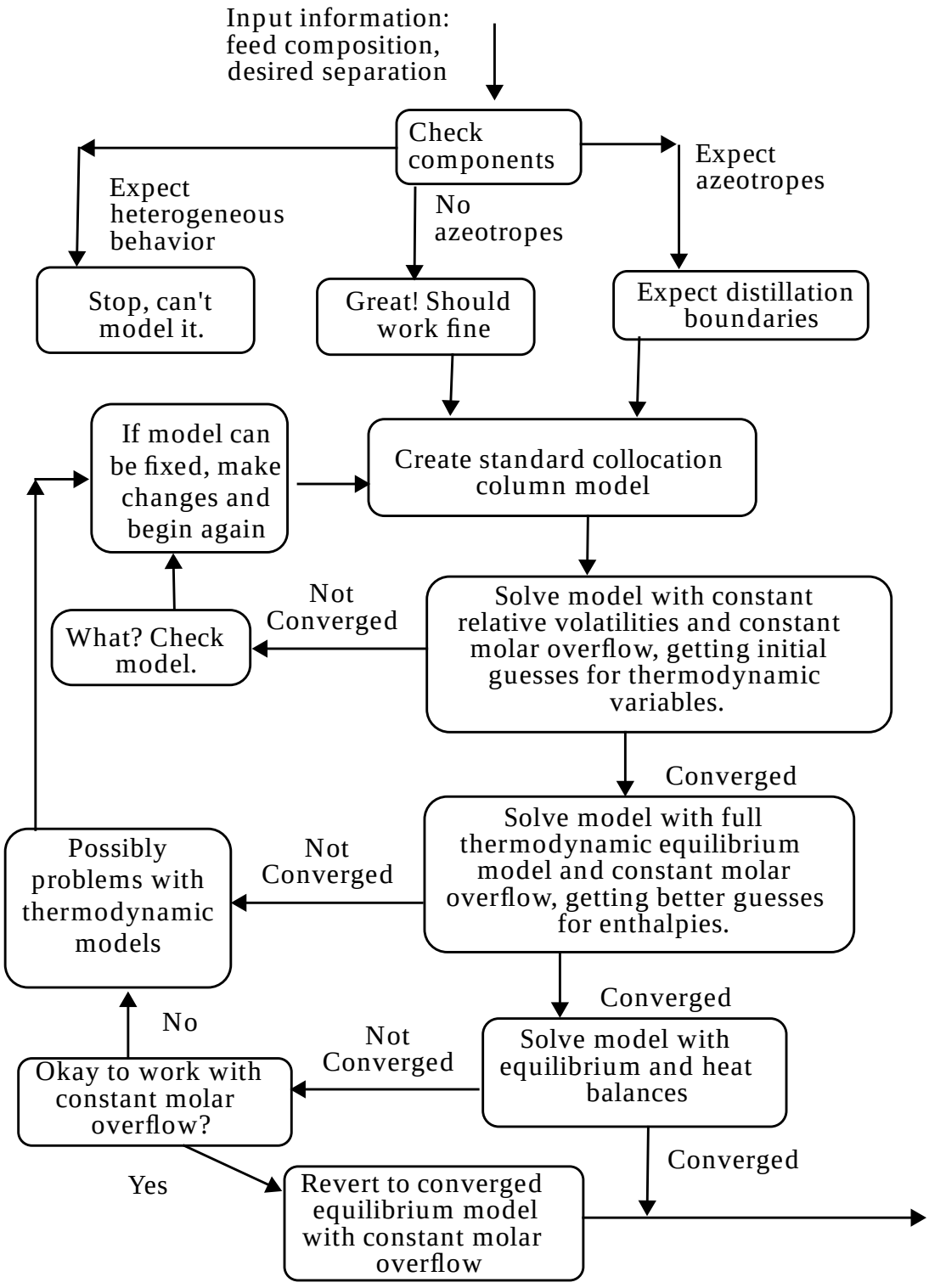

Figure 4.2. Algorithm for creating and converging a collocation model

since they may be needed and have no negative effects.

Once we create the model, we compile it and solve it with constant relative volatilities and constant molar overflow, fixing the reflux ratio, distillate flowrate, and number of trays top and bottom. From this solution we get initial guesses for the thermodynamic variables at standard conditions. This model has converged for all the systems we have tested. We next solve the model with the equilibrium equations included but still holding constant molar overflow. Then we turn on the heat balance and solve the fully rigorous model. If the heat balanced column does not converge, we can either return to the equilibrium model and gradually approach the heat balanced column by incrementally reducing the heat duties on each tray to zero using an algebraic continuation method, or we can decide to go ahead with the constant molar overflow column.

If any of the models with thermodynamics do not converge, it is possible that the thermodynamic models themselves are at fault. The collocation model and design algorithms are not dependent on what particular thermodynamics we use. If the thermodynamic models cause failure, we suggest changing them, but that is not the focus of this paper.

Once we have converged a sufficiently rigorous model of the system, we turn our attention to meeting the product specifications. Figure 4.3 shows that the first step is to adjust the degrees of freedom, fixing the recoveries of the keys and freeing the reflux ratio and distillate flowrate. Also, we define average slopes over the collocation sections, which can be used to maintain a reasonable number of trays for a given reflux ratio. Figure 4.4 shows how we define the average slope. In this case, the light component changes from 0.95 at the distillate to 0.35 at the feed tray, over a range of 50 trays. The average slope is therefore 0.012. If we fix the average slope for one component and free up the number of trays, the number of trays will change to maintain that slope if we change other variables. This allows us to avoid the problem of determining how many trays the column might

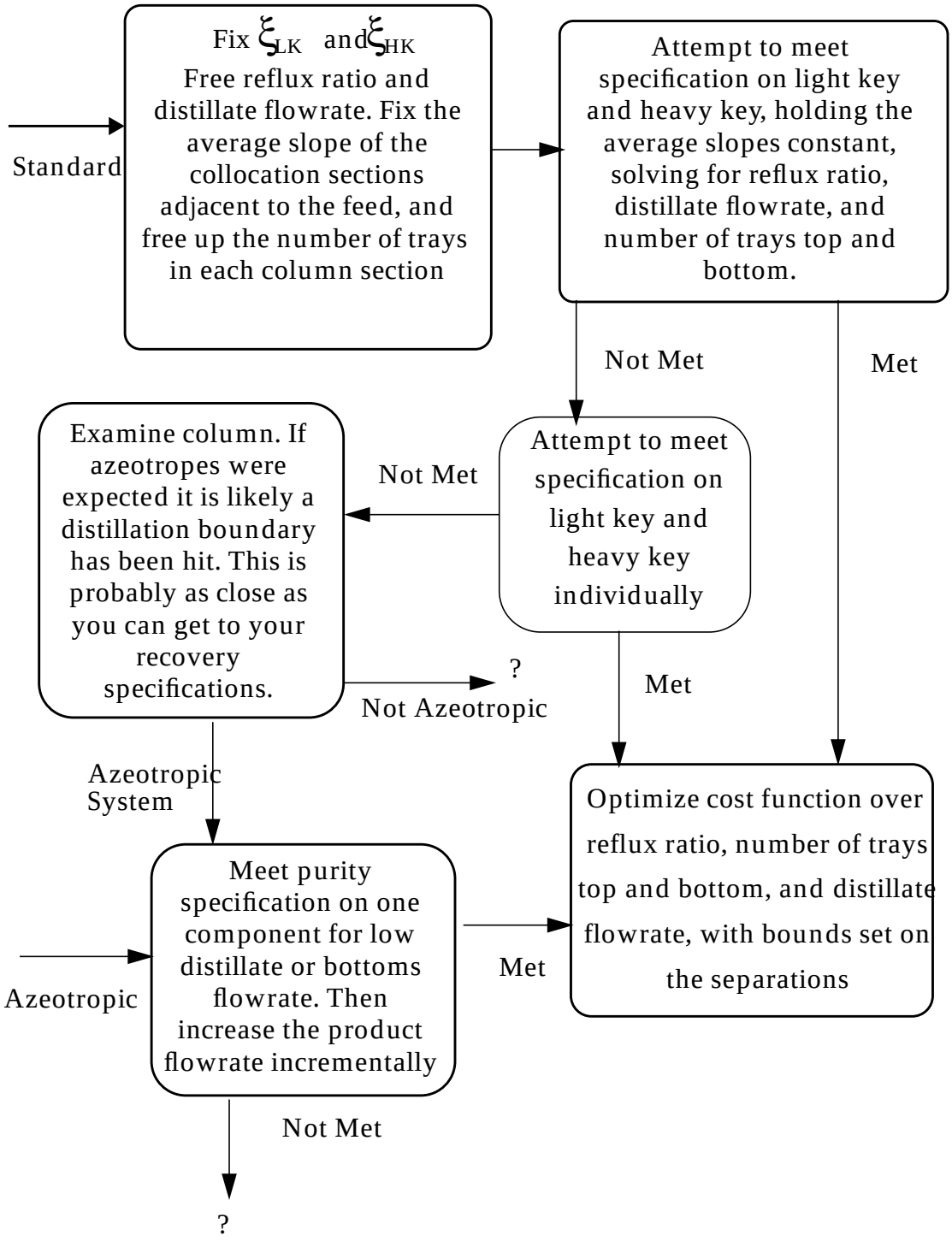

Figure 4.3. Algorithm for designing a column to meet specifications

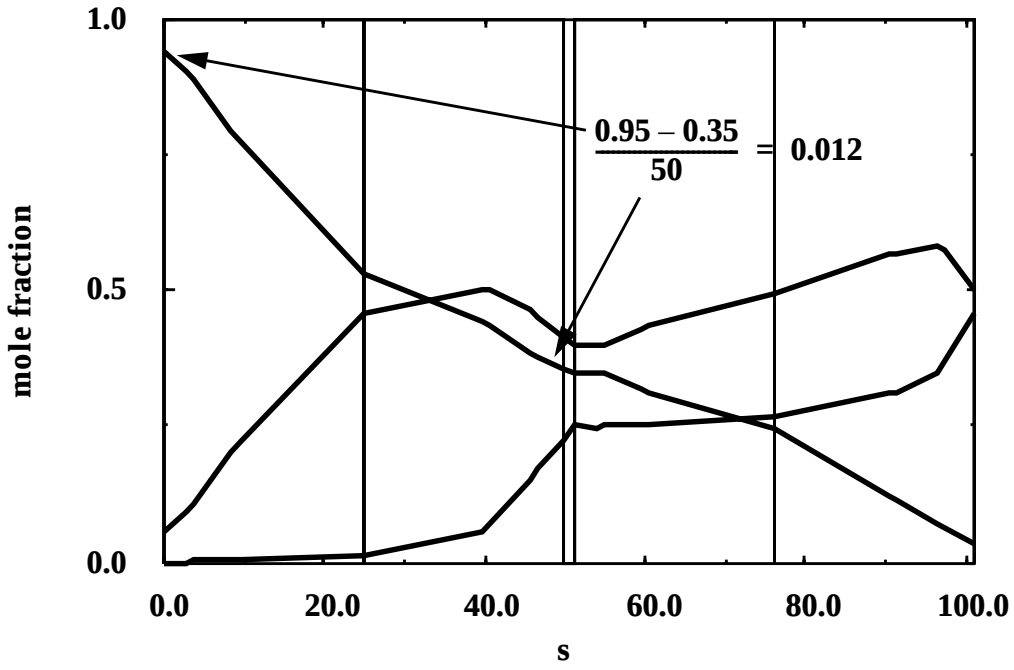

Figure 4.4. Average slope definition

need for a particular separation. For any fixed number of trays, it may be possible to solve for the reflux ratio and distillate flowrate for a given specification, but if there are not enough trays it will be impossible. By fixing the average slope of one component over each column section, the number of trays will increase as we approach the recovery specification.

In the next step, we attempt to meet the specifications on both key components while also meeting the requirement that the average slopes are 0.01 / tray, which is a reasonable value from our experience. We approach the specifications with an algebraic continuation method, taking a fractional step towards the desired settings. If the step succeeds, we take another step of the same size. If the step fails, we take a smaller step. If the four specifications (two key components and two slopes) cannot be met simultaneously, but the number of trays has increased, we fix the number of trays and just try to meet the two recovery specifications. Another alternative is to hold the slopes constant rather

than trying to move them to 0.01/tray. If the specifications cannot be met simultaneously, we then try to meet them each individually. If the specifications still cannot be met, we check to see if azeotropes were expected. If they were, it is likely that a distillation boundary has been hit.

We have an alternative for azeotropic systems where we expect to hit distillation boundaries. When we know that we want a certain purity of key components but cannot get high recoveries or do not know what the recoveries will be, we can meet purity specifications instead. If, for example, we want 99% purity of the light component in the distillate, we can first meet this purity specification with a very low distillate flowrate, solving for the reflux ratio. We will still use the slope criteria for setting the number of trays in each column section. Once we have met the purity specification, we can incrementally increase the distillate flowrate while holding the purity constant.

If the specifications can be met, we can attempt to optimize the column for cost. We have a cost function, accounting for the capital cost of the column, condenser, and reboiler, and the heat duties of both exchangers without heat integration [Douglas, 1988]. Currently, we have failed to get MINOS attached to ASCEND to solve the optimization problem for a fully thermodynamic column, due perhaps to the nonlinearity of the thermodynamic models and possibly poor variable bounding, but we can approach an optimal cost for a specified separation by doing a simple gradient based optimization routine using an ASCEND script. By specifying the recovery of both the key components and fixing the number of trays top and bottom, we can solve for the reflux ratio and distillate flowrate and the cost. By perturbing the number of trays and the feed location and resolving, we determine a slope on the cost function and take a step to reduce it. We have developed a simple routine in the script to minimize the cost over the number of trays and the feed tray location.

4.4 Design Essentials and Tricks

While developing this design algorithm, we discovered certain essential techniques and discovered some tricks that significantly improve our algorithm. The first essential lesson is that we cannot simply specify a product purity and solve for an appropriate reflux ratio. Even when there is a solution, this is a difficult problem to converge and must be approached gradually. With the standard degrees of freedom: feed, reflux ratio, distillate flowrate, number of trays, it is difficult to make large changes in any of these variables. It is even more difficult to switch the degrees of freedom for product purities or recoveries and make large changes. We have also discovered that it is better to make both key component specifications on the recoveries and release the reflux ratio and distillate flowrate than to just make one purity specification and release either the reflux ratio or the distillate flowrate. It is very easy to make purity specifications that are impossible with either the reflux ratio or distillate flowrate fixed. When we approach both key component recoveries simultaneously and solving for reflux and distillate flowrate, we give the model more room to maneuver.

The main trick we discovered is using the average slope over a column section to determine the number of trays in that section. This allowed us to solve simultaneously for the reflux ratio, distillate flowrate, number of trays, and feed location for a given recovery specification. This is not an essential technique because it is not always possible to get a given slope for a given specification, but it is very helpful for increasing the trays while approaching a product specification. In the algorithm, we first use the slope criteria to get us as far as possible towards the desired recoveries, which increases the number of trays. If it fails to go the whole way, we can generally fix the trays at this point and continue to go for the separation, solving for reflux and distillate flowrate.

4.5 Design Examples

We used the design algorithm discussed above to generate column designs

for different ideal and nonideal systems. For an ideal system with a simplified cost function we used MINOS attached to ASCEND to minimize the cost with constraints on the separation of the key components. The feed was 3 components, 3 mole/s of each component. The relative volatilities were 1.5 ,1.2, and 1.0. A nominal solution requiring 95% purity of the key components had 30 trays in each column section and a reflux ratio of 9.8, with a diameter of 1.3 meters, height of 42 meters, and an investment cost of 157,000 dollars per year. The optimal solution had 19.5 trays top, 18.8 trays bottom, a reflux ratio of 14.5, with a diameter of 1.6 meters, height of 27 meters, and an investment cost of 130,000 dollars per year.

We used the algorithm shown in Figures 4.2 and 4.3 to design a column for an equimolar feed of propanol, isobutanol and butanol, with 3 mole/s of each component. The separation specification was 99% of the propanol in the distillate and 1% of the isobutanol in the distillate. Using the algorithm it took 8 minutes to create a model, solve it up to full thermodynamics and heat balance, and incrementally meet the purity specifications, representing 9 solutions of the full column model. All that was required was to enter the components, the feed composition, and the purity specifications through the interface we placed on top of ASCEND, as shown in Figure 4.1. The nominal solution had 59.25 trays, with the feed tray 60.2% down the column, a reflux ratio of 6.12, and an annualized cost of 178,000 dollars per year. We optimized this column using a simple gradient search implemented from an ASCEND script, varying the total number of trays and the feed tray location with the separation of the key components fixed. The "optimal" solution had 52.28 trays, with the feed tray 52.7% down the column, a reflux ratio of 6.021and a cost of 174,000 dollars per year. To test the optimization, we increased the utilities cost by a factor of 10. As expected, the number of trays increased (to 75.1). The feed tray location moved to 50.2% down the column, and the reflux ratio decreased to 5.71. When we decreased the utilities cost by a factor of 10, the number of trays decreased as expected to 49.66 while the reflux ratio increased to 6.15. Each optimization took about 10 minutes to perform,

representing 30 column solutions. We performed these calculations on an HP700.

These design tests demonstrate that it is possible with this collocation model to solve for the number of trays and the feed tray location, while minimizing a cost function and holding a given specification. The time needed to execute the algorithm will decrease as we increase our understanding of the solution procedure.

4.6 Background on Flexible Distillation Design

Chemical plants use a wide variety of solvents which must be recovered for reuse. However, the feed to a solvent recovery plant will vary as the chemical plant demands change. Figure 4.5 shows a general picture of this type of problem. Any number of plants will send their waste mixed solvents to the recovery plant. Therefore, the feed to the recovery plant will change as the operation of the plants change. Azeotropic systems are particularly difficult because a change in the feed composition can move the system into a different distillation region. A simple, but expensive, solution to this problem is to require a constant feed composition to the recovery plant, using mixing to maintain the composition. A flexible solvent recovery plant, capable of separating a range of feeds with the same equipment, would be very useful to the chemical industry.

Very little work has been done on the specific problem of flexible distillation design. A significant amount of work addresses the general problem of process design flexibility. Halemane and Grossmann[1983] developed a formulation for determining design flexibility and ensuring design feasibility. For a convex constraint set, they showed that, by guaranteeing feasibility at critical vertices, the design will be feasible over the entire parameter space. They developed an approach to solving the two-stage programming formulation of the flexible design problem. Grossmann, Halemane, and Swaney[1983] presented an overview of optimization strategies for flexible chemical processes.

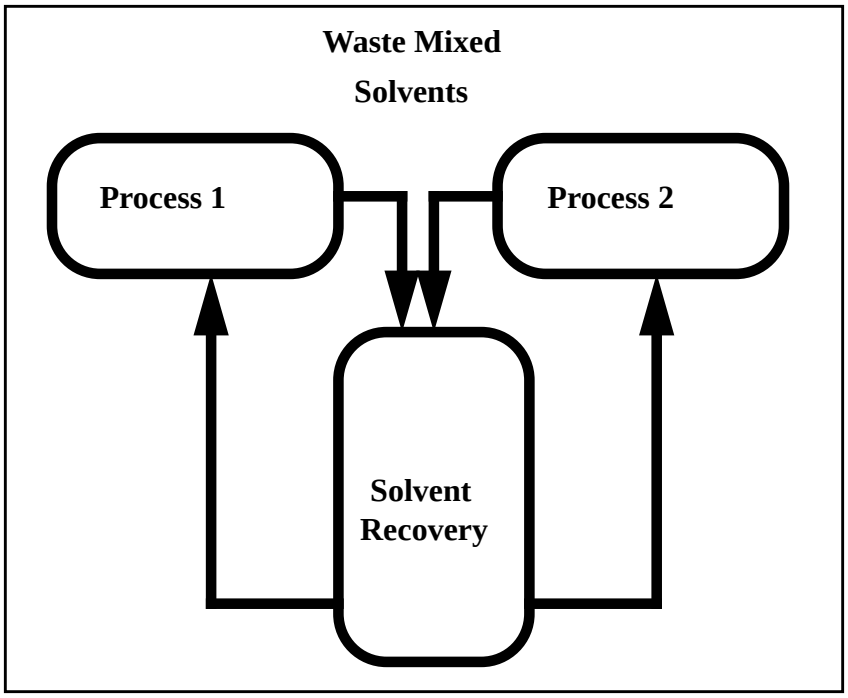

Figure 4.5. Solvent Recovery Plant Problem

Wagler and Douglas[1988] have addressed the specific problem of flexible distillation design. They used the concept of critical vertices also and simplified the problem further by combining vertices into "near critical points." They determined these near critical points by having a working knowledge of characteristics of the system. Specifically for distillation, they showed that, for the five constraints on product purity, flooding limit, weeping limit, feasible heat exchanger operation, and feasible accumulator operation, two near critical points could replace the five constraints. However, their results showed that the near critical points did not guarantee feasibility for the heat exchangers. Also, they assumed that the system they investigated had constant relative volatilities, and they used the Fenske-Underwood-Gilliland shortcut method to design the columns.

4.7 Problem Statement:

Design a separation system that is feasible over a specified range of feeds, meeting product specifications, while minimizing the cost of the system.

We can characterize the variability of the feed in a number of ways. It could be gradually but continually changing, changing infrequently but abruptly, or changing frequently and abruptly. We chose to address the problem of an infrequently changing feed which changes abruptly to known points.

Figure 4.6 shows an example of this type of problem, using acetone, chloroform, and benzene. There is a maximum boiling azeotrope between acetone and chloroform, creating a distillation boundary between the azeotrope and benzene, as shown on the ternary diagram. The three points represent the three possible feed compositions. If designing a separation system individually for these feeds, we might come up with the flowsheets shown. The solvent recovery plant designed to handle any of these feeds could have only three columns, where the same columns are used in different configurations for different feeds. For any given feed, the columns may be operating close to flooding or weeping limits rather than at more conventional flows.

4.8 Single Column Problem

For this paper, we only consider the problem of designing a single flexible column. Flexible columns will be required by the type of problem shown above. Even if different feeds to the solvent recovery plant require different flowsheets as shown in Figure 4.6, a single column can be used in multiple flowsheets. Therefore, a subproblem to the overall synthesis problem could involve designing a column that is able to handle a set of different feeds, performing a specific separation task for each.

Assuming such a problem, we allow the reflux ratio, distillate flowrate, and *feed tray position* to be control variables. We must enforce flooding and weeping limits on the column's operation for each feed and design heat exchangers for the condenser and reboiler that are large enough to handle the operation of the column for all feeds. The objective is to design the column meeting these specifications while minimizing the cost of the column. The

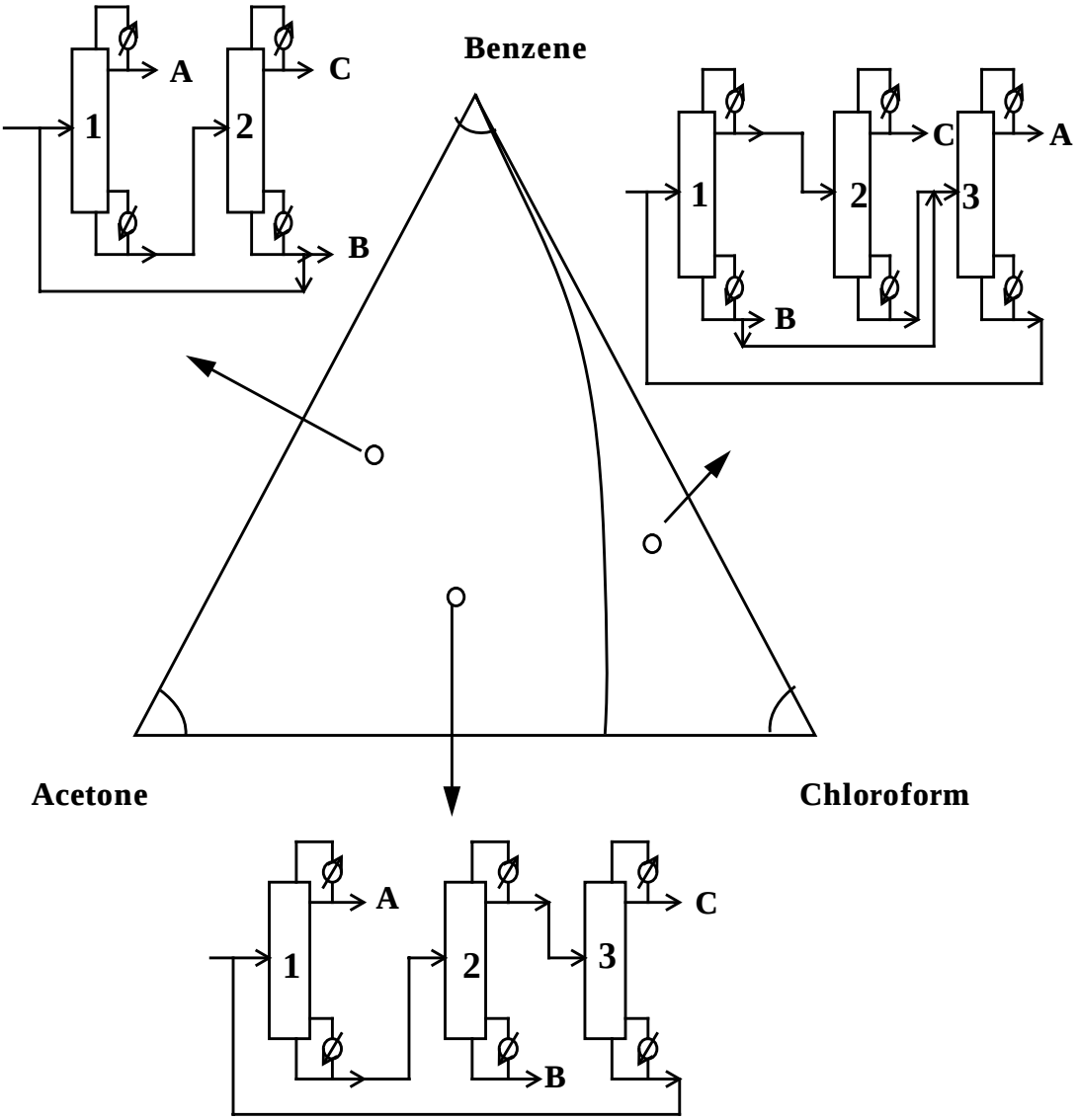

Figure 4.6. Solvent Recovery Plant Example

operating costs are averaged over the possible feeds weighting the cost by the estimated probability of each feed over time.

4.9 Flexible Design Algorithm

Figure 4.7 shows the algorithm used to design a single flexible column. The input information for this problem is the composition and flowrate of each feed

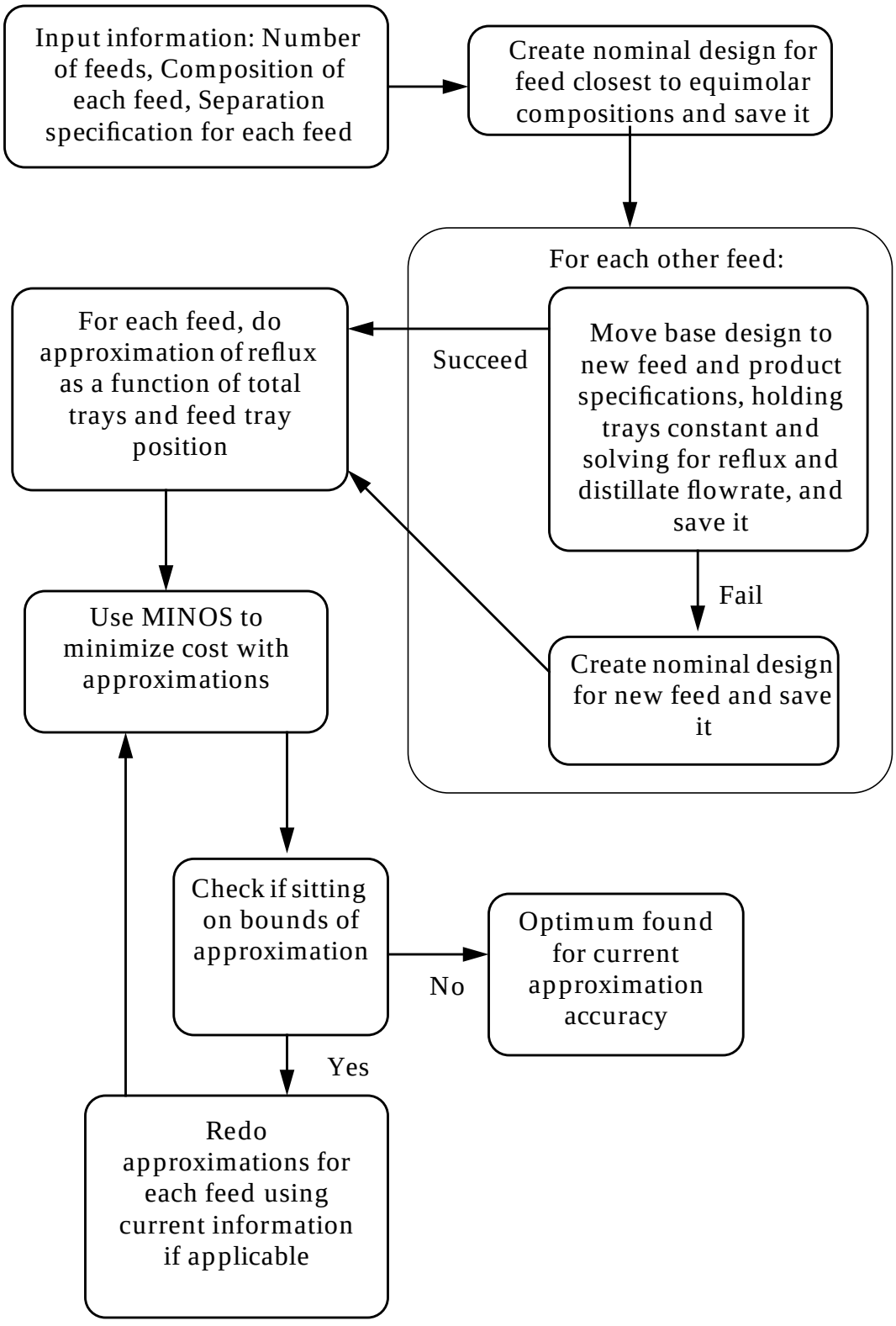

Figure 4.7. Flexible Design Algorithm

and the separation specification for each feed. With this information, we go through the column design algorithm presented earlier in this paper to create a nominal design for one of the feeds and save that solution. We choose the feed closest to the average composition of the feeds, which makes it more likely that the first nominal design will work for the other feeds.

For each other feed, we start with the nominal design and move its feed to the new feed, holding the number of trays constant. We create a nominal design for the new feed from scratch if we fail, getting a different number of trays for it. We then move the number of trays for each nominal design to the average number of trays.

We now have a nominal design for each possible feed. We save this information in a file for each feed, as well as saving it within the current ASCEND process. Within ASCEND we can make multiple saves of the values of existing models in RAM. Reading and writing these "virtual" saves takes a tenth of a second, while reading and writing to a file takes 30 seconds or more. We can read the values for these nominal designs into the one column model depending on the feed in which we are interested. Throughout the rest of the algorithm, we update the saved values whenever we change the column design.

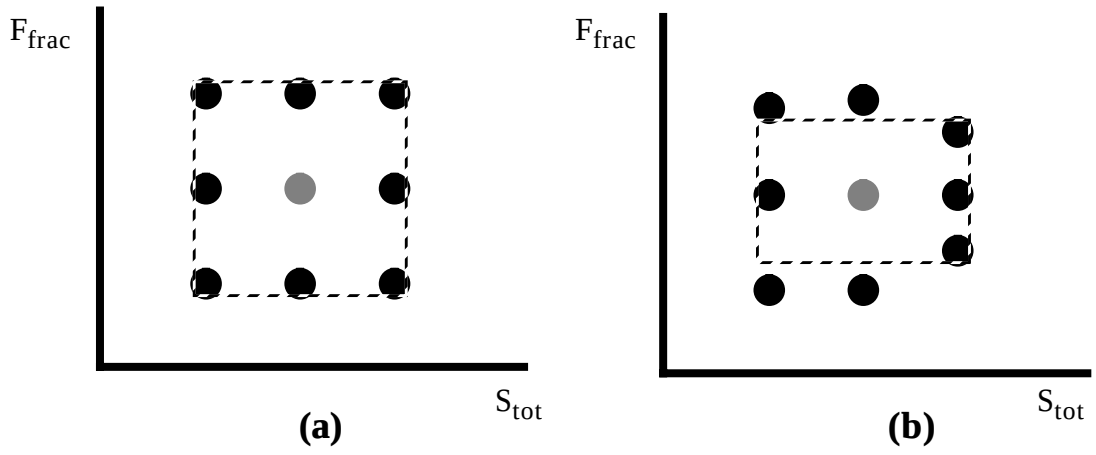

Figure 4.8. Data Point Placement

At this point we create an approximate model for each column to be used for cost minimization. We record the reflux ratio over a range of total trays and feed tray location. Figure 4.8a shows the normal placement of the data points. Generally we will move 10% away from the base point. From these nine data points we can fit reflux ratio as a quadratic function of the total number of stages, *Stot*, and the location of the feed tray, *Frac*. *Frac* is the fraction of the column above the feed tray. The approximation of the reflux ratio takes the following form.

$$R = \sum_{k=0}^{2} \hat{S}_k Stot^k \tag{4.1}$$

$$\hat{S}_k = \sum_{j=0}^{2} \hat{F}_{jk} Frac^j \tag{4.2}$$

Note that column diameter is not a concern here. For a given composition, the column trajectories will be the same, regardless of the total flowrate of the feed. The diameter can be determined once the reflux ratio, feed flowrate, and flooding factor are known.

We use the full model as a basis for the approximate model's cost calculation based on the number of trays, reflux ratio, and total feed flowrate. We set bounds for each approximation on *Stot* and *Frac* based on the range of the approximation. Figure 4.8b shows the points we might get if the column could not be moved fully 10% away from the base point. The dotted lines in both a and b show the bounds for that approximation.

Using these approximations, and requiring that the total number of trays and diameter is the same for each feed, we minimize the cost of the column. MINOS is attached to ASCEND for optimization problems. The optimization is formulated below:

$$\min ColCost + RebCost + ConCost + WCost + SCost \quad (4.3)$$

s.t. $ColCost = ColCost(H,D)$ (4.4)

$ConCost_i = ConCost_i(Feed_i,R_i,Dist_i),\ i = 1..nfeeds$ (4.5)

$RebCost_i = RebCost_i(Feed_i,R_i,Dist_i),\ i = 1..nfeeds$ (4.6)

$WCost_i = WCost_i(Feed_i,R_i,Dist_i),\ i = 1..nfeeds$ (4.7)

$SCost_i = SCost_i(Feed_i,R_i,Dist_i),\ i = 1..nfeeds$ (4.8)

$$ConCost \geq ConCost_i,\ i = 1..nfeeds \quad (4.9)$$

$$RebCost \geq RebCost_i,\ i = 1..nfeeds \quad (4.10)$$

$$WCost = \sum_{i=1}^{nfeeds} \frac{P_i(WCost_i)}{nfeeds} \quad (4.11)$$

$$SCost = \sum_{i=1}^{nfeeds} \frac{P_i(SCost_i)}{nfeeds} \quad (4.12)$$

$$R_i \geq \sum_{k=0}^{2} \hat{S}_{ik} Stot_i^k,\ i = 1..nfeeds \quad (4.13)$$

$$\hat{S}_{ik} = \sum_{j=0}^{2} \hat{F}_{ijk} Frac_i^j,\ i = 1..nfeeds \quad (4.14)$$

$H = 2.15Stot$ (4.15)

$F_i = F_i(Feed_i,R_i,D_i),\ i = 1..nfeeds$ (4.16)

$F_i \leq 2.4,\ i = 1..nfeeds$ (4.17)

$F_i \geq 0.8,\ i = 1..nfeeds$ (4.18)

$Stot \leq StotU$ (4.19)

$Stot \geq StotL$ (4.20)

$$Frac_i \leq FracU_i,\ i = 1..nfeeds \quad (4.21)$$

$$Frac_i \geq FracL_i,\ i = 1..nfeeds \quad (4.22)$$

The cost functions in equations 4.4-4.8 are Guthrie cost calculations and are functions of the total feed flowrate, the reflux ratio, distillate flowrate, and the flooding factor [Douglas, 88]. The flooding factor, *F*, is a dimensionless quantity, representing how close the column is to the flooding limit. For tray spacing of 2 feet, a flooding factor of 2.51 represents flooding [Douglas, 88]. Normally, we would design at 60% of flooding, but, for the flexible design problem, the column may need to run close to flooding for some feeds and close to weeping for others.

We are designing the condenser and the reboiler in equations 4.9 and 4.10 to be large enough for the largest demand. Since the cost of the exchangers is based on area, an exchanger with the largest area will have the highest cost and we assume can be operated to handle the other feeds. The utility costs in equations 4.11 and 4.12 are a weighted average over all the feeds, where P_i is the probability of feed *i* occurring over the time period of interest. Equations 4.13 and 4.14 are the reflux approximations for each feed. Note that equation 4.13 requires the reflux ratio to be greater than or equal to the reflux required for the specified separation. This allows the column to over-separate when the weeping limit is encountered. Equation 4.15 relates the height of the column to the number of trays. F_i is the flooding factor for column *i* and is a function of the total feed flowrate, the reflux ratio, and the diameter. We determine the bounds used in equations 4.19-4.22 based on the data points used for the approximation of each column.

If the result from MINOS is on the bounds for the number of stages or the feed tray location, *Stot* or *Frac*, we perform the approximation again about the new point. If the point is in a corner, we need to generate 5 new points. If it is on an edge, we only generate 3 new points. Figure 4.9 shows two steps from an initial approximation. The first hits only the bound on *Stot*, so we use 6 existing

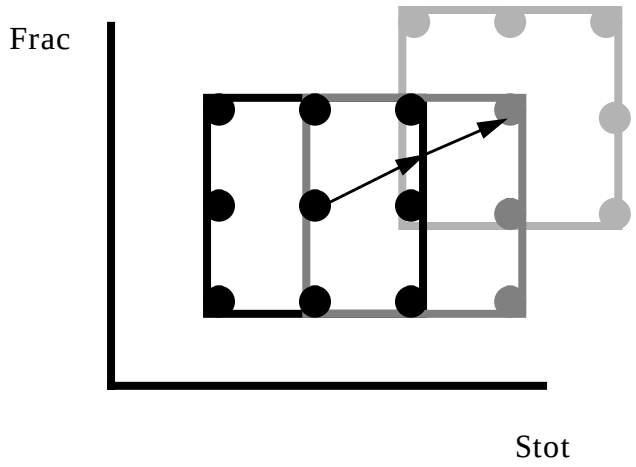

Figure 4.9. Reapproximation example

data points and create 3 more for the new approximation. The second optimization hits the upper bound on both *Stot* and *Frac*, so we can use 4 existing data points and must create 5. We keep looping through reapproximations and optimizations until the optimum is not on the bound for trays or feed tray location. This is a local minimum of cost based on the current approximation range. At this point, we could reduce the approximation range and re-optimize to get a more accurate solution.

4.10 Flexible Design Examples

We used the algorithm described above to design a flexible column for a propanol, isobutanol, butanol system. There were four possible feeds, each equally likely, and we required that 99% of the propanol and 1% of the isobutanol would be in the distillate. The component flowrates of each feed are listed in Table 4.1. Table 4.1 also shows the reflux ratios for the initial solution of 86 trays. After optimizing, the solution had 64.15 trays, and a cost of $210,000 / year. The reflux ratios, flooding factors, and feed locations for the final solution are also Table 4.1.

Table 4.1: Flexible Design Example 1

	Feed 1	Feed 2	Feed 3	Feed 4
Propanol feed	3 mole/s	1 mole/s	7 mole/s	3 mole/s
Isobutanol feed	3 mole/s	5 mole/s	3 mole/s	9 mole/s
Butanol Feed	3 mole/s	5 mole/s	3 mole/s	3 mole/s
Reflux ratio (86 trays)	5.73	17.61	3.69	5.73
Reflux ratio	6.18	20.91	3.84	5.71
Flooding factor	1.52	1.67	2.35	1.43
Feed location	0.51	0.47	0.64	0.47

Our second example is an acetone, chloroform, benzene system. We used approximately the three feeds shown on Figure 4.6. We desire a column that can perform the first separation task in each flowsheet. The two feeds to the left of the distillation boundary are separated to 99% pure acetone and the distillation boundary. The feed to the right of the boundary is separated to 99% pure benzene and a mixture of acetone and chloroform. We uses the alternative in Figure 4.3 for azeotropic systems where we first meet the purity specification of one of the product streams at a very low flowrate and then increase the flowrate of the product. For example, we want the first two feeds to have 99% pure acetone from the top, but the recovery of acetone will not be 99%. We set the distillate flowrate to a small number and met the purity specification on acetone while solving for the reflux ratio and the number of trays. Then we incrementally increased the distillate flowrate as far as it would go while maintaining the purity on acetone.

Table 4.2 shows the information for this problem. The initial solution had 24.2 trays. The final solution had 27.7 trays, and a cost of $150,000 / year. This example demonstrates how a column can be designed for completely different separation tasks.

Table 4.2: Flexible Design Example 2

	Feed 1	Feed 2	Feed 3
Acetone feed	3 mole/s	3 mole/s	1 mole/s
Chloroform feed	3 mole/s	1 mole/s	6 mole/s
Benzene feed	2 mole/s	3 mole/s	3 mole/s
Reflux ratio (24 trays)	12.4	6.8	3.6
Reflux ratio	11.5	4.8	2.5
Flooding factor	2.06	1.13	2.4
Feed location	0.78	0.85	0.5

4.11 Conclusions

This collocation method is an excellent tool for distillation design. By having the number of trays as a continuous variable, we are able to develop new concepts and techniques for distillation design and optimize the size of the distillation column. While developing the algorithms, we discovered some techniques that significantly improve the performance and enabled us to automate the design process. By modeling in the ASCEND system, we were able to learn how to solve these problems and create the tools for anyone to use our algorithms.

We demonstrated how a single column can be designed to deal with different separation tasks, providing the building block for more complex flexible design problems. This collocation method and these basic design algorithms should enable development of an algorithm for design of solvent recovery plants. However, we should note that this tool only finds the best design assuming the

column must perform all the separation tasks. It may be more economical to have extra columns or to store mixtures for a time. The overall flexible design problem needs to address these issues.

Acknowledgment

Support from Eastman Chemicals and DOE contract number DE FG02-85ER13396 provide support for this research. Facilities support is from NSF grant number EEC-8942146 which supports the EDRC, an NSF funded Engineering Research Center.

Nomenclature

R	Reflux ratio
Stot	Total number of stages in a column
$\hat{S}$	Coefficient of reflux ratio fit.
$\hat{F}$	Coefficient of reflux ratio fit.
Frac	Fractional location of the feed tray down the column.
ColCost	Material cost of column.
RebCost	Material cost of reboiler.
ConCost	Material cost of condenser.
Wcost	Operating cost of cooling water.
Scost	Operating cost of steam.
Feed	Feed stream to column.
Dist	Distillate product from column
nfeeds	Number of possible feed streams.
H	Height of column.
F	Flooding factor.
StotU	Upper bound on number of stages.
StotL	Lower bound on number of stages.
FracU	Upper bound on fractional location of feed tray for column.
FracL	Lower bound on fractional location of feed tray.

References

Douglas J. M., 1988, Conceptual Design of Chemical Processes, McGraw-Hill

Grossmann, I. E, K. P. Halemane, and R.E. Swaney. 1983 “Optimization Strategies for Flexible Chemical Processes.” *Comp. Chem Eng.* **7**, 439-462

Halemane, K. P. and I. E. Grossmann. 1983. “Optimal PRocess Design Under Uncertainty.” *AIChE J.* **29**, 425-433

Huss, R. S. and A. W. Westerberg. 1995. “Collocation Methods for Distillation Design I: Model Description and Testing” to be submitted.

Huss, R. S. and A. W. Westerberg. 1995. “Collocation Methods for Distillation Design II: Minimum Reflux” to be submitted.

Piela, P., McKelvey, R., and Westerberg, A. 1993, “An Introduction to the ASCEND Modeling System: Its Language and Interactive Environment,” *J. Management Information Systems*, **9**, 91-121

Wagler, R. M. and P. L Douglas. 1988 “A Method for the Design of FLexible Distillation Sequences.” *Canadian J. of Chem. Eng.* **66**, 565-581

CHAPTER 5

ASCEND MODEL DESCRIPTIONS

5.1 Introduction

In this chapter we will briefly describe the ASCEND models and structure that we used for this thesis. In the Appendix we include the ASCEND code so someone could reproduce the equations we have used. We only include enough of each library to model a single acetone, chloroform, benzene column. We only provide instruction for using the interface with the ASCEND system. We include this lengthy ASCEND code to present the equations of our models for anyone who wants to reproduce them. Since our formulation of the models is strongly dependent on the ASCEND system, we include the code rather than just the equations. This allows us to demonstrate the hierarchical nature of ASCEND.

For brevity, we do not include some of the standard procedures which are functionally identical for each model. Specifically, we have removed the clear and scale procedures from every model. Within each model, clear sets all the variables' fixed flags to FALSE, and includes all equations. The scale procedure sets the nominal values of all the variables within the model based on their current value and adjusts bounds relative to a scaling factor where reasonable. We also do not include the models we created to facilitate plotting of the columns. If anyone is interested in using these models in ASCEND, the system and models are available from an ftp site at Carnegie Mellon University. Contact the authors for more information.

5.2 Thermodynamics:

We are not going to describe in detail the thermodynamic models, but we will describe the structure and use of the thermodynamics library. For our purposes, the mixture model is the basic building block. The model **mixture** simply contains a set of type symbol for the names of the components in the mixture, and an array of mole fractions over that set. The only equation needed is one requiring that the sum of the mole fractions equals 1.

The model **homogeneous_mixture** refines **mixture**, but does not add anything to the model. This refinement merely gives us a type distinction for a single phase mixture model with no thermodynamics. The model **td_homogeneous_mixture** refines **homogeneous_mixture**, adding thermodynamic variables, **pure_component** models and **partial_component** models for each component. This can be refined to either **UNIFAC_mixture**, **Pitzer_mixture**, or **Wilson_mixture**. **UNIFAC_mixture** and **Wilson_mixture** are liquid mixture models, and **Pitzer_mixture** is a vapor mixture model.

The model **heterogeneous_mixture** refines **mixture**, adding structure and equations for a multiphase mixture model without thermodynamics. This model defines the phase equilibrium in terms of constant relative volatilities. The model

td_heterogeneous_mixture refines **heterogeneous_mixture**, adding thermodynamics variables, and **td_homogeneous_mixture** models for each phase. This can be refined to **equilibrium_mixture**, which merges the partial molar Gibbs free energies of each component in the different phases, and releases the constant relative volatilities. Also, **td_heterogeneous_mixture** can be refined to **murphree_equilibrium_mixture**, which assumes vapor-liquid equilibrium and adds an extra vapor mixture model to incorporate Murphree efficiencies.

5.3 Stream Models

The basic stream model which includes no thermodynamics has a total flowrate, a molar flowrate for each component in the stream, and a simple mixture model, named *state,* that has only mole fractions. The only equations in the model **molar_stream** are those defining the relationship between the mole fractions and the molar flowrates.

There are three refinements of this basic stream model, a liquid stream, a vapor stream, and a vapor-liquid stream. Since each of these models will require thermodynamics, we first create a thermodynamic stream model, **td_stream**, which is a refinement of **molar_stream**. This model adds an array of component constants data, one for each component. It also includes an energy flowrate which will be useful for energy balances.

The models **liquid_stream** and **vapor_stream** are refinements of **td_stream**, refining *state* to a **Pitzer_mixture** for the vapor and **UNIFAC_mixture** for the liquid. These are full thermodynamic models from the thermodynamics library for nonideal vapor and liquid mixtures. Both merge the molar enthalpy from the mixture model with that defined in **td_stream**.

The model **vapor_liquid_stream** model refines *state* to a **td_heterogeneous_mixture**, and defines that the phases will be vapor and liquid, using Pitzer and UNIFAC mixture models.

This hierarchy of stream models allows us to model first using molar streams, and then refine specific streams to being liquid, vapor, or liquid-vapor streams.

5.4 Flash Models

The building block for the flash library is **VLE_flash**. This model has no thermodynamics, using molar streams and simple mixture models. Inputs, liquid outputs, and vapor outputs are defined as arrays over sets that are as yet undefined. A part named VLE is defined as a **heterogeneous_mixture**, which is a multiphase mixture model using only constant relative volatilities. All of the liquid output streams are defined to have identical mixtures. The same is true for the vapor output streams. An array of fractions for the liquid and vapor outputs defines the relative flowrates of each output stream.

A standard column tray has a liquid input, vapor input, liquid output, and vapor output. The model **simple_tray** refines **VLE_flash** and defines that there are two inputs, one named *liquid* and one named *vapor*, and that there is one liquid output and one vapor output. Since this model still has no thermodynamics, we define a constant molar overflow relationship between the input and output liquid, creating a slack variable, cmo_ratio, which is fixed at 1.0 when constant molar overflow is assumed.

A standard feed tray, **simple_feed_tray** is much like any other tray in a column except that it has an extra input stream, which we name *feed*. We use the standard definition of quality to formulate the equation defining the state of the feed. For a *q* of 1.0, the liquid output will be equal to the liquid input plus the feed stream.

The model **condenser** also refines **VLE_flash**, with one input stream named *vapor* and two liquid outputs named *liquid* and *distillate*, and one vapor output name *vapor_product*. We also create a stream named *totprod* that is the sum of the distillate and vapor product streams. The *reflux_ratio* is defined relative to

the total product flowrate.

The model **reboiler** is very similar to the condenser model, but its single input is a liquid stream. It has a single liquid output stream named *bottoms* and two vapor output streams, one named *vapor* and one named *vapor_product*. Again, a *totprod* is defined as the sum of the bottoms and *vapor_product* outputs. The *reboil_ratio* is defined relative to this total product flowrate.

For each of the above models, we create a refinement to add thermodynamics to the model, putting the prefix td_ before each model name. For each model, the streams we consider liquids and vapors are refined to liquid and vapor streams, and the VLE model is refined to **td_heterogeneous_mixture**. The input stream named *feed* in the **simple_feed_tray** is refined to a **vapor_liquid_stream** so we can control the quality of the feed stream directly. For each model we create the variable *Qin* for the heat duty of the flash and define an energy balance. When using the constant molar overflow assumption, *Qin* will be non-zero, representing the non-adiabatic state. When we wish to enforce the heat balance, *cmo_ratio* or *q* can be freed up, and the heat duty set to zero.

With this hierarchy of flash models, the generic model for a distillation column is an array of **VLE_flash**, and we can refine the individual trays as required for a particular column configuration.

5.5 Collocation Models

The model **lagrange_polynomial** is the building block for the polynomial equations for any number of collocation points. The models **lgr_1_point**, **lgr_2_points**, **lgr_3_points**, **lgr_4_points**, and **lgr_5_points** refine **lagrange_polynomial** for 1,2,3,4, or 5 collocation points.

The model **collpoint** relates stage number, *s*, with the transform variable *z*, using the parameters *a* and *up_down*. The change in z across the collocation point

depends on the value of *a* and whether the numbering is going up or down the collocation section.

The model **z_set** contains an array of **collpoint** models and the **lagrange_polynomial** model. This model contains equations equating either the *s* values or the *z* values for the collocation points with the points defining the polynomial in the **lagrange_polynomial**. When we wish to use the variable transformation on stage number, we uninclude the equations for *s* (*s_based*) and include the equations for *z* (*z_based*).

The model **coll** is the generic collocation section model. It includes an array of **simple_tray** models, as well as the **z_set** model. It contains equations for polynomials of mole fraction or transformed mole fraction. Again, when we wish to use the variable transformation, we include the equations named *trans* and uninclude the equations named *frac*.

The model **coll_stack** is an array of **coll** models, merging the streams joining the models. We use **coll_stack** to model a column section. It can have any number of collocation sections.

The model **coll_column** is an array of **coll_stack** models, with **simple_feed_tray** between them and a **condenser** at the top and a **reboiler** at the bottom. We also define the recovery of each component in the distillate, *xsi[components]* and the binary separation of each pair of components, *binary_sep[components][components]*.

The model **td_coll** refines **coll** to include thermodynamics in the model. It refines the trays to **td_simple_tray**, and the end streams to **td_stream**. The model **h_coll** refines **td_coll** to include polynomial equations for the enthalpy.

The model **td_coll_stack** refines **coll_stack** to include thermodynamics. Each collocation section is refined to **td_coll**. The model **td_coll_column** refines

coll_column also to include thermodynamics. It refines each **coll_stack** to **td_coll_stack**, each feed tray to **td_simple_feed_tray**, the condenser to **td_condenser**, and the reboiler to **td_reboiler**. Finally, **equilibrium_coll_column** refines **td_coll_column** to include equilibrium equations. It refines the vapor-liquid model on each tray is to **equilibrium_mixture**.

We also have plotting models which we link to the collocation models, but we do not include them here, as they are quite long, and do not contribute to the model.

5.6 Column Cost Models

The model **cost_calc** includes the Guthrie cost calculations for the column, condenser, reboiler, cooling water, and steam. It also includes some range calculations on the operating costs if the total flowrate of the feed changed. The model **cost_column** includes both a **coll_column** model and a **cost_calc** model, linking the variables that the cost calculation needs to the column model.

We use **reflux_fit** to determine the coefficients of the approximation polynomial for reflux ratio from the data points collected with the column model. The model **appoximate_column** is a column cost model using the approximation polynomial for reflux ratio. The approximate cost calculation also uses some variables from the column model at the center point of the approximation to calculate the operating costs.

The model **apcol_set** contains an array of **approximate_column** models, linking the variables that should not change if the same column is to be used by each potential feed. It also includes an overall cost calculation, taking the cost for the reboiler and condenser from the largest of the possible columns and averaging the operating costs based on the probability of each feed.

Finally, **standard_cost** refines **column_w_plot**, linking a cost_column

model with the plotting model and the approximate models (*col_set*). It also includes a **reflux_fit** (*col_fit*) model for each potential feed and links the coefficients from *col_fit* to *col_set*. This is the model we refine to create new designs.

CHAPTER 6

USE OF THE FLEXIBLE COLUMN CREATOR

6.1 Use of the Application

To run the Flexible Column Creator, the user must be running ASCEND. From the command line in ASCEND, he can start the application by typing,

source main.tcl

assuming he is in the column design directory. This should read in the libraries needed and create the application window, shown in Figure 6.1. First we will describe the general structure of the window.

At the top are the menus, under that are entry boxes for choosing a file and

model name. Under that are two list boxes for choosing components. Under that are a set of buttons. And under the buttons is a message box, which will contain a description of whatever button or region the mouse is over.

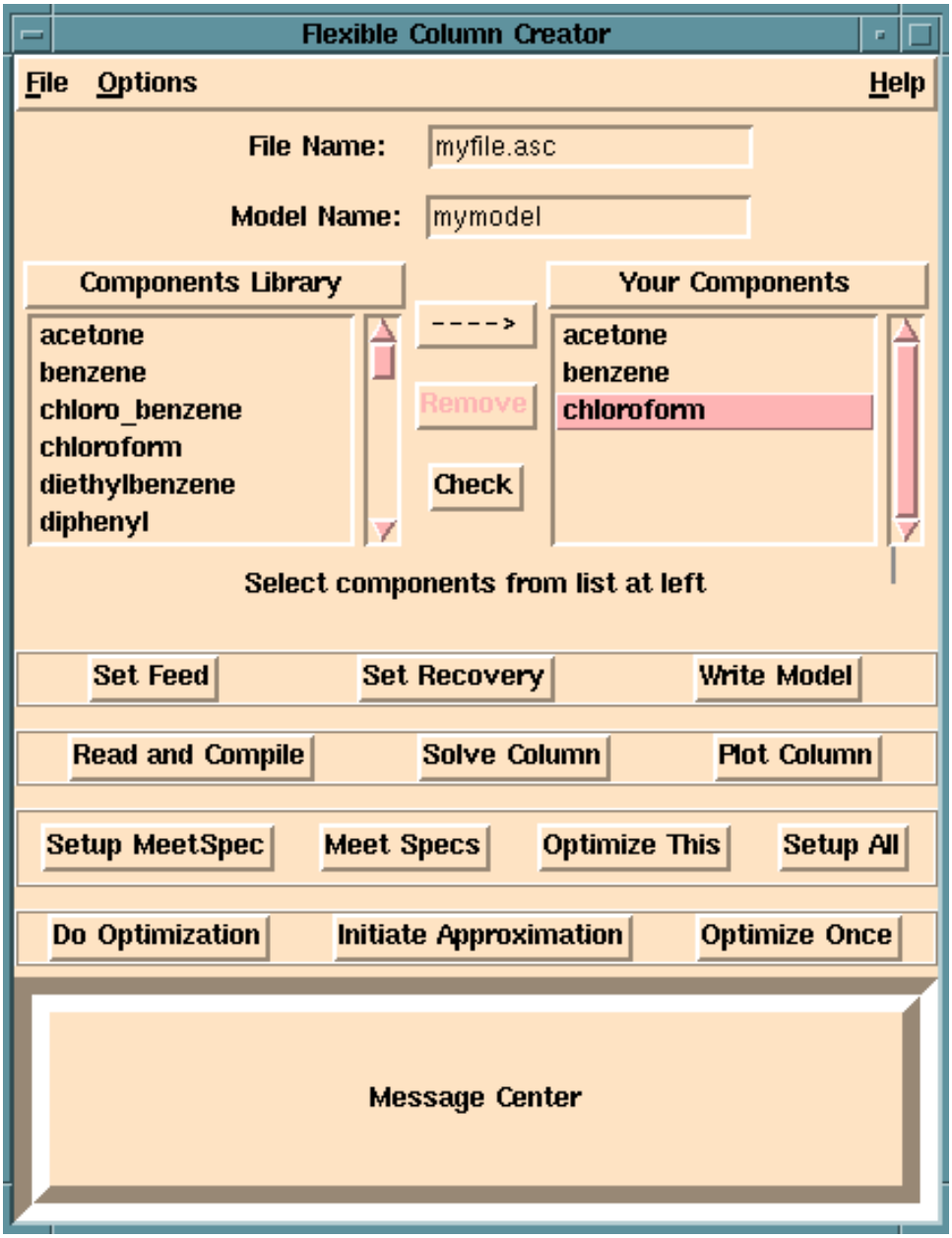

Figure 6.1. Column Design Interface

6.2 File Menu

Open Settings will bring up a file select box for the user to choose a

settings file to source. We use the suffix .col for settings files, and they contain code for regaining any choices the user has made in the application, like the components selected, the feed and recovery specifications, etc.

Save Settings will bring up a file select box for the user to choose a name for the creation of a settings file. If the user chooses an already existing filename, it will be overwritten. We use the suffix .col for settings files, but it is not mandatory.

Disable Message Box will remove the message box from the window.

Enable Message Box will restore the message box to the window.

Write Values will write the current model in ASCEND to a values file, naming it modelname.values, where modelname is the user defined name for the model. It will overwrite any existing file of that name.

Read Values will read in the values file created by **Write Values**.

Save All Values is for saving the current set of solutions for the flexible design problem. The flexible design problem will solve the column for the different potential feeds and create virtual saves of each solution. This will read each virtual save and create a separate values file for it.

Update Values is also for the flexible design problem. It will read in each of the values files created by **Save All Values** and create a virtual save for each of them. This is useful for continuing with a flexible design model on a different day. The user can read in the settings from the other day, read and compile the model, and then update the values.

Quit destroys the window. It does not quit ASCEND, destroy any models, nor clear any tcl variables.

6.3 Options Menu

List components alphabetically reorders the components library list to be alphabetical.

List components by boiling point reorders the components library list to be in the order of increasing boiling point.

Set globals creates an input window where the user can view or change the important variables that the application uses. Most of these can be changed through the interface, or are changed when different parts of the algorithm are executed. The variables plot_dir, values_dir, and logfile set the destination directory for plots, values and log files. The variable delta sets how big of a step the meet specifications procedure will take. Also max and min set the range of the approximation for the flexible design problem.

Color Settings creates a window for changing the color settings of the application. Standard names of colors can be input for each entry, or, by pressing Control-Mouse1, the user will bring up a color select box.

Procedure Menus will create extra menus for the procedures used by the application. In most cases, clicking these procedures in their menus will not work, because they require arguments, but it is useful for the more advanced user to have a list of the procedures available.

6.4 Entry Boxes

The first thing a user should do is enter their own file and model name in the two entry boxes in the top of the window.

File Name is the entry box for the name of the file to which the model will be written.

Model Name is the entry box for the name of the model.

6.5 Component Lists

Components Library is the list of all the components currently in the ASCEND components library. The library listing can be shown either in alphabetical order or by boiling point order. The options menu allows the user to change this.

Your Components is the list of the components the user has chosen.

The user can select components from the left list box by clicking on them and then hitting the **---->** key. Any number of components can be selected.

The user can remove components from his own list by selecting a component in the right list and hitting the **Remove** button.

The **Check** button checks the components for possible azeotropes and puts them in boiling point order. This check is done against a database we created by doing infinite dilution equilibrium calculations for each binary pair. From this database we can guess if binary azeotropes or heterogeneous behavior will occur. If heterogeneous behavior is expected, we warn the user that the models will be inaccurate, since we do not consider liquid-liquid equilibrium yet.

6.6 Settings Buttons

The **Set Feed** button brings up an entry window, asking the user first for how many feeds he wants. This is for the flexible design problem, not for multiple feeds to a single column. If a standard design is desired, the user should input 1. Then a dialog box will pop up, asking for the molar flowrate of each component in each feed. If the user wants to do a flexible design problem, he can input how every many feeds he wishes.

The **Set Recovery** button brings up a dialog box asking the user to pick the light and heavy key and set the recovery of each. The recovery is the fraction leaving the top of the column, so the light key recovery must be higher than the

heavy key recovery.

The **Write Model** button will create an ASCEND model named whatever the user entered in the **Model Name** entry line in the file the user entered on the **File Name** line. If the file already exists, the model will be appended to it. If it does not already exist, it will create the file.

6.7 Creating Buttons

The **Read and Compile** button will read the file into ASCEND and compile it. It will also run the values and specify procedures and export it to the Browser. At this point, or at any later point, the user can explore and manipulate this model like any other in ASCEND.

The **Solve Column** button will run the SolvCol1 procedure described above. The I/O window will show the progress of this procedure.

The **Plot Column** button can be used at any time once the model is read and compiled in ASCEND. It will create a plot file with the same name as the model with the suffix plot, e.g. mymodel.plot. It will also try to plot it with whatever plotting command the ASCEND user has set in the Utilities window.

6.8 Running Buttons

The **SetupMeetSpec** button gives the user 3 choices about how to approach the recovery specification. We suggest option 1 be tried first. It will attempt to meet the recovery specifications while trying to move the average slopes of the key components in the top and bottom to 0.01. This is not always achievable, but it is a reasonable starting point and will increase the number of trays in each column section. Option 2 has a similar effect, but assumes that the initial slopes are good ones. The third option is suggested as a second attempt option, after doing 1 or 2. Once a fairly large number of trays are modeled, it is reasonable to fix them and just go for the recoveries.

The **Meet Specs** button will run the MultipleMeetSpec procedure with the settings made with the **Setup MeetSpec** button. If the multiple specifications cannot be met, it will prompt the user to ask if he wants to try for each key recovery separately. Note that the settings given to MultipleMeetSpec can be accessed with the **Set globals** choice in the **Options** menu. They are keys (full ASCEND name of the key components), specs (specifications for the key components), and delta (initial step size towards specification).

The **Optimize This** button will use our gradient based optimization procedure to try to minimize the cost of the column by adjusting the number of trays and the feed tray location, while holding the recovery specifications and solving for the reflux ratio and distillate flowrate.

The **Setup All** button will attempt to get starting solutions for each possible feed for the flexible design problem. If it has problems, it will prompt the user for choices on meeting the specifications.

6.9 Flexible Optimization

Do Optimization will try to do the full flexible design optimization routine. First it will create the initial approximation and locally optimize this. Then it will move the approximation as needed and re-optimize iteratively until the local optimum is within the borders of the approximation. Then it will reduce the range of the approximation and initiate the approximation again and resolve the optimization problem.

Initiate Approximation will do only do the approximation for each of the columns and one optimization, setting up the optimization problem.

Optimize Once will find the local minimum for a particular level of approximation, moving the approximation region and re-optimizing until the minimum does not occur on a boundary of the approximation region.

CHAPTER 7

CONCLUSIONS AND FUTURE WORK

7.1 Conclusions

We have developed a general collocation method capable of modeling ideal, nonideal, and azeotropic distillation column over a wide range of operating conditions. The model can simulate fewer trays than there are collocation points or near pinch conditions. It can also model systems with very high separations, with trace components below 10^{-8}.

We found that the collocation method could accurately reduce the order of column models. The variable transformations on stage location and mole fraction greatly expanded the capabilities of the collocation method, overshadowing the

effect of point placement. We also demonstrated that the choice of point placement is nontrivial, and that the optimum point placement will be different for different systems.

We explored conditions near minimum reflux by simulating large column sections with the collocation model. We demonstrated that the sensitivity of reflux ratio to the amount of trace species can be significant, but that it drops sharply when the separation between key components is nonsharp. We demonstrated that a pinch point calculation connected to a finite column section is the appropriate model for nonsharp minimum reflux calculations, and that by making the finite column section large we can approach saddle pinch conditions. We encountered numerical instability with the collocation model near the saddle pinch point and compensated for this by detecting the error with an extra collocation point. To approximate sharp split minimum reflux, we approach the saddle pinch to the limit of the stability of the model. Generally we believe this approximation will over-approximate the actual minimum reflux.

We developed design algorithms for standard column design and flexible column design. We have found the ASCEND system an excellent tool for learning how to define and solve models. We designed a single column as would exist in a flexible solvent recovery plant for an azeotropic system. It was designed to handle three possible feeds, each with a distinct separation task. Our flexible optimization approach does not involve complex manipulations of the equations involved.

We have created a design tool which can be used with the ASCEND system to design standard columns or flexible columns. It can be used with little knowledge of the models involved, but it can be a powerful research tool to the experienced modeler.

7.2 Future Directions

Primarily, this collocation method can be used for designing standard distillation columns. With this model, the number of trays in each column section is a continuous variable, allowing optimization for the size of the column and feed tray location. We hope others will incorporate this model into design tools. Some companies have already expressed an interest in doing so.

This method does introduce many adjustable parameters which perform adequately at the values we have been using them. However, we believe there is room for improvement through understanding the physical effect of these parameters. As an example, the affect of the parameter *a* is to de-emphasize the section of the column where little activity occurs. For a low relative volatility system, it takes many trays to approach a pinch condition. However, for high relative volatilities the column profile can flatten out in only a few trays. This parameter will affect these two systems differently.

Our main hope for future research is in flexible distillation design. One possible direction is to employ the methods described by Sargent [1994] using the collocation model. He describes the use of a state-task-network to represent the superstructure of a distillation sequence. We could model such a superstructure with linked collocation elements. The same network could perform several separation tasks, with specific parts of the network representing different columns for different structures.

Nomenclature

A	Simplification variable for Kremser approximation (L/KV)
α	Mole fraction average relative volatility
a	Parameter for exponential transformation of stage location
α	Parameter of Jacobi polynomial
α_i	Relative volatility of species i
β	Parameter of Jacobi polynomial
ColCost	Material cost of column.
ConCost	Material cost of condenser.
Dist	Distillate product from column
f_{int}	Factor for selection of spread of collocation points
f_{mid}	Factor for selection of the midpoint of collocation points
F	Flooding factor.
$\hat{F}$	Coefficient of reflux ratio fit.
Feed	Feed stream to column.
Frac	Fractional location of the feed tray down the column.
FracU	Upper bound on fractional location of feed tray for column.
FracL	Lower bound on fractional location of feed tray.
h	Liquid molar enthalpy
H	Vapor molar enthalpy
H	Height of column.
K	Equilibrium constant used in Kremser approximation
L	Liquid molar flowrate
n	Order of polynomial
n_c	Number of components
n_p	Number of phases
n_s	Number of collocation points
nfeeds	Number of possible feed streams.
P	Pressure

P_n	Jacobi polynomial of order n
R	Reflux ratio
$RebCost$	Material cost of reboiler.
s	Stage location
$\hat{\mathbf{S}}$	Coefficient of reflux ratio fit.
$Scost$	Operating cost of steam.
$Stot$	Total number of stages in a column
$StotU$	Upper bound on number of stages.
$StotL$	Lower bound on number of stages.
T	Temperature
V	Vapor molar flowrate
W_k	kth term of Lagrange polynomial
w_k	Position of bottom of tray k
w_{mid}	Midpoint for placement of collocation points
$Wcost$	Operating cost of cooling water.
$wtop_k$	Position of top of tray k
x_i	Liquid mole fraction of component i
$\hat{x}_i$	Transformed mole fraction
y_i	Vapor mole fraction of component i
z	Transformed stage location

References

Carnahan B., Luther H. A., Wilkes J. O., 1969, Applied Numerical Methods, Wiley, New York.

Cho Y.S. and B. Joseph., 1983. "Reduced-Order Steady-State and Dynamic Models for Separation Processes." *AIChE J.* **29**, 261-269, 270-276

Douglas J. M., 1988, Conceptual Design of Chemical Processes, McGraw-Hill

Grossmann, I. E, K. P. Halemane, and R.E. Swaney. 1983 "Optimization Strategies for Flexible Chemical Processes." *Comp. Chem Eng.* **7**, 439-462

Halemane, K. P. and I. E. Grossmann. 1983. "Optimal PRocess Design Under Uncertainty." *AIChE J.* **29**, 425-433

Julka V., and M. F. Doherty, 1990. Geometric Behavior and Minimum Flows for Nonideal Multicomponent Distillation. *Chemical Engineering Science*, **45**, p. 1801-1822.

Koehler J., P Aguirre, and E. Blass., 1991. "Minimum Reflux Calculations for Nonideal Mixtures Using the Reversible Distillation Model." *Chem. Eng. Sci.*, **46**. 3007-3021.

Levy S. G., D.B. Van Dongen, and M. G. Doherty., 1985"Design and Synthesis of Homogeneous Azeotropic Distillations II: Minimum Reflux Calculations for Nonideal and Azeotropic Columns." *Ind. and Eng. Chem. Fun.*, **24** 463-474

Piela, P., McKelvey, R., and Westerberg, A. 1993, "An Introduction to the ASCEND Modeling System: Its Language and Interactive Environment," *J. Management Information Systems*, **9**, 91-121

Seferlis P and A. N. Hrymak, 1994, 'Optimization of Distillation units using collocation models', *AIChe J.* **40**, 813-825

Seferlis P and A. N. Hrymak, 1994, “Adaptive Collocation of Finite Elements Models for the Optimization of Multi-Stage Distillation Unit,” *Chem. Eng. Sci.*. **49**, 1369-1382

Stewart W. E., K. L. Levien, and M. Morari. 1984. “Collocation Methods in Distillation.” in Westerberg, A.W., and H.H. Chien (eds), Proc. 2nd Intn’l Conf. Foundations Computer-aided Process Design (FOCAPD’83), CACHE Corp, Ann Arbor, MI, 535-569.

Swartz C. L. E. and Stewart W. E. 1986, “A Collocation Approach to Distillation Column Design,” *AIChE J.* **32**, 1832-1838

Swartz C. L.E. and Stewart W. E. 1987, “Finite-Element Steady State Simulation of Multiphase Distillation,” *AIChE J.* **33**, 1977-1985

Underwood, A. J. V. *J. Inst. Pet.* 1945, **31**, 111

Underwood, A. J. V., *J. Inst. Pet.* 1946, **32**, 598

Underwood, A. J. V., *Chem. Eng. Prog*. 1948, **44**, 603

Wagler, R. M. and P. L Douglas. 1988 “A Method for the Design of FLexible Distillation Sequences.” *Canadian J. of Chem. Eng.* **66**, 565-581

Wahnschafft O. M. 1992, “Synthesis of Separation Systems for Azeotropic Mixtures with and Emphasis on Distillation-Based Methods,” Ph.D. Thesis, Dept of Chem. Eng., Technical University of Munich.

Appendix

```
(*************************************************************************\
                        system.lib
                        by Benjamin A. Allan
                        Part of the Ascend Library
This file is part of the Ascend modeling library.
Copyright (C) 1994
The Ascend modeling library is free software; you can redistribute
it and/or modify it under the terms of the GNU General Public License as
published by the Free Software Foundation; either version 2 of the
License, or (at your option) any later version.
The Ascend Language Interpreter is distributed in hope that it will be
useful, but WITHOUT ANY WARRANTY; without even the implied warranty of
MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the GNU
General Public License for more details.
You should have received a copy of the GNU General Public License along with
the program; if not, write to the Free Software Foundation, Inc., 675
Mass Ave, Cambridge, MA 02139 USA.  Check the file named COPYING.
\*************************************************************************)
(*=========================================================================*
    S Y S T E M  .  L  I  B
    -------------------------
    AUTHOR:      Benjamin A. Allan
    DATES:       06/94 - Original Code
    CONTENTS:    Basic definitions for relation, solver_var,
                 and generic_real.  This file is necessary for all
                 other ASCEND models to work on ASCEND3C
    REQUIRES:
 *=========================================================================*)
RELATION_DEFINITION
    included IS_A boolean;
    included := TRUE;
END;
ATOM solver_var REFINES real;
    lower_bound IS_A real;
    upper_bound IS_A real;
    nominal     IS_A real;
    fixed       IS_A boolean;
    fixed := FALSE;
END solver_var;
ATOM generic_real REFINES solver_var
        DIMENSIONLESS
        DEFAULT 0.5;
        lower_bound := -1e50;
        upper_bound := 1e50;
        nominal := 0.5;
END generic_real;
```

```
(*************************************************************************\
                        atoms.lib
                        by Joseph J. Zaher,
                           Benjamin A. Allan,
                           Robert S. Huss
                        Part of the Ascend Library
This file is part of the Ascend modeling library.
Copyright (C) 1994
The Ascend modeling library is free software; you can redistribute
it and/or modify it under the terms of the GNU General Public License as
published by the Free Software Foundation; either version 2 of the
License, or (at your option) any later version.
The Ascend Language Interpreter is distributed in hope that it will be
useful, but WITHOUT ANY WARRANTY; without even the implied warranty of
MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the GNU
General Public License for more details.
You should have received a copy of the GNU General Public License along with
the program; if not, write to the Free Software Foundation, Inc., 675
Mass Ave, Cambridge, MA 02139 USA.  Check the file named COPYING.
\*************************************************************************)
(*=========================================================================*
    A  T  O  M  S  .  L  I  B
    -------------------------
    AUTHOR:      Joseph J. Zaher
    DATES:       07/91 - Original code.
                 02/92 - Eliminated "display_units" assignments, added
                         commonly used UNIVERSAL constants and transport
                         quantities.
                 04/92 - Added electro-magnetic quantities.
                 06/92 - Converted by Ben Allan to be compatible with ASCEND3C
                 08/92 - Additional monetary atoms added by Bob Huss
    CONTENTS:    ASCEND atom definitions for engineering variable types.
                 Many of the anticipated dimensional variables which occur
                 in engineering design calculations are given to provide a
                 means of standardization.  Chosen defaults, nominal, and
                 lower and upper bound values should be re-specified if
                 necessary to enhance the convergence properties of
                 specific models.  Units to be displayed are to be controlled
                 using the UNITS tool kit of the environment.
    REQUIRES:    system.lib
 *=========================================================================*)
    ATOM constant REFINES real;
    END constant;
    UNIVERSAL ATOM gas_constant REFINES real
        DIMENSION M*L^2/T^2/Q/TMP
        DEFAULT 1{GAS_C};
    END gas_constant;
    UNIVERSAL ATOM circle_constant REFINES real
        DIMENSIONLESS
        DEFAULT 1{PI};
    END circle_constant;
UNIVERSAL ATOM scaling_constant REFINES real
    DIMENSIONLESS
    DEFAULT 1e8;
END scaling_constant;

    ATOM temperature REFINES solver_var
        DIMENSION TMP
        DEFAULT 298.0{K};
```

```
    lower_bound := 0.0{K};
    upper_bound := 10000{K};
    nominal := 298.0{K};
END temperature;
ATOM pressure REFINES solver_var
    DIMENSION M/L/T^2
    DEFAULT 1.0{atm};
    lower_bound := 0.0{psia};
    upper_bound := 1000{atm};
    nominal := 1.0{atm};
END pressure;
ATOM molar_mass REFINES solver_var
    DIMENSION M/Q
    DEFAULT 100.0{g/g_mole};
    lower_bound := 0.0{g/g_mole};
    upper_bound := 1e9{g/g_mole};
    nominal := 100.0{g/g_mole};
END molar_mass;
ATOM mass REFINES solver_var
    DIMENSION M
    DEFAULT 10.0{kg};
    lower_bound := 0.0{kg};
    upper_bound := 1e50{kg};
    nominal := 10.0{kg};
END mass;
ATOM mole REFINES solver_var
    DIMENSION Q
    DEFAULT 10.0{lb_mole};
    lower_bound := 0.0{lb_mole};
    upper_bound := 1e50{lb_mole};
    nominal := 10.0{lb_mole};
END mole;
ATOM mass_rate REFINES solver_var
    DIMENSION M/T
    DEFAULT 50{g/s};
    lower_bound := 0.0{g/s};
    upper_bound := 1e50{g/s};
    nominal := 100.0{g/s};
END mass_rate;
ATOM molar_rate REFINES solver_var
    DIMENSION Q/T
    DEFAULT 100.0{lb_mole/hour};
    lower_bound := 0.0{lb_mole/hour};
    upper_bound := 1e50{lb_mole/hour};
    nominal := 100.0{lb_mole/hour};
END molar_rate;
ATOM molar_volume REFINES solver_var
    DIMENSION L^3/Q
    DEFAULT 1000.0{cm^3/g_mole};
    lower_bound := 0.0{cm^3/g_mole};
    upper_bound := 1e50{cm^3/g_mole};
    nominal := 1000.0{cm^3/g_mole};
END molar_volume;
ATOM volume REFINES solver_var
    DIMENSION L^3
    DEFAULT 100.0{ft^3};
    lower_bound := 0.0{ft^3};
    upper_bound := 1e50{ft^3};
    nominal := 100.0{ft^3};
```

```
END volume;
ATOM volume_rate REFINES solver_var
    DIMENSION L^3/T
    DEFAULT 100.0{gpm};
    lower_bound := 0.0{gpm};
    upper_bound := 1e50{gpm};
    nominal := 100.0{gpm};
END volume_rate;
ATOM volume_expansivity REFINES solver_var
    DIMENSION 1/TMP
    DEFAULT 0.001{1/K};
    lower_bound := 0.0{1/K};
    upper_bound := 1e50{1/K};
    nominal := 0.001{1/K};
END volume_expansivity;
ATOM molar_density REFINES solver_var
    DIMENSION Q/L^3
    DEFAULT 0.1{mole/m^3};
    lower_bound := 0.0{mole/m^3};
    upper_bound := 1e50{mole/m^3};
    nominal := 0.1{mole/m^3};
END molar_density;
ATOM mass_density REFINES solver_var
    DIMENSION M/L^3
    DEFAULT 1.0{g/cm^3};
    lower_bound := 0.0{g/cm^3};
    upper_bound := 1e50{g/cm^3};
    nominal := 1.0{g/cm^3};
END mass_density;
ATOM molar_energy REFINES solver_var
    DIMENSION M*L^2/T^2/Q
    DEFAULT 10000.0{BTU/lb_mole};
    lower_bound := -1e50{BTU/lb_mole};
    upper_bound := 1e50{BTU/lb_mole};
    nominal := 10000.0{BTU/lb_mole};
END molar_energy;
ATOM energy REFINES solver_var
    DIMENSION M*L^2/T^2
    DEFAULT 100000.0{BTU};
    lower_bound := -1e50{BTU};
    upper_bound := 1e50{BTU};
    nominal := 100000.0{BTU};
END energy;
ATOM energy_rate REFINES solver_var
    DIMENSION M*L^2/T^3
    DEFAULT 100000.0{BTU/hour};
    lower_bound := -1e50{BTU/hour};
    upper_bound := 1e50{BTU/hour};
    nominal := 100000.0{BTU/hour};
END energy_rate;
ATOM heat_capacity REFINES solver_var
    DIMENSION M*L^2/T^2/Q/TMP
    DEFAULT 1.00e5{J/mole/K};
    lower_bound := 0.0{J/mole/K};
    upper_bound := 1e60{J/mole/K};
    nominal := 1.00e5{J/mole/K};
END heat_capacity;
ATOM molar_entropy REFINES solver_var
    DIMENSION M*L^2/T^2/Q/TMP
```

```
        DEFAULT 100.0{BTU/lb_mole/R};
        lower_bound := -1e50{BTU/lb_mole/R};
        upper_bound := 1e50{BTU/lb_mole/R};
        nominal := 100.0{BTU/lb_mole/R};
    END molar_entropy;
    ATOM entropy REFINES solver_var
        DIMENSION M*L^2/T^2/TMP
        DEFAULT 1000.0{BTU/R};
        lower_bound := -1e50{BTU/R};
        upper_bound := 1e50{BTU/R};
        nominal := 1000.0{BTU/R};
    END entropy;
    ATOM entropy_rate REFINES solver_var
        DIMENSION M*L^2/T^3/TMP
        DEFAULT 1000.0{BTU/hour/R};
        lower_bound := -1e50{BTU/hour/R};
        upper_bound := 1e50{BTU/hour/R};
        nominal := 1000.0{BTU/hour/R};
    END entropy_rate;
    ATOM factor REFINES solver_var
        DIMENSIONLESS
        DEFAULT 1.0;
        lower_bound := -1e50;
        upper_bound := 1e50;
        nominal := 1.0;
    END factor;
    ATOM fraction REFINES solver_var
        DIMENSIONLESS
        DEFAULT 0.5;
        lower_bound := 0.0;
        nominal := 1.0;
        upper_bound := 1.0;
    END fraction;
    ATOM monetary_unit REFINES solver_var
        DIMENSION C
        DEFAULT 100.0{USdollar};
        lower_bound := -1e50{USdollar};
        upper_bound := 1e50{USdollar};
        nominal := 100.0{USdollar};
    END monetary_unit;
ATOM cost_per_volume REFINES solver_var
        DIMENSION C/L^3
        DEFAULT 1.0{USdollar/gallon};
        lower_bound := 0.0{USdollar/gallon};
        upper_bound := 1e50{USdollar/gallon};
        nominal := 1.0{USdollar/gallon};
END cost_per_volume;
ATOM cost_per_mass REFINES solver_var
        DIMENSION C/M
        DEFAULT 1.0{USdollar/lbm};
        lower_bound := 0.0{USdollar/lbm};
        upper_bound := 1e50{USdollar/lbm};
        nominal := 1.0{USdollar/lbm};
END cost_per_mass;
ATOM cost_per_time REFINES solver_var
        DIMENSION C/T
        DEFAULT 1.0{USdollar/min};
        lower_bound := 0.0{USdollar/min};
        upper_bound := 1e50{USdollar/min};
```

```
        nominal := 1.0{USdollar/min};
END cost_per_time;
    ATOM distance REFINES solver_var
        DIMENSION L
        DEFAULT 10.0{ft};
        lower_bound := 0.0{ft};
        upper_bound := 1e50{ft};
        nominal := 10.0{ft};
    END distance;
    ATOM area REFINES solver_var
        DIMENSION L^2
        DEFAULT 10.0{ft^2};
        lower_bound := 0.0{ft^2};
        upper_bound := 1e50{ft^2};
        nominal := 10.0{ft^2};
    END area;
    ATOM time REFINES solver_var
        DIMENSION T
        DEFAULT 60.0{s};
        lower_bound := -1e50{s};
        upper_bound := 1e50{s};
        nominal := 60.0{s};
    END time;
```

```
(*************************************************************************\
                        components.lib
                        by Joseph J. Zaher
                        Part of the Ascend Library
This file is part of the Ascend modeling library.
Copyright (C) 1994
The Ascend modeling library is free software; you can redistribute
it and/or modify it under the terms of the GNU General Public License as
published by the Free Software Foundation; either version 2 of the
License, or (at your option) any later version.
The Ascend Language Interpreter is distributed in hope that it will be
useful, but WITHOUT ANY WARRANTY; without even the implied warranty of
MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the GNU
General Public License for more details.
You should have received a copy of the GNU General Public License along with
the program; if not, write to the Free Software Foundation, Inc., 675
Mass Ave, Cambridge, MA 02139 USA.  Check the file named COPYING.
\*************************************************************************)
(**
 **
 **  C  O  M  P  O  N  E  N  T  S  .  L  I  B
 **  ----------------------------------------
 **
 **  AUTHOR:      Joseph J. Zaher
 **
 **  DATES:       07/91 - Original code.
 **               02/92 - Made compatible with new set version of ASCEND.
 **                       Expanded data base, revised vapor pressure data,
 **                       and added UNIFAC group and subgroup sets with help
 **                       of Bob Huss.
 **               07/92 - Expanded data base with help of Kay C. Dee.
 **               08/92 - Replaced name attribute of each component with a
 **                       formula attribute.  Component names are free to be
 **                       specified by the user.
 **               03/94 - Made compatible with gnu-ascend.
 **               08/94 - Made compatible with H,G or H,S thermo library,
 **                       and with Wilson models written by Boyd Safrit.
 **
 **  CONTENTS:    ASCEND structure for component physical property constants.
 **               All anticipated constants which may be used by the models
 **               of “thermodynamics.lib” are created in a general model where
 **               a consistent reference state (298.15{K} and 1.0{atm}) is
 **               chosen.  Specific refinements to actual chemical species are
 **               made UNIVERSAL to ensure only one instantiation of the
 **               constants for each component is maintained throughout a
 **               user’s simulation.  A reference enthalpy and entropy is
 **               chosen by default to be those of formation of the component
 **               from its elements at the reference state.
 **
 **  REQUIRES:    “atoms.lib”
 **
 **)
UNIVERSAL MODEL UNIFAC_constants;
    groups                                  IS_A set OF symbol;
    sub[groups]                             IS_A set OF symbol;
    subgroups                               IS_A set OF symbol;
    group[subgroups]                        IS_A symbol;
    a[groups][groups]                       IS_A constant;
    R[subgroups], Q[subgroups]              IS_A constant;
```

```
groups                                  := ['CH2', 'C=C', 'ACH', 'ACCH2',
                                            'OH', 'CH3OH', 'H2O', 'ACOH',
                                            'CH2CO', 'CCl3', 'CCOO'];
sub['CH2']                              := ['CH3', 'CH2', 'CH', 'C'];
sub['C=C']                              := ['CH2=CH', 'CH=CH', 'CH2=C',
                                            'CH=C', 'C=C'];
sub['ACH']                              := ['ACH', 'AC'];
sub['ACCH2']                            := ['ACCH3', 'ACCH2', 'ACCH'];
sub['OH']                               := ['OH'];
sub['CH3OH']                            := ['CH3OH'];
sub['H2O']                              := ['H2O'];
sub['ACOH']                             := ['ACOH'];
sub['CH2CO']                            := ['CH3CO', 'CH2CO'];
sub['CCl3']                             := ['CHCl3', 'CCl3'];
sub['CCOO']                             := ['CH3COO', 'CH2COO'];
subgroups                               := UNION(sub[i] | i IN groups);
group[sub['CH2']]                       := 'CH2';
group[sub['C=C']]                       := 'C=C';
group[sub['ACH']]                       := 'ACH';
group[sub['ACCH2']]                     := 'ACCH2';
group[sub['OH']]                        := 'OH';
group[sub['CH3OH']]                     := 'CH3OH';
group[sub['H2O']]                       := 'H2O';
group[sub['ACOH']]                      := 'ACOH';
group[sub['CH2CO']]                     := 'CH2CO';
group[sub['CCl3']]                      := 'CCl3';
group[sub['CCOO']]                      := 'CCOO';
    a['CH2']['CH2'] := 0.0{K};
    a['CH2']['C=C'] := -200{K};
    a['CH2']['ACH'] := 61.13{K};
    a['CH2']['ACCH2'] := 76.50{K};
    a['CH2']['OH'] := 986.5{K};
    a['CH2']['CH3OH'] := 697.2{K};
    a['CH2']['H2O'] := 1318.0{K};
    a['CH2']['ACOH'] := 1333.0{K};
    a['CH2']['CH2CO'] := 476.4{K};
    a['CH2']['CCl3'] := 24.90{K};
    a['CH2']['CCOO'] := 232.1{K};
    a['C=C']['CH2'] := 2520{K};
    a['C=C']['C=C'] := 0.0{K};
    a['C=C']['ACH'] := 340.7{K};
    a['C=C']['ACCH2'] := 4102{K};
    a['C=C']['OH'] := 693.9{K};
    a['C=C']['CH3OH'] := 1509.0{K};
    a['C=C']['H2O'] := 634.2{K};
    a['C=C']['ACOH'] := 547.4{K};
    a['C=C']['CH2CO'] := 524.5{K};
    a['C=C']['CCl3'] := 4584.0{K};
    a['C=C']['CCOO'] := 71.23{K};
    a['ACH']['CH2'] := -11.12{K};
    a['ACH']['C=C'] := -94.78{K};
    a['ACH']['ACH'] := 0.0{K};
    a['ACH']['ACCH2'] := 167.0{K};
    a['ACH']['OH'] := 636.10{K};
    a['ACH']['CH3OH'] := 637.3{K};
    a['ACH']['H2O'] := 903.8{K};
    a['ACH']['ACOH'] := 1329.0{K};
    a['ACH']['CH2CO'] := 25.77{K};
    a['ACH']['CCl3'] := -231.9{K};
```

```
a['ACH']['CCOO'] := 5.994{K};
a['ACCH2']['CH2'] := -69.70{K};
a['ACCH2']['C=C'] := -269.7{K};
a['ACCH2']['ACH'] := -146.80{K};
a['ACCH2']['ACCH2'] := 0.0{K};
a['ACCH2']['OH'] := 803.20{K};
a['ACCH2']['CH3OH'] := 603.2{K};
a['ACCH2']['H2O'] := 5695.00{K};
a['ACCH2']['ACOH'] := 547.4{K};
a['ACCH2']['CH2CO'] := -52.10{K};
a['ACCH2']['CCl3'] := -12.14{K};
a['ACCH2']['CCOO'] := 5688.0{K};
a['OH']['CH2'] := 156.40{K};
a['OH']['C=C'] := 8694.0{K};
a['OH']['ACH'] := 89.60{K};
a['OH']['ACCH2'] := 25.82{K};
a['OH']['OH'] := 0.0{K};
a['OH']['CH3OH'] := -137.1{K};
a['OH']['H2O'] := 353.50{K};
a['OH']['ACOH'] := -259.7{K};
a['OH']['CH2CO'] := 84.0{K};
a['OH']['CCl3'] := -98.12{K};
a['OH']['CCOO'] := 101.1{K};
a['CH3OH']['CH2'] := 16.51{K};
a['CH3OH']['C=C'] := -52.39{K};
a['CH3OH']['ACH'] := -50.00{K};
a['CH3OH']['ACCH2'] := -44.50{K};
a['CH3OH']['OH'] := 249.1{K};
a['CH3OH']['CH3OH'] := 0.0{K};
a['CH3OH']['H2O'] := -181.0{K};
a['CH3OH']['ACOH'] := -101.7{K};
a['CH3OH']['CH2CO'] := 23.39{K};
a['CH3OH']['CCl3'] := -139.4{K};
a['CH3OH']['CCOO'] := -10.72{K};
a['H2O']['CH2'] := 300.00{K};
a['H2O']['C=C'] := 692.7{K};
a['H2O']['ACH'] := 362.30{K};
a['H2O']['ACCH2'] := 377.60{K};
a['H2O']['OH'] := -229.10{K};
a['H2O']['CH3OH'] := 289.6{K};
a['H2O']['H2O'] := 0.0{K};
a['H2O']['ACOH'] := 324.5{K};
a['H2O']['CH2CO'] := -195.40{K};
a['H2O']['CCl3'] := 353.7{K};
a['H2O']['CCOO'] := 14.42{K};
a['ACOH']['CH2'] := 275.8{K};
a['ACOH']['C=C'] := 1665.0{K};
a['ACOH']['ACH'] := 25.34{K};
a['ACOH']['ACCH2'] := 244.2{K};
a['ACOH']['OH'] := -451.6{K};
a['ACOH']['CH3OH'] := -265.2{K};
a['ACOH']['H2O'] := -601.8{K};
a['ACOH']['ACOH'] := 0.0{K};
a['ACOH']['CH2CO'] := -356.1{K};
a['ACOH']['CCl3'] := 0.0{K};
a['ACOH']['CCOO'] := -449.4{K};
a['CH2CO']['CH2'] := 26.76{K};
a['CH2CO']['C=C'] := -82.92{K};
a['CH2CO']['ACH'] := 140.10{K};
```

```
a['CH2CO']['ACCH2'] := 365.80{K};
a['CH2CO']['OH'] := 164.5{K};
a['CH2CO']['CH3OH'] := 108.7{K};
a['CH2CO']['H2O'] := 472.5{K};
a['CH2CO']['ACOH'] := -133.1{K};
a['CH2CO']['CH2CO'] := 0.0{K};
a['CH2CO']['CCl3'] := -354.6{K};
a['CH2CO']['CCOO'] := -213.7{K};
a['CCl3']['CH2'] := 36.70{K};
a['CCl3']['C=C'] := -185.1{K};
a['CCl3']['ACH'] := 288.5{K};
a['CCl3']['ACCH2'] := 33.61{K};
a['CCl3']['OH'] := 742.1{K};
a['CCl3']['CH3OH'] := 649.1{K};
a['CCl3']['H2O'] := 826.7{K};
a['CCl3']['ACOH'] := 0.0{K};
a['CCl3']['CH2CO'] := 552.1{K};
a['CCl3']['CCl3'] := 0.0{K};
a['CCl3']['CCOO'] := 176.5{K};
a['CCOO']['CH2'] := 114.8{K};
a['CCOO']['C=C'] := 269.3{K};
a['CCOO']['ACH'] := 85.84{K};
a['CCOO']['ACCH2'] := -170.0{K};
a['CCOO']['OH'] := 245.4{K};
a['CCOO']['CH3OH'] := 249.6{K};
a['CCOO']['H2O'] := 10000.0{K};
a['CCOO']['ACOH'] := -36.72{K};
a['CCOO']['CH2CO'] := 372.2{K};
a['CCOO']['CCl3'] := -209.7{K};
a['CCOO']['CCOO'] := 0.0{K};
R['CH3'] := 0.9011;
R['CH2']  := 0.6744;
R['CH']  := 0.4469;
R['C']  := 0.2195;
R['CH2=CH']  := 1.3454;
R['CH=CH']  := 1.1167;
R['CH2=C']  := 1.1173;
R['CH=C']  := 0.8886;
R['C=C']  := 0.6605;
R['ACH']  := 0.5313;
R['AC']  := 0.3652;
R['ACCH3']  := 1.2663;
R['ACCH2']  := 1.0396;
R['ACCH']  := 0.8121;
R['OH']  := 1.000;
R['CH3OH']  := 1.4311;
R['H2O']  := 0.9200;
R['ACOH']  := 0.8952;
R['CH3CO']  := 1.6724;
R['CH2CO']  := 1.4457;
R['CHCl3']  := 2.87007;
R['CCl3'] := 2.6401;
R['CH3COO'] := 1.9031;
R['CH2COO'] := 1.6764;
Q['CH3'] := 0.848;
Q['CH2'] := 0.540;
Q['CH'] := 0.228;
Q['C'] := 0.0;
Q['CH2=CH'] := 1.176;
```

```
        Q['CH=CH'] := 0.867;
        Q['CH2=C'] := 0.988;
        Q['CH=C'] := 0.676;
        Q['C=C'] := 0.485;
        Q['ACH'] := 0.40;
        Q['AC'] := 0.120;
        Q['ACCH3'] := 0.968;
        Q['ACCH2'] := 0.660;
        Q['ACCH'] := 0.348;
        Q['OH'] := 1.200;
        Q['CH3OH'] := 1.432;
        Q['H2O'] := 1.400;
        Q['ACOH'] := 0.680;
        Q['CH3CO'] := 1.488;
        Q['CH2CO'] := 1.180;
        Q['CHCl3'] := 2.410;
        Q['CCl3'] := 2.184;
        Q['CH3COO'] := 1.728;
        Q['CH2COO'] := 1.420;
END UNIFAC_constants;
MODEL component_constants;
    formula                              IS_A symbol;
    groups                               IS_A set OF symbol;
    subgroups                            IS_A set OF symbol;
    wilson_set                           IS_A set OF symbol;
    mw,
    Tb, Tc, Pc, Vc, Zc, omega,
    cpvapa, cpvapb, cpvapc, cpvapd,
    Hf, Gf, vpa, vpb, vpc, vpd,
    Hv, Tliq, Vliq, T0, P0, H0, G0, S0   IS_A constant;
    nu[subgroups]                        IS_A constant;
    lambda[wilson_set]                   IS_A constant;
    del_ip[wilson_set]                   IS_A constant;
        T0 := 298.15{K};
        P0 := 1.0{atm};
END component_constants;
UNIVERSAL MODEL chloroform REFINES component_constants;
    formula                              := 'CHCl3';
    groups                               := ['CCl3'];
    subgroups                            := ['CHCl3'];
    wilson_set                           := ['CHCl3','C6H6','(CH3)2CO'];
        mw := 119.378{g/g_mole};
        Tb := 334.3{K};
        Tc := 536.4{K};
        Pc := 54.0{atm};
        Vc := 239.0{cm^3/g_mole};
        Zc := 0.293;
        omega := 0.216;
        cpvapa := 5.733{cal/g_mole/K};
        cpvapb := 4.522e-2{cal/g_mole/K^2};
        cpvapc := -4.397e-5{cal/g_mole/K^3};
        cpvapd := 1.590e-8{cal/g_mole/K^4};
        Hf := -101238{J/g_mole};
        Gf := -68524{J/g_mole};
        vpa := -6.95546;
        vpb := 1.16625;
        vpc := -2.13970;
        vpd := -3.44421;
        Hv := 29702{J/g_mole};
```

```
        Vliq := 80.173{cm^3/g_mole};
        Tliq := 293.15{K};
        H0 := -101238{J/g_mole};
        G0 := -68524{J/g_mole};
        S0 := (-101238{J/g_mole} +68524{J/g_mole})/298.15{K};
        nu['CHCl3'] := 1;
        lambda['(CH3)2CO'] := 1.49288;
        lambda['C6H6'] := 1.92309;
        lambda['CHCl3'] := 1.0;
        del_ip['(CH3)2CO'] := -1359.32 {J/g_mole};
        del_ip['C6H6'] := -1638.28 {J/g_mole};
        del_ip['CHCl3'] := 0.0 {J/g_mole};
END chloroform;
UNIVERSAL MODEL acetone REFINES component_constants;
    formula                              := '(CH3)2CO';
    groups                               := ['CH2', 'CH2CO'];
    subgroups                            := ['CH3', 'CH3CO'];
    wilson_set := ['H2O','CH3OH','CHCl3','C6H6','(CH3)2CO'];
        mw := 58.08{g/g_mole};
        Tb := 329.4{K};
        Tc := 508.1{K};
        Pc := 46.4{atm};
        Vc := 209.0{cm^3/g_mole};
        Zc := 0.232;
        omega := 0.309;
        cpvapa := 1.505{cal/g_mole/K};
        cpvapb := 6.244e-2{cal/g_mole/K^2};
        cpvapc := -2.992e-5{cal/g_mole/K^3};
        cpvapd := 4.867e-9{cal/g_mole/K^4};
        Hf := -217536{J/g_mole};
        Gf := -153028{J/g_mole};
        vpa := -7.45514;
        vpb := 1.20200;
        vpc := -2.43926;
        vpd := -3.35590;
        Hv := 29116{J/g_mole};
        Vliq := 73.333{cm^3/g_mole};
        Tliq := 293.15{K};
        H0 := -217536.0{J/g_mole};
        G0 := -153028{J/g_mole};
        S0 := (-217536{J/g_mole} +153028{J/g_mole})/298.15{K};
        nu['CH3']  := 1;
        nu['CH3CO'] := 1;
        lambda['H2O'] := 0.16924;
        lambda['CH3OH'] := 0.65675;
        lambda['CHCl3'] := 1.29452;
        lambda['C6H6'] := 0.75632;
        lambda['(CH3)2CO'] := 1.0;
        del_ip['CH3OH'] := 23.2321 {J/g_mole};
        del_ip['H2O'] := 1674.817 {J/g_mole};
        del_ip['C6H6'] := 1364.63 {J/g_mole};
        del_ip['CHCl3'] := -468.831 {J/g_mole};
        del_ip['(CH3)2CO'] := 0.0 {J/g_mole};
END acetone;
UNIVERSAL MODEL benzene REFINES component_constants;
    formula                              := 'C6H6';
    groups                               := ['ACH'];
    subgroups                            := ['ACH'];
    wilson_set                           := ['CHCl3','C6H6','(CH3)2CO'];
```

```
        mw := 78.114{g/g_mole};
        Tb := 353.252{K};
        Tc := 562.2{K};
        Pc := 48.3{atm};
        Vc := 259.0{cm^3/g_mole};
        Zc := 0.271;
        omega := 0.212;
        cpvapa := -8.101{cal/g_mole/K};
        cpvapb := 1.133e-1{cal/g_mole/K^2};
        cpvapc := -7.206e-5{cal/g_mole/K^3};
        cpvapd := 1.703e-8{cal/g_mole/K^4};
        Hf := 82980.0{J/g_mole};
        Gf := 129662.0{J/g_mole};
        vpa := -6.98273;
        vpb := 1.33213;
        vpc := -2.62863;
        vpd := -3.33399;
        Hv := 30760.0{J/g_mole};
        Vliq := 88.2644{cm^3/g_mole};
        Tliq := 289.0{K};
        H0 := 82980.0{J/g_mole};
        G0 := 129662.0{J/g_mole};
        S0 := (82980.0{J/g_mole} -129662.0{J/g_mole})/298.15{K};
        nu['ACH'] := 6;
        lambda['CHCl3'] := 0.52431;
        lambda['(CH3)2CO'] := 0.87629;
        lambda['C6H6'] := 1.0;
        del_ip['CHCl3'] := 1614.0 {J/g_mole};
        del_ip['(CH3)2CO'] := -156.458 {J/g_mole};
        del_ip['C6H6'] := 0.0 {J/g_mole};
END benzene;
```

```
(*************************************************************************\
  H_G_thermodynamics.lib
  by Joseph J. Zaher,
  Modified structurally by Robert S. Huss
  Part of the Ascend Library
  This file is part of the Ascend modeling library.
  Copyright (C) 1994
  The Ascend modeling library is free software; you can redistribute
  it and/or modify it under the terms of the GNU General Public License as
  published by the Free Software Foundation; either version 2 of the
  License, or (at your option) any later version.
  The Ascend Language Interpreter is distributed in hope that it will be
  useful, but WITHOUT ANY WARRANTY; without even the implied warranty of
  MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the GNU
  General Public License for more details.
  You should have received a copy of the GNU General Public License along with
  the program; if not, write to the Free Software Foundation, Inc., 675
  Mass Ave, Cambridge, MA 02139 USA.  Check the file named COPYING.
  \*************************************************************************)
(**
  **
  **  T  H  E  R  M  O  D  Y  N  A  M  I  C  S  .  L  I  B
  **  ---------------------------------------------------
  **
  **  AUTHOR:      Joseph J. Zaher
  **
  **  DATES:       07/91 - Original code.
  **               02/92 - Made compatible with new set version of ASCEND.
  **                       Scaled equations to enhance convergence, updated
  **                       vapor pressure correlation, added Pitzer extension
  **                       to vapor mixtures and UNIFAC extension to liquid
  **                       mixtures with help of Bob Huss.
  **               03/92 - Removed stream model.  Library remains purely
  **                       intensive without any assumption to static or
  **                       dynamic modeling.
  **               07/92 - Structural changes to provide a common thermodynamic
  **                       properties root model as the library interface.
  **                       Modified the existing phase distribution model
  **                       to incorporate an intensive mass balance over the
  **                       phases.  Residual quantities for pure vapor
  **                       components estimate corrections from ideal gas
  **                       behavior while residual quantities for pure liquid
  **                       components estimate corrections from incompressible
  **                       fluid behavior.
  **               08/92 - Allowed component names in mixtures to be freely
  **                       specified by user.
  **               03/94 - Made compatible with gnu-ascend.
  **               05/94 - Removed refinement link of models correction and
  **                       and partial_component which should not contain T,
  **                       P, and R anyway.  The interface to the library
  **                       is now returned to model thermodynamic_properties
  **                       where refinement to pure_component,
  **                       homogeneous_mixture, or heterogeneous_mixture
  **                       is possible.
  **               06/94 - Changed canonical variables from V, H, and S to
  **                       V, H, and G. Also, liquid component model was
  **                       condensed, eliminating instance saturated.
  **
  **               08/94 - Slight structural changes made by Bob Huss to
```

```
  **                      allow refinement of non-thermodynamic models,
  **                      and to include Wilson liquid mixture written
  **                      by Boyd Safrit.
  **
  **  CONTENTS:    ASCEND structure for calculating the basic set of intensive
  **               thermodynamic properties molar volume, enthalpy, and
  **               entropy for single and multiple phase streams of pure and
  **               mixed components. Specify procedures are included which
  **               have been designed to provide a means of calculating ideal
  **               approximations when base models are used.  For pure
  **               component vapors, the ideal gas law can be obtained whereas
  **               for pure component liquids, incompressibility can be
  **               specified.  Ideal vapor and liquid mixtures are maintained
  **               by setting all partial molar excess properties to zero.
  **               Distribution of components among multiple phases can be
  **               ideally computed using constant relative volatilities.
  **
  **               For more rigorous non-ideal calculations, some generalized
  **               refinements of the base models are provided.  For pure
  **               component vapors, a Pitzer correlation of the two term
  **               virial equation allows a more accurate compressibility and
  **               residual calculation.  The widely used Rackett correlation
  **               is accurate in estimating the effect of temperature on
  **               liquid volumes.  Non-ideal vapor mixtures are computed using
  **               an extension of the Pitzer correlation where the exact
  **               composition dependence of the second virial coefficient is
  **               given by statistical mechanics.  A reliable UNIFAC model
  **               estimates non-ideal liquid mixtures.  Phase equilibrium
  **               can be enforced rigorously among multiple phases which
  **               in turn will allow calculation of the true relative
  **               volatilities.
  **
  **  REQUIRES:    “atoms.lib”
  **               “components.lib”
  **
  **)
MODEL thermodynamic_properties;
        T                                       IS_A temperature;
        P                                       IS_A pressure;
        V                                       IS_A molar_volume;
        H                                       IS_A molar_energy;
        G                                       IS_A molar_energy;
        R                                       IS_A gas_constant;
        scale                             IS_A scaling_constant;
  INITIALIZATION
    PROCEDURE specify;
        T.fixed := TRUE;
        P.fixed := TRUE;
        V.fixed := TRUE;
        H.fixed := TRUE;
        G.fixed := TRUE;
    END specify;
    PROCEDURE reset;
        RUN clear;
        RUN specify;
    END reset;
END thermodynamic_properties;
MODEL pure_component REFINES thermodynamic_properties;
        data                                    IS_A component_constants;
```

```
END pure_component;
MODEL Pitzer_component REFINES pure_component;
        P*V/R/data.Tc = T/data.Tc + (P/data.Pc)*
            (0.083 - 0.422*(data.Tc/T)^1.6 + data.omega*
            (0.139 - 0.172*(data.Tc/T)^4.2));
        H/R/data.Tc = data.H0/R/data.Tc +
            data.cpvapa*(T - data.T0)/R/data.Tc +
            data.cpvapb*(T^2 - data.T0^2)/2/R/data.Tc +
            data.cpvapc*(T^3 - data.T0^3)/3/R/data.Tc +
            data.cpvapd*(T^4 - data.T0^4)/4/R/data.Tc +
            (P/data.Pc)*
            (0.083 - 1.097*(data.Tc/T)^1.6 + data.omega*
            (0.139 - 0.894*(data.Tc/T)^4.2));
        G/R/data.Tc = data.G0/R/data.Tc -
            (data.H0 - data.G0)*(T/data.T0 - 1)/R/data.Tc -
            data.cpvapa*(T*ln(T/data.T0) - T + data.T0)/R/data.Tc -
            data.cpvapb*(T^2 - 2*T*data.T0 + data.T0^2)/2/R/data.Tc -
            data.cpvapc*(T^3/2 - 3*T*data.T0^2/2 + data.T0^3)/3/R/data.Tc -
            data.cpvapd*(T^4/3 - 4*T*data.T0^3/3 + data.T0^4)/4/R/data.Tc +
            T*ln(P/data.P0)/data.Tc +
            (P/data.Pc)*
            (0.083 - 0.422*(data.Tc/T)^1.6 + data.omega*
            (0.139 - 0.172*(data.Tc/T)^4.2));
        T.lower_bound := 1.0e-12{K};
        P.lower_bound := 1.0e-12{Pa};
        V := 24{liter/mole};
  INITIALIZATION
    PROCEDURE specify;
        T.fixed := TRUE;
        P.fixed := TRUE;
    END specify;
END Pitzer_component;
MODEL Rackett_component REFINES pure_component;
        VP                                      IS_A pressure;
        ln(VP/data.Pc)*T/data.Tc =
            data.vpa*sqrt(sqr(1.0 - T/data.Tc)) +
            data.vpb*sqrt(sqr(1.0 - T/data.Tc))^1.5 +
            data.vpc*sqrt(sqr(1.0 - T/data.Tc))^3.0 +
            data.vpd*sqrt(sqr(1.0 - T/data.Tc))^6.0;
        V/data.Vliq =
            data.Zc^(sqrt(sqr(1.0 - T/data.Tc))^(2/7))/
            data.Zc^(sqrt(sqr(1.0 - data.Tliq/data.Tc))^(2/7));
        H/R/data.Tc = data.H0/R/data.Tc +
            data.cpvapa*(T - data.T0)/R/data.Tc +
            data.cpvapb*(T^2 - data.T0^2)/2/R/data.Tc +
            data.cpvapc*(T^3 - data.T0^3)/3/R/data.Tc +
            data.cpvapd*(T^4 - data.T0^4)/4/R/data.Tc +
            (VP/data.Pc)*
            (0.083 - 1.097*(data.Tc/T)^1.6 + data.omega*
            (0.139 - 0.894*(data.Tc/T)^4.2)) -
            (data.Hv/R/data.Tc)*sqrt(sqr((data.Tc-T)/(data.Tc-data.Tb)))^0.38 +
            (P - VP)*(data.Vliq/R/data.Tc)*
            (data.Zc^(sqrt(sqr(1.0 - T/data.Tc))^(2/7))/
            data.Zc^(sqrt(sqr(1.0 - data.Tliq/data.Tc))^(2/7)))*(1.0 -
            (-2/7)*(T/data.Tc)*(sqrt(sqr(1 - T/data.Tc))^(-5/7))*ln(data.Zc));
        G/R/data.Tc = data.G0/R/data.Tc -
            (data.H0 - data.G0)*(T/data.T0 - 1)/R/data.Tc -
            data.cpvapa*(T*ln(T/data.T0) - T + data.T0)/R/data.Tc -
            data.cpvapb*(T^2 - 2*T*data.T0 + data.T0^2)/2/R/data.Tc -
```

```
            data.cpvapc*(T^3/2 - 3*T*data.T0^2/2 + data.T0^3)/3/R/data.Tc -
            data.cpvapd*(T^4/3 - 4*T*data.T0^3/3 + data.T0^4)/4/R/data.Tc +
            T*ln(VP/data.P0)/data.Tc +
            (VP/data.Pc)*
            (0.083 - 0.422*(data.Tc/T)^1.6 + data.omega*
            (0.139 - 0.172*(data.Tc/T)^4.2)) +
            (P - VP)*(data.Vliq/R/data.Tc)*
            (data.Zc^(sqrt(sqr(1.0 - T/data.Tc))^(2/7))/
            data.Zc^(sqrt(sqr(1.0 - data.Tliq/data.Tc))^(2/7)));
        VP.lower_bound := 1.0e-12{Pa};
        V := 0.1{liter/mole};
  INITIALIZATION
    PROCEDURE specify;
        T.fixed := TRUE;
        P.fixed := TRUE;
    END specify;
END Rackett_component;
MODEL partial_component;
        V                                   IS_A molar_volume;
        H                                   IS_A molar_energy;
        G                                   IS_A molar_energy;
        scale                           IS_A scaling_constant;
  INITIALIZATION
    PROCEDURE specify;
        V.fixed := TRUE;
        H.fixed := TRUE;
        G.fixed := TRUE;
    END specify;
    PROCEDURE reset;
        RUN clear;
        RUN specify;
    END reset;
END partial_component;
MODEL mixture;
        components                      IS_A set OF symbol;
        y[components]                   IS_A fraction;
        scale                           IS_A scaling_constant;
        SUM(y[i] | i IN components) = 1.0;
  INITIALIZATION
    PROCEDURE specify;
        y[components-[CHOICE(components)]].fixed := TRUE;
    END specify;
    PROCEDURE reset;
        RUN clear;
        RUN specify;
    END reset;
END mixture;
MODEL homogeneous_mixture REFINES mixture;
END homogeneous_mixture;
MODEL td_homogeneous_mixture REFINES homogeneous_mixture;
        T                                   IS_A temperature;
        P                                   IS_A pressure;
        V                                   IS_A molar_volume;
        H                                   IS_A molar_energy;
        G                                   IS_A molar_energy;
        R                                   IS_A gas_constant;
        data[components]                    IS_A component_constants;
        pure[components]                    IS_A pure_component;
        partial[components]                 IS_A partial_component;
```

```
        T, pure[components].T ARE_THE_SAME;
        P, pure[components].P ARE_THE_SAME;
        pure[components] ARE_ALIKE;
        V*data[CHOICE(components)].Pc/R/
            data[CHOICE(components)].Tc =
            SUM(y[i]*partial[i].V | i IN components)*
            data[CHOICE(components)].Pc/R/
            data[CHOICE(components)].Tc;
        H/R/data[CHOICE(components)].Tc =
            SUM(y[i]*partial[i].H | i IN components)/R/
            data[CHOICE(components)].Tc;
        G/R/data[CHOICE(components)].Tc =
            SUM(y[i]*partial[i].G | i IN components)/R/
            data[CHOICE(components)].Tc;
        FOR i IN components CREATE
            data[i], pure[i].data ARE_THE_SAME;
        END;
        y[components].lower_bound := 1.0e-12;
  INITIALIZATION
    PROCEDURE specify;
        RUN pure[components].specify;
        RUN partial[components].specify;
        y[components].fixed := TRUE;
        y[CHOICE(components)].fixed := FALSE;
    END specify;
END td_homogeneous_mixture;
MODEL Pitzer_mixture REFINES td_homogeneous_mixture;
        pure[components]                    IS_REFINED_TO Pitzer_component;
        FOR i IN components CREATE
            (partial[i].V - pure[i].V)*data[i].Pc/R/data[i].Tc =
                -1.0*
                (0.083 - 0.422*(data[i].Tc/T)^1.6 + data[i].omega*
                (0.139 - 0.172*(data[i].Tc/T)^4.2))*(1.0 - y[i])^2 +
                0.50*(1.0 - y[i])*1.0*
                SUM(y[j]*((1.0 + (data[j].Vc/data[i].Vc)^(1/3))^3/
                (1.0 + data[j].Zc/data[i].Zc))*
                (0.083 - 0.422*(sqrt(data[i].Tc*data[j].Tc)/T)^1.6 +
                0.5*(data[i].omega + data[j].omega)*
                (0.139 - 0.172*(sqrt(data[i].Tc*data[j].Tc)/T)^4.2))
                | j IN components - [i]);
            (partial[i].H - pure[i].H)/R/data[i].Tc =
                -(P/data[i].Pc)*
                (0.083 - 1.097*(data[i].Tc/T)^1.6 + data[i].omega*
                (0.139 - 0.894*(data[i].Tc/T)^4.2))*(1.0 - y[i])^2 +
                0.50*(1.0 - y[i])*(P/data[i].Pc)*
                SUM(y[j]*((1.0 + (data[j].Vc/data[i].Vc)^(1/3))^3/
                (1.0 + data[j].Zc/data[i].Zc))*
                (0.083 - 1.097*(sqrt(data[i].Tc*data[j].Tc)/T)^1.6 +
                0.5*(data[i].omega + data[j].omega)*
                (0.139 - 0.894*(sqrt(data[i].Tc*data[j].Tc)/T)^4.2))
                | j IN components - [i]);
            (partial[i].G - pure[i].G - R*T*ln(y[i]))/R/data[i].Tc =
                -(P/data[i].Pc)*
                (0.083 - 0.422*(data[i].Tc/T)^1.6 + data[i].omega*
                (0.139 - 0.172*(data[i].Tc/T)^4.2))*(1.0 - y[i])^2 +
                0.50*(1.0 - y[i])*(P/data[i].Pc)*
                SUM(y[j]*((1.0 + (data[j].Vc/data[i].Vc)^(1/3))^3/
                (1.0 + data[j].Zc/data[i].Zc))*
                (0.083 - 0.422*(sqrt(data[i].Tc*data[j].Tc)/T)^1.6 +
```

```
                0.5*(data[i].omega + data[j].omega)*
                (0.139 - 0.172*(sqrt(data[i].Tc*data[j].Tc)/T)^4.2))
                | j IN components - [i]);
        END;
        V := 24{liter/mole};
        partial[components].V := 24{liter/mole};
        pure[components].V := 24{liter/mole};
  INITIALIZATION
    PROCEDURE specify;
        RUN td_homogeneous_mixture::specify;
        partial[components].V.fixed := FALSE;
        partial[components].H.fixed := FALSE;
        partial[components].G.fixed := FALSE;
    END specify;
END Pitzer_mixture;
MODEL UNIFAC_mixture REFINES td_homogeneous_mixture;
        pure[components]                        IS_REFINED_TO Rackett_component;
        subgroups                               IS_A set OF symbol;
        groups                                  IS_A set OF symbol;
        comps[subgroups]                        IS_A set OF symbol;
        rv[components]                          IS_A constant;
        qs[components]                          IS_A constant;
        Jv[components]                          IS_A factor;
        Ls[components]                          IS_A factor;
        theta[subgroups]                        IS_A factor;
        eta[subgroups]                          IS_A factor;
        uc                                      IS_A UNIFAC_constants;
        subgroups := UNION(data[i].subgroups | i IN components);
        groups := UNION(data[i].groups | i IN components);
        FOR k IN subgroups CREATE
            comps[k] := [i IN components | k IN data[i].subgroups];
        END;
        FOR k IN subgroups CREATE
            theta[k] = uc.Q[k]*SUM(data[i].nu[k]*y[i] | i IN comps[k]);
            eta[k] =
                SUM(theta[m] | m IN subgroups*uc.sub[uc.group[k]]) +
                SUM(SUM(theta[m]*exp(-uc.a[g][uc.group[k]]/T)
                | m IN subgroups*uc.sub[g])
                | g IN groups - [uc.group[k]]);
        END;
        FOR i IN components CREATE
            rv[i] = Jv[i]*SUM(rv[j]*y[j] | j IN components);
            qs[i] = Ls[i]*SUM(qs[j]*y[j] | j IN components);
            partial[i].V,
                pure[i].V       ARE_THE_SAME;

            (partial[i].H - pure[i].H)/R/data[i].Tc =
                SUM(theta[k]*
                SUM(SUM(theta[n]*
                ((uc.a[g][uc.group[k]] -
                uc.a[uc.group[n]][uc.group[k]])/data[i].Tc)*
                exp(-(uc.a[g][uc.group[k]] +
                uc.a[uc.group[n]][uc.group[k]])/T)*
                SUM(data[i].nu[m]*uc.Q[m]
                | m IN data[i].subgroups*uc.sub[g])
                | g IN data[i].groups - [uc.group[n]])
                | n IN subgroups)/eta[k]/eta[k]
                | k IN subgroups) -
                SUM((data[i].nu[k]*uc.Q[k]/(
```

```
                SUM(data[i].nu[m]*uc.Q[m]
                | m IN data[i].subgroups*uc.sub[uc.group[k]]) +
                SUM(SUM(data[i].nu[m]*uc.Q[m]*exp(-uc.a[g][uc.group[k]]/T)
                | m IN data[i].subgroups*uc.sub[g])
                | g IN data[i].groups - [uc.group[k]])))*
                SUM(SUM(theta[n]*
                ((uc.a[g][uc.group[k]] -
                uc.a[uc.group[n]][uc.group[k]])/data[i].Tc)*
                exp(-(uc.a[g][uc.group[k]] +
                uc.a[uc.group[n]][uc.group[k]])/T)*
                SUM(data[i].nu[m]*uc.Q[m]
                | m IN data[i].subgroups*uc.sub[g])
                | g IN data[i].groups - [uc.group[n]])
                | n IN subgroups)/eta[k]
                | k IN data[i].subgroups);
            (partial[i].G - pure[i].G - R*T*ln(y[i]))/R/data[i].Tc =
                (1.0 - Jv[i] + ln(Jv[i]) -
                5.0*qs[i]*(1.0 - Jv[i]/Ls[i] + ln(Jv[i]/Ls[i])) +
                qs[i]*(1 - ln(Ls[i])))*T/data[i].Tc -
                SUM(theta[k]*(
                SUM(data[i].nu[m]*uc.Q[m]
                | m IN data[i].subgroups*uc.sub[uc.group[k]]) +
                SUM(SUM(data[i].nu[m]*uc.Q[m]*exp(-uc.a[g][uc.group[k]]/T)
                | m IN data[i].subgroups*uc.sub[g])
                | g IN data[i].groups - [uc.group[k]]))/eta[k]
                | k IN subgroups)*T/data[i].Tc +
                SUM(data[i].nu[k]*uc.Q[k]*ln((
                SUM(data[i].nu[m]*uc.Q[m]
                | m IN data[i].subgroups*uc.sub[uc.group[k]]) +
                SUM(SUM(data[i].nu[m]*uc.Q[m]*exp(-uc.a[g][uc.group[k]]/T)
                | m IN data[i].subgroups*uc.sub[g])
                | g IN data[i].groups - [uc.group[k]]))/eta[k])
                | k IN data[i].subgroups)*T/data[i].Tc;
        END;
        Jv[components].lower_bound := 1.0e-12;
        Ls[components].lower_bound := 1.0e-12;
        theta[subgroups].lower_bound := 0.0;
        eta[subgroups].lower_bound := 0.0;
        V := 0.1{liter/mole};
        partial[components].V := 0.1{liter/mole};
        pure[components].V := 0.1{liter/mole};
  INITIALIZATION
    PROCEDURE specify;
        RUN td_homogeneous_mixture::specify;
        partial[components].V.fixed := FALSE;
        partial[components].H.fixed := FALSE;
        partial[components].G.fixed := FALSE;
        FOR i IN components DO
            rv[i] := 0.0;
            qs[i] := 0.0;
            FOR k IN data[i].subgroups DO
                rv[i] := rv[i] + data[i].nu[k]*uc.R[k];
                qs[i] := qs[i] + data[i].nu[k]*uc.Q[k];
            END;
        END;
    END specify;
END UNIFAC_mixture;
MODEL Wilson_mixture REFINES td_homogeneous_mixture;
        pure[components]                        IS_REFINED_TO Rackett_component;
```

```
        lambda[components][components]          IS_A factor;
        FOR i IN components CREATE
            FOR j IN components CREATE
                lambda[i][j] = (pure[j].V/pure[i].V)*
                    exp(-pure[i].data.del_ip[pure[j].data.formula]/(R*T));
            END;
        END;
        FOR i IN components CREATE
            partial[i].V, pure[i].V             ARE_THE_SAME;
            partial[i].G - pure[i].G - R*T*ln(y[i]) = R*T*(-ln(
                SUM(y[j]*lambda[i][j]
                | j IN components)) + 1 -
                SUM((y[k]*lambda[k][i]) / (
                SUM(y[j]*lambda[k][j]
                | j IN components))
                | k IN components));
            partial[i].H - pure[i].H = R*T^2*((SUM(y[j]*
                (-lambda[i][j] * ln(lambda[i][j]))/T
                | j IN components)) /
                SUM(y[j]*lambda[i][j]
                | j IN components) +
                SUM((
                SUM(y[l]*lambda[k][l]*y[k]*((-lambda[k][i] *
                ln(lambda[k][i]))/T)
                | l IN components) - y[k]*lambda[k][i] *
                SUM(y[l]*((-lambda[k][l] * ln(lambda[k][l]))/T)
                | l IN components)) / (
                SUM(y[l]*lambda[k][i]
                | l IN components)^2)
                | k IN components));
        END;
        V := 0.1{liter/mole};
        partial[components].V := 0.1{liter/mole};
        pure[components].V := 0.1{liter/mole};
  INITIALIZATION
    PROCEDURE specify;
        RUN td_homogeneous_mixture::specify;
        partial[components].V.fixed := FALSE;
        partial[components].G.fixed := FALSE;
        partial[components].H.fixed := FALSE;
    END specify;
    PROCEDURE reset;
        RUN clear;
        RUN specify;
    END reset;
END Wilson_mixture;
MODEL heterogeneous_mixture REFINES mixture;
        reference                       IS_A symbol;
        phases                          IS_A set OF symbol;
        mix[phases]                     IS_A homogeneous_mixture;
        alpha[phases-[reference]][components]           IS_A factor;
        ave_alpha[phases-[reference]]                   IS_A factor;
        phi[phases]                     IS_A fraction;
        components, mix[phases].components ARE_THE_SAME;
        FOR i IN components CREATE
            y[i] = SUM(phi[j]*mix[j].y[i] | j IN phases);
            FOR j IN phases - [reference] CREATE
                ave_alpha[j]*mix[j].y[i] =
                    alpha[j][i]*mix[reference].y[i];
```

```
            END;
        END;
  INITIALIZATION
    PROCEDURE specify;
        RUN mix[phases].specify;
        alpha[phases - [reference]][components].fixed := TRUE;
        y[components-[CHOICE(components)]].fixed := TRUE;
        mix[phases].y[components].fixed := FALSE;
        phi[phases-[reference]].fixed := TRUE;
    END specify;
END heterogeneous_mixture;
MODEL td_heterogeneous_mixture REFINES heterogeneous_mixture;
        T                                       IS_A temperature;
        P                                       IS_A pressure;
        V                                       IS_A molar_volume;
        H                                       IS_A molar_energy;
        G                                       IS_A molar_energy;
        R                                       IS_A gas_constant;
        data[components]                        IS_A component_constants;
        mix[phases]             IS_REFINED_TO  td_homogeneous_mixture;
        FOR i IN components CREATE
            data[i],
                mix[phases].data[i]     ARE_THE_SAME;
        END;
        T, mix[phases].T ARE_THE_SAME;
        P, mix[phases].P ARE_THE_SAME;
        V*data[CHOICE(components)].Pc/R/
            data[CHOICE(components)].Tc =
            SUM(phi[j]*mix[j].V | j IN phases)*
            data[CHOICE(components)].Pc/R/
            data[CHOICE(components)].Tc;
        H/R/data[CHOICE(components)].Tc =
            SUM(phi[j]*mix[j].H | j IN phases)/R/
            data[CHOICE(components)].Tc;
        G/R/data[CHOICE(components)].Tc =
            SUM(phi[j]*mix[j].G | j IN phases)/R/
            data[CHOICE(components)].Tc;
        components, mix[phases].components ARE_THE_SAME;
        V := 30{liter/mol};
END td_heterogeneous_mixture;
MODEL equilibrium_mixture REFINES td_heterogeneous_mixture;
        FOR i IN components CREATE
            FOR j IN phases - [reference] CREATE
                mix[j].partial[i].G, mix[reference].partial[i].G ARE_THE_SAME;
            END;
        END;
  INITIALIZATION
    PROCEDURE specify;
        RUN td_heterogeneous_mixture::specify;
        T.fixed := FALSE;
        alpha[phases - [reference]][components].fixed := FALSE;
        ave_alpha[phases - [reference]] := 1.0;
        ave_alpha[phases - [reference]].fixed := TRUE;
    END specify;
END equilibrium_mixture;
MODEL murphree_equilibrium_mixture REFINES td_heterogeneous_mixture;
        (*         ASSUMES vapor-liquid pd, with liquid reference *)
        vap_eq                                  IS_A Pitzer_mixture;
        equil_alpha[components]                 IS_A factor;
```

```
        ref_y[components]                       IS_A fraction;
        murph_eff                               IS_A factor;
        vap_eq, mix['vapor']                    ARE_ALIKE;
        T, vap_eq.T                             ARE_THE_SAME;
        P, vap_eq.P                             ARE_THE_SAME;
        components, vap_eq.components           ARE_THE_SAME;
        FOR i IN components CREATE
            data[i],
                vap_eq.data[i]                  ARE_THE_SAME;
        END;
        SUM(ref_y[components]) = 1;
        FOR i IN components CREATE
            vap_eq.y[i] = equil_alpha[i]*mix[reference].y[i];
            vap_eq.partial[i].G,
                mix[reference].partial[i].G     ARE_THE_SAME;
        END;
        FOR i IN components - [CHOICE(components)] CREATE
            murph_eff*(vap_eq.y[i] - ref_y[i]) =
                mix['vapor'].y[i] - ref_y[i];
        END;
    INITIALIZATION
      PROCEDURE specify;
        RUN td_heterogeneous_mixture::specify;
        alpha[phases - [reference]][components].fixed := FALSE;
        ave_alpha[phases - [reference]] := 1.0;
        ave_alpha[phases - [reference]].fixed := TRUE;
        RUN vap_eq.specify;
        vap_eq.y[components].fixed := FALSE;
        equil_alpha[components].fixed := FALSE;
        ref_y[components - [CHOICE(components)]].fixed := TRUE;
        murph_eff.fixed := TRUE;
        T.fixed := FALSE;
      END specify;
END murphree_equilibrium_mixture;
```

```
(*************************************************************************\
  stream.lib
  by Robert S. Huss
  Part of the Ascend Library
  This file is part of the Ascend modeling library.
  Copyright (C) 1993,1994
  The Ascend modeling library is free software; you can redistribute
  it and/or modify it under the terms of the GNU General Public License as
  published by the Free Software Foundation; either version 2 of the
  License, or (at your option) any later version.
  The Ascend Language Interpreter is distributed in hope that it will be
  useful, but WITHOUT ANY WARRANTY; without even the implied warranty of
  MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the GNU
  General Public License for more details.
  You should have received a copy of the GNU General Public License along with
  the program; if not, write to the Free Software Foundation, Inc., 675
  Mass Ave, Cambridge, MA 02139 USA.  Check the file named COPYING.
  \*************************************************************************)
(*
  S T R E A M . L I B
  -------------------
  AUTHOR:               Robert S. Huss
  DATES:                5/93 - Original code.
  9/93 - Slight changes to original code,
         added vapor_liquid_stream model.
  6/94 - Made compatible with ASCEND3C
  8/94 - Made compatible with H_G_thermodynamics.lib.
         Can still be used with H_S_thermodynamics.lib
  3/95 - Added relation scaling for absolute convergence
  CONTENTS:     Stream definitions, ranging from a simple molar
                stream to a thermodynamic multiphase stream.
                Molar stream can be refined to include
                thermodyanmic models, if data are available
                for each component.
  REQUIRES:
  "system.lib"
  "atoms.lib"
  "components.lib"
  "H_S_thermodynamics.lib" or "H_G_thermodynamics.lib"
  *)
MODEL molar_stream;
        components                      IS_A set OF symbol;
        state                           IS_A mixture;
        Ftot,f[components]              IS_A molar_rate;
        scale                           IS_A scaling_constant;
        Ftot_scale                      IS_A real;
        components, state.components    ARE_THE_SAME;
        FOR i IN components CREATE
            f_def[i]: f[i]/Ftot_scale = Ftot*state.y[i]/Ftot_scale;
        END;
        Ftot_scale := 1000 {mole/s};
  INITIALIZATION
    PROCEDURE seqmod;
        RUN state.specify;
        state.y[components].fixed               := FALSE;
    END seqmod;
    PROCEDURE specify;
        RUN seqmod;
        f[components].fixed                     := TRUE;
```

```
    END specify;
    PROCEDURE reset;
        RUN clear;
        RUN specify;
    END reset;
END molar_stream;
MODEL td_stream REFINES molar_stream;
        data[components]                IS_A component_constants;
        Htot                            IS_A energy_rate;
        H                               IS_A molar_energy;
        Htot_scale                      IS_A real;
        Htot_def: Htot/Htot_scale = H*Ftot/Htot_scale;
        Htot_scale := 1{MW};
  INITIALIZATION
    PROCEDURE seqmod;
        RUN molar_stream::seqmod;
        H.fixed                                 := TRUE;
    END seqmod;
END td_stream;
MODEL vapor_stream REFINES td_stream;
        state IS_REFINED_TO Pitzer_mixture;
        FOR i IN components CREATE
            data[i],state.data[i]       ARE_THE_SAME;
        END;
        H, state.H                      ARE_THE_SAME;
  INITIALIZATION
    PROCEDURE seqmod;
        RUN molar_stream::seqmod;
    END seqmod;
END vapor_stream;
MODEL liquid_stream REFINES td_stream;
        state IS_REFINED_TO UNIFAC_mixture;
        FOR i IN components CREATE
            data[i],state.data[i]       ARE_THE_SAME;
        END;
        H, state.H                      ARE_THE_SAME;
  INITIALIZATION
    PROCEDURE seqmod;
        RUN molar_stream::seqmod;
    END seqmod;
END liquid_stream;
MODEL multiphase_stream REFINES td_stream;
        state IS_REFINED_TO td_heterogeneous_mixture;
        FOR i IN components CREATE
            data[i],state.data[i]       ARE_THE_SAME;
        END;
        H, state.H                      ARE_THE_SAME;
        phases                          IS_A set OF symbol;
        phase_flow[phases]              IS_A molar_rate;
        phases,state.phases             ARE_THE_SAME;
        FOR k IN phases CREATE
            phase_flow_def[k]: phase_flow[k]/Ftot_scale =
                state.phi[k]*Ftot/Ftot_scale;
        END;
  INITIALIZATION
    PROCEDURE seqmod;
        RUN molar_stream::seqmod;
    END seqmod;
END multiphase_stream;
```

```
MODEL vapor_liquid_stream REFINES multiphase_stream;
        phases                          := [‘liquid’,’vapor’];
        state.reference                 := ‘liquid’;
        state.mix[‘liquid’]             IS_REFINED_TO UNIFAC_mixture;
        state.mix[‘vapor’]              IS_REFINED_TO Pitzer_mixture;
END vapor_liquid_stream;
```

```
(*************************************************************************\
                        flash.lib
                        by Robert S. Huss
                        Part of the Ascend Library
This file is part of the Ascend modeling library.
Copyright (C) 1994
The Ascend modeling library is free software; you can redistribute
it and/or modify it under the terms of the GNU General Public License as
published by the Free Software Foundation; either version 2 of the
License, or (at your option) any later version.
The Ascend Language Interpreter is distributed in hope that it will be
useful, but WITHOUT ANY WARRANTY; without even the implied warranty of
MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the GNU
General Public License for more details.
You should have received a copy of the GNU General Public License along with
the program; if not, write to the Free Software Foundation, Inc., 675
Mass Ave, Cambridge, MA 02139 USA.  Check the file named COPYING.
\*************************************************************************)
(*
        F L A S H . L I B
        -----------------
        AUTHOR:         Robert S. Huss
        DATES:          5/93 - Original code.
                        9/93 - Slight changes to original code.
                        6/94 - Made compatible with ASCEND3C
        CONTENTS:       Vapor-liquid-equilibrium flash models.
                        Molar flash models:
                        VLE_flash:  Basic molar flash model with constant
                        relative volatility.  Any number of input streams,
                        any number of liquid and vapor output streams.
                        simple_tray:  Refines VLE_flash.  Specifies
                        1 liquid and 1 vapor input streams, 1 liquid
                        and 1 vapor output streams.
                        simple_feed_tray: Refines VLE_flash.  Specifies
                        1 liquid, 1 vapor, and 1 feed input streams,
                        1 liquid, 1 vapor output streams;
                        multiple_feed_tray: Refines VLE_flash.  Any
                        number of 'feed' inputs, 1 liquid, 1 vapor input,
                        1 liquid, 1 vapor output.
                        condenser: Refines VLE_flash.  1 vapor input,
                        2 liquid outputs (liquid, distillate), 1 vapor
                        output (vapor_product).
                        reboiler: Refines VLE_flash.  1 liquid input,
                        2 vapor outputs (vapor, vapor_product), 1 liquid
                        output (bottoms).
                        Thermodynamic flash models (td):  Refine molar
                        flash models to incorporate thermodynamics.
                        data[components] must be defined, and feed streams
                        must be refined to thermodynamic stream models. With td
                        models, ideal and rigorous thermodynamic models are
                        possible. Models are: td_VLE_flash, td_simple_tray,
                        td_simple_feed_tray, td_multiple_feed_tray,
                        td_condenser, td_rebolier.  The variable
                        'can_be_adiabatic' defines whether or not the
                        flash type would typically be adiabatic in a
                        standard column.
                        murphree_equilibrium_mixture: Refinement of
                        multiphase_mixture in thermodynamics library
                        to incorporate Murphree Vapor Efficiency
```

```
                        into a tray model.  When used with a tray
                        model, the ref_y variables should be ARE_THE_SAME'd
                        to the incoming vapor to get Murphree Vapor
                        Efficiency.  This model can be used to get
                        a pseudo efficiency by using the overall
                        mixture composition y[components] as the
                        reference composition.
                        murph_tray:  Refinement of td_simple_tray to
                        include Murphree efficiency.  Uses
                        murphree_equilibrium_mixture.
        REQUIRES:       "atoms.lib"
                        "components.lib"
                        "H_S_thermodynamics.lib" or "H_G_thermodynamics.lib"
                        "stream.lib"
*)
MODEL VLE_flash;
        components                      IS_A set OF symbol;
        alpha[components]               IS_A factor;
        inputs, liqouts, vapouts        IS_A set OF symbol;
        input[inputs],
        liqout[liqouts],
        vapout[vapouts]                 IS_A molar_stream;
        VLE                             IS_A heterogeneous_mixture;
        scale                           IS_A scaling_constant;
        F_scale                         IS_A real;
        F_scale, totfeed.Ftot_scale     ARE_THE_SAME;
        (* Linking *)
        components,
        input[inputs].components,
        totfeed.components,
        VLE.components                          ARE_THE_SAME;
        FOR i IN components CREATE
          alpha[i], VLE.alpha['vapor'][i]       ARE_THE_SAME;
        END;
        (* defining equilibrium *)
        VLE.phases                              := ['liquid','vapor'];
        liqout[liqouts].state,
        VLE.mix['liquid']                       ARE_THE_SAME;
        vapout[vapouts].state,
        VLE.mix['vapor']                        ARE_THE_SAME;
        VLE.reference                           := 'liquid';
        (* Mass  balances       *)
        totfeed                         IS_A molar_stream;
        liqsplit[liqouts],
        vapsplit[vapouts]               IS_A fraction;
        FOR i IN components CREATE
          totfeedflow[i]: totfeed.f[i]/F_scale =
              SUM(input[inputs].f[i])/F_scale;
        END;
        FOR i IN components - [CHOICE(components)] CREATE
          totfeed.state.y[i], VLE.y[i]                  ARE_THE_SAME;
        END;
        FOR j IN liqouts CREATE
          liqoutflow[j]: liqout[j].Ftot/F_scale =
              VLE.phi['liquid']*totfeed.Ftot*liqsplit[j] / F_scale;
        END;
        FOR j IN vapouts CREATE
          vapoutflow[j]: vapout[j].Ftot/F_scale =
                VLE.phi['vapor']*totfeed.Ftot*vapsplit[j]/F_scale;
```

```
        END;
        sum_liqsplit: SUM(liqsplit[liqouts]) = 1.0;
        sum_vapsplit: SUM(vapsplit[vapouts]) = 1.0;
  INITIALIZATION
    PROCEDURE seqmod;
        RUN VLE.specify;
        VLE.y[components].fixed                := FALSE;
        liqsplit[liqouts].fixed                := TRUE;
        liqsplit[CHOICE(liqouts)].fixed        := FALSE;
        vapsplit[vapouts].fixed                := TRUE;
        vapsplit[CHOICE(vapouts)].fixed        := FALSE;
    END seqmod;
    PROCEDURE specify;
        RUN seqmod;
        RUN input[inputs].specify;
    END specify;
    PROCEDURE reset;
        RUN clear;
        RUN specify;
    END reset;
END VLE_flash;
MODEL simple_tray REFINES VLE_flash;
        inputs := [‘liquid’,’vapor’];
        liqouts := [‘liquid’];
        vapouts := [‘vapor’];
        cmo_ratio                      IS_A factor;
        cmo: cmo_ratio*input[‘liquid’].Ftot/F_scale
            = liqout[‘liquid’].Ftot/F_scale;
  INITIALIZATION
    PROCEDURE seqmod;
        RUN VLE.specify;
        VLE.y[components].fixed                := FALSE;
        VLE.phi[VLE.phases].fixed              := FALSE;
        cmo_ratio.fixed                        := TRUE;
    END seqmod;
END simple_tray;
MODEL simple_feed_tray REFINES VLE_flash;
        q                                      IS_A factor;
        inputs := [‘feed’,’liquid’,’vapor’];
        liqouts := [‘liquid’];
        vapouts := [‘vapor’];
        qeq: liqout[‘liquid’].Ftot/F_scale =
                (input[‘liquid’].Ftot
            + q*input[‘feed’].Ftot)/F_scale;
        q := 1.0;
  INITIALIZATION
    PROCEDURE seqmod;
        RUN VLE.specify;
        VLE.y[components].fixed                := FALSE;
        VLE.phi[VLE.phases].fixed              := FALSE;
        q.fixed                                := TRUE;
    END seqmod;
END simple_feed_tray;
MODEL condenser REFINES VLE_flash;
        reflux_ratio                   IS_A factor;
        prodsplit[‘distillate’,
            ‘vapor_product’]           IS_A fraction;
        totprod                        IS_A molar_stream;
        inputs                                := [‘vapor’];
```

```
        liqouts                                 := ['liquid','distillate'];
        vapouts                                 := ['vapor_product'];

        components, totprod.components          ARE_THE_SAME;

        reflux_def: liqout['liquid'].Ftot =
        reflux_ratio*totprod.Ftot;
        FOR i IN components CREATE
                totprodflow[i]: totprod.f[i]/F_scale =
                    (liqout['distillate'].f[i] +
                    vapout['vapor_product'].f[i])/F_scale;
        END;
        split_def_distillate:
            totprod.Ftot*prodsplit['distillate']/F_scale
            = liqout['distillate'].Ftot/F_scale;
        split_def_vapor_product:
            totprod.Ftot*prodsplit['vapor_product']/F_scale =
            vapout['vapor_product'].Ftot/F_scale;
        prodsplit['distillate'] := 1.0;
        prodsplit['vapor_product'] := 0.0;
  INITIALIZATION
    PROCEDURE seqmod;
        RUN VLE.specify;
        VLE.y[components].fixed                 := FALSE;
        reflux_ratio.fixed                      := TRUE;
        VLE.phi[VLE.phases].fixed               := FALSE;
        prodsplit['vapor_product'].fixed        := TRUE;
    END seqmod;
END condenser;
MODEL reboiler REFINES VLE_flash;
        reboil_ratio                    IS_A factor;
        prodsplit['bottoms',
            'vapor_product']            IS_A fraction;
        totprod                         IS_A molar_stream;
        inputs                  := ['liquid'];
        liqouts                 := ['bottoms'];
        vapouts                 := ['vapor','vapor_product'];
        components, totprod.components          ARE_THE_SAME;
        reboil_def: vapout['vapor'].Ftot/F_scale =
            reboil_ratio*totprod.Ftot/F_scale;
        FOR i IN components CREATE
                totprodflow[i]: totprod.f[i]/F_scale =
                    liqout['bottoms'].f[i]/F_scale +
                    vapout['vapor_product'].f[i]/F_scale;
        END;
        split_def_bottoms:
            totprod.Ftot*prodsplit['bottoms']/F_scale
            = liqout['bottoms'].Ftot/F_scale;
        split_def_vapor_product:
            totprod.Ftot*prodsplit['vapor_product']/F_scale
            = vapout['vapor_product'].Ftot/F_scale;
        prodsplit['bottoms'] := 1.0;
        prodsplit['vapor_product'] := 0.0;
  INITIALIZATION
    PROCEDURE seqmod;
        RUN VLE.specify;
        VLE.y[components].fixed                 := FALSE;
        VLE.phi[VLE.phases].fixed               := FALSE;
        reboil_ratio.fixed                      := TRUE;
```

```
        prodsplit['vapor_product'].fixed        := TRUE;
    END seqmod;
END reboiler;
MODEL td_VLE_flash REFINES VLE_flash;
        Qin                             IS_A energy_rate;
        data[components]                IS_A component_constants;
        can_be_adiabatic                IS_A boolean;
        can_be_adiabatic                := TRUE;
        VLE                             IS_REFINED_TO td_heterogeneous_mixture;
        input[inputs]                   IS_REFINED_TO td_stream;
        liqout[liqouts]                 IS_REFINED_TO liquid_stream;
        vapout[vapouts]                 IS_REFINED_TO vapor_stream;
        H_scale                         IS_A real;
        H_scale,
            vapout[CHOICE(vapouts)].Htot_scale  ARE_THE_SAME;
        FOR i IN components CREATE
          data[i],
          input[inputs].data[i],
          VLE.data[i]                           ARE_THE_SAME;
        END;
        (* heat balance *)
        energy_bal:  (SUM(input[inputs].Htot) + Qin)/H_scale =
            (SUM(liqout[liqouts].Htot) +
            SUM(vapout[vapouts].Htot))/H_scale;
  INITIALIZATION
END td_VLE_flash;
MODEL td_simple_tray REFINES simple_tray;
        Qin                             IS_A energy_rate;
        data[components]                IS_A component_constants;
        can_be_adiabatic                IS_A boolean;
        can_be_adiabatic                := TRUE;
        VLE                             IS_REFINED_TO td_heterogeneous_mixture;
        input[inputs]                   IS_REFINED_TO td_stream;
        liqout[liqouts]                 IS_REFINED_TO liquid_stream;
        vapout[vapouts]                 IS_REFINED_TO vapor_stream;
        H_scale                         IS_A real;
        H_scale, vapout['vapor'].Htot_scale     ARE_THE_SAME;
        FOR i IN components CREATE
          data[i],
          input[inputs].data[i],
          VLE.data[i]                           ARE_THE_SAME;
        END;
        (* heat balance *)
        energy_bal:  (SUM(input[inputs].Htot) + Qin)/H_scale =
            (SUM(liqout[liqouts].Htot) +
            SUM(vapout[vapouts].Htot))/H_scale;
  INITIALIZATION
    PROCEDURE heat_balance;
        cmo_ratio.fixed := FALSE;
        Qin.fixed := TRUE;
    END heat_balance;
END td_simple_tray;
MODEL td_simple_feed_tray REFINES simple_feed_tray;
        Qin                             IS_A energy_rate;
        data[components]                IS_A component_constants;
        can_be_adiabatic                IS_A boolean;
        can_be_adiabatic                := TRUE;
        VLE             IS_REFINED_TO td_heterogeneous_mixture;
        input['feed']           IS_REFINED_TO vapor_liquid_stream;
```

```
        input[inputs]           IS_REFINED_TO td_stream;
        liqout[liqouts]         IS_REFINED_TO liquid_stream;
        vapout[vapouts]         IS_REFINED_TO vapor_stream;
        H_scale                       IS_A real;
        H_scale, vapout['vapor'].Htot_scale     ARE_THE_SAME;
        FOR i IN components CREATE
          data[i],
          input[inputs].data[i],
          VLE.data[i]                           ARE_THE_SAME;
        END;
        liqout[liqouts].state,
        input['feed'].state.mix['liquid']       ARE_ALIKE;
        vapout['vapor'].state,
        input['feed'].state.mix['vapor']        ARE_ALIKE;
        input['feed'].state.phi['liquid'] := 1.0;
        input['feed'].state.phi['vapor'] := 0.0;
        (* heat balance *)
        energy_bal:  (SUM(input[inputs].Htot) + Qin)/H_scale =
            (SUM(liqout[liqouts].Htot) +
            SUM(vapout[vapouts].Htot))/H_scale;
  INITIALIZATION
    PROCEDURE heat_balance;
        q.fixed := FALSE;
        Qin.fixed := TRUE;
    END heat_balance;
END td_simple_feed_tray;
MODEL td_reboiler REFINES reboiler;
        Qin                           IS_A energy_rate;
        data[components]              IS_A component_constants;
        can_be_adiabatic              IS_A boolean;
        can_be_adiabatic              := FALSE;
        VLE                     IS_REFINED_TO td_heterogeneous_mixture;
        input[inputs]           IS_REFINED_TO td_stream;
        liqout[liqouts]         IS_REFINED_TO liquid_stream;
        vapout[vapouts]         IS_REFINED_TO vapor_stream;
        H_scale                 IS_A real;
        H_scale, vapout['vapor'].Htot_scale     ARE_THE_SAME;
        FOR i IN components CREATE
          data[i],
          input[inputs].data[i],
          VLE.data[i]                           ARE_THE_SAME;
        END;
        (* heat balance *)
        energy_bal:  (SUM(input[inputs].Htot) + Qin)/H_scale =
            (SUM(liqout[liqouts].Htot) +
            SUM(vapout[vapouts].Htot))/H_scale;
  INITIALIZATION
END td_reboiler;
MODEL td_condenser REFINES condenser;
        Qin                           IS_A energy_rate;
        data[components]              IS_A component_constants;
        can_be_adiabatic              IS_A boolean;
        can_be_adiabatic              := FALSE;
        VLE                           IS_REFINED_TO td_heterogeneous_mixture;
        input[inputs]                 IS_REFINED_TO td_stream;
        liqout[liqouts]               IS_REFINED_TO liquid_stream;
        vapout[vapouts]               IS_REFINED_TO vapor_stream;
        H_scale                       IS_A real;
        H_scale, input['vapor'].Htot_scale      ARE_THE_SAME;
```

```
        FOR i IN components CREATE
          data[i],
          input[inputs].data[i],
          VLE.data[i]                          ARE_THE_SAME;
        END;
        (* heat balance *)
        energy_bal:  (SUM(input[inputs].Htot) + Qin)/H_scale =
            (SUM(liqout[liqouts].Htot) +
            SUM(vapout[vapouts].Htot))/H_scale;
  INITIALIZATION
END td_condenser;
MODEL murph_tray REFINES td_simple_tray;
        murph_eff                              IS_A factor;
        VLE IS_REFINED_TO murphree_equilibrium_mixture;
        murph_eff, VLE.murph_eff               ARE_THE_SAME;
        FOR i IN components - [CHOICE(components)] CREATE
                VLE.ref_y[i],
                input['vapor'].state.y[i]      ARE_THE_SAME;
        END;
  INITIALIZATION
    PROCEDURE seqmod;
        RUN VLE.specify;
        VLE.y[components].fixed                := FALSE;
        VLE.phi[VLE.phases].fixed              := FALSE;
        cmo_ratio.fixed                        := TRUE;
        VLE.ref_y[components].fixed            := FALSE;
    END seqmod;
END murph_tray;
```

```
(*************************************************************************
  collocation.lib
  by Robert S. Huss
  Part OF the ASCEND Library
This file is part of the Ascend modeling library.
Copyright (C) 1994
The Ascend modeling library is free software; you can redistribute
it and/or modify it under the terms of the GNU General Public License as
published by the Free Software Foundation; either version 2 of the
License, or (at your option) any later version.
The Ascend Language Interpreter is distributed in hope that it will be
useful, but WITHOUT ANY WARRANTY; without even the implied warranty of
MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the GNU
General Public License for more details.
You should have received a copy of the GNU General Public License along with
the program; if not, write to the Free Software Foundation, Inc., 675
Mass Ave, Cambridge, MA 02139 USA.  Check the file named COPYING.

    *************************************************************************)
(*
        C O L L O C A T I O N . L I B
        -----------------------------
        AUTHOR:         Robert S. Huss
        DATES:          5/95 - First Public Release
        CONTENTS:       Collocation models for distillation modeling.
        REQUIRES:       “atoms.lib”
                        “components.lib”
                        “H_S_thermodynamics.lib” or “H_G_thermodynamics.lib”
                        “plot.lib”
                        “stream.lib”
                        “flash.lib”
*)
MODEL lagrange_polynomial;
        npoints,
        ntrays,
        order                                   IS_A integer;
        W[0..npoints][0..order],
        w_tray[0..ntrays+1],
        w_points[0..npoints],
        w_mid                                   IS_A factor;
        f_int,
        f_mid                                   IS_A fraction;
        scale                                   IS_A scaling_constant;
        change[1..ntrays]                       IS_A factor;
        change[1..ntrays] := 1.0;
        npoints := 2*ntrays + 1;
        order := ntrays;
        f_mid := 0.5;
        w_tray[0],
            w_points[0]                         ARE_THE_SAME;
        w_tray[ntrays+1],
        w_points[npoints]                       ARE_THE_SAME;
        w_mid = w_tray[0] + f_mid*(w_tray[ntrays+1] - w_tray[0]);
        FOR i IN [0..npoints] CREATE
            FOR k IN [0..ntrays] CREATE
                W[i][k] = PROD((w_points[i] - w_tray[j])
                    / (w_tray[k] - w_tray[j]) | j IN
                    [0..k-1,k+1..ntrays]);
            END;
```

```
        END;
  INITIALIZATION
     PROCEDURE specify;
        w_points[0..npoints].fixed      := TRUE;
        f_int.fixed                     := TRUE;
        f_mid.fixed                     := TRUE;
        change[1..ntrays].fixed         := TRUE;
     END specify;
     PROCEDURE reset;
        RUN clear;
        RUN specify;
     END reset;
END lagrange_polynomial;
MODEL lgr_1_point REFINES lagrange_polynomial;
        ntrays := 1;
        w_mid,
        w_tray[1]                       ARE_THE_SAME;
END lgr_1_point;
MODEL lgr_2_points REFINES lagrange_polynomial;
        ntrays := 2;
        f_int := 0.333;
        w_tray[1] = w_mid - change[1]*f_int
            *(w_mid - w_tray[0]);
        w_tray[2] = w_mid + change[2]*f_int
            *(w_tray[ntrays+1] - w_mid);
END lgr_2_points;
MODEL lgr_3_points REFINES lagrange_polynomial;
        ntrays := 3;
        f_int := 0.5;
        w_tray[1] = w_mid - change[1]*f_int
            *(w_mid - w_tray[0]);
        w_tray[2], w_mid                ARE_THE_SAME;
        w_tray[3] = w_mid + change[3]*f_int
            *(w_tray[ntrays+1] - w_mid);
END lgr_3_points;
MODEL lgr_4_points REFINES lagrange_polynomial;
        ntrays := 4;
        f_int := 0.6;
        w_tray[1] = w_mid - f_int*change[1]
            *(w_mid - w_tray[0]);
        w_tray[2] = w_mid - f_int*change[2]
            *(w_mid - w_tray[0])/3;
        w_tray[3] = w_mid + f_int*change[3]
            *(w_tray[ntrays+1] - w_mid)/3;
        w_tray[4] = w_mid + f_int*change[4]
            *(w_tray[ntrays+1] - w_mid);
END lgr_4_points;
MODEL lgr_5_points REFINES lagrange_polynomial;
        ntrays := 5;
        f_int := 0.66666667;
        w_tray[1] = w_mid - f_int*(w_mid - w_tray[0]);
        w_tray[2] = w_mid - f_int*(w_mid - w_tray[0])/2;
        w_tray[3], w_mid                ARE_THE_SAME;
        w_tray[4] = w_mid + f_int*(w_tray[ntrays+1] - w_mid)/2;
        w_tray[5] = w_mid + f_int*(w_tray[ntrays+1] - w_mid);
END lgr_5_points;
MODEL collpoint;
        a,
            z,s,
```

```
        ztop                                IS_A factor;
    scale                           IS_A scaling_constant;
    up_down                         IS_A real;
    s_def: z = 1-exp(-a*s);
    ztopdefn: ztop = 1-(1-z)*exp(-up_down*a);
    a                                       := 0.1;
    z                                       := 0.5;
    ztop                                    := 0.5;
    s                                       := 1;
    a.lower_bound                           := 0.0;
    a.upper_bound                           := 3.0;
    z.lower_bound                           := -0.5;
    ztop.lower_bound                        := -0.5;
    z.upper_bound                           := 1.0;
    ztop.upper_bound                        := 1.0;
    s.lower_bound                           := 0.0;

  INITIALIZATION
    PROCEDURE seqmod;
        a.fixed                             := TRUE;
        s.fixed                             := TRUE;
    END seqmod;
    PROCEDURE specify;
        RUN seqmod;
    END specify;
    PROCEDURE reset;
        RUN clear;
        RUN specify;
    END reset;
    PROCEDURE s_off;
        s_def.included                      := FALSE;
        s.fixed                             := TRUE;
    END s_off;
END collpoint;
MODEL z_set;
    ntrays                          IS_A integer;
    a,
        s_values[0..ntrays+1],
        z_values[0..ntrays+1],
        stot                        IS_A factor;
    up_down                         IS_A real;
    z[0..ntrays+1]                  IS_A collpoint;
    ztop,
        zbot                        IS_A factor;
    lgr                             IS_A lagrange_polynomial;
    z_on                            IS_A boolean;
    scale                           IS_A scaling_constant;
    z_on := FALSE;
    lgr.ntrays, ntrays              ARE_THE_SAME;
    stot = s_values[ntrays+1] - s_values[0];
    FOR j IN [0..ntrays+1] CREATE
        z[j].z,
            z_values[j]             ARE_THE_SAME;
        z[j].s,
            s_values[j]             ARE_THE_SAME;
    END;
    up_down,
        z[0..ntrays+1].up_down              ARE_THE_SAME;
    s_values[0] := 0;
```

```
      z_values[0] := 0;
      a,z[0..ntrays+1].a              ARE_THE_SAME;
      stot.lower_bound := 0.0;
      FOR j IN [1..ntrays] CREATE
          lgr.w_points[2*j-1],
              lgr.w_tray[j]           ARE_THE_SAME;
      END;
      (*      z_based         *)
      FOR j IN [1..ntrays] CREATE
          z_based_odd[2*j-1]: lgr.w_points[2*j-1] =
              z_values[j];
          z_based_even[2*j]: lgr.w_points[2*j] =
              z[j].ztop;
      END;
      z_based_0: lgr.w_points[0] = z_values[0];
      z_based_n: lgr.w_points[lgr.npoints] =
          z_values[ntrays+1];
      (*      s_based         *)
      FOR j IN [1..ntrays] CREATE
          s_based_odd[2*j-1]: lgr.w_points[2*j-1] =
              s_values[j];
          s_based_even[2*j]: lgr.w_points[2*j] =
              s_values[j] + z[0].up_down;
      END;
      s_based_0: lgr.w_points[0] = s_values[0];
      s_based_n: lgr.w_points[lgr.npoints] =
          s_values[ntrays+1];
      ztop = (up_down+1)*z_values[ntrays+1]/2 +(1-up_down)*z_values[0]/2;
      zbot = (up_down+1)*z_values[0]/2 +(1-up_down)*z_values[ntrays+1]/2;
      tray_delta              IS_A factor;
      s_values[ntrays] + tray_delta = s_values[ntrays+1];
INITIALIZATION
  PROCEDURE specify;
      RUN lgr.specify;
      lgr.w_points[0..lgr.npoints].fixed      := FALSE;
      a.fixed                                 := TRUE;
      s_values[0].fixed                       := TRUE;
      stot.fixed                              := TRUE;
      IF (z_on) THEN
          RUN z_based_poly;
      ELSE
          RUN s_based_poly;
      END;
  END specify;
  PROCEDURE reset;
      RUN clear;
      RUN specify;
  END reset;
  PROCEDURE z_based_poly;
      FOR j IN [1..ntrays] DO
          z_based_odd[2*j-1].included         := TRUE;
          z_based_even[2*j].included          := TRUE;
          s_based_odd[2*j-1].included         := FALSE;
          s_based_even[2*j].included          := FALSE;
      END;
      z_based_0.included                      := TRUE;
      z_based_n.included                      := TRUE;
      s_based_0.included                      := FALSE;
      s_based_n.included                      := FALSE;
```

```
        z_on                                    := TRUE;
    END z_based_poly;
    PROCEDURE s_based_poly;
        FOR j IN [1..ntrays] DO
            z_based_odd[2*j-1].included         := FALSE;
            z_based_even[2*j].included          := FALSE;
            s_based_odd[2*j-1].included         := TRUE;
            s_based_even[2*j].included          := TRUE;
        END;
        z_based_0.included                      := FALSE;
        z_based_n.included                      := FALSE;
        s_based_0.included                      := TRUE;
        s_based_n.included                      := TRUE;
        z_on                                    := FALSE;
    END s_based_poly;
    PROCEDURE s_off;
        RUN z[ntrays..ntrays+1].s_off;
    END s_off;
    PROCEDURE pin_end;
        tray_delta.fixed                        := TRUE;
        tray_delta := 1.0;
        lgr.change[ntrays].fixed                := FALSE;
    END pin_end;
END z_set;
MODEL coll;
        components,
            active_components                   IS_A set OF symbol;
        inactive_component              IS_A symbol;
        ntrays,
            x_order,
            y_order                             IS_A integer;
        topliq,
            botliq,
            topvap,
            botvap                              IS_A molar_stream;
        x_coeff[active_components]
            [0..x_order],
            y_coeff[active_components]
            [0..y_order],
            cmo[1..ntrays],
            cmotot                              IS_A factor;
        tray[1..ntrays]                 IS_A simple_tray;
        z_set                           IS_A z_set;
        z_on,
            hat_on                              IS_A boolean;
        x[1..ntrays][components]        IS_A fraction;
        y[1..ntrays][components]        IS_A fraction;
        x_hat[1..ntrays][components]    IS_A factor;
        scale                           IS_A scaling_constant;
        active_components := components - [inactive_component];
        z_on            := FALSE;
        hat_on          := FALSE;
        FOR i IN components CREATE
            FOR j IN [1..ntrays] CREATE
                x[j][i] = (1+z_set.up_down)
                    *tray[ntrays+1-j].liqout['liquid'].state.y[i]/2
                    + (1-z_set.up_down)
                    *tray[j].liqout['liquid'].state.y[i]/2;
                y[j][i] = (1+z_set.up_down)
```

```
            *tray[ntrays+1-j].input['vapor'].state.y[i]/2
            + (1-z_set.up_down)
            *tray[j].input['vapor'].state.y[i]/2;
        x_hat[j][i] = (1+z_set.up_down)
            *tray_x_hat['out'][i][ntrays+1-j]/2
            + (1-z_set.up_down)
            *tray_x_hat['out'][i][j]/2;
    END;
END;
x_order         := ntrays;
y_order         := ntrays;
z_on,
    z_set.z_on                  ARE_THE_SAME;
y_coeff[active_components][0..y_order] := 0.5;
x_coeff[active_components][0..x_order] := 0.5;
components,
    topliq.components,
    topvap.components,
    botliq.components,
    botvap.components,
    tray[1..ntrays].components  ARE_THE_SAME;
ntrays,z_set.ntrays                     ARE_THE_SAME;
tray[1..ntrays].VLE             ARE_ALIKE;
tray[1..ntrays]                 ARE_ALIKE;
(* constant molar overflow model - instead of heat balance *)
FOR j IN [1..ntrays] CREATE
    cmo[j]*tray[j].input['liquid'].Ftot = botliq.Ftot;
END;
cmotot*topliq.Ftot = botliq.Ftot;
FOR i IN components CREATE
    overall_MB[i]: topliq.f[i] - topvap.f[i] =
        botliq.f[i] - botvap.f[i];
END;
tray[1..ntrays].liqout['liquid'].state  ARE_ALIKE;
tray[1..ntrays].vapout['vapor'].state   ARE_ALIKE;
(* xtrans stuff *)
tray_x_hat['in','out'][components][1..ntrays],
    tray_y_hat['in','out'][components][1..ntrays],
    end_x_hat['top','bot'][components],
    end_y_hat['top','bot'][components]  IS_A factor;
td              IS_A real;
td := 1.0;
tray_x_hat['in','out'][components][1..ntrays].nominal := 20;
tray_y_hat['in','out'][components][1..ntrays].nominal := 20;
end_x_hat['top','bot'][components].nominal := 20;
end_y_hat['top','bot'][components].nominal := 20;
tray_x_hat['in','out'][components][1..ntrays] := 1;
tray_y_hat['in','out'][components][1..ntrays] := 1;
end_x_hat['top','bot'][components] := 1;
end_y_hat['top','bot'][components] := 1;
FOR i IN components CREATE
    (2.0*botliq.state.y[i] - 1.0)
        = tanh(end_x_hat['bot'][i]*td);
    (2.0*topliq.state.y[i] - 1.0)
        = tanh(end_x_hat['top'][i]*td);
    (2.0*botvap.state.y[i] - 1.0)
        = tanh(end_y_hat['bot'][i]*td);
    (2.0*topvap.state.y[i] - 1.0)
        = tanh(end_y_hat['top'][i]*td);
```

```
END;
FOR i IN components CREATE
    FOR j IN [1..ntrays] CREATE
        (2.0*tray[j].liqout['liquid'].state.y[i] - 1.0)
            = tanh(tray_x_hat['out'][i][j]*td);
        (2.0*tray[j].input['liquid'].state.y[i] - 1.0)
            = tanh(tray_x_hat['in'][i][j]*td);
        (2.0*tray[j].vapout['vapor'].state.y[i] - 1.0)
            = tanh(tray_y_hat['out'][i][j]*td);
        (2.0*tray[j].input['vapor'].state.y[i] - 1.0)
            = tanh(tray_y_hat['in'][i][j]*td);
    END;
END;
(* polynomial *)
(* Overall material balances *)
FOR j IN [1..ntrays] CREATE
    tot_trayMB[j]: botvap.Ftot - botliq.Ftot =
        tray[j].vapout['vapor'].Ftot -
        tray[j].input['liquid'].Ftot;
END;
FOR i IN active_components CREATE
    FOR j IN [1..ntrays] CREATE
        frac_x_in[j][i]:  tray[j].input['liquid'].state.y[i] =
            SUM(z_set.lgr.W[2*j][k]*x_coeff[i][k]
            | k IN [0..x_order]);
        frac_y_in[j][i]:  tray[j].input['vapor'].state.y[i] =
            SUM(z_set.lgr.W[2*j-1][k]*y_coeff[i][k]
            | k IN [0..y_order]);
        frac_x_out[j][i]:  tray[j].liqout['liquid'].state.y[i] =
            SUM(z_set.lgr.W[2*j-1][k]*x_coeff[i][k]
            | k IN [0..x_order]);
        frac_y_out[j][i]:  tray[j].vapout['vapor'].state.y[i] =
            SUM(z_set.lgr.W[2*j][k]*y_coeff[i][k]
            | k IN [0..y_order]);
    END;
END;
FOR i IN active_components CREATE
    FOR j IN [1..ntrays] CREATE
        trans_x_out[j][i]: tray_x_hat['out'][i][j] =
            SUM(z_set.lgr.W[2*j-1][k]*x_coeff[i][k]
            | k IN [0..x_order]);
        trans_x_in[j][i]: tray_x_hat['in'][i][j] =
            SUM(z_set.lgr.W[2*j][k]*x_coeff[i][k]
            | k IN [0..x_order]);
        trans_y_out[j][i]: tray_y_hat['out'][i][j] =
            SUM(z_set.lgr.W[2*j][k]*y_coeff[i][k]
            | k IN [0..y_order]);
        trans_y_in[j][i]: tray_y_hat['in'][i][j] =
            SUM(z_set.lgr.W[2*j-1][k]*y_coeff[i][k]
            | k IN [0..y_order]);
    END;
END;
FOR i IN active_components CREATE
    frac_x_top[1][i]:  topliq.state.y[i]
        = SUM(((z_set.up_down+1)
        *z_set.lgr.W[z_set.lgr.npoints][k]/2
        + (1-z_set.up_down)
        *z_set.lgr.W[0][k]/2)*x_coeff[i][k]
        | k IN [0..x_order]);
```

```
        frac_y_top[1][i]:  topvap.state.y[i]
            = SUM(((z_set.up_down+1)
            *z_set.lgr.W[z_set.lgr.npoints][k]/2
            + (1-z_set.up_down)
            *z_set.lgr.W[0][k]/2)*y_coeff[i][k]
            | k IN [0..y_order]);
        frac_y_bot[1][i]: botvap.state.y[i]
            = SUM(((z_set.up_down+1)
            *z_set.lgr.W[0][k]/2
            + (1-z_set.up_down)
            *z_set.lgr.W[z_set.lgr.npoints][k]/2)
            *y_coeff[i][k]
            | k IN [0..y_order]);
    END;
    FOR i IN active_components CREATE
        trans_x_top[1][i]: end_x_hat[‘top’][i]
            = SUM((((z_set.up_down+1)*z_set.lgr.W[z_set.lgr.npoints][k]/2
            + (1-z_set.up_down)*z_set.lgr.W[0][k]/2))*x_coeff[i][k]
            | k IN [0..x_order]);
        trans_y_top[1][i]: end_y_hat[‘top’][i]
            = SUM((((z_set.up_down+1)*z_set.lgr.W[z_set.lgr.npoints][k]/2
            + (1-z_set.up_down)*z_set.lgr.W[0][k]/2))*y_coeff[i][k]
            | k IN [0..y_order]);
        trans_y_bot[1][i]: end_y_hat[‘bot’][i]
            = SUM(((z_set.up_down+1)*z_set.lgr.W[0][k]/2
            + (1-z_set.up_down)
            *z_set.lgr.W[z_set.lgr.npoints][k]/2)*y_coeff[i][k]
            | k IN [0..y_order]);
    END;
    slope[components]       IS_A factor;
    FOR i IN components CREATE
        slope[i] = (sqrt(sqr(topliq.state.y[i]
              - botliq.state.y[i])))/z_set.stot;
    END;
    aslope[components]                      IS_A factor;
    intercept[components]                   IS_A fraction;
    slope_slack[1..ntrays][components]      IS_A factor;
    FOR i IN components CREATE
        botpoint[i]: botliq.state.y[i] = aslope[i]*z_set.zbot +
            intercept[i];
        toppoint[i]: topliq.state.y[i] = aslope[i]*z_set.ztop +
            intercept[i];
        FOR j IN [1..ntrays] CREATE
            midpoint[j][i]: tray[j].liqout[‘liquid’].state.y[i] =
                aslope[i]*(z_set.z_values[j] - z_set.zbot) +
                intercept[i] + slope_slack[j][i];
        END;
    END;
INITIALIZATION
  PROCEDURE standard_poly;
    frac_x_in[1..ntrays][active_components].included        :=TRUE;
    frac_y_in[1..ntrays][active_components].included        :=TRUE;
    frac_x_out[1..ntrays][active_components].included       :=TRUE;
    frac_y_out[1..ntrays][active_components].included       :=TRUE;
    frac_x_top[1][active_components].included               :=TRUE;
    frac_y_top[1][active_components].included               :=TRUE;
    frac_y_bot[1][active_components].included               :=TRUE;
    trans_x_out[1..ntrays][active_components].included      :=FALSE;
    trans_x_in[1..ntrays][active_components].included       :=FALSE;
```

```
        trans_y_out[1..ntrays][active_components].included      :=FALSE;
        trans_y_in[1..ntrays][active_components].included       :=FALSE;
        trans_x_top[1][active_components].included              :=FALSE;
        trans_y_top[1][active_components].included              :=FALSE;
        trans_y_bot[1][active_components].included              :=FALSE;
        hat_on                                                  :=FALSE;
    END standard_poly;
    PROCEDURE trans_poly;
        frac_x_in[1..ntrays][active_components].included        :=FALSE;
        frac_y_in[1..ntrays][active_components].included        :=FALSE;
        frac_x_out[1..ntrays][active_components].included       :=FALSE;
        frac_y_out[1..ntrays][active_components].included       :=FALSE;
        frac_x_top[1][active_components].included               :=FALSE;
        frac_y_top[1][active_components].included               :=FALSE;
        frac_y_bot[1][active_components].included               :=FALSE;
        trans_x_out[1..ntrays][active_components].included      :=TRUE;
        trans_x_in[1..ntrays][active_components].included       :=TRUE;
        trans_y_out[1..ntrays][active_components].included      :=TRUE;
        trans_y_in[1..ntrays][active_components].included       :=TRUE;
        trans_x_top[1][active_components].included              :=TRUE;
        trans_y_top[1][active_components].included              :=TRUE;
        trans_y_bot[1][active_components].included              :=TRUE;
        hat_on                                                  :=TRUE;
    END trans_poly;
    PROCEDURE seqmod;
        cmo[1..ntrays].fixed                            := TRUE;
        cmotot.fixed                                    := TRUE;
        RUN tray[1..ntrays].seqmod;
        RUN z_set.specify;
        RUN tray[1..ntrays].input[tray[1].inputs].seqmod;
        RUN topliq.seqmod;
        RUN botvap.seqmod;
        RUN topvap.seqmod;
        RUN botliq.seqmod;
        IF (hat_on) THEN
            RUN trans_poly;
        ELSE
            RUN standard_poly;
        END;
    END seqmod;
    PROCEDURE specify;
        RUN seqmod;
        RUN topliq.specify;
        RUN botvap.specify;
    END specify;
    PROCEDURE reset;
        RUN clear;
        RUN specify;
    END reset;
    PROCEDURE s_off;
        RUN z_set.s_off;
    END s_off;
    PROCEDURE z_based_poly;
        RUN z_set.z_based_poly;
    END z_based_poly;
    PROCEDURE s_based_poly;
        RUN z_set.s_based_poly;
    END s_based_poly;
END coll;
```

```
MODEL coll_stack;
        components,active_components    IS_A set OF symbol;
        inactive_component              IS_A symbol;
        straight_choice                 IS_A symbol;
        ncolls                          IS_A integer;
        coll[1..ncolls]                 IS_A coll;
        split[1..ncolls]                IS_A fraction;
        stot                            IS_A factor;
        scale                           IS_A scaling_constant;
        components,
        coll[1..ncolls].components      ARE_THE_SAME;
        active_components,
        coll[1..ncolls].active_components       ARE_THE_SAME;
        inactive_component,
        coll[1..ncolls].inactive_component      ARE_THE_SAME;
        FOR j IN [1..ncolls-1] CREATE
                coll[j].botvap,
                coll[j+1].topvap        ARE_THE_SAME;
                coll[j].botliq,
                coll[j+1].topliq        ARE_THE_SAME;
        END;
        FOR j IN [1..ncolls] CREATE
                tray_split[j]: coll[j].z_set.stot = split[j]*stot;
                split[j] := 1.0/ncolls;
        END;
        coll[1..ncolls].tray[1]         ARE_ALIKE;
        coll[1..ncolls].tray[1].VLE     ARE_ALIKE;
        stot_def: stot = SUM(coll[j].z_set.stot | j IN [1..ncolls]);
        stot.lower_bound := 1e-8;
  INITIALIZATION
     PROCEDURE seqmod;
        RUN coll[1..ncolls].seqmod;
        FOR j IN [1..ncolls] DO
            coll[j].z_set
                .z_values[coll[j].ntrays+1].fixed       := FALSE;
            coll[j].z_set.stot.fixed                    := FALSE;
        END;
        stot.fixed                                      := TRUE;
        split[1..ncolls-1].fixed                        := TRUE;
     END seqmod;
     PROCEDURE specify;
        RUN seqmod;
        RUN coll[1].topliq.specify;
        RUN coll[ncolls].botvap.specify;
     END specify;
     PROCEDURE reset;
        RUN clear;
        RUN specify;
     END reset;
END coll_stack;
MODEL coll_column;
        nfeeds                          IS_A integer;
        condenser                       IS_A condenser;
        coll_stack[1..nfeeds+1]         IS_A coll_stack;
        feed_tray[1..nfeeds]            IS_A simple_feed_tray;
        reboiler                        IS_A reboiler;
        components                      IS_A set OF symbol;
        stot,
        s_stack[1..nfeeds+1]            IS_A factor;
```

```
split[1..nfeeds+1]              IS_A fraction;
xsi[components]                 IS_A fraction;
xsi_set[components]             IS_A fraction;
xsi_diff[components]            IS_A fraction;
scale                           IS_A scaling_constant;
components,
condenser.components,
coll_stack[1..nfeeds+1].components,
feed_tray[1..nfeeds].components,
reboiler.components             ARE_THE_SAME;
FOR j IN [1..nfeeds+1] CREATE
        s_stack[j],
        coll_stack[j].stot      ARE_THE_SAME;
END;
stot = SUM(s_stack[1..nfeeds+1]);
FOR j IN [1..nfeeds+1] CREATE
        tray_split[j]: s_stack[j] = split[j]*stot;
        split[j] := 1.0/(nfeeds+1);
END;
condenser.liqout['liquid'],
coll_stack[1].coll[1].topliq                    ARE_THE_SAME;
condenser.input['vapor'],
coll_stack[1].coll[1].topvap                    ARE_THE_SAME;
FOR j IN [1..nfeeds] CREATE
        coll_stack[j].coll[coll_stack[j].ncolls].botliq,
        feed_tray[j].input['liquid']            ARE_THE_SAME;
        coll_stack[j].coll[coll_stack[j].ncolls].botvap,
        feed_tray[j].vapout['vapor']            ARE_THE_SAME;
        coll_stack[j+1].coll[1].topliq,
        feed_tray[j].liqout['liquid']           ARE_THE_SAME;
        coll_stack[j+1].coll[1].topvap,
        feed_tray[j].input['vapor']             ARE_THE_SAME;
END;
coll_stack[nfeeds+1].coll[coll_stack[nfeeds+1].ncolls].botliq,
reboiler.input['liquid']                ARE_THE_SAME;
coll_stack[nfeeds+1].coll[coll_stack[nfeeds+1].ncolls].botvap,
reboiler.vapout['vapor']                ARE_THE_SAME;
condenser.VLE,
coll_stack[1..nfeeds+1].coll[1].tray[1].VLE,
feed_tray[1..nfeeds].VLE,
reboiler.VLE                                    ARE_ALIKE;
condenser.liqout['liquid'].state,
coll_stack[1..nfeeds+1].coll[1].tray[1].liqout['liquid'].state,
feed_tray[1..nfeeds].liqout['liquid'].state,
reboiler.liqout['bottoms'].state                ARE_ALIKE;
condenser.vapout['vapor_product'].state,
coll_stack[1..nfeeds+1].coll[1].tray[1].vapout['vapor'].state,
feed_tray[1..nfeeds].vapout['vapor'].state,
reboiler.vapout['vapor'].state                  ARE_ALIKE;
FOR i IN components CREATE
        OverallMB[i]: SUM(feed_tray[1..nfeeds].input['feed'].f[i]) =
                condenser.totprod.f[i] +
                reboiler.totprod.f[i];
END;
FOR i IN components CREATE
        xsi[i]*SUM(feed_tray[k].input['feed'].f[i] | k IN [1..nfeeds])
                = condenser.totprod.f[i];
        xsi_diff[i] = 0.5*sqr(xsi[i] - xsi_set[i]);
END;
```

```
      recovery: MINIMIZE SUM(xsi_diff[i] | i IN components);
      binary_sep[components][components]      IS_A factor;
      FOR i IN components CREATE
          FOR j IN components CREATE
              binary_sep[i][j] = (condenser.totprod.f[i] /
              (condenser.totprod.f[i] +
              condenser.totprod.f[j]) -
              reboiler.totprod.f[i] /
              (reboiler.totprod.f[i] +
              reboiler.totprod.f[j]));
              sep_opt[i][j]: MINIMIZE -sqr(binary_sep[i][j]);
          END;
      END;
INITIALIZATION
   PROCEDURE seqmod;
      RUN condenser.seqmod;
      RUN coll_stack[1..nfeeds+1].seqmod;
      RUN feed_tray[1..nfeeds].seqmod;
      RUN reboiler.seqmod;
      OverallMB[components].included                  := FALSE;
      reboiler.reboil_ratio.fixed                     := FALSE;
      condenser.totprod.Ftot.fixed                    := TRUE;
      xsi_set[components].fixed                       := TRUE;
      recovery.included                               := FALSE;
      sep_opt[components][components].included        := FALSE;
   END seqmod;
   PROCEDURE specify;
      RUN seqmod;
      RUN feed_tray[1..nfeeds].input['feed'].specify;
   END specify;
   PROCEDURE reset;
      RUN clear;
      RUN specify;
   END reset;
   PROCEDURE standard_poly;
      RUN coll_stack[1..nfeeds+1].standard_poly;
   END standard_poly;
   PROCEDURE trans_poly;
      RUN coll_stack[1..nfeeds+1].trans_poly;
   END trans_poly;
   PROCEDURE z_based_poly;
      RUN coll_stack[1..nfeeds+1].z_based_poly;
   END z_based_poly;
   PROCEDURE s_based_poly;
      RUN coll_stack[1..nfeeds+1].s_based_poly;
   END s_based_poly;
   PROCEDURE propogate_feed;
      FOR i IN components DO
          condenser.alpha[i] := feed_tray[1].alpha[i];
          FOR k IN [2..nfeeds] DO
              feed_tray[k].alpha[i] :=
                  feed_tray[1].alpha[i];
          END;
        FOR k IN [1..nfeeds+1] DO
         FOR j IN [1..coll_stack[k].ncolls] DO
              coll_stack[k].coll[j].tray[1..coll_stack[k]
                .coll[j].ntrays].alpha[i]
                      := feed_tray[1].alpha[i];
         END;
```

```
          END;
                reboiler.alpha[i] := feed_tray[1].alpha[i];
          END;
       END propogate_feed;
       PROCEDURE overallMB;
          OverallMB[components].included                  := TRUE;
          feed_tray[1].totfeedflow[components].included   := FALSE;
       END overallMB;
END coll_column;
MODEL td_coll REFINES coll;
          data[components]                IS_A component_constants;
          tray[1..ntrays] IS_REFINED_TO td_simple_tray;
          topliq          IS_REFINED_TO td_stream;
          topvap          IS_REFINED_TO td_stream;
          botliq          IS_REFINED_TO td_stream;
          botvap          IS_REFINED_TO td_stream;
          tray[1..ntrays].liqout['liquid'].state  ARE_ALIKE;
          tray[1..ntrays].vapout['vapor'].state   ARE_ALIKE;
          FOR i IN components CREATE
                  data[i],
                  tray[1..ntrays].data[i],
                  topliq.data[i],
                  topvap.data[i],
                  botliq.data[i],
                  botvap.data[i]          ARE_THE_SAME;
          END;
END td_coll;
MODEL h_coll REFINES td_coll;
          h_order                                 IS_A integer;
          h_order,ntrays  ARE_THE_SAME;
          h_coeff['liquid','vapor'][0..h_order]   IS_A molar_energy;
          Qtot,
          Qin[1..ntrays]                          IS_A energy_rate;
          h_exist,
              hb_on                               IS_A boolean;
          h_exist := TRUE;
          hb_on := FALSE;
          Overall_HB: Qtot + topliq.Htot + botvap.Htot =
              botliq.Htot + topvap.Htot;
          FOR j IN [1..ntrays] CREATE
                  tot_trayHB[j]: Qin[j] +topliq.Htot - topvap.Htot =
                                  tray[j].liqout['liquid'].Htot -
                                  tray[j].input['vapor'].Htot;
          END;
(* end points *)
          h_end_topliq: topliq.H = SUM(((z_set.up_down+1)
              *z_set.lgr.W[z_set.lgr.npoints][k]/2
                           + (1-z_set.up_down)*z_set.lgr.W[0][k]/2)
                           *h_coeff['liquid'][k]
                           | k IN [0..h_order]);
          h_end_topvap: topvap.H = SUM(((z_set.up_down+1)
                           *z_set.lgr.W[z_set.lgr.npoints][k]/2
                           + (1-z_set.up_down)*z_set.lgr.W[0][k]/2)
                           *h_coeff['vapor'][k]
                           | k IN [0..h_order]);
          h_end_botliq: botliq.H =  SUM(((z_set.up_down+1)
                           *z_set.lgr.W[0][k]/2
                           + (1-z_set.up_down)
                           *z_set.lgr.W[z_set.lgr.npoints][k]/2)
```

```
                        *h_coeff['liquid'][k]
                        | k IN [0..h_order]);
        h_end_botvap: botvap.H = SUM(((z_set.up_down+1)
                        *z_set.lgr.W[0][k]/2
                        + (1-z_set.up_down)
                        *z_set.lgr.W[z_set.lgr.npoints][k]/2)
                        *h_coeff['vapor'][k]
                        | k IN [0..h_order]);
(* interior points *)
        FOR j IN [1..ntrays] CREATE
                h_int_liqout[j]: tray[j].liqout['liquid'].H =
                    SUM(z_set.lgr.W[2*j-1][k]*h_coeff['liquid'][k]
                    | k IN [0..h_order]);
                h_int_liqin[j]: tray[j].input['liquid'].H =
                    SUM(z_set.lgr.W[2*j][k]*h_coeff['liquid'][k]
                    | k IN [0..h_order]);
                h_int_vapout[j]: tray[j].vapout['vapor'].H =
                    SUM(z_set.lgr.W[2*j][k]*h_coeff['vapor'][k]
                    | k IN [0..h_order]);
                h_int_vapin[j]: tray[j].input['vapor'].H =
                    SUM(z_set.lgr.W[2*j-1][k]*h_coeff['vapor'][k]
                    | k IN [0..h_order]);
        END;
   INITIALIZATION
     PROCEDURE seqmod;
         RUN td_coll::seqmod;
        tray[1..ntrays].input[tray[1].inputs].H.fixed          := FALSE;
        topvap.H.fixed                                         := FALSE;
        botliq.H.fixed                                         := FALSE;
        IF (hb_on) THEN
            RUN heat_balance;
        END;
     END seqmod;
     PROCEDURE heat_balance;
        RUN tray[1..ntrays].heat_balance;
        Qtot.fixed                                             := TRUE;
        cmotot.fixed                                           := FALSE;
        Qin[1..ntrays].fixed                                   := TRUE;
        cmo[1..ntrays].fixed                                   := FALSE;
        hb_on                                                  := TRUE;
     END heat_balance;
     PROCEDURE CMO;
         tray[1..ntrays].cmo_ratio.fixed                       := TRUE;
         tray[1..ntrays].Qin.fixed                             := FALSE;
         Qtot.fixed                                            := FALSE;
        cmotot.fixed                                           := TRUE;
        Qin[1..ntrays].fixed                                   := FALSE;
        cmo[1..ntrays].fixed                                   := TRUE;
        hb_on                                                  := FALSE;
     END CMO;
END h_coll;
MODEL td_coll_stack REFINES coll_stack;
        data[components]        IS_A component_constants;
        coll[1..ncolls]         IS_REFINED_TO td_coll;
        hb_on                   IS_A boolean;
        hb_on := FALSE;
        FOR i IN components CREATE
                data[i],
                coll[1..ncolls].data[i] ARE_THE_SAME;
```

```
        END;
        coll[1..ncolls].tray[1].liqout['liquid'].state          ARE_ALIKE;
        coll[1..ncolls].tray[1].vapout['vapor'].state           ARE_ALIKE;
    INITIALIZATION
      PROCEDURE heat_balance;
        RUN coll[1..ncolls].heat_balance;
        hb_on                                                   := TRUE;
      END heat_balance;
      PROCEDURE CMO;
         RUN coll[1..ncolls].CMO;
         hb_on                                                  := FALSE;
      END CMO;
      PROCEDURE seqmod;
         RUN coll_stack::seqmod;
        FOR j IN [1..ncolls] DO
          IF coll[j].h_exist THEN
               coll[j].botliq.H.fixed                           := FALSE;
          END;
        END;
        IF (hb_on) THEN
            RUN heat_balance;
        END;
      END seqmod;
END td_coll_stack;
MODEL td_coll_column REFINES coll_column;
        reduce                          IS_A factor;
        reduce := 0.5;
        data[components]                IS_A component_constants;
        condenser                       IS_REFINED_TO td_condenser;
        feed_tray[1..nfeeds]            IS_REFINED_TO td_simple_feed_tray;
        reboiler                        IS_REFINED_TO td_reboiler;
        coll_stack[1..nfeeds+1]         IS_REFINED_TO td_coll_stack;
        hb_on                           IS_A boolean;
        hb_on := FALSE;
        FOR i IN components CREATE
                data[i],
                condenser.data[i],
                coll_stack[1..nfeeds+1].coll[1].data[i],
                feed_tray[1..nfeeds].data[i],
                reboiler.data[i]                                ARE_THE_SAME;
        END;
        condenser.liqout['liquid'].state,
        coll_stack[1..nfeeds+1].coll[1].tray[1].liqout['liquid'].state,
        feed_tray[1..nfeeds].liqout['liquid'].state,
        reboiler.liqout['bottoms'].state                        ARE_ALIKE;
        condenser.vapout['vapor_product'].state,
        coll_stack[1..nfeeds+1].coll[1].tray[1].vapout['vapor'].state,
        feed_tray[1..nfeeds].vapout['vapor'].state,
        reboiler.vapout['vapor_product'].state                  ARE_ALIKE;
    INITIALIZATION
      PROCEDURE heat_balance;
        RUN coll_stack[1..nfeeds+1].heat_balance;
        RUN feed_tray[1..nfeeds].heat_balance;
        hb_on := TRUE;
      END heat_balance;
      PROCEDURE CMO;
         RUN coll_stack[1..nfeeds+1].CMO;
         feed_tray[1..nfeeds].q.fixed := TRUE;
         feed_tray[1..nfeeds].Qin.fixed := FALSE;
```

```
        hb_on := FALSE;
     END CMO;
     PROCEDURE reduce_Q;
        FOR i IN [1..nfeeds+1] DO
          FOR j IN [1..coll_stack[i].ncolls] DO
                coll_stack[i].coll[j].Qtot :=
                coll_stack[i].coll[j].Qtot*reduce;
            FOR k IN [1..coll_stack[i].coll[j].ntrays] DO
                coll_stack[i].coll[j].Qin[k] :=
                coll_stack[i].coll[j].Qin[k]*reduce;
                coll_stack[i].coll[j].tray[k].Qin :=
                coll_stack[i].coll[j].tray[k].Qin*reduce;
            END;
          END;
        END;
        FOR i IN [1..nfeeds] DO
                feed_tray[i].Qin :=
                feed_tray[i].Qin*reduce;
        END;
     END reduce_Q;
     PROCEDURE zero_Q;
        reduce := 0;
        RUN reduce_Q;
     END zero_Q;
     PROCEDURE seqmod;
         RUN coll_column::seqmod;
        IF (hb_on) THEN
            RUN heat_balance;
        END;
     END seqmod;
END td_coll_column;
MODEL equilibrium_coll_column REFINES td_coll_column;
        condenser.VLE                   IS_REFINED_TO equilibrium_mixture;
        feed_tray[1..nfeeds]
                .input['feed'].state    IS_REFINED_TO equilibrium_mixture;
   INITIALIZATION
    PROCEDURE seqmod;
        RUN td_coll_column::seqmod;
        condenser.VLE.T.fixed                           := FALSE;
     END seqmod;
END equilibrium_coll_column;
MODEL basic_coll_column REFINES coll_column;
        nfeeds := 1;
        coll_stack[1..2].ncolls := 2;
        coll_stack[1..2].coll[1].z_set.up_down := -1.0;
        coll_stack[1..2].coll[2].z_set.up_down := 1.0;
END basic_coll_column;
MODEL cost_calc;
        Afrac           IS_A real; (* fraction of area taken by tray *)
        Fp1,
            Fm1         IS_A real; (* material factors for column  p 574*)
        Fd2,
            Fp2,
            Fm2         IS_A real; (* material factors for exchangers p 572*)
        M_S             IS_A real;
        Tin             IS_A real; (* in temperature of cooling water *)
        Uc              IS_A real; (* heat transfer coefficient for condenser *)
        CpW             IS_A real; (* heat capacity of cooling water *)
        Hs              IS_A real; (* heat of vaporization of steam *)
```

```
Cw              IS_A real; (* price of cooling water *)
Cs              IS_A real; (* price of steam *)
Tray_height     IS_A real; (* height of each tray *)

cost            IS_A factor;
column_cost,
    condenser_cost,
    condenser_min,
    condenser_max,
    reboiler_cost,
    reboiler_min,
    reboiler_max,
    water_cost,
    water_min,
    water_max,
    steam_cost,
    steam_min,
    steam_max           IS_A cost_per_time;
scale                   IS_A scaling_constant;
Area             IS_A area; (* total cross-sectional area of column *)
V[1..nsections]         IS_A molar_rate; (* vapor molar flowrate *)
V_bar[1..nsections]     IS_A molar_volume; (* vapor molar volume *)
M_g                    IS_A molar_mass; (*average molar mass of vapor *)
D                       IS_A distance; (* diameter of column *)
H                       IS_A distance; (* height of column *)
pi                      IS_A circle_constant;
DT_C        IS_A temperature; (* change in cooling water temperature *)
Ac,
    Ar,
    Acmin,
    Acmax,
    Armin,
    Armax               IS_A area; (* area of condenser and reboiler *)
QC,
   QR         IS_A energy_rate; (* heat duty of condenser and reboiler *)
Tc                      IS_A temperature; (* temperature of condenser *)
Tout                    IS_A temperature;
Fc1                     IS_A factor;
Fc2                     IS_A factor;
nsections               IS_A integer;
Feedtot,
    Feedmax,
    Feedmin             IS_A molar_rate;
F[1..nsections],
    Fmax,
    Fmin                IS_A factor; (* flooding factor *)
LMT      IS_A factor; (* log mean temperature difference in condenser *)
stot               IS_A factor; (* total number of trays in column *)
F[1..nsections] := 1.51{};
Fmax            := 2.5;
Fmin            := 0.75;
Afrac := 0.88{};
Fp1 := 1.0{};
Fm1 := 1.0{};
Fd2 := 1.0{};
Fp2 := 0.0{};
Fm2 := 1.0{};
M_S := 900{USdollar};
Tin := 459.67{R} + 70{R};
```

```
        Tout := 459.67{R} + 90{R};
        Uc := 100{BTU/hr/ft^2/R};
        CpW := 1{cal/mole/K};
        Hs := 933{BTU/lbm};
        Cw := 0.03{USdollar}/1000{gallon};
        Cs := 2.5{USdollar}/1000{lbm};
        Tray_height := 2.0{ft};

        V_bar[1..nsections] := 24{liter/mol};
        M_g := 70{g/mol};
        Tc := 350{K};
        QC := -30{kW};
        QR := 30{kW};
        FOR j IN [1..nsections] CREATE
            Area = 1{ft^2}*V[j]*1{hr/lb_mole}
                *sqrt(M_g*1{lb_mole^2/lbm/ft^3}
                *V_bar[j])/Afrac/F[j]/3600;
        END;
        F[1]*Feedmin = Fmin*Feedtot;
        F[1]*Feedmax = Fmax*Feedtot;
        Acmin*F[1] = Fmin*Ac;
        Acmax*F[1] = Fmax*Ac;
        Armin*F[1] = Fmin*Ar;
        Armax*F[1] = Fmax*Ar;

        D = (4*Area/pi)^0.5;
        H = Tray_height*1.15*stot;
        Fc1 = Fm1*Fp1;
        Fc2 = (Fd2+Fp2)*Fm1;
        DT_C = (Tout - Tin);
        LMT = ln((Tc-Tin)/(Tc-Tout));
        Ac = -QC*LMT/((Tout-Tin)*Uc);
        Ar = QR/11250{BTU/hr/ft^2};
        Tout IS_REFINED_TO temperature;
        Tout = Tc - 5{K};
        c1: column_cost = (M_S/280/3{yr})*(120*(D/1{ft})
            *((H/1{ft})^0.8))*(2.18 + Fc1);
        c2: condenser_cost = (M_S/280/3{yr})*(101.3)
            *(2.29+Fc2)*((Ac/1{ft^2})^0.65);
        c3: reboiler_cost = (M_S/280/3{yr})*(101.3)
            *(2.29+Fc2)*((Ar/1{ft^2})^0.65);
        c4: water_cost =  Cw*(-QC)*1{ml/g}*18{g/mole}/(CpW*DT_C);
        c5: steam_cost = Cs*QR/Hs;
        condenser_min = (M_S/280/3{yr})*(101.3)*(2.29+Fc2)
            *((Acmin/1{ft^2})^0.65);
        condenser_max = (M_S/280/3{yr})*(101.3)*(2.29+Fc2)
            *((Acmax/1{ft^2})^0.65);
        reboiler_min = (M_S/280/3{yr})*(101.3)*(2.29+Fc2)
            *((Armin/1{ft^2})^0.65);
        reboiler_max = (M_S/280/3{yr})*(101.3)*(2.29+Fc2)
            *((Armax/1{ft^2})^0.65);
        water_min*F[1] = Fmin*water_cost;
        water_max*F[1] = Fmax*water_cost;
        steam_min*F[1] = Fmin*steam_cost;
        steam_max*F[1] = Fmax*steam_cost;
        c_tot1: cost*1.0{USdollar/yr} = column_cost
            + condenser_cost + reboiler_cost
            + water_cost + steam_cost;
INITIALIZATION
```

```
    PROCEDURE seqmod;
        F[1].fixed              := TRUE;
        Fmin.fixed              := TRUE;
        Fmax.fixed              := TRUE;
    END seqmod;
    PROCEDURE specify;
        Tc.fixed := TRUE;
        M_g.fixed := TRUE;
        QC.fixed := TRUE;
        QR.fixed := TRUE;
        V[1..nsections].fixed := TRUE;
        V_bar[1..nsections].fixed := TRUE;
        stot.fixed := TRUE;
    END specify;
    PROCEDURE reset;
        RUN clear;
        RUN specify;
    END reset;
END cost_calc;
MODEL cost_column;
        cost_calc               IS_A cost_calc;
        col                     IS_A td_coll_column;
        cost_calc.Tc,
            col.condenser.VLE.T ARE_THE_SAME;
        cost_calc.QC,
            col.condenser.Qin   ARE_THE_SAME;
        cost_calc.QR,
            col.reboiler.Qin    ARE_THE_SAME;
        cost_calc.nsections := col.nfeeds+1;
        FOR j IN [1..col.nfeeds+1] CREATE
            cost_calc.V[j],
                col.coll_stack[j].coll[1]
                .tray[1].vapout['vapor'].Ftot  ARE_THE_SAME;
            cost_calc.V_bar[j],
                col.coll_stack[j].coll[1]
                .tray[1].vapout['vapor'].state.V  ARE_THE_SAME;
        END;
        cost_calc.stot,
            col.stot            ARE_THE_SAME;
        cost_calc.M_g = SUM(col.feed_tray[1].data[i].mw
            *col.feed_tray[1].vapout['vapor'].state.y[i]
            | i IN col.components);
        cost_calc.Feedtot,
            col.feed_tray[1].input['feed'].Ftot ARE_THE_SAME;
  INITIALIZATION
    PROCEDURE seqmod;
        RUN col.seqmod;
        RUN cost_calc.seqmod;
    END seqmod;
    PROCEDURE specify;
        RUN seqmod;
        RUN col.feed_tray[1..col.nfeeds].input['feed'].specify;
    END specify;
    PROCEDURE reset;
        RUN clear;
        RUN specify;
    END reset;
END cost_column;
MODEL reflux_fit;
```

```
        npoints,
            order               IS_A integer;
        reflux[1..npoints],
            stot[1..npoints],
            frac[1..npoints],
            R_hat[1..npoints],
            error[1..npoints],
            stot_coeff[1..npoints][0..order],
            frac_coeff[0..order][0..order],
            tot_error                   IS_A factor;
        FOR i IN [1..npoints] CREATE
            R_hat[i] = stot_coeff[i][0] +
                SUM(stot_coeff[i][k]*(PROD(stot[i] | m IN [1..k]))
                | k IN [1..order]);
            FOR k IN [0..order] CREATE
                stot_coeff[i][k] = frac_coeff[k][0] +
                    SUM(frac_coeff[k][j]*(PROD(frac[i] | m IN [1..j]))
                    | j IN [1..order]);
            END;
            error[i] = 0.5*sqr(reflux[i] - R_hat[i]);
        END;
        tot_error = SUM(error[i] | i IN [1..npoints]);
        MINIMIZE tot_error;
  INITIALIZATION
    PROCEDURE specify;
        reflux[1..npoints].fixed        := TRUE;
        stot[1..npoints].fixed          := TRUE;
        frac[1..npoints].fixed          := TRUE;
        frac_coeff[0..order][0..order].fixed    := TRUE;
    END specify;
    PROCEDURE reset;
        RUN clear;
        RUN specify;
    END reset;
    PROCEDURE free;
        frac_coeff[0..order][0..order].fixed           := FALSE;
    END free;
    PROCEDURE zero_error;
        error[1..9].fixed := TRUE;
        frac_coeff[0..order][0..order].fixed           := FALSE;
        error[1..9] := 0.0;
    END zero_error;
END reflux_fit;
MODEL approximate_column;
        order           IS_A integer;
        frac_coeff[0..order][0..order],
            stot_coeff[0..order],
            reflux_ratio,
            calc_reflux_ratio,
            stot,
            s_stack[1..2]       IS_A factor;
        split[1..2]             IS_A fraction;
        cost                    IS_A factor;
        column_cost,
            condenser_cost,
            reboiler_cost,
            water_cost,
            steam_cost          IS_A cost_per_time;
        Afrac,
```

```
    M_S,
    Tin,
    Uc,
    CpW,
    Hs,
    Cw,
    Cs,
    Tray_height,
    Qc0,
    Qr0,
    V0                  IS_A real;
Area,
    Ac,
    Ar                  IS_A area;
Feed,
    Distillate,
    V                   IS_A molar_rate;
V_bar                   IS_A molar_volume;
M_g                     IS_A molar_mass;
D,
    H                   IS_A distance;
pi                      IS_A circle_constant;
DT_C,
    Tout                IS_A temperature;
QC,
    QR                  IS_A energy_rate;
Fc1,
    Fc2,
    F,
    LMT                 IS_A factor;
D_F                     IS_A fraction;
F := 1.51 {};
Afrac := 0.88{};
M_S := 900{USdollar};
Tin := 459.67{R} + 70{R};
Tout := 459.67{R} + 90{R};
Uc := 100{BTU/hr/ft^2/R};
CpW := 1{cal/mole/K};
Hs := 933{BTU/lbm};
Cw := 0.03{USdollar}/1000{gallon};
Cs := 2.5{USdollar}/1000{lbm};
Tray_height := 2.0{ft};

V_bar := 24{liter/mol};
M_g := 70{g/mol};
QC := -30{kW};
QR := 30{kW};
Qc0 := -30{kW};
Qr0 := 30{kW};
V0 := 3{mol/s};
Area = 1{ft^2}*V*1{hr/lb_mole}
    *sqrt(M_g*1{lb_mole^2/lbm/ft^3}
    *V_bar)/Afrac/F/3600;
D = (4*Area/pi)^0.5;
H = Tray_height*1.15*stot;
QC = V*Qc0/V0;
QR = V*Qr0/V0;
Ac = -QC*LMT/((Tout-Tin)*Uc);
Ar = QR/11250{BTU/hr/ft^2};
```

```
V = Distillate*(reflux_ratio+1);
Distillate = D_F*Feed;
c1: column_cost =
    (M_S/280/3{yr})*120*(D/1{ft})*(H/1{ft})^0.8*(2.18+Fc1);
c2: condenser_cost =
    (M_S/280/3{yr})*(101.3)*(2.29+Fc2)*(Ac/1{ft^2})^0.65;
c3: reboiler_cost = (M_S/280/3{yr})*(101.3)
    *(2.29+Fc2)*((Ar/1{ft^2})^0.65);
c4: water_cost =  Cw*(-QC)*1{ml/g}*18{g/mole}/(CpW*DT_C);
c5: steam_cost = Cs*QR/Hs;
cost*1.0{USdollar/yr} = column_cost
    + condenser_cost + reboiler_cost +
    water_cost + steam_cost;
calc_reflux_ratio = stot_coeff[0] +
    SUM(stot_coeff[k]*(PROD(stot | i IN [1..k]))
    | k IN [1..order]);
FOR k IN [0..order] CREATE
    stot_coeff[k] = frac_coeff[k][0] + SUM(frac_coeff[k][j]
        *(PROD(split[1] | i IN [1..j]))
        | j IN [1..order]);
END;
stot = SUM(s_stack[i] | i IN [1..2]);
FOR i IN [1..2] CREATE
    stot*split[i] = s_stack[i];
END;
equal: reflux_ratio = calc_reflux_ratio;
inequal: calc_reflux_ratio <= reflux_ratio;
(*      test_opt: MINIMIZE cost; *)
frac_coeff[0..order][0..order].lower_bound := -1e6;
stot_coeff[0..order].lower_bound := -1e6;
reflux_ratio.lower_bound        := 0.1;
calc_reflux_ratio.lower_bound   := 0.1;
stot.lower_bound                := 1;
split[1..2].lower_bound         := 0.1;
s_stack[1..2].lower_bound       := 1;
cost.lower_bound                := 0.0;
column_cost.lower_bound         := 0.0{USdollar/yr};
condenser_cost.lower_bound      := 0.0{USdollar/yr};
reboiler_cost.lower_bound       := 0.0{USdollar/yr};
water_cost.lower_bound          := 0.0{USdollar/yr};
steam_cost.lower_bound          := 0.0{USdollar/yr};
Area.lower_bound                := 0.0001{ft^2};
V.lower_bound                   := 0.0{mol/s};
V_bar.lower_bound               := 0.0{liter/mol};
M_g.lower_bound                 := 0.0{g/mol};
D.lower_bound                   := 1.0{ft};
H.lower_bound                   := 4.0{ft};
DT_C.lower_bound                := 0.0{R};
QC.lower_bound                  := -10000{kW};
QR.lower_bound                  := -10000{kW};
F.lower_bound                   := 0.2;
D_F.lower_bound                 := 0.01;
Distillate.lower_bound          := 0.0{mol/s};
Feed.lower_bound                := 0.0{mol/s};
Fc1.lower_bound                 := 0.0;
Fc2.lower_bound                 := 0.0;
frac_coeff[0..order][0..order].upper_bound := 1e6;
stot_coeff[0..order].upper_bound := 1e6;
reflux_ratio.upper_bound        := 40.0;
```

```
        calc_reflux_ratio.upper_bound   := 40.0;
        stot.upper_bound                := 100.0;
        split[1..2].upper_bound         := 0.9;
        s_stack[1..2].upper_bound       := 100.0;
        cost.upper_bound                := 1e9;
        column_cost.upper_bound         := 1e9{USdollar/yr};
        condenser_cost.upper_bound      := 1e9{USdollar/yr};
        reboiler_cost.upper_bound       := 1e9{USdollar/yr};
        water_cost.upper_bound          := 1e9{USdollar/yr};
        steam_cost.upper_bound          := 1e9{USdollar/yr};
        Area.upper_bound                := 1e9{ft^2};
        V.upper_bound                   := 1e9{mol/s};
        V_bar.upper_bound               := 500{liter/mol};
        M_g.upper_bound                 := 500{g/mol};
        D.upper_bound                   := 50.0{ft};
        H.upper_bound                   := 200.0{ft};
        DT_C.upper_bound                := 300.0{R};
        QC.upper_bound                  := 10000{kW};
        QR.upper_bound                  := 10000{kW};
        F.upper_bound                   := 2.4;
        D_F.upper_bound                 := 0.99;
        Distillate.upper_bound          := 1e9{mol/s};
        Feed.upper_bound                := 1e9{mol/s};
        Fc1.upper_bound                 := 2.4;
        Fc2.upper_bound                 := 2.40;
  INITIALIZATION
    PROCEDURE seqmod;
        frac_coeff[0..order][0..order].fixed := TRUE;
        inequal.included := FALSE;
    END seqmod;
    PROCEDURE specify;
        RUN seqmod;
        stot.fixed              := TRUE;
        split[1].fixed          := TRUE;
        V_bar.fixed             := TRUE;
        M_g.fixed               := TRUE;
        DT_C.fixed              := TRUE;
        F.fixed                 := TRUE;
        D_F.fixed               := TRUE;
        Feed.fixed              := TRUE;
        Fc1.fixed               := TRUE;
        Fc2.fixed               := TRUE;
        Tout.fixed              := TRUE;
        LMT.fixed               := TRUE;
    END specify;
    PROCEDURE reset;
        RUN clear;
        RUN specify;
    END reset;
END approximate_column;
MODEL apcol_set;
        nfeed_pointsIS_A integer;
        apcol[1..nfeed_points]  IS_A approximate_column;
        prob[1..nfeed_points]   IS_A fraction;
        apcol[1..nfeed_points].order := 2;
        cost,
            min_d,
            max_d               IS_A factor;
        column_cost,
```

```
        condenser_cost,
        reboiler_cost,
        water_cost,
        steam_cost          IS_A cost_per_time;
    SUM(prob[j] | j IN [1..nfeed_points]) = 1;
    column_cost, apcol[1].column_cost       ARE_THE_SAME;
    water_cost = SUM(prob[j]*apcol[j].water_cost | j IN
        [1..nfeed_points]);
    steam_cost = SUM(prob[j]*apcol[j].steam_cost | j IN
        [1..nfeed_points]);
    con_equal: condenser_cost =
        SUM(prob[j]*apcol[j].condenser_cost | j IN
        [1..nfeed_points]);
    reb_equal: reboiler_cost =
        SUM(prob[j]*apcol[j].reboiler_cost | j IN
        [1..nfeed_points]);
    FOR j IN [1..nfeed_points] CREATE
        con_inequal[j]: apcol[j].condenser_cost <= condenser_cost;
        reb_inequal[j]: apcol[j].reboiler_cost <= reboiler_cost;
    END;
    cost*1.0{USdollar/yr} = column_cost + condenser_cost +
        reboiler_cost + water_cost + steam_cost;

    FOR j IN [1..nfeed_points] CREATE
        prob[j] := 1.0/nfeed_points;
    END;
    apcol[1..nfeed_points].stot     ARE_THE_SAME;
    apcol[1..nfeed_points].D        ARE_THE_SAME;
    MINIMIZE cost;
INITIALIZATION
  PROCEDURE bounds;
    FOR j IN [1..nfeed_points] DO
        apcol[j].stot.lower_bound := min_d*apcol[j].stot;
        apcol[j].split[1].lower_bound := min_d*apcol[j].split[1];
        apcol[j].stot.upper_bound := max_d*apcol[j].stot;
        apcol[j].split[1].upper_bound := max_d*apcol[j].split[1];
    END;
  END bounds;
  PROCEDURE F_bounds_on;
    apcol[1..nfeed_points].F.lower_bound := 0.8;
    apcol[1..nfeed_points].F.upper_bound := 2.4;
  END F_bounds_on;
  PROCEDURE F_bounds_off;
    apcol[1..nfeed_points].F.lower_bound := 0.0;
    apcol[1..nfeed_points].F.upper_bound := 20;
  END F_bounds_off;
  PROCEDURE seqmod;
    RUN apcol[1..nfeed_points].seqmod;
    prob[2..nfeed_points].fixed := TRUE;
    apcol[2..nfeed_points].F.fixed := FALSE;
    con_inequal[1..nfeed_points].included := FALSE;
    reb_inequal[1..nfeed_points].included := FALSE;
  END seqmod;
  PROCEDURE specify;
    RUN apcol[1..nfeed_points].specify;
    prob[2..nfeed_points].fixed := TRUE;
    apcol[2..nfeed_points].F.fixed := FALSE;
    con_inequal[1..nfeed_points].included := FALSE;
    reb_inequal[1..nfeed_points].included := FALSE;
```

```
    END specify;
    PROCEDURE reset;
        RUN clear;
        RUN specify;
    END reset;
    PROCEDURE free;
        RUN reset;
        apcol[1].stot.fixed := FALSE;
        apcol[1].F.fixed := FALSE;
        apcol[1..nfeed_points].split[1..2].fixed := FALSE;
        apcol[1..nfeed_points].equal.included := FALSE;
        apcol[1..nfeed_points].inequal.included := TRUE;
        con_equal.included := FALSE;
        reb_equal.included := FALSE;
        con_inequal[1..nfeed_points].included := TRUE;
        reb_inequal[1..nfeed_points].included := TRUE;
    END free;
END apcol_set;
MODEL standard_cost REFINES column_w_plot;

        nfeed_points            IS_A integer;
        current_col             IS_A mutable_integer;
        feed[1..nfeed_points]   IS_A molar_stream;
        col_set                 IS_A apcol_set;
        col_fit[1..nfeed_points]        IS_A reflux_fit;
        col_fit[1..nfeed_points].order,
            col_set.apcol[1].order      ARE_THE_SAME;
        FOR k IN [1..nfeed_points] CREATE
            FOR i IN [0..col_fit[1].order] CREATE
                FOR j IN [0..col_fit[1].order] CREATE
                    col_fit[k].frac_coeff[i][j],
                        col_set.apcol[k].frac_coeff[i][j]       ARE_THE_SAME;
                END;
            END;
        END;

        current_col := 1;
        col_fit[1..nfeed_points].npoints := 9;

        col_set.nfeed_points,
            nfeed_points        ARE_THE_SAME;
        cc      IS_A cost_column;
        feed[1..nfeed_points].components,
            components          ARE_THE_SAME;

        cc.col,
            col ARE_THE_SAME;
        col.nfeeds := 1;
        col.coll_stack[1..2].ncolls := 2;
        col.coll_stack[1..2].coll[1].z_set.up_down := -1.0;
        col.coll_stack[1..2].coll[2].z_set.up_down := 1.0;
        col.coll_stack[1..col.nfeeds+1].coll
            [1..col.coll_stack[1].ncolls].z_set.lgr IS_REFINED_TO lgr_2_points;
        col IS_REFINED_TO td_coll_column;
        col.coll_stack[1..2].coll[1..2] IS_REFINED_TO h_coll;
        cc.cost_calc.Afrac,
            col_set.apcol[1..nfeed_points].Afrac                ARE_THE_SAME;
        cc.cost_calc.M_S,
            col_set.apcol[1..nfeed_points].M_S          ARE_THE_SAME;
```

```
        cc.cost_calc.Fc1,
            col_set.apcol[1..nfeed_points].Fc1          ARE_THE_SAME;
        cc.cost_calc.Fc2,
            col_set.apcol[1..nfeed_points].Fc2          ARE_THE_SAME;
        cc.cost_calc.Cw,
            col_set.apcol[1..nfeed_points].Cw           ARE_THE_SAME;
        cc.cost_calc.CpW,
            col_set.apcol[1..nfeed_points].CpW          ARE_THE_SAME;
        cc.cost_calc.Cs,
            col_set.apcol[1..nfeed_points].Cs           ARE_THE_SAME;
        cc.cost_calc.Hs,
            col_set.apcol[1..nfeed_points].Hs           ARE_THE_SAME;
        cc.cost_calc.V_bar[1],
            col_set.apcol[1..nfeed_points].V_bar                ARE_THE_SAME;
        cc.cost_calc.Tray_height,
            col_set.apcol[1..nfeed_points].Tray_height          ARE_THE_SAME;
        cc.cost_calc.Uc,
            col_set.apcol[1..nfeed_points].Uc           ARE_THE_SAME;

INITIALIZATION
  PROCEDURE seqmod;
      plots.z_space.fixed := TRUE;
      plots.box_height.fixed := TRUE;
      RUN cc.seqmod;
      RUN col_set.seqmod;
      RUN feed[1..nfeed_points].specify;
  END seqmod;
  PROCEDURE specify;
      RUN seqmod;
      RUN col.feed_tray[1..col.nfeeds].input['feed'].specify;
  END specify;
  PROCEDURE spec1;
      RUN reset;
      col.stot.fixed := TRUE;
      col.split[1].fixed := TRUE;
      col.s_stack[1..2].fixed := FALSE;
  END spec1;
  PROCEDURE setapcol;
      col_set.apcol[current_col].Qc0 := col.condenser.Qin;
      col_set.apcol[current_col].Qr0 := col.reboiler.Qin;
      col_set.apcol[current_col].V0 := col.condenser.input['vapor'].Ftot;
      col_set.apcol[current_col].D_F :=
          col.condenser.totprod.Ftot/col.feed_tray[1].input['feed'].Ftot;
      col_set.apcol[current_col].F := cc.cost_calc.F[1];
      col_set.apcol[current_col].DT_C := cc.cost_calc.DT_C;
      col_set.apcol[current_col].M_g := cc.cost_calc.M_g;
      col_set.apcol[current_col].LMT := cc.cost_calc.LMT;
      col_set.apcol[current_col].Tout := cc.cost_calc.Tout;
      col_set.apcol[current_col].Tin := cc.cost_calc.Tin;
      col_set.apcol[current_col].Feed := cc.cost_calc.Feedtot;
      col_set.apcol[current_col].stot := col.stot;
      col_set.apcol[current_col].split[1] := col.split[1];
  END setapcol;
  PROCEDURE setfeed;
      FOR i IN components DO
          col.feed_tray[1].input['feed'].f[i] :=
              feed[current_col].f[i];
      END;
  END setfeed;
```

```
    PROCEDURE setup_opt;
        RUN col_set.bounds;
        RUN col_set.F_bounds_on;
        FOR j IN [1..nfeed_points] DO
            FOR k IN [1..3] DO
                IF (col_fit[j].stot[k] > col_set.apcol[j].stot.lower_bound) THEN
                    col_set.apcol[j].stot.lower_bound :=
                        col_fit[j].stot[k];
                END;
            END;
            FOR k IN [7..9] DO
                IF (col_fit[j].stot[k] < col_set.apcol[j].stot.upper_bound) THEN
                    col_set.apcol[j].stot.upper_bound :=
                        col_fit[j].stot[k];
                END;
            END;
            FOR k IN [1,4,7] DO
                IF (col_fit[j].frac[k] <
                    col_set.apcol[j].split[1].upper_bound) THEN
                    col_set.apcol[j].split[1].upper_bound :=
                        col_fit[j].frac[k];
                END;
            END;
            FOR k IN [3,6,9] DO
                IF (col_fit[j].frac[k] >
                    col_set.apcol[j].split[1].lower_bound) THEN
                    col_set.apcol[j].split[1].lower_bound :=
                        col_fit[j].frac[k];
                END;
            END;
        END;
    END setup_opt;
END standard_cost;
MODEL acb1 REFINES standard_cost;
        col.components := ['c1_acetone','c2_chloroform','c3_benzene'];
        col.coll_stack[1..col.nfeeds+1].inactive_component := 'c2_chloroform';
        col.data['c1_acetone'] IS_REFINED_TO acetone;
        col.data['c2_chloroform'] IS_REFINED_TO chloroform;
        col.data['c3_benzene'] IS_REFINED_TO benzene;
        nfeed_points := 1;
  INITIALIZATION
    PROCEDURE values;
        col.feed_tray[1].alpha['c1_acetone'] := 1.4;
        col.feed_tray[1].alpha['c2_chloroform'] := 1.2;
        col.feed_tray[1].alpha['c3_benzene'] := 1.0;
        col.feed_tray[1].input['feed'].f['c1_acetone'] := 3.0 {mole/s};
        col.feed_tray[1].input['feed'].f['c2_chloroform'] := 3.0 {mole/s};
        col.feed_tray[1].input['feed'].f['c3_benzene'] := 3.0 {mole/s};
        feed[1].f['c1_acetone'] := 3.0 {mole/s};
        feed[1].f['c2_chloroform'] := 3.0 {mole/s};
        feed[1].f['c3_benzene'] := 3.0 {mole/s};
        col.condenser.totprod.Ftot := 3.0 {mol/s};
        RUN col.propogate_feed;
        FOR j IN [1..2] DO
            col.coll_stack[j].split[1] := 0.5;
            col.coll_stack[j].stot := 2;
            col.coll_stack[j].coll[1].z_set.stot := 1;
            col.coll_stack[j].coll[1..col.coll_stack[1].ncolls].z_set.a := 0.1;
        END;
```

```
        col.feed_tray[1].q := 1.0;
        col.condenser.prodsplit[‘vapor_product’] := 0.0;
        col.reboiler.prodsplit[‘vapor_product’] := 0.0;
        col.condenser.reflux_ratio := 0.5;
    END values;
END acb1;
```