Development of Computational Tools for the Design, Simulation and Optimization of Cyclic Steady State (CSS) Chromatographic Processes

Kevin Wood

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

Doctor of Philosophy

In

Chemical Engineering

Y. A. Liu, Chair

Donald G. Baird

Richey M. Davis

Stephen M. Martin

June 6, 2016

Blacksburg, VA

Keywords: Simulated Moving Bed Chromatography, Process Design, Simulation

Development of Computational Tools for the Design, Simulation and Optimization of Cyclic Steady State (CSS) Adsorption and Chromatographic Processes

Kevin Wood

Abstract

This dissertation presents an analysis of two aspects of the chromatographic separation process known as Simulated Moving Bed (SMB) chromatography. The first aspect is system design, and the second is improving computer simulations to generate heuristics for choosing operational modes.

For the past 15-20 years, there has been a surge of interest in the use of Simulated Moving Bed systems for the chromatographic separation of chemicals[1]. A wide variety of methods, nomenclatures, and conventions have been adopted over the years[2–4], as teams from different backgrounds adopt and improve on the SMB technology. This work presents a unifying discussion of the two major design methods, Triangle Theory and Standing Wave Design, used in the SMB field. We provide the complete computer code required to execute both design methods. A sample problem is worked, which demonstrates the novelty and ease of use that such tools provide. Mathematica was chosen for the implementation of these design methods, because of its strong symbolic analysis capabilities, and simplicity of creating interfaces for new users. We present derivations of the classic Langmuir results in Mathematica, and proceed to extend those implementations. When analytic solutions are impossible, we use Mathematica's numerical methods.

This work also develops a distributed computing tool known as ChromRunner which allows large numbers of detailed numerical simulations to be run simultaneously. The motivations and benefits of this approach are discussed alongside implementation details. We apply the distributed computing system to two separate SMB separations in order to optimize them, as well as determine heuristics governing their operational modes. We wrote ChromRunner in C#, and took advantage of Visual Studio's Entity Framework to create the database backend. The user interface for this software was created using Microsoft's "Windows Presentation Foundation" (WPF) technologies.

Acknowledgement

We would very much like to thank Aspen Tech, SINOPEC, MATRIC, and BAE systems for making our research possible.

I am grateful to my advisor Dr. Y. A. Liu who has consistently provided me with the motivation and resources I needed to complete my work.

I extend a thank you to my committee members, Professors Donald Baird, Richey Davis, and Stephen Martin, who provided valuable feedback and guidance.

I am indebted to Stuart Higgins for valuable technical discussions on my many programming projects. I would like to thank Yueying Yu with whom I have worked very hard on many projects. I would also like to thank Wei Shan Chin and Wei-Jun Wang for their encouragement.

Finally, I would like to thank my wife Jessica for her patience and support.

TABLE OF CONTENTS

1 Introduction to the Dissertation 1
1.1 Scope and Organization 1
1.2 Motivations for the Research 1
1.3 Significance of the Research 2
2 Mathematical Basis of Adsorptive and Chromatographic Separations 3
2.1 Physical Properties in Adsorption: Characterizing Solids 3
2.1.1 Void Fractions 4
2.1.2 Densities 6
2.1.3 Relationships 6
2.1.4 Characterizing the Liquid Phase 7
2.2 The Adsorption Process 8
2.2.1 Mass Transfer Resistance 8
2.2.2 Dead Volume 8
2.3 Isotherm Equations 9
2.3.1 The Langmuir Isotherm 9
2.3.2 The Linear Isotherm 10
2.3.3 The Brunauer–Emmett–Teller Isotherm B.E.T. (1938) 10
2.3.4 The Freundlich Isotherm (1906) 10
2.3.5 The Sips (Langmuir-Freundlich) Isotherm (1948) 10
2.3.6 The Toth Isotherm (1963) 10
2.3.7 Summary 11
2.4 Mass Balances 11
2.5 Derivation of the Differential Equations 13
2.6 Differential Equations 15
2.7 Analytic solutions: The Method of Characteristics 16
2.8 Steady State vs Cyclic Steady State 16
2.9 SMB Chromatography 17
2.9.1 Introduction 17
2.9.2 A Model System 17
2.10 Overview of the Computer Tools 19

2.10.1 Aspen Custom Modeler 20
2.10.2 Mathematica 20
3 Simulated Moving Bed Design Methods 21
3.1 Introduction to SMB 21
3.1.1 Key Features and Terms 21
3.2 General Concepts 23
3.2.1 The Design Problem 23
3.2.2 SMB vs TMB 26
3.2.3 Solution Approaches 26
3.3 Triangle Theory 28
3.3.1 Introduction 28
3.3.2 Constraints on the System 30
3.3.3 Differential Equations – Analysis 31
3.3.4 Constructing the Solution from Eigenvectors and Eigenvalues 34
3.3.5 Exercize 1: Calculating Transitions in a Fixed Bed using Mathematica 35
3.3.6 Use the Steady State Information to Constrain Operating Conditions 47
3.3.7 Exercise 2: Constructing the Constraints on the TMB System in Mathematica 49
3.4 Standing Wave Design 54
3.4.1 Standing Wave Design in a Nonlinear Ideal System 54
3.4.2 Notation Used in the Standing Wave Design Literature 56
3.4.3 Standing Wave Design in a System with Nonlinear Isotherm 60
3.5 Remarks 61
4 Application of Mathematica Tools 63
4.1 The Triangle Theory Method Applied to Non-Langmuir Isotherms 63
4.2 Application of the User Interface for SMB Design 64
4.2.1 Workshop: Calculating the Triangle Theory Region for a System with a Toth Isotherm 64
5 Optimization of SMB systems 70
5.1 Explanation of Optimization Variables and Solution Strategies 70
5.1.1 Analytic Solutions 70
5.1.2 Optimization Variables 70
5.1.3 Optimization in Aspen Chromatography 70
5.1.4 Summary 71
5.2 Distributed System 73

5.2.1 Motivation 73
5.2.2 System Requirements 73
5.2.3 Use 73
5.2.4 Code Structure 75
5.2.5 Specific Algorithms 76
5.2.6 Optimization Algorithms 83
6 Applications 84
6.1 SMB Operational Modes for Binary Separation 84
6.2 Varicol 85
6.2.1 Description 85
6.2.2 Varicol Modeling and Optimization 86
6.2.3 Varicol Results 87
6.3 PowerFeed 88
6.3.1 Description 88
6.3.2 Modeling and Optimization 88
6.4 ModiCon 92
6.4.1 Description 92
6.4.2 Modeling and Optimization 92
6.5 Comparison of Separation Feasibility Regions of Basic Operational Modes 97
6.5.1 Varicol 97
6.5.2 PowerFeed 97
6.5.3 ModiCon 98
6.6 Combined Operational Modes 100
6.7 Comparison of Sensitivities across All Operational Modes 102
6.8 Five-Zone SMB and Operational Modes for Ternary Separation 103
6.9 Comparison of Separation Feasibility regions of Basic Operational Modes 104
6.9.1 Varicol Model 105
6.9.2 PowerFeed Model 105
6.9.3 ModiCon Model 105
6.10 Combined Operational Modes 105
6.11 Findings 106
7 Concluding Remarks 107
8 Literature Cited 109

9 Appendix 112

• TABLE OF FIGURES

Figure 2.1: The Locations of the Various Reference Volumes 4
Figure 2.2: Reference Volumes Used in Solid Characterization 5
Figure 2.3: The Phases involved in the Adsorption Process 8
Figure 2.4: The Locations of Dead Volumes in a Packed Bed 9
Figure 2.5: Sample Isotherm Curves 11
Figure 2.6: Diagram of a Countercurrent Adsorber 12
Figure 2.7: A Differential Column Element 14
Figure 2.8: Diagram of a Basic 4-zone True Moving Bed 17
Figure 2.9: Diagram of 4-zone Simulated Moving Bed Operation 18
Figure 2.10: Net Motion of the components in a 4-zone system 19
Figure 3.1: True Moving Bed (Left) compared to a Simulated Moving Bed (Right) 22
Figure 3.2: Ideal Separation Case 24
Figure 3.3: Zone Roles in TMB separation 24
Figure 3.4: SMB Unit under Ideal Separation Conditions 25
Figure 3.5: SMB Concentration Profile under Ideal Separation Conditions 25
Figure 3.6: Degrees of Freedom in an SMB System 26
Figure 3.7: A history of the design approaches 27
Figure 3.8: The differences between triangle theory and Standing Wave Design 28
Figure 3.9: Net flow through a zone 29
Figure 3.10: Examples of Steady State Conditions 29
Figure 3.11: Boundary Conditions in an Absorber 34
Figure 3.12: The eigenvector directions depend on which component is more strongly adsorbed 39
Figure 3.13: Transitions in the $\mathbf{c1}, \mathbf{c2}$ plane; the eigenvector prediction (blue line) vs numerical simulation in Aspen (red x). The yellow curve is the path that would be taken during the reverse transition. 41
Figure 3.14 transitions in the $\mathbf{x} - \boldsymbol{\tau}$ plane 42

Figure 3.15: The breakthrough curve defined by the curves in Figure 3.14, without accounting for shocks 44
Figure 3.16 Formation of shockwaves 45
Figure 3.17 transitions in the $\mathbf{x}-\boldsymbol{\tau}$ plane with shockwaves accounted for 46
Figure 3.18 Breakthrough curves with shockwaves accounted for 47
Figure 3.19 TMB operation with zone steady states labeled 47
Figure 3.20 Transitions in a TMB column 48
Figure 3.21 Velocities are proportional to $\mathbf{m}-\boldsymbol{\theta}$ 49
Figure 3.22: Constraints on column operation by the four solutions for $\boldsymbol{c2II}$ and $\boldsymbol{c1III}$. Blue and yellow denote regions where a constraint is satisfied. When both colors are present, operation is only possible where the regions intersect 53
Figure 3.23: Region where all constraints are satisfied 53
Figure 3.24: Simple Standing Wave Design Algorithm 55
Figure 3.25: Feed State Imposes Upper Bound on $\boldsymbol{c2II}$ 57
Figure 3.26: Standing Wave Design in a Nonlinear Nonideal System 60
Figure 4.1: Inputting the values into the Mathematica User Interface 65
Figure 4.2: Optimum Operating Region in the $\boldsymbol{m2}$, $\boldsymbol{m3}$ plane 66
Figure 4.3: Optimum Operating Region in the $\boldsymbol{m4}$,$\boldsymbol{m1}$ plane 67
Figure 5.1: ChromRunner Main Window with 4 Aspen Instances Waiting for Instructions 74
Figure 5.2: Defining an Optimization Experiment 74
Figure 5.3: Configuration Options for ChromRunner 75
Figure 5.4: How Experiments Are Stored in the Database 76
Figure 5.5: How Simulations Are Stored in the Database 77
Figure 6.1 Diagram of the asynchronous shift of inlet/outlet ports in a Varicol process. 86
Figure 6.2 Varicol Optimization Strategy 87
Figure 6.3 Contour plot of A and B purities in the (Z_1, Z_3) plane with $Z_2 = 2.4$. 87
Figure 6.4 (a) An illustration of variable feed and recycle flow rates within subintervals in the modified PowerFeed process. (b) A comparison of feed and recycle flow rates policies of the basic SMB and PowerFeed in three switching subintervals. In this modified PowerFeed with three switching subintervals, t= 0 ~ t_1, F =F_1, Recy=$Recy_1$; t = t_1~ t_2, F = F_2, Recy=$Recy_2$; t = t_2 ~ t_3, F = F_3, Recy=$Recy_3$; t = $t_{sw,}$ F = F_1, Recy=$Recy_1$ 88

Figure 6.5 PowerFeed Optimization Steps. 89
Figure 6.6 Diagrams and Results for PowerFeed operation. 90
Figure 6.7 PowerFeed (PF) Concentration Profiles. 92
Figure 6.8 (a) An illustration of variable feed concentrations within three subintervals in the ModiCon process. (b) A comparison of feed concentration policies of the basic SMB and ModiCon within three switching subintervals. In this ModiCon with three switching subintervals, $t= 0 \sim t_1$, $C = C_1$; $t = t_1 \sim t_2$, $C = C_2$; $t = t_2 \sim t_3$, $C = C_3$; $t = t_{sw,}$ $C = C_1$. 92
Figure 6.9 ModiCon optimization steps 93
Figure 6.10 Final trend diagram for the ModiCon operation 94
Figure 6.11 Product concentrations in SMB and ModiCon (MC) operations. 95
Figure 6.12 Modicon (MC) Concentration Profiles with 40 grid points models 97
Figure 6.13 Modicon (MC) Concentration Profiles with 80 grid points models 97
Figure 6.14 Operating Region Comparison Plots for Single Operational Modes 99
Figure 6.15 Final trend diagram for the hybrid PowerFeed and ModiCon (PF&MC) operation. 101
Figure 6.16 Operating Region Comparison Plots for Combined Operational Modes 102
Figure 6.17 A purity effect of the m_2 value. 103
Figure 6.18 B recovery effect of the m_2 value. 103
Figure 6.19 B purity effect of the m_3 value. 103
Figure 6.20 A recovery effect of the m_3 value. 103
Figure 6.21 A purity effect on the change of feed flow rate. 103
Figure 6.22 B purity effect on the change of feed flow rate. 103
Figure 6.23 Operating Region Comparison Plots for Single Operational Modes for 5-Zone Ternary Separation. 105
Figure 6.24 Operating Region Comparison Plots for Combined Operational Modes for 5-Zone Ternary Separation. 106

1 Introduction to the Dissertation

1.1 Scope and Organization

This work tackles two major facets of Simulated Moving Bed system design.

In Chapter 3, this work unifies and clarifies the existing design and simulation methods, namely, the "Triangle Theory" and "Standing Wave Design" (SWD). We accomplish this through the development of tools in the Mathematica algebra system that implement both design methods. We explain the design methods step-by-step, and provide the corresponding Mathematica code alongside the derivation. The complete set of code used in the tools is provided in the Appendices. The implementations are used to produce documents that can be used simply and for free, without the need for expensive software or requiring the end user to compile code. The most visible benefit of this approach is the ease of use and flexibility of the Mathematica language. It allows the interactive exploration of the design method predictions and easy adjustment of the underlying assumptions. The second benefit is that the mathematical procedures and assumptions are explicitly laid out in the code, making it a valuable learning tool.

Shortcut design methods are only one piece of the SMB puzzle, however. Chapter 4 of this work addresses the challenges associated with detailed numerical simulations of SMB systems. Specifically, we discuss the development of a program to speed up detailed dynamic simulations and optimization in Aspen Chromatography, or indeed in any Aspen Custom Modeler variant. The program distributes the rigorous simulations across multiple computers, as well as CPU cores. This will allow engineers to more quickly and reliably optimize SMB processes and cyclic adsorption processes and open the door for complex analyses that would be impractical to do manually, or with simple scripts.

Chapter 5 discusses the use of the distributed simulation platform was used in a literature paper[5]. The speedup afforded by the distributed system allowed a more in depth analysis of "operational modes" than would have been possible via manual operation.

1.2 Motivations for the Research

The published literature contains a number of different methods and procedures for achieving the same goals in an SMB system[6–8]. Moreover, these methods and procedures tend to be developed for a narrow set of assumptions and system conditions (e.g. mass transfer assumptions and isotherm models.) This has the effect of requiring engineers and researchers to either shoehorn their system into a form that works with these methods, or spend a long time to re-derive the methods for their specific system (for example[2]). The end result is a fair amount of confusion regarding the applicability of SMB to systems involving less common mass transfer or isotherms, as well as duplicate effort. A suite of tools that allows easy access to these methods will produce some novel results (extension of methods to systems not previously considered), bring the current understanding of SMB design into a clearer focus, as well as provide a valuable learning resource for engineers who want to explore the applicability of SMB to their own processes.

Furthermore, the use of detailed simulations for rigorous optimization is not easily accessible to people outside academia. In the literature, there are numerous examples of using detailed simulation for optimization[9,10], but there is a distinct lack of any tools that make this feasible for people not well versed

in writing their own computer modeling code. In the literature, detailed simulations are typically developed by the researcher in a tool like gPROMS or MATLAB, and tools built from or into those software packages are used for optimization. Occasionally, software tools such as Aspen Chromatography are used[11–13], which dramatically simplifies the process of developing a process model. However, the optimization tools built into Aspen are not well suited for the long-running cyclic-steady-state (CSS) models that are inherent to SMB modeling and other adsorption processes like temperature swing adsorption (TSA) and pressure swing adsorption (PSA.) A tool that improves on the optimizations in Aspen Chromatography and Aspen Adsorption therefore provides the best of both worlds: engineers can develop models in Aspen Chromatography without needing to code their own model, and subsequently optimize the Aspen model with methods that were previously impractical.

1.3 Significance of the Research

This work coherently presents a unifying analysis of the two major design methods in SMB Chromatography and provides the following:

1. An explanation of the two methods on the same conceptual basis
2. A concrete and open source implementation of both design methods
3. A flexible platform for extending design methods to less conventional systems

The optimization section describes a system for distributing complex simulations across a network, and provides the following:

1. A high level description of the goals and approach of the project
2. Implementations of robust “wrappers” for the Aspen Custom Modeler family of products
3. Implementation details of the database support required for distributed operation
4. Evaluation of the strengths and weaknesses of the approach

Finally, the application section covers the use of the distributed system in a literature paper[5] designed to:

1. Find heuristics governing the selection and optimization of SMB operational modes
2. Investigate the combination of operational modes
3. Evaluate the effect of operational modes on the robustness of SMB systems

2 Mathematical Basis of Adsorptive and Chromatographic Separations

Adsorptive separations have a long history, and are important in many different disciplines. Because of this, there has been a great deal of high quality work done to build the mathematical models that describe these systems. However, the diversity of researchers has also led to a variety of nomenclatures and conventions that cause confusion and can make approaching the field difficult. In this chapter, we will cover some of the classical mathematical descriptions of adsorption with an emphasis on relating this field's jargon to the physical concepts.

2.1 Physical Properties in Adsorption: Characterizing Solids

Adsorptive separations fundamentally involve the contact between fluids and solids, and so it is important to first establish the basic concepts that govern heat and mass transfer in those conditions.

In order to describe any adsorptive separation, it is important to think clearly about how to characterize the particles and the voids within the adsorbent bed. As with most systems involving solid particles, the terms and conventions can be confusing or misleading, because many different disciplines have developed their own partially-overlapping naming schemes.

In adsorptive separations, the questions that we typically ask ourselves about the solid phase is essentially this: "Given some bed that is filled with both fluid and solid particles, what volume of the bed represents the solid phase and what is the volume of the liquid phase?" It turns out that we can answer that question in several different ways, and which answer is "correct" depends on the details of the system. However, despite the variety of possible answers, the answer always involves two physical properties: density and void fraction.

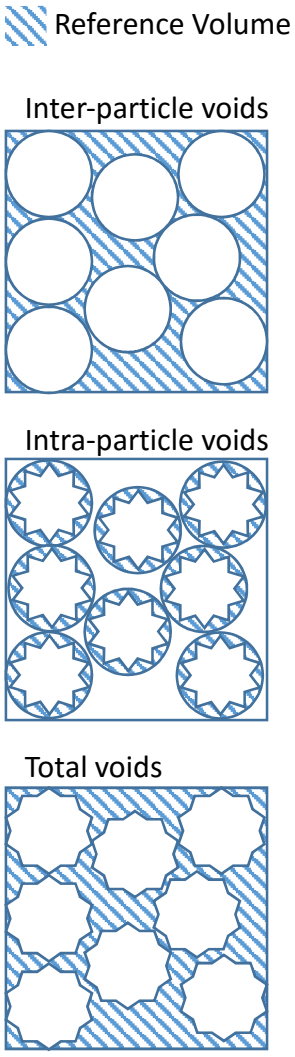


Figure 2.1: The Locations of the Various Reference Volumes

2.1.1 Void Fractions

The first set of important parameters, which are intimately tied to density, are the void fractions (or porosities) of a packed bed. Figure 2.1 depicts the three voids that are most relevant to adsorptive separations. Of course, the drawings in Figure 2.1 represent the voids themselves, but we want to know about the void *fractions*, that is, the size of the voids relative to some reference volume

2.1.1.1 External Voids

The first and most intuitive void fraction is the volume of space *not* occupied by particles in the bed to total bed volume. This ratio is called the **inter-particle, external,** or **interstitial** void fraction. While this seems simple at first, it is somewhat difficult to exactly determine what space is "inside" a particle and what space is "outside." Particles can have very irregular geometry, which can make the boundaries very fuzzy.

2.1.1.2 Internal Voids

The second void fraction follows naturally from the first; since external porosity defines the volume *outside* the particles, it is natural to want a ratio to describe the voids *inside* a particle. Unfortunately,

towards this end, two conventions have arisen. In chromatographic practice, it is common to define the **internal** or **intra-particle** void fraction as the ratio of voids in the particles to the total column volume. However, in chemical engineering circles, it is more common to hear **internal** or **intra-particle** void fraction to refer to the ratio of voids in the particles to the total volume of *particles* instead of the *column*. To see this in mathematical terms, consult Table 2.1.

The final void fraction is simply derived from the sum of both internal and external voids. The **total** void fraction is therefore the ratio of all voids in the system, to the total system volume.

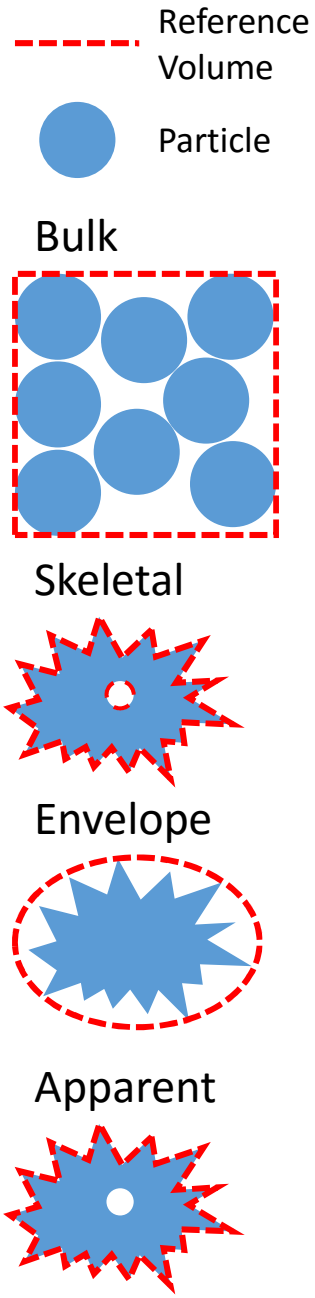


Figure 2.2: Reference Volumes Used in Solid Characterization

2.1.2 Densities

Density is simply characterized by dividing a mass of particles by a reference volume, and as you might have guessed, there are several meaningful ways to characterize the density of a group of particles. Figure 2.2 provides a visual summary of the different ways that we can create reference volumes to divide our particle mass by.

2.1.2.1 Bulk Density

The simplest and most intuitive is the density of many particles in aggregate. This is the density found by pouring the particles in a beaker and measuring their mass and volume. It is commonly referred to as the **bulk** density.

2.1.2.2 Skeletal Density

The next relevant density is that of the pure solid. This is the density you would get if there were a perfect cube of a solid (i.e. with no holes or voids) and divided its mass by the volume of the cube. Since it is impractical to make such a perfect solid, this density is often measured by forcing gas into all the pores of the particles. When the volume of the gas is known, and the volume of the gas-particles mix is known, the volume of the pure solid "skeleton" is simply their difference. The density is therefore the mass of the particles divided by the "skeleton" volume. This density is referred to as the **skeletal** or **solid** density.

The skeletal density is very similar to another density, which includes a particle's internal voids. When the reference volume includes internal voids, the density is referred to as the **apparent** density.

2.1.2.3 Envelope Density

The final density is less intuitive and falls between the bulk and skeletal densities. We find this density by defining a hypothetical "envelope" around each particle; the envelope contains both the solid mass as well as the particle's internal and external pores. The density is then found by dividing the mass of the particles by the volume of their envelopes. This density is troublesome, thanks to the difficulty in selecting and measuring an appropriate envelope; what constitutes appropriate frequently depends on the application. There are some techniques, such as mercury porosimetry, which allow measurement of envelope volumes. We refer to this ratio as the **envelope** or **particle** density.

2.1.2.4 Caveats

Some authors use **true** density to mean **skeletal** density, while others use it to refer to **apparent** density. Some authors (and ISO 80) use **apparent** density to refer to **bulk** density, while others use it to refer to **envelope** density. When gathering data from the literature, vendors, or even your own experiments, great care should be used to ensure that you know exactly what your density is referring to.

2.1.3 Relationships

As might be expected, all these parameters are related to each other and Table 2.1 summarizes their relationships. This text will exclusively use the chemical engineering convention, but the chromatographic convention is presented here for completeness. The source of the difference between the engineering and chromatographic conventions is the definition of the inter-particle porosity, ε_p, where the engineering conventions defines the fraction in terms of the solid volume, while the chromatographic convention defines it in terms of the total system volume.

Table 2.1

Quantity	Symbol	Units	Example Relationships
Total system volume	V_T	$Volume$	For example, in a cylindrical column $V_T = \pi R^2 L$
External void volume	V_X	$Volume$	$V_X = \varepsilon_i V_T$
Internal void volume (Pore volume)	V_P	$Volume$	Chromatographic convention: $V_P = \varepsilon_p V_T$ Engineering convention: $V_P = \varepsilon_p(1-\varepsilon_i)V_T$
Bulk density	ρ_b	$\frac{Mass}{Volume}$	$\rho_b = \frac{M_{particles}}{V_T} = (1-\varepsilon_i)\rho_e = (1-\varepsilon_t)\rho_s$
Skeletal or Solid density	ρ_s	$\frac{Mass}{Volume}$	$\rho_s = \frac{M_{particles}}{V_T - V_X - V_P} = \frac{\rho_b}{1-\varepsilon_t}$
Envelope density	ρ_e	$\frac{Mass}{Volume}$	$\rho_e = \frac{M_{particles}}{V_T - V_X} = \frac{\rho_b}{1-\varepsilon_i}$
External or Inter-particle porosity	ε_i	$\frac{Volume}{Volume}$	$\varepsilon_i = \frac{V_X}{V_T} = 1 - \frac{\rho_b}{\rho_e}$
Internal or Intra-particle porosity	ε_p	$\frac{Volume}{Volume}$	Chromatographic convention: $\varepsilon_p = \frac{V_P}{V_T} = (1-\varepsilon_i) - \frac{\rho_b}{\rho_s}$ Engineering convention: $\varepsilon_P = \frac{V_P}{V_T - V_X} = 1 - \frac{\rho_b}{\rho_s}$ Conversion: $\varepsilon_{p,eng} = \varepsilon_{p,cro} + \varepsilon_i$
Total porosity	ε_t	$\frac{Volume}{Volume}$	$\varepsilon_t = \frac{V_P + V_X}{V_T}$ Chromatographic convention: $\varepsilon_t = \varepsilon_i + \varepsilon_p$ Engineering convention: $\varepsilon_t = \varepsilon_i + \varepsilon_p(1-\varepsilon_i)$

2.1.4 Characterizing the Liquid Phase

Fortunately for us, the liquid phase is better understood and more homogeneous than the solid phase, so the conventions are not quite as diverse.

2.2 The Adsorption Process

Adsorption is a multi-step process by which a molecule moves from the bulk liquid phase and becomes attached to the surface of a solid particle. For our purposes, we will treat this process as involving three distinct regions, as shown in Figure 2.3. Specifically, the adsorption involves:

1. Diffusion of a component from the bulk fluid into the pores of a solid particle
2. The binding of that component to the solid surface inside a pore

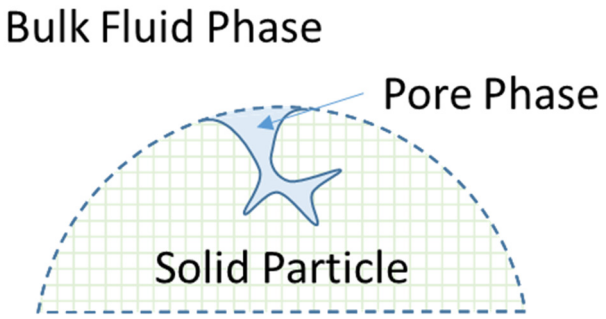


Figure 2.3: The Phases involved in the Adsorption Process

The driving force for the migration of the chemical species into the pore phase is a difference in concentration. This is a very intuitive concept, if the pore phase is poor in a component that the bulk phase is rich in, then that component will be driven to diffuse into the pores. The actual adsorption onto the solid surface involves more complicated concepts, see section 2.3 for more details.

There are more detailed models of this process, for example, we could distinguish between bulk phase and the laminar boundary layer surrounding the solid particle. We could also attempt to describe the motion of the adsorbed components once they have been adsorbed onto the solid surface. We usually do not need this level of detail for the purpose of modeling the adsorptive unit operations.

2.2.1 Mass Transfer Resistance

Having defined where the adsorbed components flow during the adsorption process, we need a concept that allows us to describe how quickly they will move. As we have already established, the difference in concentrations between the pore and bulk phases is the biggest driving force in the system, but there are a number of other possible contributing factors. To address this, the most common technique is to use of a *lumped mass transfer parameter* that lumps all the sources of mass transfer resistance into a single number. This has the nice effect of making the rate of mass transfer into and out of the pore phase directly proportional to the concentration difference, as shown in equation (2.1).

$$Rate\ of\ Mass\ Transfer\ From\ Bulk\ to\ Solid = k_{f,i}(c_{bulk,i} - c_{pore,i}) \qquad (2.1)$$

Where $k_{f,i}$ is the lumped mass transfer parameter, while $c_{bulk,i}$ and $c_{pore,i}$ are the bulk and pore concentrations respectively.

2.2.2 Dead Volume

In both liquid and gas adsorption systems, there are frequently “dead” volumes that do not contain any adsorbent. We need to consider these volumes because they introduce lag times in the movement of material through the systems. The most common sources of dead volumes in both gas and liquid

systems are the *bed head* and the *bed lines*, as shown in Figure 2.4. Each of the multiple beds in a SMB, TSA, or PSA system will have these voids.

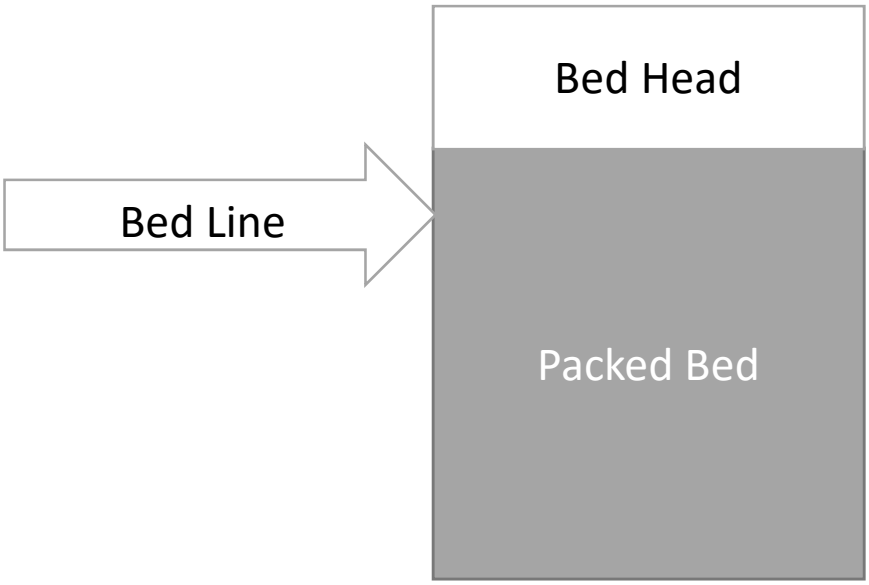


Figure 2.4: The Locations of Dead Volumes in a Packed Bed

2.3 Isotherm Equations

There are a number of equations that are commonly used to describe the relationship between the fluid phase concentration (i.e. the pore phase concentration) and the mass of the components that can become affixed to the solid surface. Many of these relationships are essentially empirical; the equations were originally derived as convenient best-fits. Some, most notably the Langmuir isotherm, are backed by physical models for describing how molecules transition from a solvated state to an adsorbed state. Not surprisingly, the model used to derive the Langmuir isotherm is called the Langmuir Adsorption Model.

2.3.1 The Langmuir Isotherm

The Langmuir isotherm[14], equation (2.4), is the most popular isotherm in adsorptive and SMB applications. Although it was originally derived for ideal gasses (and, in fact, has the ideal gas behavior as an assumption), it turns out that it works well with liquid adsorption. There are a handful of Langmuir variations that are also used, for example the "Dual Site" or "Bi-" Langmuir consists of two basic Langmuir equations summed together, see eq. (2.3). The Bi-langmuir case is equivalent to assuming that the adsorbent has two separate classes of homogeneous adsorption sites.

$$n_i = \frac{a_i c_i}{1 + \sum_{j=1}^{NC} b_j c_j} \tag{2.2}$$

$$n_i = \frac{a_i c_i}{1 + \sum_{j=1}^{NC} b_j c_j} + \frac{\alpha_i c_i}{1 + \sum_{j=1}^{NC} \beta_j c_j} \tag{2.3}$$

In these equations, a_i and b_i are the Langmuir isotherm parameters. The a_i value is related to the Henry's constant in the linear isotherm, since the Langmuir isotherm approaches the linear case as the concentrations become smaller. NC is the number of components in the system, while n_i is the solid loading of component i.

2.3.2 The Linear Isotherm

The linear isotherm is essentially a Henry's law model. It is considered useful in cases of low concentrations and loadings.

$$n_i = H_i c_i \tag{2.4}$$

2.3.3 The Brunauer–Emmett–Teller Isotherm B.E.T. (1938)

This isotherm extends the Langmuir model to work in cases where adsorbed molecules can form layers on the surface of the adsorbent[15]. Relatively recent work[16] found that the appropriate form of the BET equation for use in liquid systems is given by

$$n_i = \frac{q_m K_i^S c_i^{eq}}{\left(1 - K_L c_i^{eq}\right)\left(1 - K_L c_i^{eq} + K_i^S c_i^{eq}\right)} \tag{2.5}$$

In this isotherm, q_m is used to denote the maximum loading of all components.

2.3.4 The Freundlich Isotherm (1906)

An empirical equation originally used on gasses[17], eq (2.6). It has applications in protein and other bio-separations. The model is almost entirely empirical.

$$n_i = h_i c_i^{1/n} \tag{2.6}$$

The competitive form of this isotherm was found by Digiano[18] to be

$$n_i = \frac{h_i^{1/n} c_i}{\left(h_1^{1/n} c_1 + h_2^{1/n} c_2\right)^{1-n}} \tag{2.7}$$

When all components have the same value of the freundlich exponent, n.

2.3.5 The Sips (Langmuir-Freundlich) Isotherm (1948)

This model is an attempt to put the Freundlich isotherm into a more physically meaningful form. At low concentrations, it mimics Freundlich behavior, while at high concentrations it predicts a monolayer like the Langmuir isotherm.

$$n_i = \frac{a_i c_i^n}{1 + \sum_{j=1}^{NC} b_j c_j^n} \tag{2.8}$$

2.3.6 The Toth Isotherm (1963)

This empirical isotherm is primarily used in gas-phase adsorption, and is similar in form to the Sips equation. Nevertheless, it has found some application in liquid chiral separations[19]. This isotherm assumes that the adsorption surface is heterogeneous, with a continuous adsorption energy distribution.

$$n_i = \frac{h_i c_i}{(a_i + c_i)^{1/t}} \tag{2.9}$$

The competitive form of this isotherm was found be

$$n_i = \frac{q_s b_i c_i}{\left(1 + \left(\sum_{j=1}^{NC} b_j c_j\right)^t\right)^{1/t}} \tag{2.10}$$

2.3.7 Summary

Figure 2.5 shows what these isotherms look like in the single component case.

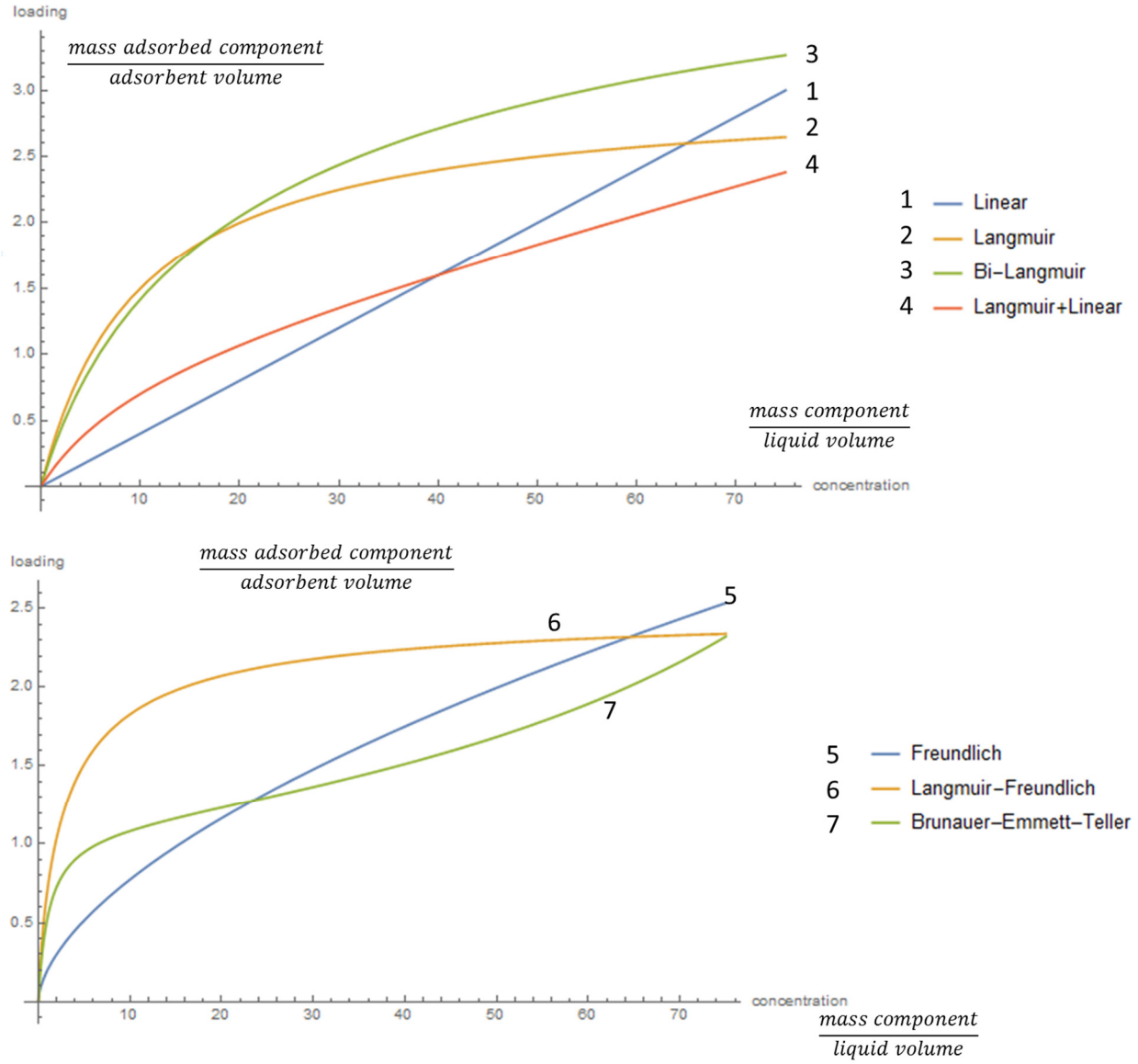


Figure 2.5: Sample Isotherm Curves

2.4 Mass Balances

Using the concepts and nomenclature developed in the previous sections, we can start discussing the mathematics of an adsorption bed. The obvious first step towards that end is to perform a mass balance on the system. Let us start by considering the case of a countercurrent bed, where the solid and liquid are both moving. A fixed bed will be a special case of this scenario, with the solid velocity equal to zero.

A column with solid particles moving downwards and the liquid moving upwards can be represented by Figure 2.6. $\dot{M}_L$ is the mass flow rate of the liquid phase, while $\dot{M}_s$ is the mass flow rate of the solid phase. Keep in mind that "the solid phase" refers to both the solid particles and the liquid entrained in the particle pores. The overall steady state mass balance is therefore

$$\left[\dot{\boldsymbol{M}}_{\boldsymbol{L}}(\boldsymbol{z}=\boldsymbol{0})+\dot{\boldsymbol{M}}_{\boldsymbol{s}}(\boldsymbol{z}=\boldsymbol{L})\right]-\left[\dot{\boldsymbol{M}}_{\boldsymbol{L}}(\boldsymbol{z}=\boldsymbol{L})+\dot{\boldsymbol{M}}_{\boldsymbol{s}}(\boldsymbol{z}=\boldsymbol{0})\right]=\boldsymbol{0} \tag{2.11}$$

Unfortunately, without further assumptions, this mass balance is not very useful. We establish the following definitions:

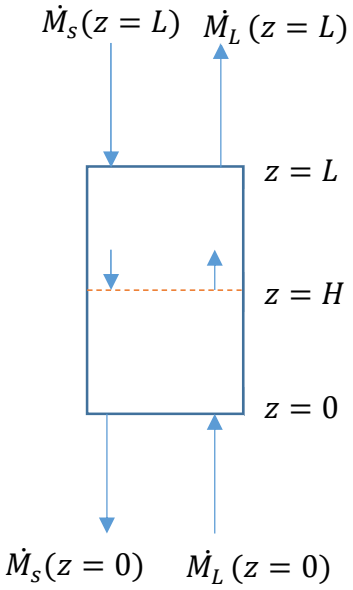


Figure 2.6: Diagram of a Countercurrent Adsorber

1. Volumetric flow rates will be denoted $\dot{Q}_L$ for the liquid phase and $\dot{Q}_s$ for the solid phase.
2. Loading, which has the symbol q, is the mass of the liquid adsorbed per unit volume of the solid phase.
3. Concentration, which has the symbol C, is the mass of the liquid per unit volume in the liquid phase.
4. Linear velocities will be referred to as v for the liquid system, and u_s for the solid system.

Assuming the following allows more useful statements about the system:

1. The volumetric flow rate of the solid phase is constant, and the actual "solids" part of the solids phase has a constant density, ρ_e
2. The cross sectional area of the column, A, is constant.
3. The fraction of the column's volume that is comprised of the upward-moving liquid is constant and equal to the external porosity, ε_i
4. The superficial (or interstitial, or linear) velocity in the liquid phase is given by $v=\frac{\dot{Q}_L}{A\varepsilon_i}$ this can also be thought of as assuming that the bulk porosity value holds true at differential length scales.

Thanks to definitions 1 and 3, we can immediately write:

$$\dot{\boldsymbol{M}}_{\boldsymbol{L}} = \boldsymbol{C}\dot{\boldsymbol{Q}}_{\boldsymbol{L}} \tag{2.12}$$

And definition 2 gives:

$$\dot{\boldsymbol{M}}_{\boldsymbol{s}} = \boldsymbol{q}\dot{\boldsymbol{Q}}_{\boldsymbol{s}} + \boldsymbol{\rho}_{\boldsymbol{s}}\dot{\boldsymbol{Q}}_{\boldsymbol{s}} = (\boldsymbol{q} + \boldsymbol{\rho}_{\boldsymbol{e}})\dot{\boldsymbol{Q}}_{\boldsymbol{s}} \tag{2.13}$$

Assumptions 2-4 allow us to substitute $vA\varepsilon_i$ for $\dot{Q}_L$ and $u_s A(1-\varepsilon_i)$ for $\dot{Q}_S$ to get:

$$\dot{\boldsymbol{M}}_{\boldsymbol{L}} = \boldsymbol{C}\dot{\boldsymbol{Q}}_{\boldsymbol{L}} = \boldsymbol{CvA\varepsilon_i} \tag{2.14}$$

$$\dot{\boldsymbol{M}}_{\boldsymbol{s}} = \boldsymbol{q}\dot{\boldsymbol{Q}}_{\boldsymbol{s}} + \boldsymbol{\rho}_{\boldsymbol{s}}\dot{\boldsymbol{Q}}_{\boldsymbol{s}} = (\boldsymbol{q} + \boldsymbol{\rho}_{\boldsymbol{e}})\boldsymbol{u}_{\boldsymbol{s}}\boldsymbol{A}(\boldsymbol{1} - \boldsymbol{\varepsilon}_{\boldsymbol{i}}) \tag{2.15}$$

Our overall mass balance is therefore:

$$\begin{aligned}[\boldsymbol{CvA\varepsilon_i}]_{\boldsymbol{z=0}} + [(\boldsymbol{q} + \boldsymbol{\rho}_{\boldsymbol{e}})\boldsymbol{u}_{\boldsymbol{s}}\boldsymbol{A}(\boldsymbol{1} - \boldsymbol{\varepsilon}_{\boldsymbol{i}})]_{\boldsymbol{z=L}} \\ = [\boldsymbol{CvA\varepsilon_i}]_{\boldsymbol{z=L}} + [(\boldsymbol{q} + \boldsymbol{\rho}_{\boldsymbol{e}})\boldsymbol{u}_{\boldsymbol{s}}\boldsymbol{A}(\boldsymbol{1} - \boldsymbol{\varepsilon}_{\boldsymbol{i}})]_{\boldsymbol{z=0}}\end{aligned} \tag{2.16}$$

We know that A, ε_i, and ρ_e are not functions of z, so their values are the same at $z = 0$ and $z = L$ and we can treat them as constants. It is also generally assumed that the solid velocity u_s is constant.

$$\begin{aligned}\boldsymbol{A\varepsilon_i}[\boldsymbol{Cv}]_{\boldsymbol{z=0}} + \boldsymbol{u}_{\boldsymbol{s}}\boldsymbol{A}(\boldsymbol{1} - \boldsymbol{\varepsilon}_{\boldsymbol{i}})\boldsymbol{\rho}_{\boldsymbol{e}} + \boldsymbol{u}_{\boldsymbol{s}}\boldsymbol{A}(\boldsymbol{1} - \boldsymbol{\varepsilon}_{\boldsymbol{i}})[\boldsymbol{q}]_{\boldsymbol{z=L}} \\ = \boldsymbol{A\varepsilon_i}[\boldsymbol{Cv}]_{\boldsymbol{z=L}} + \boldsymbol{u}_{\boldsymbol{s}}\boldsymbol{A}(\boldsymbol{1} - \boldsymbol{\varepsilon}_{\boldsymbol{i}})\boldsymbol{\rho}_{\boldsymbol{e}} + \boldsymbol{u}_{\boldsymbol{s}}\boldsymbol{A}(\boldsymbol{1} - \boldsymbol{\varepsilon}_{\boldsymbol{i}})[\boldsymbol{q}]_{\boldsymbol{z=0}}\end{aligned} \tag{2.17}$$

Simplifying by dividing through by $A\varepsilon_i$ and removing the $u_s A(1-\varepsilon_i)\rho_e$ term which appears on both sides:

$$[\boldsymbol{Cv}]_{\boldsymbol{z=0}} + \boldsymbol{u}_{\boldsymbol{s}}\frac{(\boldsymbol{1} - \boldsymbol{\varepsilon}_{\boldsymbol{i}})}{\boldsymbol{\varepsilon}_{\boldsymbol{i}}}[\boldsymbol{q}]_{\boldsymbol{z=L}} = [\boldsymbol{Cv}]_{\boldsymbol{z=L}} + \boldsymbol{u}_{\boldsymbol{s}}\frac{(\boldsymbol{1} - \boldsymbol{\varepsilon}_{\boldsymbol{i}})}{\boldsymbol{\varepsilon}_{\boldsymbol{i}}}[\boldsymbol{q}]_{\boldsymbol{z=0}} \tag{2.18}$$

Before the final simplification, we can assume the overall liquid concentration, is constant. This is equivalent to assuming that the components being adsorbed are only present in small quantities. An immediate consequence is that the superficial velocity, v, is constant. Our final equation is therefore:

$$\boldsymbol{v}(\boldsymbol{C}_{\boldsymbol{L}} - \boldsymbol{C}_{\boldsymbol{0}}) = \boldsymbol{u}_{\boldsymbol{s}}\frac{(\boldsymbol{1} - \boldsymbol{\varepsilon}_{\boldsymbol{i}})}{\boldsymbol{\varepsilon}_{\boldsymbol{i}}}(\boldsymbol{q}_{\boldsymbol{L}} - \boldsymbol{q}_{\boldsymbol{0}}) \tag{2.19}$$

2.5 Derivation of the Differential Equations

Now that we have established some background in equations used in adsorption, let us consider flow in packed beds, with adsorption.

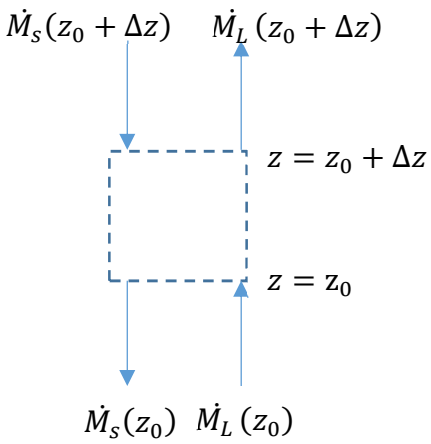


Figure 2.7: A Differential Column Element

Starting with a differential volume element of the column, and write the non-steady state mass balance.

$$Accumulation = In - Out$$

As Figure 2.7 illustrates, the net mass flow in is $\left[\dot{M}_L(z) + \dot{M}_s(z + \Delta z)\right]$ so in a short period of time, the total mass entering would be $\left[\dot{M}_L(z) + \dot{M}_s(z + \Delta z)\right]\Delta t$ Likewise, the total mass leaving the system in that time period would be $\left[\dot{M}_L(z + \Delta z) + \dot{M}_s(z)\right]\Delta t$ and we can write:

$$M_{t+\Delta t} - M_t = \left[\dot{M}_L(z) + \dot{M}_s(z + \Delta z)\right]\Delta t - \left[\dot{M}_L(z + \Delta z) + \dot{M}_s(z)\right]\Delta t \quad (2.20)$$

We can now substitute the relationships we found in the mass balance section to find that

$$M_{t+\Delta t} - M_t = A u_s(1 - \varepsilon_i)(q_{z+\Delta z} - q_z)\Delta t - A\varepsilon_i v(C_{z+\Delta z} - C_z)\Delta t \quad (2.21)$$

The mass of our differential volume section can be written $M = C\varepsilon_i A\Delta z + q(1 - \varepsilon_i)A\Delta z$ so the change over time can be written:

$$M_{t+\Delta t} - M_t = (C_{t+\Delta t} - C_t)\varepsilon_i A\Delta z + (q_{t+\Delta t} - q_t)(1 - \varepsilon_i)A\Delta z \quad (2.22)$$

Equations (2.20) and (2.21) have equal left-hand sides, we can therefore write:

$$\begin{aligned}(C_{t+\Delta t} - C_t)\varepsilon_i A\Delta z + (q_{t+\Delta t} - q_t)(1 - \varepsilon_i)A\Delta z \\ = A\, u_s(1 - \varepsilon_i)q_z\Delta t - A\, v\, (C_{z+\Delta z} - C_z)\Delta t\end{aligned} \quad (2.23)$$

Dividing both sides of equation (2.23) by $A\varepsilon_i\Delta t\Delta z$ and we end up with

$$\frac{(C_{t+\Delta t} - C_t)}{\Delta t} + \frac{(1 - \varepsilon_i)}{\varepsilon_i}\frac{(q_{t+\Delta t} - q_t)}{\Delta t} = u_s\frac{(1 - \varepsilon_i)}{\varepsilon_i}\frac{(q_z - q_{z-\Delta z})}{\Delta z} - \frac{v}{\varepsilon_i}\frac{(C_{z+\Delta z} - C_z)}{\Delta z} \quad (2.24)$$

By taking the limit as Δz and Δt go to 0, equation (2.24) becomes this PDE:

$$\frac{\partial C}{\partial t} + \frac{1 - \varepsilon_i}{\varepsilon_i}\frac{\partial q}{\partial t} = u_s\frac{1 - \varepsilon_i}{\varepsilon_i}\frac{\partial q}{\partial z} - v\frac{\partial C}{\partial z} \quad (2.25)$$

This differential equation does not include an axial dispersion term. It represents limit of TMB operation as axial dispersion approaches 0. A derivation of the full differential equations is not provided here, but

if we were to include mass transfer and dispersive effects, we would actually be left with a pair of differential-algebraic equations:

$$\frac{\partial C}{\partial t} = E_i \frac{\partial^2 C}{\partial z^2} - v\frac{\partial C}{\partial x} - k\frac{(1-\varepsilon_i)}{\varepsilon_i}(q - q^{eq}) \tag{2.26}$$

$$\epsilon_p \frac{\partial C^*}{\partial t} + \left(1-\epsilon_p\right)\frac{\partial q}{\partial t} = k\frac{(1-\varepsilon_i)}{\varepsilon_i}(C - C^*) + v\epsilon_p \frac{\partial C^*}{\partial x} + (1-\epsilon_p)v\frac{\partial q}{\partial x} \tag{2.27}$$

2.6 Differential Equations

Liquid and gas adsorption are mathematically very similar. As seen in the previous section, the case of liquid adsorption has the advantage of incompressibility, which allows some simplifications to the differential equations. Indeed, with some simplifying assumptions, liquid chromatography can be described with PDE's that we can work with analytically instead of numerically.

Every lab group seems to have its preferred rearrangement of the well-understood differential equations governing chromatographic separation in a moving bed, i.e. equations (2.26) and (2.27). Generally, the equations can be put in the form

$$\partial_t c_i = E_i \partial_{z,z} c_i - u_0 \partial_z c_i + k\frac{(1-\epsilon)}{\epsilon}\partial_z q_i \tag{2.28}$$

$$v(\epsilon + (1-\epsilon)k)\partial_z q_i = -k(h_i c_i - q_i) \tag{2.29}$$

To proceed beyond this point, we have some choices to make. It is possible to numerically approximate the solutions to these algebraic PDEs using finite element and difference methods. This is automatically done for us in commercial software (e.g. Aspen Adsorption) or we could locate and compile an appropriate Fortran PDE solver, (e.g. COLNEW.) The specific solution, used in many commercial solvers, is to discretize the space dimension, then integrate numerically through time. With some simplifications, it is possible to perform the numerical simulations relatively simply in tools such as Matlab.

However, attempts to find analytic solutions typically proceeds in one of two ways:

A. Assume that the loading is always a function of concentration: $q = f(C)$ This is an *equilibrium* assumption, equivalent to assuming an instantaneous mass transfer. The result is the linear first order PDE shown in equation (2.30) that can be solved with the method of characteristics.

$$\partial_t c_i + \partial_z (c_i + v n_i) = 0 \tag{2.30}$$

B. Assume the system is at steady state and the $\partial/\partial t$ terms drop out of the equations. This results in a system of 2nd order ordinary differential equations (ODE) that can be solved with a variety of methods.

From method A, we learn about the *wave velocities* in the system. That is to say, we can know how quickly components will move through each zone in the system. That information allows us to put constraints on the flow rates in each zone.

In this work, we will provide a brief introduction of both analytic solutions methods, as well as a discussion of how to use commercial simulators to solve these equations numerically. The analytic solutions are found with the help of the Mathematica software package, and numerical solutions are calculated with various packages from Aspen Technology.

2.7 Analytic solutions: The Method of Characteristics

The method of characteristics is a handy method for solving PDEs of the form

$$A(z,t)\partial_t c + B(z,t)\partial_z c = C(z,t) \tag{2.31}$$

Specifically, the method asks us to find parametric curves, $(z(s), t(s))$ with parameter s such that

$$\frac{dt}{ds} = A(z(s), t(s)) \tag{2.32}$$

$$\frac{dz}{ds} = B(z(s), t(s)) \tag{2.33}$$

$$\frac{dc}{ds} = C(z(s), t(s)) \tag{2.34}$$

Now we will notice that we can, in principle, solve equations (2.32) to (2.34) to find three distinct equations:

$$t(s) = \int A(z(s), t(s))ds \tag{2.35}$$

$$z(s) = \int B(z(s), t(s))ds \tag{2.36}$$

$$c(s) = \int C(z(s), t(s))ds \tag{2.37}$$

We can subsequently combine the equations with their initial conditions and eliminate s completely. Specifically, we know that $c(z,t) = c(s)$ and $s = g(z,t)$ we will end up with a unique solution of the form

$$c(z,t) = f\big(g(z,t)\big) \tag{2.38}$$

This process is examined in more detail in chapter 3

2.8 Steady State vs Cyclic Steady State

It is common to want to define chemical unit operations in terms of a steady state. That is to say, the state where the system performance no longer changes over time. However, in this work, we are considering systems which never attain a steady state, but rather a cyclic steady state. That means the system will eventually achieve a state that is periodic over time. In commercial software, this is typically achieved through the use of a "Cycle Organizer," which we will discuss in Chapter 4.

2.9 SMB Chromatography

2.9.1 Introduction

Chromatographic separations are a powerful technology, and have been in use for a long time. However, such separations were traditionally carried out in batches, which limited their efficiency. The search for a solid-liquid separation method that would eliminate the inefficiencies of batch operation by working as an analogue of an absorber/stripper came to an end in 1961, when Broughton and Gerhold[20] developed the concept of a Simulated Moving Bed and found applications in petrochemical and sugar separations. The SMB concept allowed a column to approximate countercurrent solid/liquid separation, without the need for solid-handling equipment.

The complexity and novelty of the approach restricted its use to high-value applications for a long time, but this has changed as understanding has improved. There has been a surge in interest in applying SMB to much smaller-scale problems in pharmaceutical, biochemical, and chiral separations. Depending on the economics of the separations, SMB can offer a number of significant benefits over a more conventional batch chromatographic separation, from reduced desorbent consumption and lower adsorbent requirements, to higher productivity or recovery.

2.9.2 A Model System

As explained in the introduction, the SMB concept was developed to implement a true countercurrent separator. To learn about these systems, lets consider the classic case of a binary separation. To perform such a separation with a true moving bed separation (TMB), we would configure our column as shown in Figure 2.8.

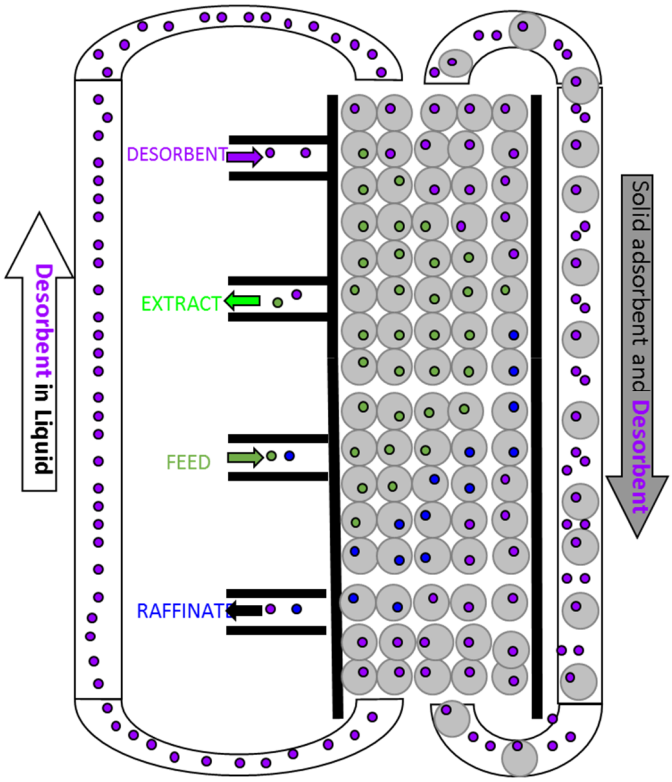


Figure 2.8: Diagram of a Basic 4-zone True Moving Bed

In a TMB unit, the liquid and solid phases continuously flow in the opposite direction. Figure 2.8 shows the two required inlet streams (feed and desorbent), and two outlet streams (extract and raffinate). This configuration would allow for the attainment for high-purity products even when using a low-selectivity adsorbent. However, from an engineering point of view, the use of this type of equipment with the actual movement of the stationary phase is not practical for industrial implementation. This complex solid handling may lead to some technical problems, namely, equipment abrasion, mechanical erosion of adsorbent, and difficulties in maintaining plug flow for the solid (especially in beds with large diameters).

The key insight that allows engineers to sidestep these drawbacks is to realize that this separation is possible because of the relative motion of the solid and the ports. However, we do not actually need to move the solids to create this relative motion, we can move the ports instead, as shown in Figure 2.9. Ideally we could move the ports continuously, and perfectly replicate a TMB. If we only switch the port locations at discrete times and positions, we can *simulate* TMB performance. The finer in space and time we make these steps, the closer we will approximate TMB performance. In practice, this means that we can more closely approximate TMB performance by creating more possible locations for the ports to switch to.

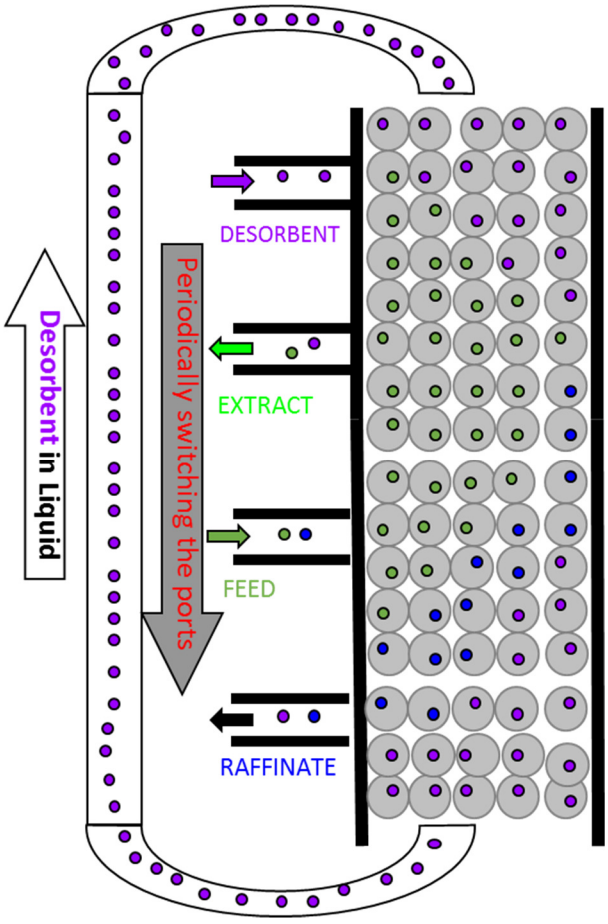


Figure 2.9: Diagram of 4-zone Simulated Moving Bed Operation

The four streams in Figure 2.8 and Figure 2.9 divide the column into four regions that have different roles in the separation. If we assume that component A is attracted to the solid phase more strongly

than component B, our separator can achieve the state described in Figure 2.10. The conditions in zones 2-4 are such that component A will get adsorbed onto the solid phase, and moved upwards; but in zone 1, the desorbent is capable of regenerating the solid phase by desorbing component A. The conditions in zones 1-3 are such that component B will not be significantly adsorbed, and move down the column. In zone 4, the solvent will be regenerated and B will move back up the column with the solid. Figure 2.10 also introduces the concept of zone flowrates, Q_i, that denote the liquid flow rate in each zone, i.

In SMB chromatography, it Is typical to make what is called the **trace liquid assumption** that states the density of the liquid is not related to the concentrations of the components being separated, and therefore the fluid flow rate in each zone is constant. This is valid when

- The components being separated have a similar density to the solvent
- The components being separated are present in low (i.e. trace) concentrations
- The fluid is incompressible

These properties allow us to treat the entire SMB unit as though volume is conserved; we can do volume balances instead of mass balances if we so choose.

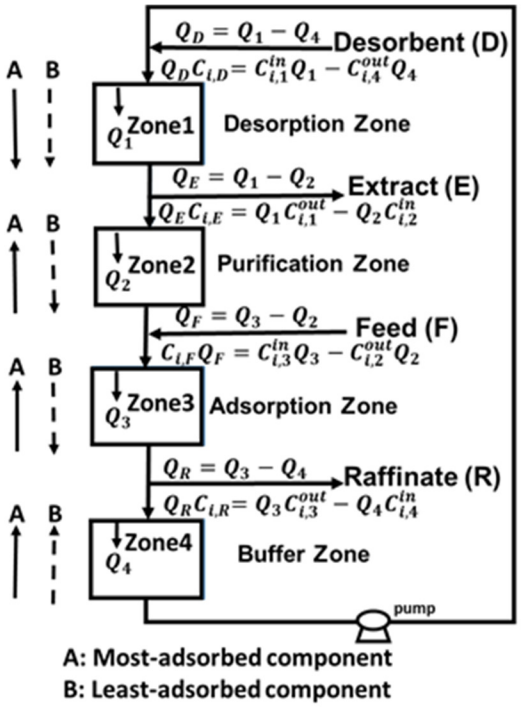


Figure 2.10: Net Motion of the components in a 4-zone system

We discuss the balances and constraints required to understand the SMB system's design in the next section.

2.10 Overview of the Computer Tools

This textbook will provide some instruction use of two powerful pieces of software. We assume that the reader has some familiarity with basic office tools, such as Microsoft Excel.

2.10.1 Aspen Custom Modeler

Aspen Custom Modeler is a platform technology sold by Aspen Tech. It allows us to develop detailed numerical simulations involving dynamic operation, process control, and creation of custom process models. Aspen provides a number of packages built on top of this platform that contain domain specific models for ease of use. For example, Aspen Chromatography has a large number of built-in models dealing with liquid solid separation, including a basic SMB system. All of these models are accessible through a flowsheet-style interface.

We use Aspen's technology to quickly develop highly rigorous simulations of SMB processes, and extend the default models to handle more complex operation, such as the operational modes discussed in Chapter 6.

2.10.2 Mathematica

Mathematica is an advanced computer algebra system that was originally released in 1988. Produced by a company called Wolfram Research, Mathematica makes many seemingly complex mathematical tasks accessible to "mere mortals." Unlike Apsen, you do not need a Mathematica license to use the Mathematica content in this work; Mathematica's "Computational Document Format" is viewable with a free program called the "CDF Player" and is available on the Wolfram website, at https://www.wolfram.com/cdf-player/ . Some installation instructions are provided in this work.

Some of Mathematica's features are explained here, so that the reader can get some sense of why it is a valuable program to use in the context of chromatographic separation.

2.10.2.1 Powerful Math System

Mathematica is a powerful computer algebra system. It allows the user to solve a wide variety of mathematical problems, both numeric and symbolic. It also has a wide variety of built-in functions that make solving engineering problems relatively straightforward. Specifically, Mathematica has powerful differential equation solving abilities, it can solve a wide variety of ODEs and PDEs. In addition, Mathematica has the ability to do a wide variety of symbolic manipulations, matrix manipulations, and optimizations.

2.10.2.2 Functional-style Programming

Mathematica is not intended to be used in the same way as a language like FORTRAN. In Mathematica, the user is encouraged to think in terms of defining and applying mathematical functions. For example, suppose we had a list of numbers that we wanted to take the square root of. In a traditional language like FORTRAN, we would immediately set up a for loop that would select each element of the list, find its square root, and put it back in the list. In Mathematica, however, we can *Map* the square root function to each element in the list in one step, and the result is a *new* list with all the square roots. Mathematica does not need us to write out an explicit "for" loop. More generally, if we have some function "f" and some list {a,b,c,d,e}, *Mapping* (the /@ operator below) applies f to each element of the list.

```
Input:  f/@{a,b,c,d,e}
Output: {f[a],f[b],f[c],f[d],f[e]}
```

In most cases, using explicit loops (e.g. do, while, for) is not the right way to write Mathematica code. One other important and relevant example of this is convergence. When we have a problem that we

want to solve by iterating to convergence, the preferred way to do this in Mathematica is to "nest" a function. Essentially, you want to define your iteration scheme as a function that takes the previous iteration as input, and outputs the next iteration. If your iteration procedure is f, and the first guess of your solution is x, then three iterations looks like:

```
Input:  Nest[f,x,3]
Output: f[f[f[x]]]
```

If you want to continue to apply f until the output no longer changes (i.e. it converges), Mathematica has provided us with the "FixedPoint" function which applies f until the output no longer changes. For example, if our function was f, and our initial guess, then FixedPoint would calculate:

```
FixedPoint[f,x]
x
f[x]
f[f[x]]
f[f[f[x]]]
f[f[f[f[x]]]]
…
```

And it would stop once it found an instance where f[y]=y or the iteration limit is reached.

2.10.2.3 Easy Creation of User Interface

One final noteworthy feature of Mathematica is the ease by which we can create simple user interfaces. Specifically, user interfaces that involve changing numerical values with sliders. These capabilities have direct and obvious application to the design methods discussed in chapter 2. For example, it is very easy to create a graph representing the triangle theory separation region, and allow the user to adjust the isotherm parameters.

3 Simulated Moving Bed Design Methods

3.1 Introduction to SMB

3.1.1 Key Features and Terms

As covered in the introduction, the SMB process is a complex, but practical implementation of a separator whose goal is to approximate the performance of a unit where the solid and the liquid move countercurrent to each other. To understand the SMB process, we need to be very clear about the use of the following terms.

3.1.1.1 Zones

A "Zone" in a SMB unit is the set of fixed beds between two ports. The convention is to start numbering the zones from the desorbent port in the direction of liquid flow. Be very careful when reading literature published prior to the 1990s because the convention had not yet been established and they may number the zones differently. Each zone may contain many *packed beds*, and the zones may have different numbers of beds. For example, in Figure 3.2, zone 2 may always consist of 2 packed beds, while zone 1 may consist of only one packed bed.

3.1.1.2 True Moving Bed Vs Simulated Moving Bed

As we discussed briefly in chapter 1, a "Simulated" moving bed is a more practical implementation of a "True" moving bed. Figure 3.1 shows the differences between the two configurations. It turns out that the key feature required to achieve the continuous chromatographic separation is the relative motion of the port position and the solid phase. A TMB unit achieves that relative motion by moving the solid, while a SMB unit achieves the relative motion by moving the ports.

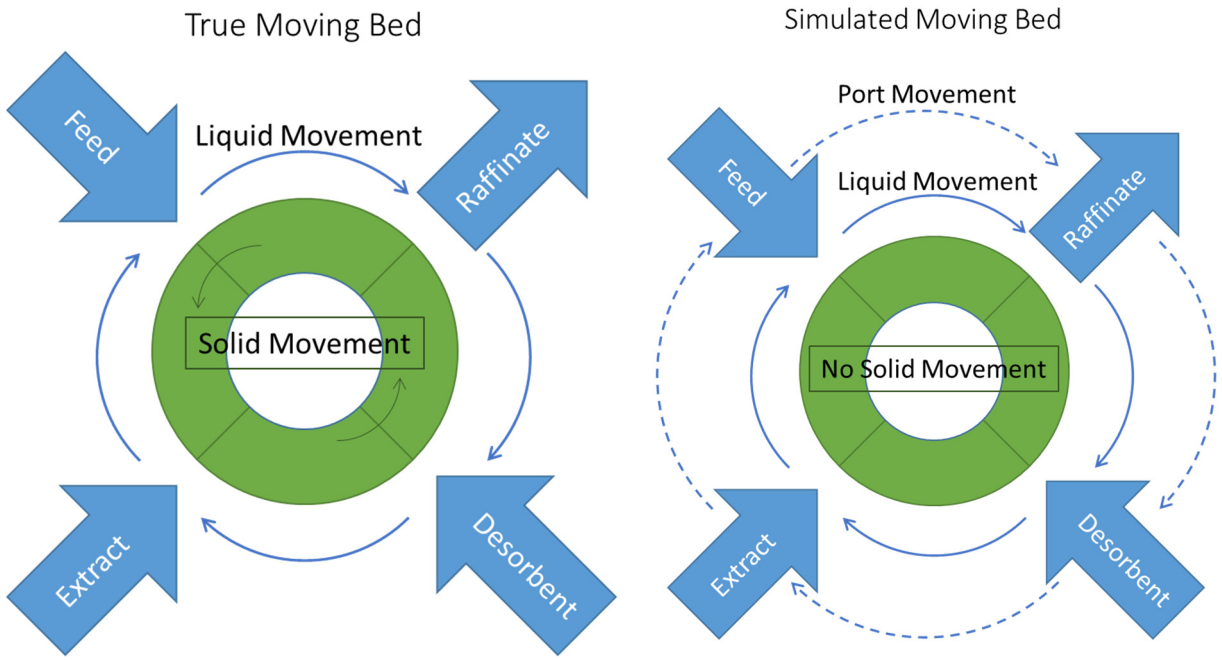


Figure 3.1: True Moving Bed (Left) compared to a Simulated Moving Bed (Right)

The process can be thought of in the following way. As the concentration waves in the liquid phase move through the fixed beds, there are two possible ways to continue feeding and extracting material in the right places. Either we can move the solid phase at the right speed to keep the waves stationary, or we can move the ports at the right speed to keep up with the concentration waves.

Because it is not feasible to move the ports continuously, the ports in a SMB unit are moved at discreet time intervals. It is this discontinuous action that makes the SMB case only an approximation of the TMB case. If it were possible to move the ports continuously, the TMB and SMB would be completely equivalent. In fact, the more fixed beds there are in each zone, the more fine-grained the switches can be; this means that we can make the SMB case more closely approximate the TMB case by adding more port positions by adding more beds.

3.1.1.3 Flow Rate Ratios

In each zone, it turns out that the key parameter to characterize the relative movement of the solid and liquid are the flow rate ratios. These are defined differently, depending on if the system is a true moving bed, or a simulated moving bed. The specific equations are as follows:

TMB
$$m_j = \frac{(Q_j - \varepsilon_p Q_s)}{Q_s(1-\varepsilon_p)} \quad (3.1)$$

SMB
$$m_j = \frac{(Q_j t_{sw} - \varepsilon^* V_{bed})}{V_{bed}(1-\varepsilon^*)} \quad (3.2)$$

The key benefit of using these ratios is that they allow us to flip back and forth between TMB and SMB models. Because the equations are different, the exact flow rates will be different, but they will be have the same theoretical performance. That is to say, if an SMB unit has the same m-values as a TMB unit, the SMB unit will be *simulating* the TMB unit, and the TMB unit represents, in some sense, a bound on the SMB's performance. If the SMB unit is underperforming its TMB counterpart, adding more beds to each zone will cause the SMB unit's performance to get closer to the theoretical TMB performance. Note that this is only strictly true in the case of basic SMB systems; units that employ the advanced operational modes described in Chapter 6 no longer strictly represent approximations of TMB units.

3.1.1.4 Performance Indices

The two numbers used to describe SMB performance are purity and recovery, which are specified in Equations (3.3) through (3.6). These conceptually describe the quality of the separation, and perfect separation would produce streams with 100% purity and recovery.

$$purity_i = \frac{c_i}{\sum_{j=1}^{nc} c_j} \tag{3.3}$$

$$recovery_i = \frac{Q_{product,i}}{Q_{feed,i}} \tag{3.4}$$

To describe the *efficiency* of the separation, it is common to use the following measures of productivity, which quantify the volume of feed that can be processed on a per-desorbent basis, or a per-adsorbent basis.

$$productivityA = \frac{Q_{feed}}{Q_{desorbent}} \tag{3.5}$$

$$productivityB = \frac{Q_{feed}}{V_{col}} \tag{3.6}$$

3.2 General Concepts

3.2.1 The Design Problem

In a perfect system, we would be able to feed in a stream containing two components dissolved in a solvent, and get two product streams, each stream containing one of the components in the same solvent. Essentially, we would want the situation depicted in Figure 3.2. The most adsorbed component travels in the direction of the solid stream, to the Extract. The least adsorbed component travels in the direction of the liquid stream towards the Raffinate.

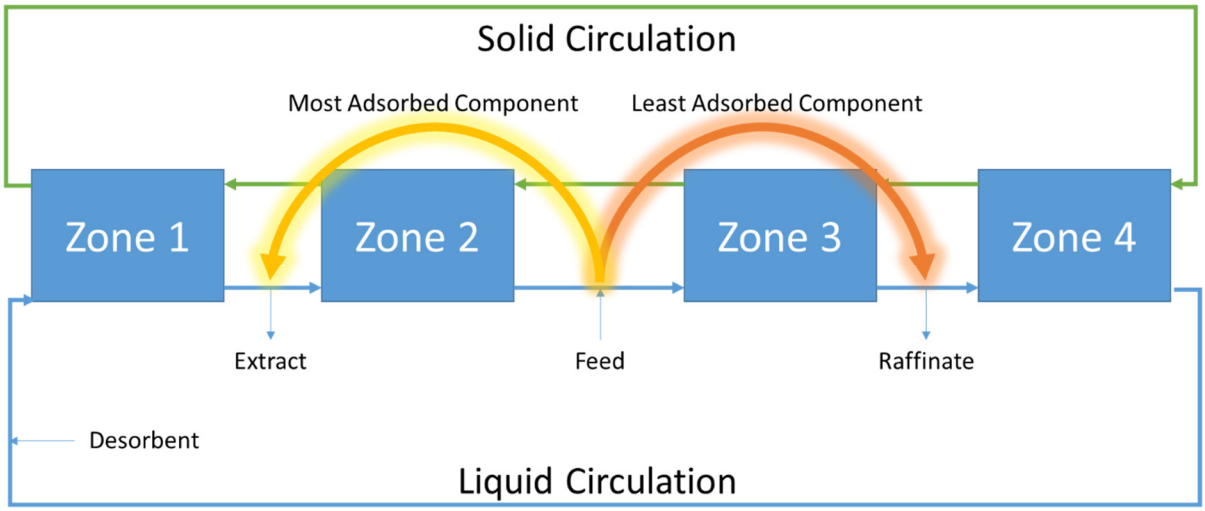


Figure 3.2: Ideal Separation Case

In order to achieve this ideal scenario, each zone has a role to play. These roles are depicted in Figure 3.3.

Zone 1: Component 2 is desorbed from the solid adsorbent into the liquid phase. This ensures that all of component 2 is present in the extract by preventing the solid phase from carrying it through to zones 4 and 3

Zone 2: Component 1 is desorbed from the solid adsorbent into the liquid phase. This ensures that there will not be any of this component in zone 1 and therefore the extract will be uncontaminated.

Zone 3: Component 2 is adsorbed into the solid phase. This means the liquid will only contain component 1 so the raffinate will remain uncontaminated.

Zone 4: The remainder of component 1 is adsorbed onto the solid. This ensures that component 1 will not be present in the liquid recycle, and will therefore not contaminate zone 1 and the extract.

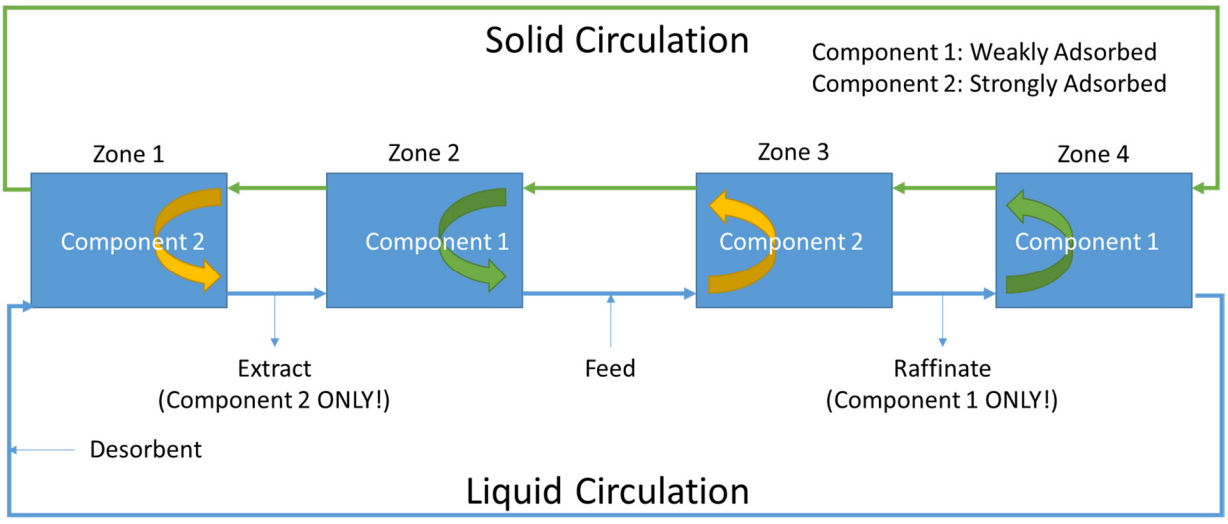


Figure 3.3: Zone Roles in TMB separation

Given those roles and our goals for the separation, we can assign variables as shown in Figure 3.4. Each stream is characterized by either a pair of concentrations (c_1, c_2) or solid loadings (w_1, w_2) and we know that some streams have zero concentrations.

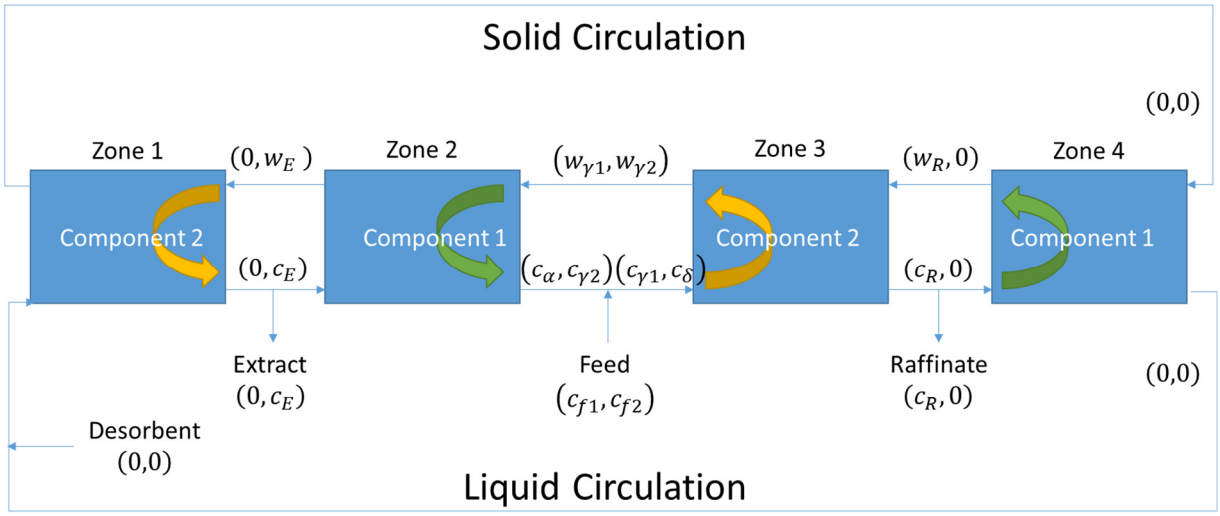


Figure 3.4: SMB Unit under Ideal Separation Conditions

In an ideal binary chromatographic system, the two components introduced in the feed stream would be perfectly separated into the extract and raffinate streams. Mass transfer resistance would be absent, and the liquid phase would instantly reach equilibrium with the solid adsorbent. Under such conditions, we achieve the ideal situation we described in Figure 3.2. The net effect is that component 1 will travel through zone 3 in the liquid phase to be removed in the raffinate stream. Component 2, on the other hand, will move through zone 2 to be removed in the extract stream. In a real separation unit, the concentration profile that would achieve this outcome can look qualitatively like Figure 3.5.

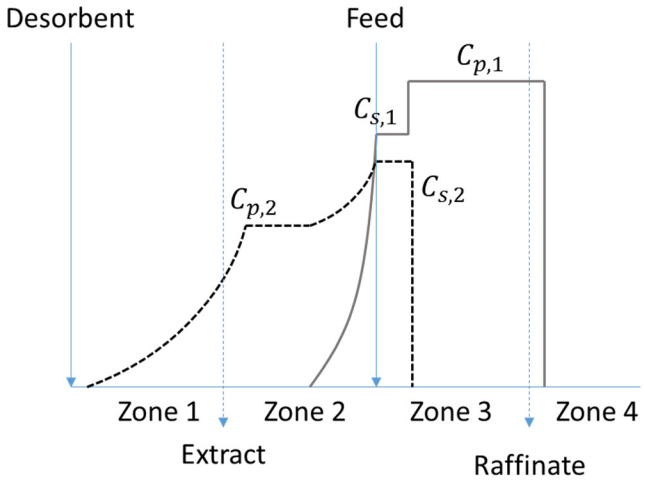


Figure 3.5: SMB Concentration Profile under Ideal Separation Conditions

Given the background we have discussed so far, the design problem now boils down to the question: For a given system configuration (chemical choices, adsorbent choices, column sizes) what set of flow rates allows us to achieve the ideal separation?

3.2.2 SMB vs TMB

It is important to note that so far we have been discussing the TMB case, in which the solid phase actually circulates. The reason for this is that the TMB case reaches an actual steady state, while the SMB case only ever reaches a cyclic steady state. The algebra in the case of a proper steady state is both simpler and easier to understand than the moving bed case. As we discussed previously, the number of beds in each zone determines how closely the real SMB model approximates the TMB case.

We must be careful, however, because the raw flow rates required by the TMB and SMB cases are related, but different. Specifically, as we will see, they are related through the flow rate ratios, m. In other words, the m values are the same between TMB and SMB, so the goal of the design equations derived from the TMB case are to calculate the operating conditions in terms of these flow rate ratios.

$$m_j = \frac{\left(Q_j^{TMB} - \varepsilon_p Q_s\right)}{Q_s\left(1-\varepsilon_p\right)} = \frac{\left(Q_j^{SMB} t_{sw} - \varepsilon^* V_{bed}\right)}{V_{bed}\left(1-\varepsilon^*\right)} \tag{3.7}$$

The flow rate ratios in each zone are not enough to completely satisfy all the degrees of freedom in the system. This is shown in Figure 3.6, with the final degree of freedom from a pressure drop constraint. In practice, the pressure drop constraint is generally a nuanced parameter to pin down. In this work we assume a whole-column pressure drop, but there are cases where the pressure drop may apply on a per-zone basis.

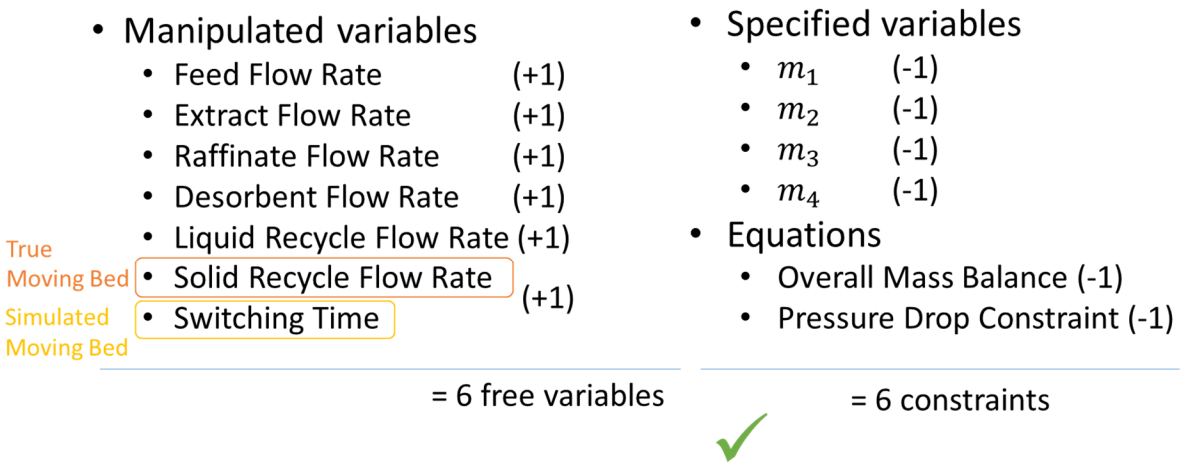


Figure 3.6: Degrees of Freedom in an SMB System

3.2.3 Solution Approaches

As we have seen, the design problem for SMB systems boils down to attempting to find the flow rate ratios that are consistent with complete TMB separation. There are two main approaches to this problem, and they are summarized in Figure 3.8.

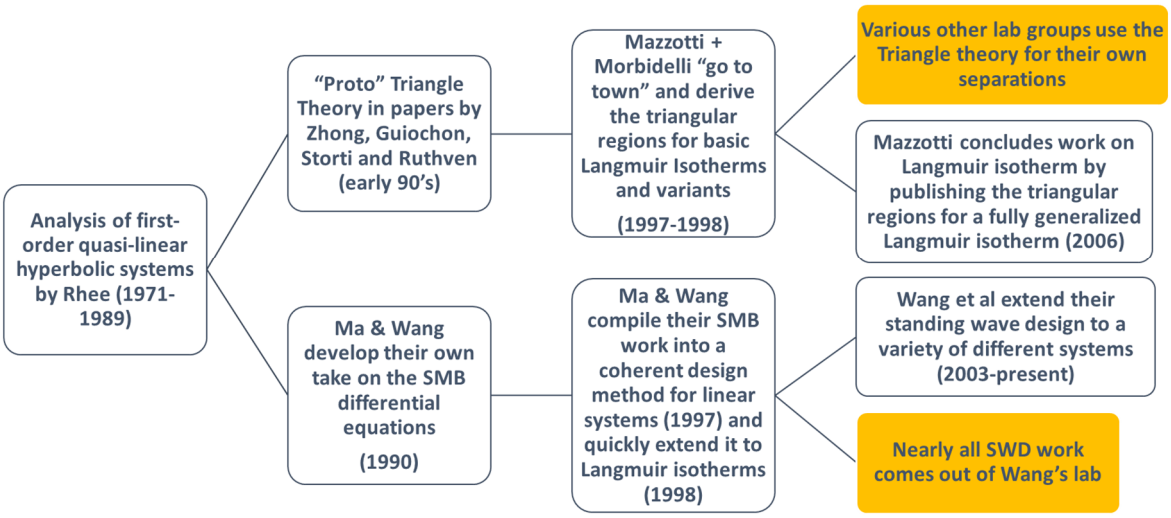


Figure 3.7: A history of the design approaches

The first method, triangle theory, asserts the simplifying assumptions we have previously discussed. By using solutions of the resulting PDEs derived using the method of characteristics, it is possible to analytically define regions where it is possible to satisfy the mass balances implied by Figure 3.4. This approach was originally developed in the early 1990s for linear isotherms as shown in Figure 3.7; the approach was named after the perfectly triangular regions that were found. In 2006, Mazzotti published a comprehensive analysis of the general Langmuir isotherm which found several vaguely triangular regions for a series of Langmuir cases. This approach has been widely used both as a design tool, as well as a method for interpreting SMB behavior. For example, process control studies often describe their performance by plotting their operating trajectory on top of a triangle theory plot.

The second method, Standing Wave Design, was developed at Purdue and officially made its debut in 1997. This method includes mass transfer information in its calculations, but rather than defining an operating region, it simply seeks to find an optimum operating point that guarantees the desired separation. It works by iteratively estimating the concentrations after the feed, using them to calculate the internal velocities of the system, and comparing the raffinate concentration predicted by mass balance to the raffinate concentrations predicted by an estimate based on estimated solutions to the differential equations.

Triangle Theory

- Assume no mass transfer resistance
  - Solid and liquid phases are everywhere in equilibrium (isotherm)
- Investigate
  - When is it possible to have physically meaningful solutions to the mass balances?
  - When are the complete separation conditions consistent with the differential equations?

Standing Wave Design

- There *is* mass transfer resistance… sometimes!

1. Solve simplified diff. eqs to get a first approximation of column conditions
2. Solve steady-state diff eqs (with mass transfer) to get an estimate of the effect of mass transfer resistance
3. Correct step 1 values with info from step 2.
4. Iterate until the correction in step 3 is small

Figure 3.8: The differences between triangle theory and Standing Wave Design

Given these facts, and the advantages that Standing Wave Design brings to the table, it is natural to ask why anyone would prefer the triangle theory. The reason that no one outside of Purdue has used the SWD has less to do with the relative merits of the approaches, and more to do with their approachability. There have been several good explanations of the Triangle Theory approach, and mechanistic tutorials exist to help people get started. Until now, Standing Wave Design has not received the same attention, and so actually executing the Standing Wave Design would require detailed study of the original papers in which it was derived.

3.3 Triangle Theory

3.3.1 Introduction

The Triangle Theory starts off by making the following simplifying assumptions:

- The system is a True Moving Bed
- Mass transfer resistance and dispersion are negligible
- Complete separation is achieved, as described in Figure 3.4
- The system is at steady state

By carefully examining Figure 3.4, we can see that the "net effect" of the SMB column is to carry one component to the extract stream, and the other component to the raffinate. In order to quantify this net effect, we can define a new term, $f_{i,j}^{net}$ which will be the net flow rate of component i in zone j. How can it be calculated? Let us again consider the mass balance discussed in Chapter 1.

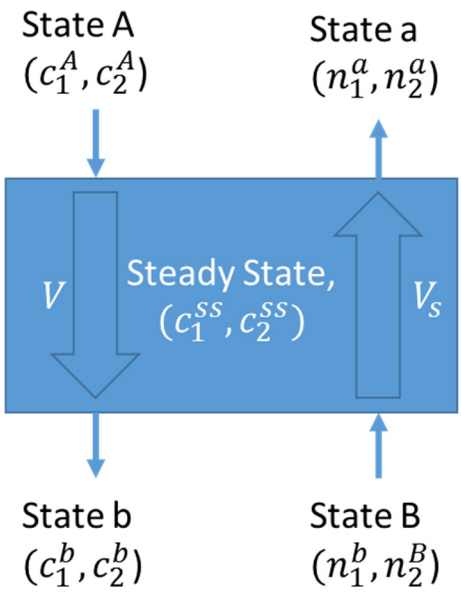


Figure 3.9: Net flow through a zone

Unfortunately, defining "the net flow through a zone" is easier to express in everyday language than in mathematics. How we quantify it crucially depends on what happens at steady state inside the zone. For example, we could imagine that the flow rate of the components in the liquid phase is very large compared to the solid phase's capacity. In such a scenario, the solid phase would immediately become fully loaded, and the concentration inside the zone would be equal to the liquid inlet concentration. This scenario is sketched graphically in Figure 3.10a. On the other hand, if the liquid and solid capacities are balanced, then the zone may take on an intermediate state at steady state. This is drawn in Figure 3.10b.

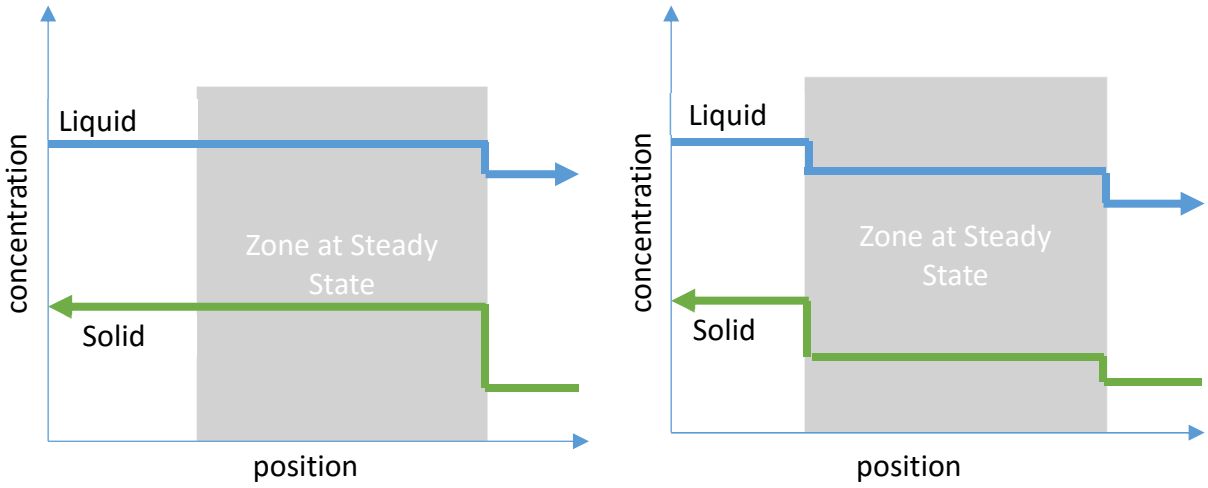


Figure 3.10: Examples of Steady State Conditions

Which state actually occurs depends on the various flow rates, concentrations, and isotherm. However, once we have that relationship, we can use the following definition for "net flow through the zone"

$$F_{i,j}^{net} = \left(Q_j - \varepsilon_p Q_s\right) c_i^{SS} - Q_s\left(1 - \varepsilon_p\right) n_i^{SS} \tag{3.8}$$

It is not too difficult to see how this equation expresses the concept of net flow. The first term is the net flow rate of each component in the liquid phase. Note that we subtract $\varepsilon_p Q_s$ which represents the flow

rate of the *voids* in the solid stream. We can think of this as discounting the liquid that is entrained in the solid. The second term represents the movement of the components that have been adsorbed onto the solid. Note that the $(1-\varepsilon_p)$ term converts the overall solid volume *including* voids to the solid volume *without* voids. It is also possible to write equation (3.8) in the following form:

$$F_{i,j}^{net} = \left(\frac{(Q_j - \varepsilon_p Q_s)}{Q_s(1-\varepsilon_p)} c_i^{SS} - n_i^{SS}\right) Q_s(1-\varepsilon_p) \tag{3.9}$$

$$F_{i,j}^{net} = (m_j c_i^{SS} - n_i^{SS}) Q_s(1-\varepsilon_p) \tag{3.10}$$

$$f_{i,j}^{net} = \frac{F_{i,j}^{net}}{Q_s(1-\varepsilon_p)} = m_j c_i^{SS} - n_i^{SS} \tag{3.11}$$

In equation (3.10) we have defined a very important quantity in SMB chromatography. Specifically, the flow rate ratio, m_j which represents the ratio of the net liquid flow rate to the net solid flow rate, as defined in equation (3.1)

Finally, we can perform a mass balance on the zone in Figure 3.9 as follows

$$\left(m + \frac{\varepsilon_p}{1-\varepsilon_p}\right) c_i^b = f_{i,j}^{net} + n_i^B + \frac{\varepsilon_p}{1-\varepsilon_p} c_i^B \tag{3.12}$$

$$\left(m + \frac{\varepsilon_p}{1-\varepsilon_p}\right) c_i^A = f_{i,j}^{net} + n_i^a + \frac{\varepsilon_p}{1-\varepsilon_p} c_i^a \tag{3.13}$$

Now that we have sketched out a few equations governing our system, we can begin the process of determining exactly which steady state conditions prevail in each zone. This process is more easily said than done, because finding the answer requires delving into the partial differential equations. Before we do that, however, we will make some remarks.

It will turn out that the steady state that is achieved inside the bed is a function of the two inlet streams (liquid and solid) as well as the m value. That is to say $f_{i,j}^{net} = f(c_1^A, c_2^A, n_1^B, n_2^B, m)$. That means that if we are able to calculate the net flow rate given values for each of those variables, we will also be able to calculate the values of the outlets at steady state.

3.3.2 Constraints on the System

In section 3.3.1, we discussed the concept of "net flow rate" through a bed. If we think back to Figure 3.2 and Figure 3.3, we can express that ideal separation in terms of net flow rate. Specifically, we would say that the only net flow rates that we want to be nonzero are the flow rates of component 2 through the second zone, and the flow rate of component 1 in the third zone. Because we want 100% of component 1 to flow through zone 3 and into the raffinate, the net flow through zone 3 should be the same as the net flow of component 1 in the feed. Similarly, for component 2, we want the net flow through zone 2 to be equal to the feed of component 2, except negative since the flow is moving in the direction of the solid phase.

$$F_1^I = 0, F_2^I = 0 \tag{3.14}$$

$$F_1^{II} = 0, F_2^{II} = -(Q_{feed}) c_2^f \tag{3.15}$$

$$F_1^{III} = (Q_{feed}) c_1^f, F_2^{III} = 0 \tag{3.16}$$

$$F_1^{IV} = 0, F_2^{IV} = 0 \tag{3.17}$$

If these are satisfied, we will have everything that we need for a perfect separation. We should also notice that it is possible to express the feed flow rate as

$$\frac{Q_{feed}}{Q_s(1-\varepsilon_p)} = m_3 - m_2 \tag{3.18}$$

which allows us to simply say

$$f_2^{II} = -(m_3 - m_2)c_2^f,\ f_1^{III} = (m_3 - m_2)c_1^f \tag{3.19}$$

If we look at the definition of net flow rate in equation (3.11) then we can immediately say something about the steady state conditions in zones 2 and 3:

$$(m_3 - m_2)c_2^f = -(m_2\, c_2^{II} - n_2^{II}) \tag{3.20}$$

$$(m_3 - m_2)c_1^f = m_3\, c_1^{III} - n_1^{III} \tag{3.21}$$

Where c^{II} and c^{III} are the steady state conditions in zones 2 and 3 respectively.

Applying the mass balances described in (3.12)and (3.13) to each zone gives:

Zone 1	$\left(m_1 + \frac{\varepsilon_p}{1-\varepsilon_p}\right)c_2^E = n_2^{ES} + \frac{\varepsilon_p}{1-\varepsilon_p}c_2^{ES}$	(3.22)
Zone 2	$\left(m_2 + \frac{\varepsilon_p}{1-\varepsilon_p}\right)c_1^\alpha = n_1^\gamma + \frac{\varepsilon_p}{1-\varepsilon_p}c_1^\gamma$	(3.23)
	$\left(m_2 + \frac{\varepsilon_p}{1-\varepsilon_p}\right)c_2^\alpha = -(m_2\, c_2^{II} - n_2^{II}) + n_2^\gamma + \frac{\varepsilon_p}{1-\varepsilon_p}c_2^\gamma$	(3.24)
	$\left(m_2 + \frac{\varepsilon_p}{1-\varepsilon_p}\right)c_2^E = -(m_2\, c_2^{II} - n_2^{II}) + n_2^{ES} + \frac{\varepsilon_p}{1-\varepsilon_p}c_2^{ES}$	(3.25)
Zone 3	$\left(m_3 + \frac{\varepsilon_p}{1-\varepsilon_p}\right)c_1^R = m_3\, c_1^{III} - n_1^{III} + n_1^{RS} + \frac{\varepsilon_p}{1-\varepsilon_p}c_1^{RS}$	(3.26)
	$\left(m_3 + \frac{\varepsilon_p}{1-\varepsilon_p}\right)c_1^\beta = m_3\, c_1^{III} - n_1^{III} + n_1^\gamma + \frac{\varepsilon_p}{1-\varepsilon_p}c_1^\gamma$	(3.27)
	$\left(m_3 + \frac{\varepsilon_p}{1-\varepsilon_p}\right)c_2^\beta = n_2^\gamma + \frac{\varepsilon_p}{1-\varepsilon_p}c_2^\gamma$	(3.28)
Zone 4	$\left(m_4 + \frac{\varepsilon_p}{1-\varepsilon_p}\right)c_1^R = n_1^{RS} + \frac{\varepsilon_p}{1-\varepsilon_p}c_1^{RS}$	(3.29)

We also know from the overall mass balances (or rearranging the previous equations) that

$$(m_3 - m_2)c_1^f = (m_3 - m_4)c_1^R \tag{3.30}$$

$$(m_3 - m_2)c_2^f = (m_1 - m_2)c_2^E \tag{3.31}$$

Unfortunately for us, this is about as far as we can go without knowing anything about the steady state concentrations in zones 2 and 3. To calculate what those are, we must delve into the differential equations that govern those two zones.

3.3.3 Differential Equations – Analysis

To start off, let us define the simplified differential equation we will be using. Starting from the more general differential equation in Chapter 1, we make the following assumptions:

- Negligible mass transfer resistance
    - The liquid and solid phases are in equilibrium at all times
- No axial dispersion
- The density of the liquid does not depend on the component concentrations

With these assumptions, we can reduce the differential equation to (3.32) with $f_i = f_i(c_1 \dots c_n)$ and $g_i = g_i(c_1 \dots c_n)$

$$\frac{\partial g_i}{\partial x} + \frac{\partial f_i}{\partial \tau} = 0 \tag{3.32}$$

Specifically, f_i is given by

$$f_i = c_i + \varepsilon\, n_i \tag{3.33}$$

in the case of fixed beds. In moving beds:

$$f_i = \psi(c_i + v^*\, n_i) \tag{3.34}$$

Where n_i depends on the choice of isotherm.

We can also define g_i for moving beds as:

$$g_i = \left((1 - \mu\epsilon_p)c_i - \mu(1 - \epsilon_p)n_i\right) \tag{3.35}$$

But in fixed beds,

$$g_i = c_i \tag{3.36}$$

To solve a PDE of this form, we can use the method of characteristics, as described in chapter 1. Unfortunately, since we have multiple components, and so we need some slightly more advanced methods. The overall goal is the same: to find a set of curves that reduces our differential equations from PDEs to ODEs.

The advanced solution method proceeds in the following way. To start with, we try to rewrite (3.32) in matrix form. We can use the chain rule to say:

$$\frac{\partial f_i}{\partial \tau} = \frac{\partial f_i}{\partial c_1}\frac{\partial c_1}{\partial \tau} + \dots + \frac{\partial f_i}{\partial c_n}\frac{\partial c_n}{\partial \tau} \tag{3.37}$$

$$\frac{\partial g_\mathrm{i}}{\partial x} = \frac{\partial g_i}{\partial c_1}\frac{\partial c_1}{\partial x} + \dots + \frac{\partial g_i}{\partial c_n}\frac{\partial c_n}{\partial x} \tag{3.38}$$

That means, we can write the system of equations defined in (3.32) as

$$\boldsymbol{g}\frac{\partial \vec{c}}{\partial x} + \boldsymbol{f}\frac{\partial \vec{c}}{\partial \tau} = 0 \tag{3.39}$$

where g and f are actually defined as the Jacobean of the f and g functions defined equations (3.33) through (3.36)

$$\boldsymbol{f} = \begin{bmatrix} \frac{\partial f_1}{\partial c_1} & \cdots & \frac{\partial f_1}{\partial c_n} \\ \vdots & \ddots & \vdots \\ \frac{\partial f_n}{\partial c_1} & \cdots & \frac{\partial f_n}{\partial c_n} \end{bmatrix} \tag{3.40}$$

$$\boldsymbol{g} = \begin{bmatrix} \frac{\partial g_1}{\partial c_1} & \cdots & \frac{\partial g_1}{\partial c_n} \\ \vdots & \ddots & \vdots \\ \frac{\partial g_n}{\partial c_1} & \cdots & \frac{\partial g_n}{\partial c_n} \end{bmatrix} \tag{3.41}$$

Notice that equation (3.39) is a relatively simple matrix equation, and we can make it even simpler by taking an inverse

$$\boldsymbol{g^{-1}f} = \boldsymbol{h} \tag{3.42}$$

$$\frac{\partial \vec{c}}{\partial x} + \boldsymbol{g^{-1}f}\frac{\partial \vec{c}}{\partial \tau} = \frac{\partial \vec{c}}{\partial x} + \boldsymbol{h}\frac{\partial \vec{c}}{\partial \tau} = 0 \tag{3.43}$$

Now this equation would be even more approachable if we were able to find some scalar multiple, σ, such that

$$\boldsymbol{h}\frac{\partial \vec{c}}{\partial \tau} = \sigma\frac{\partial \vec{\boldsymbol{c}}}{\partial \tau} = \frac{\partial \vec{\boldsymbol{c}}}{\partial x} \tag{3.44}$$

After all, we know that

$$\frac{\partial \vec{\boldsymbol{c}}}{\partial \tau}\frac{\partial \tau}{\partial x} = \frac{\partial \vec{\boldsymbol{c}}}{\partial x} \tag{3.45}$$

which would mean that

$$\sigma = \frac{\partial \tau}{\partial x} \tag{3.46}$$

But if we recall from linear algebra, the definition of eigenvalues is:

$$\boldsymbol{h}\,\vec{\boldsymbol{v}} = \sigma\,\vec{\boldsymbol{v}} \tag{3.47}$$

which is exactly what we are trying to find! Moreover, finding eigenvalues is a fairly straightforward process, we can find the correct value of σ by calculating the eigenvalues of $\boldsymbol{h}$ with the equations defined in

$$|\boldsymbol{h} - \sigma\boldsymbol{I}| = 0 \tag{3.48}$$

The exact form of the eigenvalues will depend on the isotherm, and they may be functions of concentrations. But so far so good! We have identified how to calculate a curve in the x/τ plane. But what does that curve tell us? Not much, by itself, but we do know that each eigenvalue will have a corresponding eigenvector. For simplicity, let us consider the two component case. When we identify the eigenvalues σ in equation (3.44) we also implicitly identified the corresponding eigenvector to be

$$\frac{\partial \vec{c}}{\partial \tau} = \left(\frac{\partial c_1}{\partial \tau}, \frac{\partial c_2}{\partial \tau}\right) = Eigenvector(\boldsymbol{h}) \tag{3.49}$$

That vector defines a second curve through its slope:

$$\frac{\partial c_2}{\partial \tau} \Big/ \frac{\partial c_1}{\partial \tau} = \frac{\partial c_2}{\partial c_1} \tag{3.50}$$

Now we have two curves:

- $\frac{\partial \tau}{\partial x}$ in the x, τ plane is defined by the eigenvalues of $\boldsymbol{h}$ and may be a function of concentration
- $\frac{\partial c_2}{\partial c_1}$ in the c_1, c_2 plane is defined by the eigenvectors of $\boldsymbol{h}$ and is not a function of x or τ

The eigenvalue, σ will be constant along the x, τ curve, i.e. the x t curve will be a straight line. However, the same is not true of the c_1, c_2 curve, it may not be a straight line. Regardless, the curves in the two planes allow us to find the complete solution to the differential equations. In the next section we will cover how this solution is constructed.

3.3.4 Constructing the Solution from Eigenvectors and Eigenvalues

To progress beyond the analysis in part 1, we need to introduce boundary conditions. In a moving bed adsorber, these are most commonly feed concentrations and initial column loading. From those initial conditions, actual instances of the c_1, c_2 curves are defined, and the solution can be constructed.

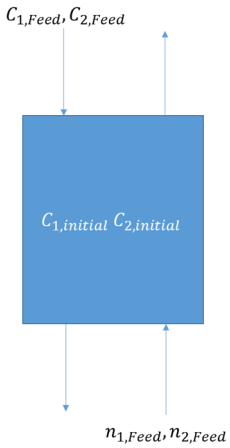


Figure 3.11: Boundary Conditions in an Absorber

Figure 3.11 shows the three pairs of boundary conditions that are used in the solution. Specifically,

$$c_i = c_i^0 \; at \; \tau = 0 \; and \; 0 \leq x \leq 1 \tag{3.51}$$

$$c_i = c_{i,Feed} \; for \; all \; \tau > 0 \; and \; x = 0 \tag{3.52}$$

$$n_i = n_{i,Feed} \; for \; all \; \tau > 0 \; and \; x = 1 \tag{3.53}$$

To completely describe this situation requires that we find a solution for the top of the column and the bottom of the column separately, at least until they begin interacting. Once that happens, we need to describe the evolution of the interacting systems until they reach steady state. All of that is possible, but it is not necessary to describe that process here. It turns out that "all roads lead to Rome," because the eventual steady state is independent of the initial column state; it is characterized by equations (3.51) and (3.52) alone. Therefore, in the following analysis, we will make the assumption that the initial column state is the same as (i.e. is in equilibrium with) the solid stream's state. This will simplify the process since we only need to consider the transition taking place at the top of the column, and we will get the correct steady state.

To construct the solution, we will follow this procedure:

1. Calculate the $\boldsymbol{g}, \boldsymbol{f}$, and $\boldsymbol{h}$ matrices defined in equations (3.40) (3.41) and (3.42)
2. Calculate the eigenvalues and eigenvectors of these matrices
3. Find the path that the transition will take in the (c_1, c_2) plane. This is done by
    a. Calculating the two curves defined by the eigenvectors at the initial condition
    b. Calculating the two curves defined by the eigenvectors at the final condition
    c. The transition path follows the second eigenvector curve from the initial condition until it intersects the first eigenvector curve from the final condition. See Figure 3.13.

4. At each point along the transition, calculate the value of the corresponding eigenvalue. Thanks to equation (3.46) we know $\tau = \sigma\, x$, so for each concentration pair in a given transition curve, we have an eigenvalue and therefore a curve in the x, τ plane.
    a. Along the curve defined by the second eigenvector, use the second eigenvalue
    b. Along the curve defined by the first eigenvector, use the first eigenvalue
5. Take a slice of the x, τ plane curves at x=1 to generate breakthrough curves for our bed.
6. Take a slice of the x, τ plane curves at $\tau = \tau_i$ to generate the concentration profile in the bed at time τ_i

3.3.5 Exercize 1: Calculating Transitions in a Fixed Bed using Mathematica

In this exercise, we will follow the procedure described in the previous section to construct the breakthrough curve and a concentration profile.

We will perform the calculations for the following scenario. For our purposes, let us consider a fixed bed column under the following conditions:

- A bed porosity of 0.4, with no particle voids
- A liquid velocity of 0.5 units
- A fixed bed (solid velocity is 0)
- A Langmuir isotherm with $a_1 = 1, a_2 = 2, b_1 = 0.1, b_2 = 0.1$
- An initially empty column, $c_{i,initial} = 0$
- A feed with composition $\left(c_{1,feed} = 5, c_{2,feed} = 5\right)$

We will also use the following notation:

Quantity	Symbol
Concentration of component i in state j	c_i^j
Bed Length	L
Flow Rate Ratio	$m = \dfrac{1 - \mu\epsilon_p}{\mu(1 - \epsilon_p)}$
Solid phase loading of component i	n_i
Derivative of solid phase loading	$n_{i,j} = \dfrac{\partial n_i}{\partial c_j}$
Solid Velocity	V_s
Liquid Velocity	V
Phase Ratio	$v^* = \dfrac{1 - \epsilon_b - (1 - \epsilon_b)\epsilon_p}{\epsilon_b + (1 - \epsilon_b)\epsilon_p}$
Dimensionless Position	$x = z/L$
States achieved during SMB operation between zones 2 and 3	α, β, γ
External or Inter-particle porosity	ε_b
Internal or Intra-particle porosity	ε_p

Ratio of solid to fluid velocity	$\mu = \frac{(1-\epsilon_b)V_s}{\epsilon_b V}$
Bed void fraction	$\psi = \epsilon_b + (1-\epsilon_b)\epsilon_p$
Eigenvalues of the Isotherm Matrix	θ_1, θ_2
Eigenvalues of the F Matrix	ϕ_1, ϕ_2
Eigenvalues of the G Matrix	γ_1, γ_2
Eigenvalues of the H Matrix	η_1, η_2

3.3.5.1 Step 1: Defining the Matrices

This step is straightforward in Mathematica. First, we write out the equations just as they appear in equations (3.34) and (3.35). We use the Table[] command in Mathematica to say that i is allowed to vary from 1 to 2, and the result is a list of equations, for example $\{f_1, f_2\}$

```
f = Table[(cᵢ+v* nᵢ)ψ,{i,1,2}]
g = Table[((1-μ ϵp)cᵢ-μ(1-ϵp)nᵢ),{i,1,2}]
isotherm = Table[(aᵢ cᵢ)/(1+b₁ c₁+b₂ c₂),{i,1,2}]
```

For the next step, we need to define our own Jacobian function. We can make such a function with the derivative command, D. Once defined, the jacobian[x] function will take the Jacobian of anything we put inside the brackets.

$$\text{jacobian[x\_]} := D[x, \{\{c_1, c_2\}\}, \text{NonConstants} \rightarrow \{n_1, n_2\}]$$

Since our original equations were in terms of the loading, n_i, we need to tell Mathematica that the loading may actually be a function of c_1 and c_2. That is accomplished by the use of the NonConstants option.

Finally, we can calculate the $\boldsymbol{g}, \boldsymbol{f}$, and $\boldsymbol{h}$ matrices by using our jacobian function on f, g, and the isotherm. Note that we use the Simplify[] function to perform algebraic rearrangements and cancellations that make the result simpler.

```
jacobianF = Simplify[jacobian[f]]
jacobianG = Simplify[jacobian[g]]
jacobianIso = Simplify[jacobian[isotherm]]
jacobianH = Simplify[Inverse[jacobianG].jacobianF]
```

The output of these commands is rather long, so we will only look at the jacobianF as an example. //MatrixForm simply formats the output.

```
jacobianF//MatrixForm
```

$$\begin{pmatrix} \psi + v^*\psi n_{1,1} & v^*\psi n_{1,2} \\ v^*\psi n_{2,1} & \psi + v^*\psi n_{2,2} \end{pmatrix}$$

3.3.5.2 Step 2: Calculating the Eigenvalues and Eigenvectors

This step is very straightforward, although our earlier decision to write the f and g equations in terms of n_1 and n_2 means that we have to do some extra work to replace those values with the actual isotherm

equation now. In other words, we want our equations to be explicitly in terms of c_1 and c_2 instead of c_1, c_2, n_1 and n_2

We can start by calculating both the eigenvalues and eigenvectors in one command:

```
eigensFn = Simplify@Eigensystem[jacobianF]
eigensGn = Simplify@Eigensystem[jacobianG]
eigensIso = Simplify@Eigensystem[jacobianIso]
eigensHn = Simplify@Eigensystem[jacobianH]
```

The Eigensystem is a built-in Mathematica function that returns a list that contains both the eigenvalues and vectors of a matrix. They are in the form: {eigenvalues, eigenvectors}

So, for example, the eigenvalues of the f matrix are the first element of eigensFn:

```
eigensFn[[1]]//MatrixForm
```

$$\begin{pmatrix} \frac{1}{2}\psi\left(2 + v^* n_{1,1} + v^* n_{2,2} - v^*\sqrt{n_{1,1}^2 + 4n_{1,2}n_{2,1} - 2n_{1,1}n_{2,2} + n_{2,2}^2}\right) \\ \frac{1}{2}\psi\left(2 + v^* n_{1,1} + v^* n_{2,2} + v^*\sqrt{n_{1,1}^2 + 4n_{1,2}n_{2,1} - 2n_{1,1}n_{2,2} + n_{2,2}^2}\right) \end{pmatrix}$$

While the eigenvectors are the second:

```
eigensFn[[2]]//MatrixForm
```

$$\begin{pmatrix} -\frac{-n_{1,1} + n_{2,2} + \sqrt{n_{1,1}^2 + 4n_{1,2}n_{2,1} - 2n_{1,1}n_{2,2} + n_{2,2}^2}}{2n_{2,1}} & 1 \\ \frac{n_{1,1} - n_{2,2} + \sqrt{n_{1,1}^2 + 4n_{1,2}n_{2,1} - 2n_{1,1}n_{2,2} + n_{2,2}^2}}{2n_{2,1}} & 1 \end{pmatrix}$$

Now, in order to eliminate the loading terms, $n_{i,j}$ we need to insert the Langmuir isotherm into these equations. This is accomplished in the following way. First, we create a list of "Replacement Rules"

```
nReplacement = Table[n_{i,j} → jacobianIso[[i, j]],  {i, 1,2}, {j, 1,2}]//Flatten
```

These rules look like:

$$n_{1,1} \rightarrow \frac{a_1(1 + b_2 c_2)}{(1 + b_1 c_1 + b_2 c_2)^2}$$

And tell Mathematica to replace $n_{1,1}$ with the expression $\frac{a_1(1+b_2c_2)}{(1+b_1c_1+b_2c_2)^2}$ in whatever equations we decide to apply the replacement rules to. So since we have generated the rules already, we just have to apply them with the /. operator.

```
eigensF = Simplify[eigensFn/.nReplacement, Assumptions->1+b1c1+b2c2>0]
eigensG = Simplify[eigensGn/.nReplacement, Assumptions->1+b1c1+b2c2>0]
eigensH = Simplify[eigensHn/.nReplacement, Assumptions->1+b1c1+b2c2>0]
```

Note that in addition to applying the replacement rules with `/.nReplacement` we have also added an "Assumption" to our Simplify function. In this case we are assuming that the denominator of the Langmuir isotherm is always positive. This is a valid assumption because we know that the alternative results in a non-physical situation: the isotherm would predict negative loading. The reason we add this assumption is because Mathematica will not simplify an equation of the form $\sqrt{x^2}$ to x unless it knows that $x > 0$. And it turns out that making such a simplification is possible and worthwhile in the eigenvectors and eigenvalues.

Before we move on to the next step, there are some interesting relationships we can ask Mathematica to calculate for us. For example, the eigenvalues of all four of our matrices are equivalent:

```
eigensHn[[2]]==eigensFn[[2]]==eigensGn[[2]]
```

True

The isotherm eigenvectors are more stubborn. Since it was not in terms of n, unlike the f,g, and h eigenvalues, it did not simplify to exactly the same form as the others. Therefore, we have to compare them on an element-by-element basis:

```
Simplify@Table[eigensH[[2,i,j]] == eigensIso[[2,i,j]], {i,1,2},{j,1,2}]
```

{{True, True}, {True, True}}

Since our equality test returned true four times, both components of both eigenvectors are the same between the h matrix and the isotherm matrix.

Finally, we may want to know if there is any relationship between the eigenvalues. Mathematica can do this for us as well, although the details of the implementation are left in the notebook on the CD. If we name the eigenvalues as follows:

Table 3.1: Names of the eigenvalues

Isotherm Matrix	θ_1, θ_2
F Matrix	ϕ_1, ϕ_2
G Matrix	γ_1, γ_2
H Matrix	η_1, η_2

Then we can find the relationships:

	$\phi_1 = \psi + v^* \psi \theta_1$ and $\phi_2 = \psi + v^* \psi \theta_2$	(3.54)
	$\gamma_1 = 1 - \epsilon_p \mu + \left(-1 + \epsilon_p\right)\mu\theta_1$ and $\gamma_2 = 1 - \epsilon_p \mu + \left(-1 + \epsilon_p\right)\mu\theta_2$	(3.55)
	$\eta_1 = \frac{\psi + v^* \psi \theta_1}{1 - \epsilon_p \mu + (-1 + \epsilon_p)\mu\theta_1}$ and $\eta_2 = \frac{\psi + v^* \psi \theta_2}{1 - \epsilon_p \mu + (-1 + \epsilon_p)\mu\theta_2}$	(3.56)

It turns out that these relationships are very important! Especially the one given by equation

3.3.5.3 Step 3: Calculate the Curves Defined by the Eigenvectors

The equations for the eigenvectors are somewhat complex, and they contain a problematic feature: there is a ratio whose denominator is proportional to c_2. This means that in all cases where $c_2 = 0$ we

are going to have problems with division by zero. To understand what is going on, let us consider the eigenvectors at the origin, $c_1 = 0, c_2 = 0$. We will ask Mathematica to calculate the direction of the eigenvectors as c_2 approaches zero. To do this, we will first normalize the eigenvectors, and let $c_1 = 0$

```
vectors = eigensIso[[2]];
vectorDirections = Simplify[Normalize/@vectors/.c1 → 0]
```

The next step is to take the limit as c_2 goes to 0. It turns out that the result of this limit depends on which component is more strongly adsorbed. Therefore, we will consider both cases, assuming that component 2 is more strongly adsorbed, and then assuming that component 1 is more strongly adsorbed:

```
type1 = Limit[Abs[#],c2->0, Assumptions->{a1<a2, b1∈Reals}]
&/@vectorDirections


type2 = Limit[Abs[#],c2->0, Assumptions->{a1>a2, b1∈Reals}]
&/@vectorDirections
```

The results show what is happening at the origin:

```
type1//MatrixForm
type2//MatrixForm
```

$$\begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}$$

$$\begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}$$

In other words, we have the situation depicted in Figure 3.12

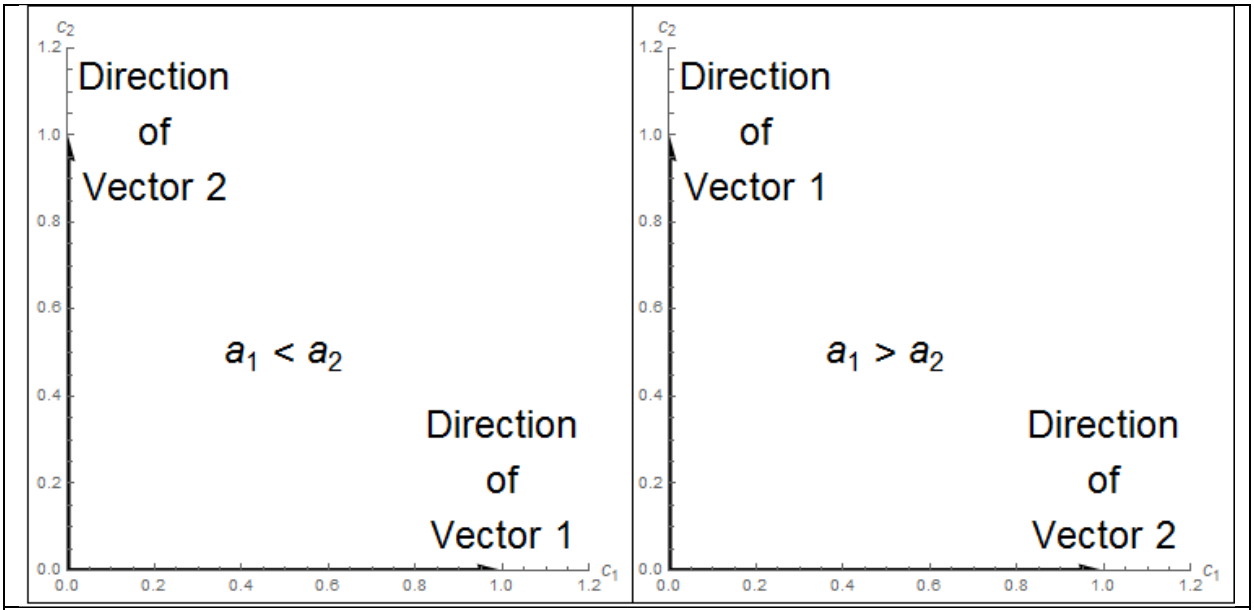


Figure 3.12: The eigenvector directions depend on which component is more strongly adsorbed

With this in mind, we will assume that $a_1 < a_2$ for the rest of this exercise.

Now, we can return to the main task: we need to calculate the curves in the eigenvector field from the initial and final conditions. To start with, we will calculate the slopes as discussed in equation (3.50)

```
slopeEq1=vectors[[1,2]]/vectors[[1,1]]//Simplify
slopeEq2=vectors[[2,1]]/vectors[[2,2]]//Simplify
```

Note that we have treated the first and second vector differently. This is because of what we know from Figure 3.12. Specifically, we have said that:

$$slopeEq1 = \frac{\partial c_2}{\partial \tau} / \frac{\partial c_1}{\partial \tau} = \frac{\partial c_2}{\partial c_1} \quad (3.57)$$

$$slopeEq2 = \frac{\partial c_1}{\partial \tau} / \frac{\partial c_2}{\partial \tau} = \frac{\partial c_1}{\partial c_2} \quad (3.58)$$

This is necessary, because $\frac{\partial c_1}{\partial c_2}$ is infinite along the first vector, while $\frac{\partial c_2}{\partial c_1}$ is infinite along the second vector.

Now that we have these two slopes in the c_1, c_2 plane, we can ask Mathematica to do the following:

```
DSolve[ c2'[c1] == (slopeEq1/. c2 → c2[c1]),      c2[c1],      c1]
```

DSolve is Mathematica's differential equation solver. The above command asks Mathematica to find an analytic solution to the differential equation we have specified so far. In other words, if Mathematica can, it will output the function, $c_2[c_1]$, in terms of c_1. Since we did not specify an initial condition, the resulting equation would contain an unknown integration constant, but we could use our initial conditions to find the value. Unfortunately, Mathematica's DSolve function is currently unable to solve this differential equation in a reasonable amount of time, so we must resort to more manual methods.

For our manual method, it turns out that the curves we are looking for are linear. We can prove this to ourselves in the following way. First, we tell Mathematica that along the first curve, c_2 is a function of c_1 and along the second curve, c_1 is a function of c_2:

```
funcSlopes1=slopeEq1/.c2->c2[c1]
funcSlopes2=slopeEq2/.c1->c1[c2]
```

Then we take the derivative of those slopes

```
secondDerivative1 = Simplify[∂c1 funcSlopes1]
secondDerivative2 = Simplify[∂c2 funcSlopes2]
```

And finally, we tell Mathematica to replace the first derivatives with the first derivatives we already knew.

```
Simplify[secondDerivative1/.c2'[c1]-> funcSlopes1]
Simplify[secondDerivative2/.c1'[c2]-> funcSlopes2]
```

```
0
0
```

Since the output is zero for both cases, we have effectively demonstrated that the second derivatives are zero, and therefore the curves are linear. This is explained in equations (3.59) and (3.60)

$$\frac{\partial slopeEq1}{\partial c_1} = \frac{\partial^2 c_2}{\partial {c_1}^2} = 0 \tag{3.59}$$

$$\frac{\partial slopeEq2}{\partial c_2} = \frac{\partial^2 c_1}{\partial {c_2}^2} = 0 \tag{3.60}$$

which implies that the curves we are looking for are linear. That means that the slope of the curves is constant at all points including the initial conditions. Therefore, the slope can be calculated once and for all at the initial and final state of the column. We can easily calculate these slopes, and subsequently the equations of the lines in the c_1, c_2 plane.

```
initialSlope1 = slopeEq1/.{c1->c01,c2->c02}
initialSlope2 = slopeEq2/.{c1->c01,c2->c02}

line1 = Simplify[c2==c1*initialSlope1+(c02-c01*initialSlope1)]
line2 = Simplify[c1==c2*initialSlope2+(c01-c02*initialSlope2)]
```

Finally, we are ready to insert the values for this problem, and plot the result. Figure 3.13 shows a plot of the lines that we have calculated, once we substitute in the isotherm parameters and operating conditions. The lines represent the analytic equations, while the x marks denote the results of a detailed numeric simulation. Clearly we are not just wasting our time, and the eigenvector transitions are able to correctly describe the loading of a packed bed.

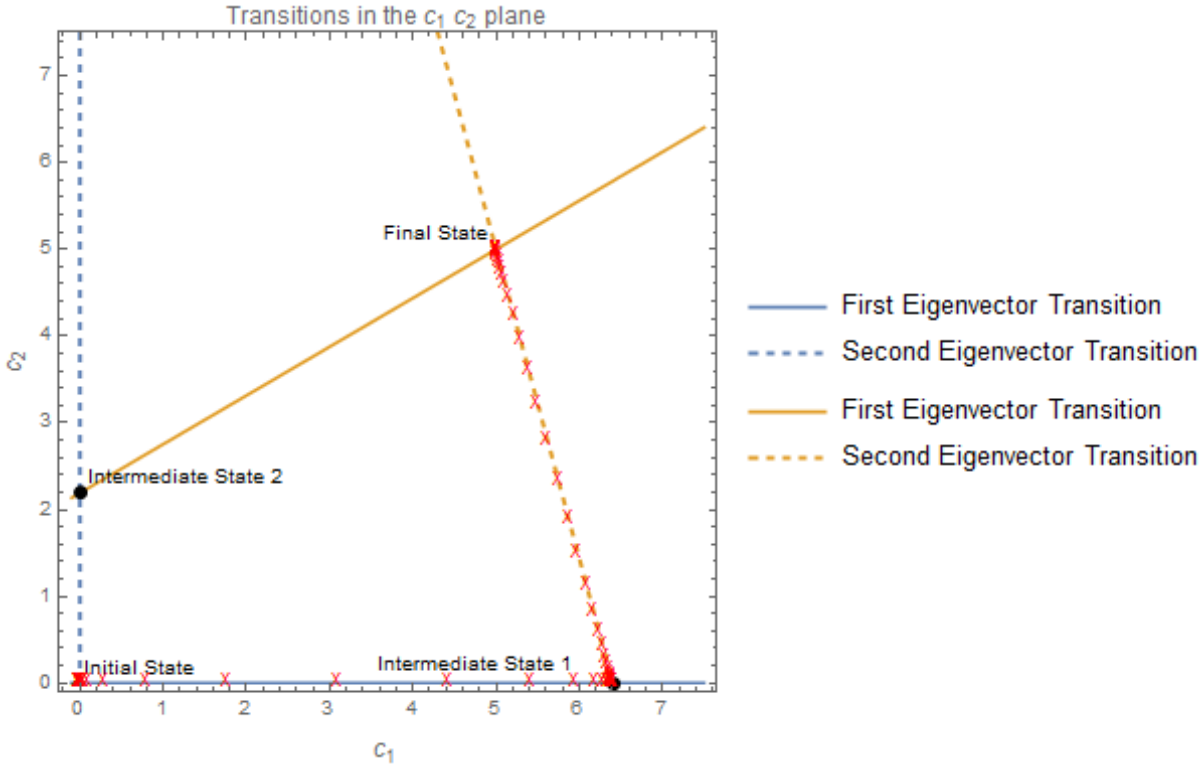


Figure 3.13: Transitions in the c_1, c_2 plane; the eigenvector prediction (blue line) vs numerical simulation in Aspen (red x). The yellow curve is the path that would be taken during the reverse transition.

3.3.5.4 Step 4: Calculate the Eigenvalues Along the Transition

As shown in Figure 3.13, we now have information about how the column gets loaded in the concentration plane. However, we have no information about how that plot corresponds to position and time. That information comes from the eigenvalues, as we saw in equation (3.46). For reasons that will be discussed later, we will calculate the eigenvalues of the isotherm matrix first (θ_i) which will then be used to calculate the eigenvalues of the other matrices using equations (3.54) through (3.56). So to get started, we will set up a temporary variable containing the eigenvalues we calculated in step 2. We will also create replacement rules by solving the line equations we produced in step 3.

```
eigenvalues=eigensIso[[1]]
line1Replacement=Solve[line1,c2]//First
line2Replacement=Solve[line2,c1]//First
```

Now we must apply the replacement rules to the corresponding eigenvalue. Along line1, the curve defined by the first eigenvector, we must use the first eigenvalue. The second eigenvalue is used along the second curve.

```
eigenvalueEquation1=Simplify[eigenvalues[[1]]/.line1Replacement]
eigenvalueEquation2=Simplify[eigenvalues[[2]]/.line2Replacement]
```

The end result are two equations, the first one is in terms of c_1 and the second is in terms of c_2. These equations allow us to calculate the eigenvalues at a particular point along the transition curves. For example, if we wanted to calculate the θ eigenvalue at the point (4.5, 0) in Figure 3.13, we would do the following:

```
eigenvalueEquation1/.{c01->0, c02->0, c1->4.5, a1->1, a2->2, b1->0.1, b2->0.1}
                                    0.475624
```

As we see in Figure 3.13, the first transition proceeds from (0,0) to (6.5,0). We can therefore create a nice plot of these slopes using our eigenvalueEquations, as seen in Figure 3.14

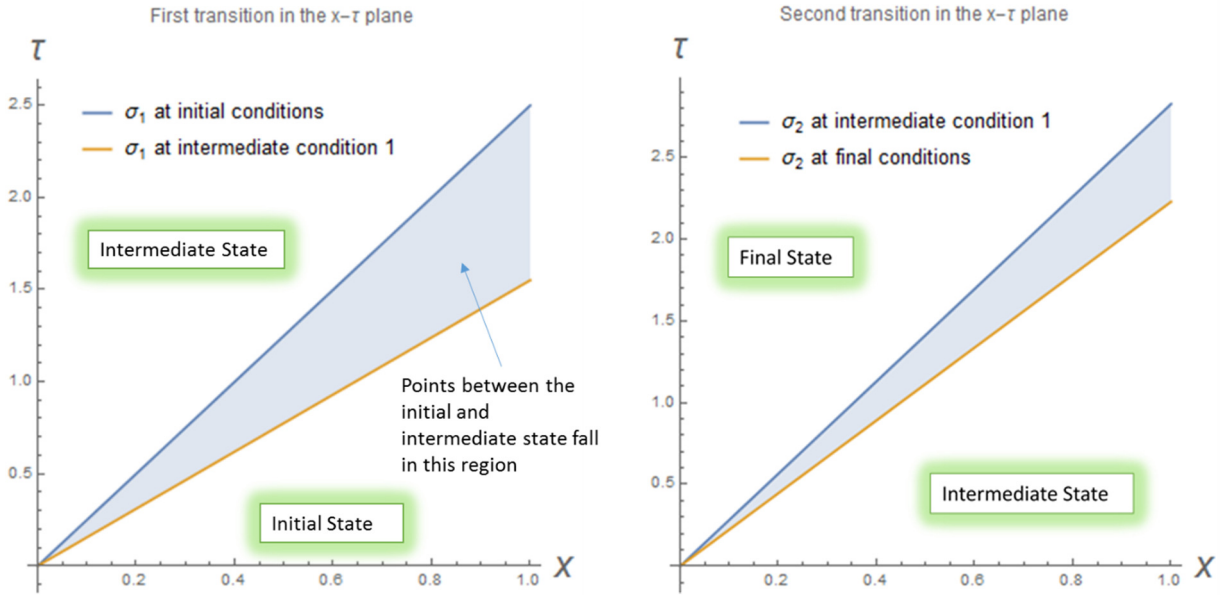


Figure 3.14 transitions in the x − τ plane

If we are paying close attention, we will notice that these transitions do not quite make sense. For example, why would the σ_1 corresponding to the intermediate condition 1 be adjacent to the initial state, instead of the intermediate state? The answer is that these transitions actually form shock waves, which we will discuss after we try to calculate the breakthrough curves in the next step.

3.3.5.5 Step 5: Calculate the Concentrations in Time and Space

Now that each concentration pair in the transition has a curve in the $\tau - x$ plane, we can choose a constant position (x=1) and calculate the time corresponding to each concentration pair. To do this, we start by calculating the τ equations at x=1:

```
τprofile1=1*ϕ1/.ϕSolutions/.θ1->eigenvalueEquation1
τprofile2=1*ϕ2/.ϕSolutions/.θ2->eigenvalueEquation2
```

These two equations tell us τ as a function of c_1 or c_2. We could immediately make a plot of this, but having τ on the vertical axis is somewhat awkward. We also need a way to plot both concentrations at all times, but these equations only involve one of the components each.

To resolve this, we will set ourselves up to create a parametric plot.

```
parametricCurves1={{τprofile1,c1},{τprofile1,(c2/.line1Replacement)}}
parametricCurves2={{τprofile2,c1/.line2Replacement},{τprofile2,c2}}
```

What we have just done is to define two pairs of parametric curves. Specifically, parametricCurve1 is now defined as

$$parametricCurve1 = \begin{pmatrix} \tau[c_1] & c_1 \\ \tau[c_1] & c_2[c_1] \end{pmatrix} \tag{3.61}$$

The first row of that matrix is a (τ, c_1) pair, where τ is a function of c_1. The second row is a (τ, c_2) pair, where both τ and c_2 are functions of c_1. This means that we can make a parametric plot where we vary c_1 from $c_{1,initial}$ to $c_{1,intermediate\ state\ 1}$. The exact same procedure applies to the second parametric curve 2, except we vary c_2 from $c_{2,intermediate\ state\ 1}$ to $c_{2,final}$. When we do this, we find the curves shown in Figure 3.15.

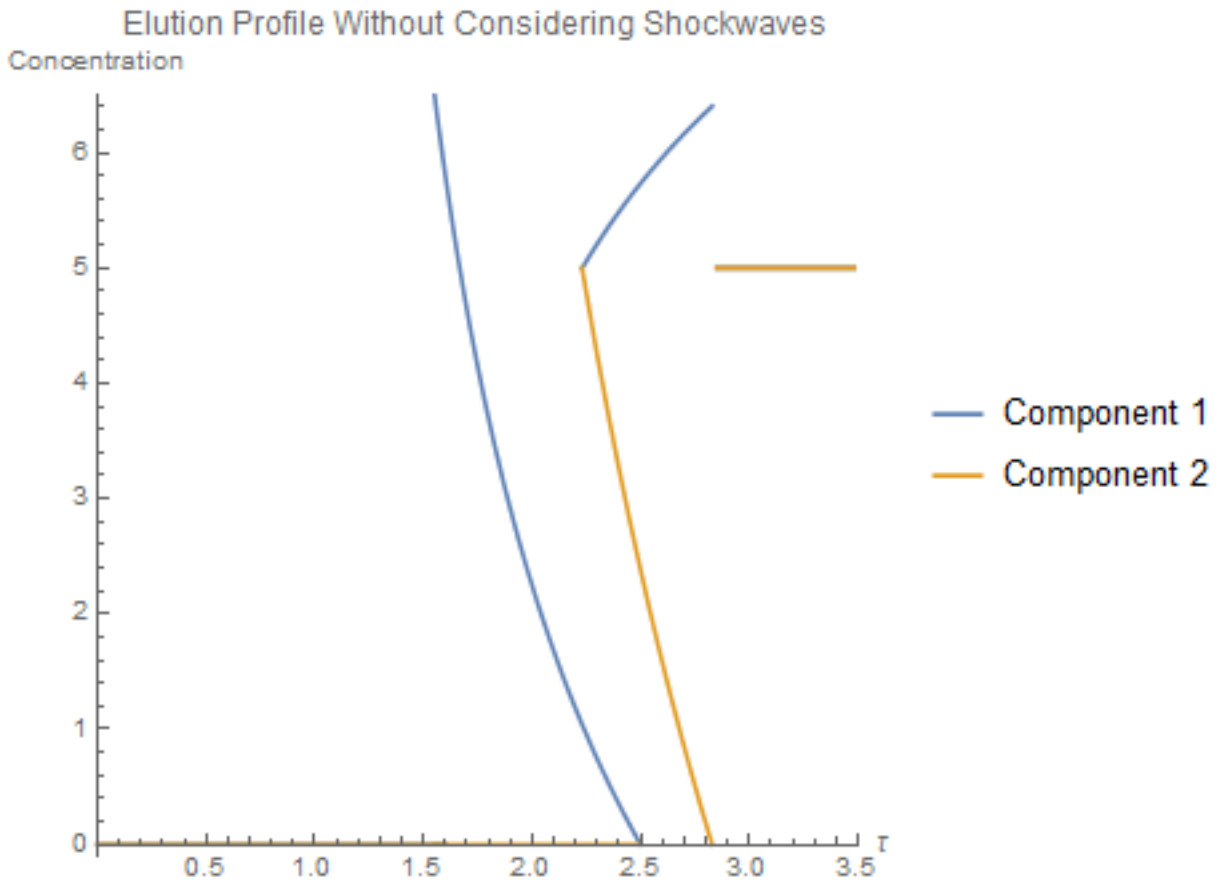


Figure 3.15: The breakthrough curve defined by the curves in Figure 3.14, without accounting for shocks

Clearly, something has gone wrong, the curves look to be reversed in time. The first transition looks like it starts from the intermediate condition, (6.4,0) and ends at the initial condition (0,0). The second curve transitions from the final state (5,5) to the intermediate state. Why are the transitions backwards? The answer, which we might have noticed earlier, is that these transitions form shockwaves.

3.3.5.6 Step 6: Account for Shockwaves

Clearly our solution in Step 5 is not physically meaningful because we have not accounted for shockwave formation. What are shockwaves and how can we calculate a solution that takes them into account?

Shockwaves occur when the state at the end of a transition moves through the column faster than the state at the start. This process is shown in Figure 3.16, where the fast final state is labeled with a rabbit, while the slow initial state is labeled with a turtle.

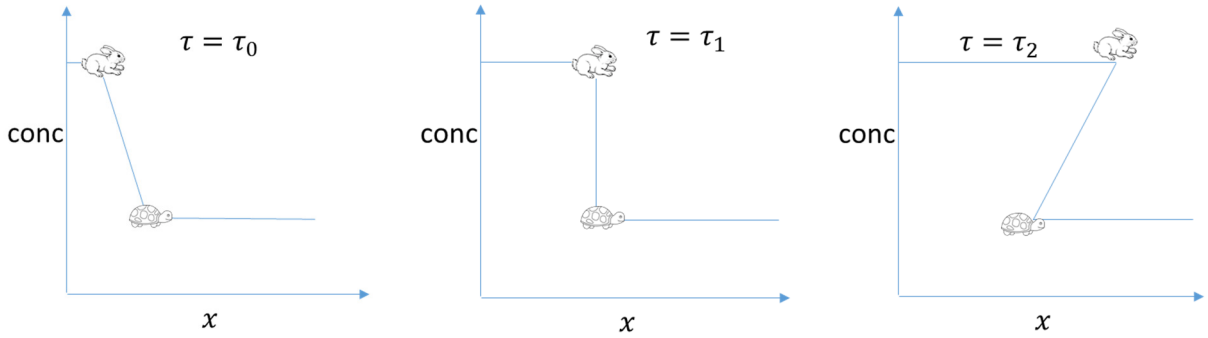


Figure 3.16 Formation of shockwaves

As we can see, the situation in Figure 3.16 causes the same kind of backwards transitions that we calculated in Figure 3.15. The backwards state shown in the $\tau = \tau_2$ plot is not physically meaningful or achievable. Instead, the $\tau = \tau_1$ state puts an end to the speed difference between the initial and final states. The instantaneous transition in that state is called a shockwave, and it has its own velocity that is calculated differently. To calculate the velocity of a transition that has formed a shockwave, we replace the θ values in equations (3.54) through (3.56) with a value based on the magnitude of the jump in the shockwave, here denoted by the brackets.

$$\theta_i|_{shock} = \left.\frac{[n_i]}{[c_i]}\right|_{shock} \tag{3.62}$$

For example, $[c_1] = |c_{1,final} - c_{1,intermediate}|$. This is our reason for using θ_i, the eigenvalue of the isotherm matrix defined in Table 3.1, instead of just working with the explicit eigenvalues originally calculated by Mathematica. Once the θ value has been calculated using equation (3.62), equation (3.56) allows us to calculate the velocity of the shockwave.

In Mathematica we can calculate this easily:

$$\theta_{\text{shocks}} = \frac{(\text{isotherm}/.\{c_1 \to \text{cF}_1, c_2 \to \text{cF}_2\}) - (\text{isotherm}/.\{c_1 \to \text{c0}_1, c_2 \to \text{c0}_2\})}{\{\text{cF}_1 - \text{c0}_1, \text{cF}_2 - \text{c0}_2\}}$$

With these θ values, we can generate a new version of the plots in Figure 3.14 that takes into account the shockwave formation. To do this, we calculate the τ profiles just like we did in step 5.

```
τprofile1shock=ϕ1/.ϕSolutions/.θ1->θshocks[[1]]
τprofile2shock=ϕ2/.ϕSolutions/.θ2->θshocks[[2]]
```

Then we plot these two shock transitions:

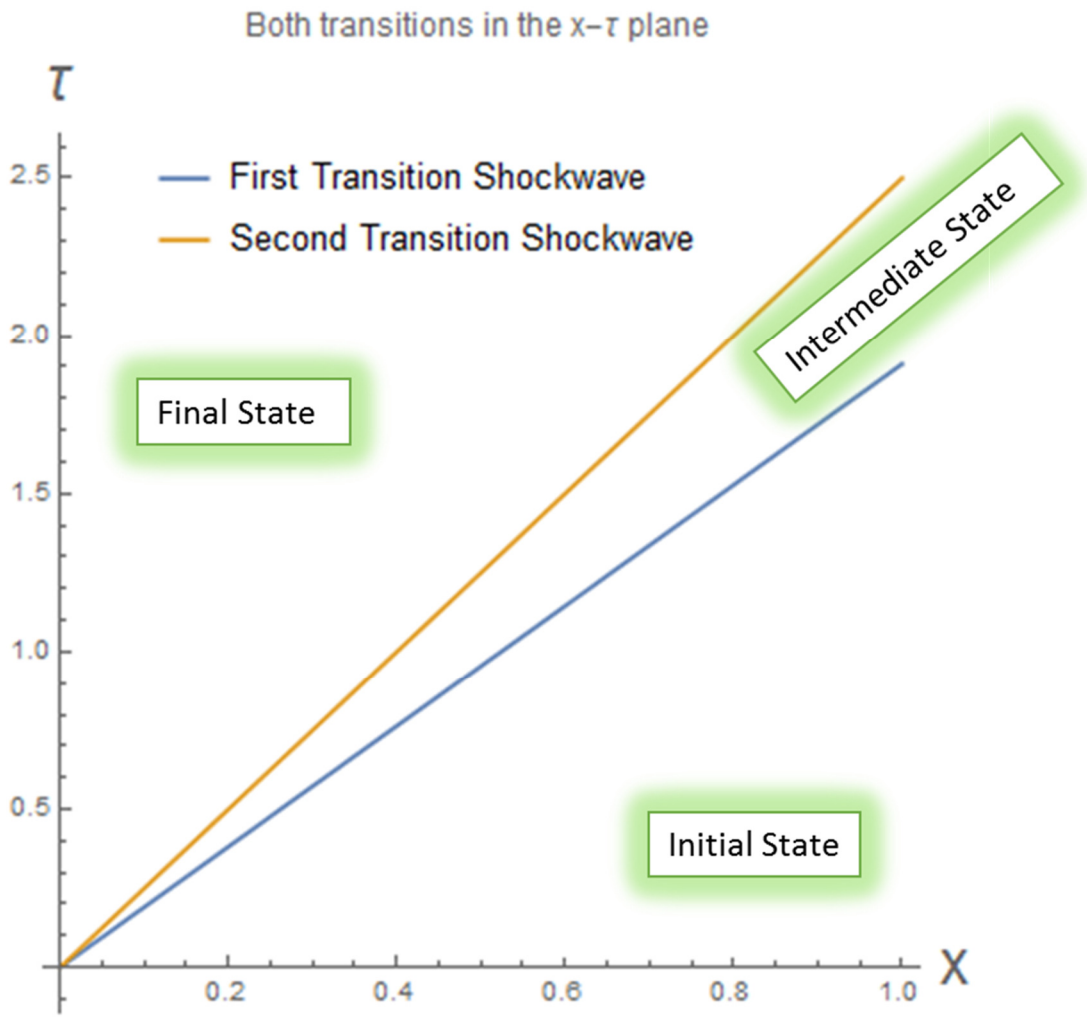


Figure 3.17 transitions in the $x - \tau$ plane with shockwaves accounted for

Pay attention to the differences between Figure 3.17 and Figure 3.14. In Figure 3.14 we plotted the two transitions separately, and the transitions contained a range of slopes in the $x - \tau$ plane. Figure 3.14 failed to account for shocks, and so the plot put the slopes corresponding to initial conditions next to the final conditions; the results of this showed up in Figure 3.15. In Figure 3.17, the transitions are represented by a single slope, because the entire transition occurs instantly. Figure 3.18 shows what the these shockwaves look like as a breakthrough curve.

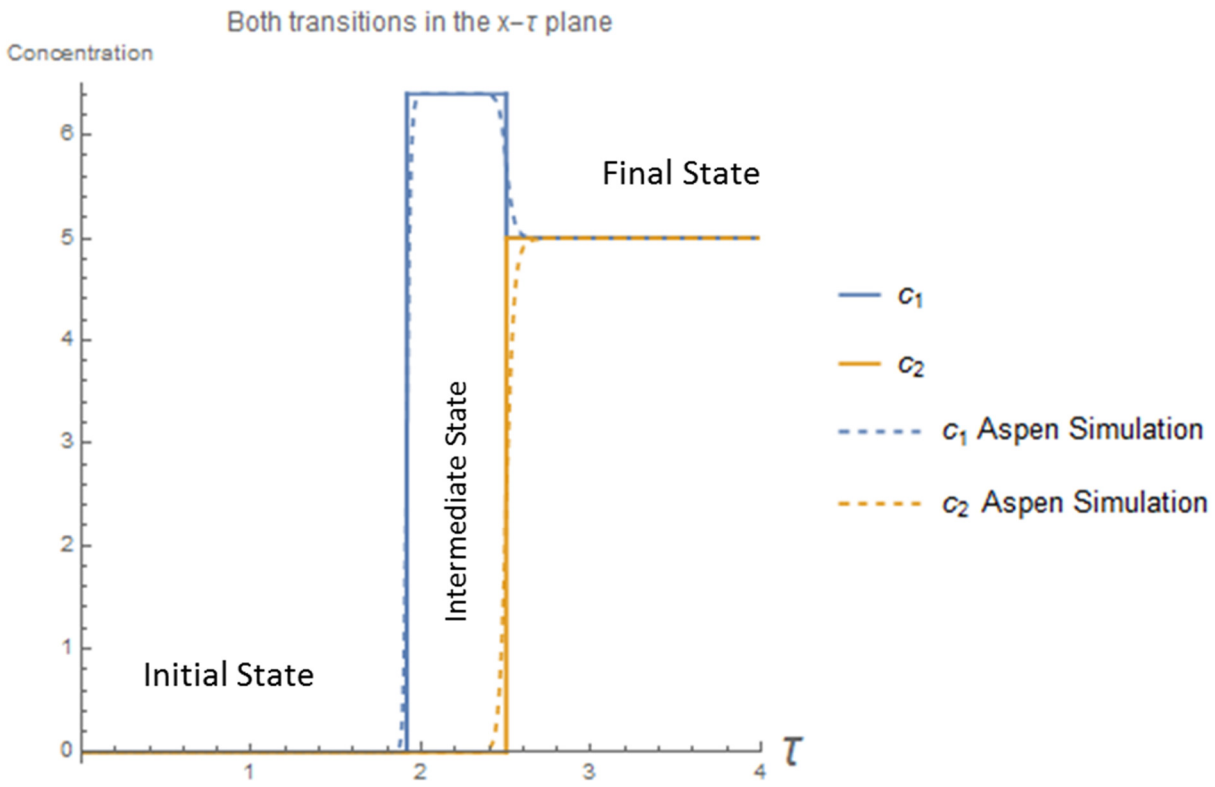


Figure 3.18 Breakthrough curves with shockwaves accounted for

3.3.6 Use the Steady State Information to Constrain Operating Conditions

At long last, we are now able to return to the constraints we were discussing in section 3.3.2. Armed with the new knowledge about transitions in the c_1, c_2 plane, we are now able to make some statements about the operating conditions in the SMB unit. As it turns out, we can place the entire SMB unit onto the same type of (c_1, c_2) plot that we used in the transition exercise. Specifically, the separation can be laid out in this way:

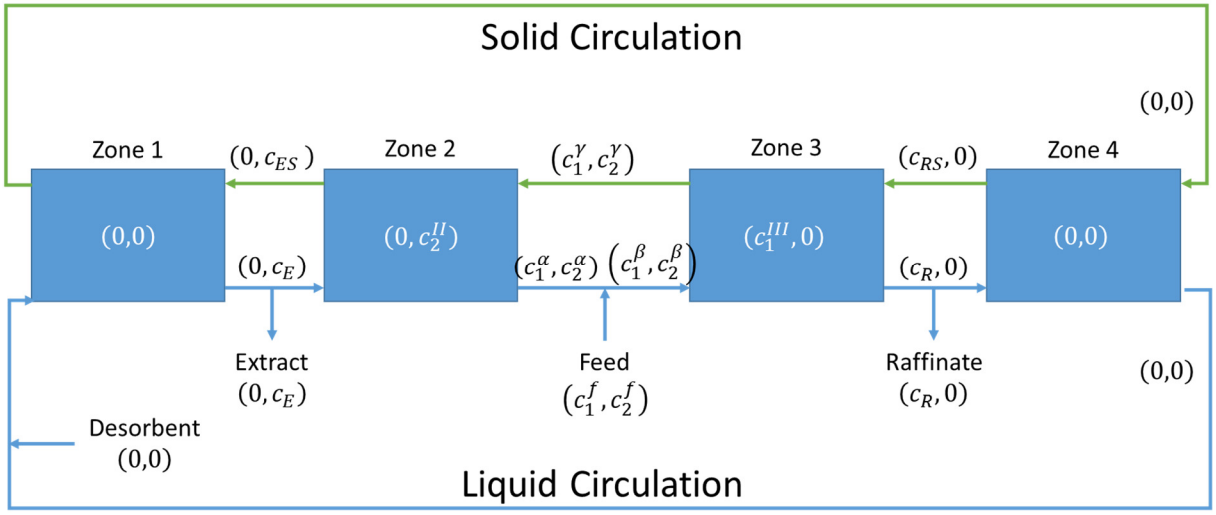


Figure 3.19 TMB operation with zone steady states labeled

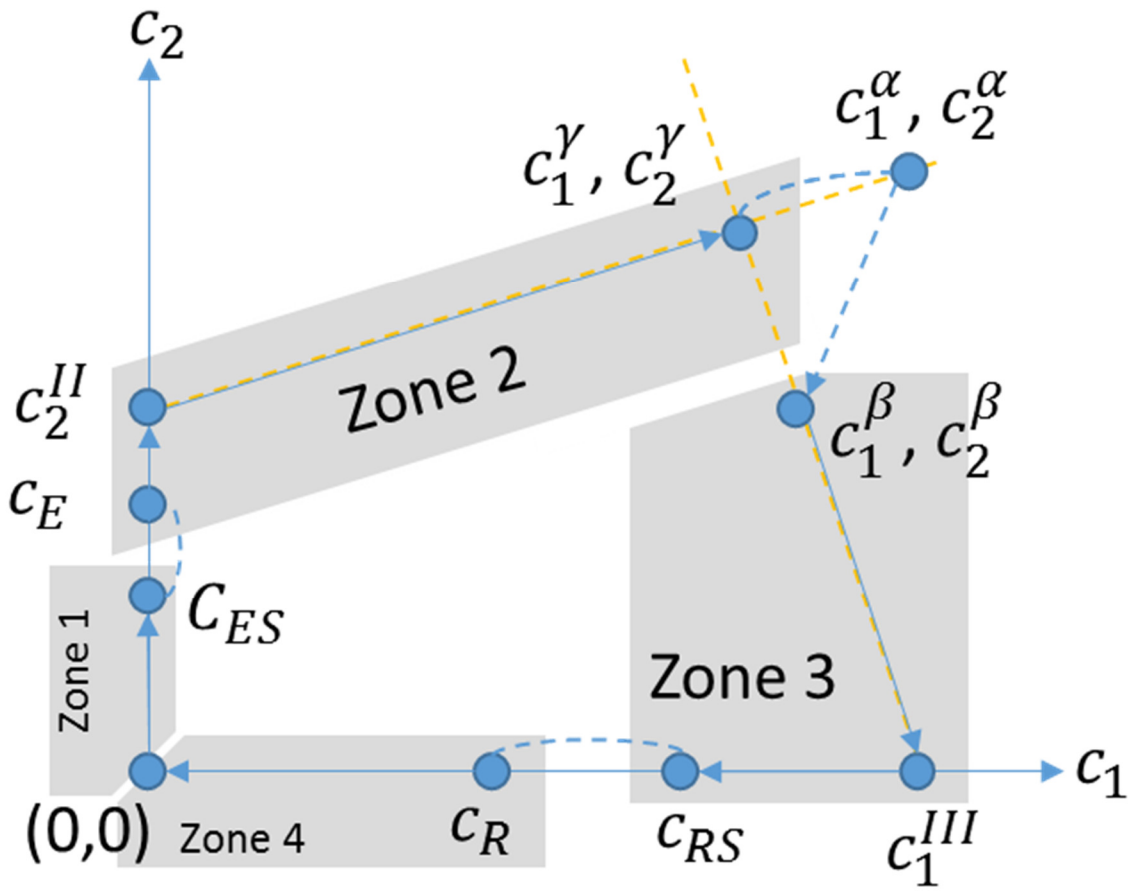


Figure 3.20 Transitions in a TMB column

Figure 3.19 shows what states will be achieved in a TMB column. Figure 3.20 shows how those states are laid out in the c_1, c_2 plane. The *most important* feature of these diagrams is to notice that all the states labeled in Figure 3.19 fall on the nice curves defined by the eigenvectors (i.e. the yellow dashed lines in Figure 3.20.) It is also important to note that the intermediate state achieved in zones 2 and 3 are intermediate states, and the concentration of one component is zero. Taken together, those features suggest a line of attack on this problem. Thanks to the exercise in section 3.3.5, we already have a general method for calculating c_2^{II} and c_1^{III} given $\left(c_1^{\gamma}, c_2^{\gamma}\right)$.

Once we have calculated c_2^{II} and c_1^{III} from $\left(c_1^{\gamma}, c_2^{\gamma}\right)$ we can use equations (3.20) (3.21) to solve for $\left(c_1^{\gamma}, c_2^{\gamma}\right)$ as a function of the feed concentrations, m_2, m_3 and the isotherm parameters. Ensuring that these concentrations are physically meaningful (i.e. real and positive) constitutes one set of constraints on this system, and it is important to note that these constraints are **not** functions of the raffinate or extract flow rates, and also do not depend on m_1 or m_4.

The second set of constraints comes from equation (3.56). Notice that the velocity of the concentrations in each zone is proportional to $(\theta - m)$ where θ is an eigenvector of the isotherm Jacobean. In order to actually achieve the intermediate state, we need to ensure that the velocity of all states past the intermediate state have positive velocities, so $(\theta_{state} - m) \geq 0$ and that the velocity of all states before the intermediate state have negative velocities, so $(\theta_{state} - m) \leq 0$. In practice, what

this means is that in order to achieve the intermediate state, we need to choose an m-value in such a way:

$$\sigma_2^{Int} \geq 0 \text{ and } \sigma_1^{Int} \leq 0 \quad (3.63)$$

$$\theta_2^{Int} \geq m \geq \theta_1^{Int} \quad (3.64)$$

We can see how this works in Figure 3.21. As in Exercise 1, the intermediate state is flanked by the two transitions, with boundaries defined by the two eigenvalues. If we decreased the m value, eventually σ_1^{Int} will become 0, and when that happens, the column will achieve the intermediate state at steady state.

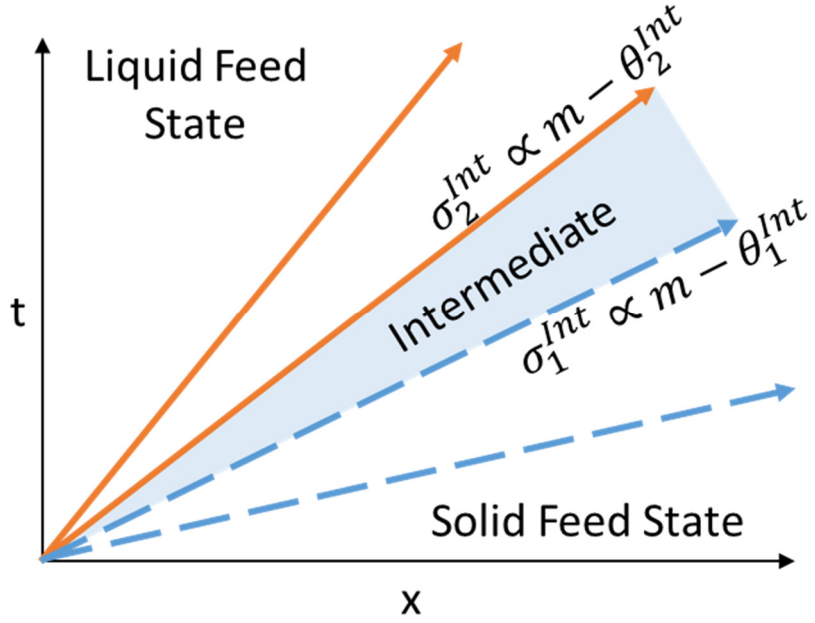


Figure 3.21 Velocities are proportional to $\mathbf{m - \theta}$

3.3.7 Exercise 2: Constructing the Constraints on the TMB System in Mathematica

3.3.7.1 Step 1: Define the Zone 2 and 3 Constraints

To do this, we simply need to tell Mathematica to use equation (3.64). That can be easily done with the following code

```
thetas=eigensIso[[1]]
zone2state={c1->0,c2->cII2}
zone2thetas=Simplify[thetas/.zone2state]
zone2constraint={ zone2thetas[[1]]∈Reals && zone2thetas[[2]]∈Reals &&
zone2thetas[[1]]<=m2<=zone2thetas[[2]]}
```

Here we set up the zone2constraint which consists of the inequality in (3.64) as well as a test to ensure that the θ values are real. We can do the same thing for zone 3, although we need to remember that as we saw in section 3.3.5, the transitions in this zone will be shockwaves. Therefore we need to use the θ_{shocks} instead of the eigenvalues.

```
zone3states={{cF₁->cIII₁,cF₂->0},{c0₁->cβ₁,c0₂->cβ₂}}
zone3theta=θshocks[[1]]/.Flatten[zone3states]
zone3constraint={zone3theta∈Reals&&m₃<=zone3theta}
```

Fortunately for us, Mazzotti 2006[4] has proven that the zone 3 constraints are always satisfied when the various other constraints are satisfied, and so there will be no need to use these relationships any further.

3.3.7.2 Step 2: Define the feed and c^γ constraints

The feed concentrations provide some constraints on the c^γ values indirectly. Specifically, the intermediate concentrations inside zones 2 and 3 cannot exceed the values predicted from the feed concentration. Therefore, c^γ values leading to steady state concentrations that are too large are forbidden, and we have an upper bound on the c^γ values. We can tell Mathematica this by calculating c_2^{II} and c_1^{III} from the feed concentrations, and setting the limit as follows:

```
relevantIntersections2[{0,0},{cF₁,cF₂}]
feedConstraint=Simplify[{cIII₁<=c₁,cII₂<=c₂}]/.Diagonal[%]
```

Naturally, the c^γ values must also be real and positive:

```
cγConstraints={cγ₁∈Reals && cγ₁>0, cγ₂∈Reals && cγ₂>0}
```

3.3.7.3 Step 3: Calculate the intersection concentrations as functions of c^γ

This step is fairly self explanatory. We use the relevantIntersections function to calculate the steady state concentrations for zones 2 and 3. The Diagonal function simply selects the c_1 and c_2 functions we care about

```
intersectionSolutions={cIII₁->c₁,cII₂->c₂}/.
Diagonal@relevantIntersections2[{0,0},{cγ₁,cγ₂}]
```

3.3.7.4 Step 4: Calculate the intersections as functions of the flow rate ratios, using mass balances

Now we can calculate the values a second way, through the mass balances on zones 2 and 3. This is very simply done with:

$$\text{balanceEq1} = -(m_3 - m_2)\text{cF}_2 == m_2\text{cII}_2 - \frac{a_2\text{cII}_2}{1 + b_2\text{cII}_2};$$
$$\text{balanceEq2} = (m_3 - m_2)\text{cF}_1 == m_2\text{cIII}_1 - \frac{a_1\text{cIII}_1}{1 + b_1\text{cIII}_1};$$
$$\text{balSolutions} = \text{Simplify@Solve}[\{\text{balanceEq1}, \text{balanceEq2}\}, \{\text{cII}_2, \text{cIII}_1\}]$$

Here we simply solve the quadratic equations for c_2^{II} and c_1^{III}. Unfortunately, we get four possible solutions to these equations, which we will have to keep track of.

3.3.7.5 Step 4: Combine the results of steps 3 and 4 to solve for c^γ

Now we have two separate equations for c_2^{II} and c_1^{III} that we can combine to find an equation for c^γ in terms of the flow rate ratios.

```
combinedEquations={
    (cIII₁/.intersectionSolutions)==cIII₁,
    (cII₂/.intersectionSolutions)==cII₂
    }/.balSolutions;
```

In this equation, we are starting from the equation $cIII_1 == cIII_1$ but replacing the left hand side with the solutions we got from the eigenvector curves, and the right hand side with the solutions from the mass balances. From this point, we need to simplify the resulting equations, then solve them for c^{γ}

```
simplifiedEquations=FullSimplify[combinedEquations,
Assumptions->{a₁<a₂,a₂>0}];


cγSolutions=First@Simplify@Solve[#,{cγ₁,cγ₂}]&/@simplifiedEquations
```

This input uses the advanced Mathematica technique called *pure functions*. Since we have four possible solutions from our mass balances, our simplifiedEquations variable contains four different pairs of equations. We want to solve each pair, so we define a function using # and the &

First@Simplify@Solve[#,{cγ₁,cγ₂}]&

This function is "pure" because we have not assigned it a name. Whatever we pass into this function will be placed in the position of the hashtag # symbol, and then the function will solve (and simplify, and take the first answer) that input. Since we want Mathematica to solve each pair of equations, we can do so by telling Mathematica to pass the pairs into the function one at a time with the code:

/@simplifiedEquations

The result will be four separate solutions for $\{c_1^{\gamma}, c_2^{\gamma}\}$

3.3.7.6 Step 5: View the regions specified by each constraint

First, we have to enumerate all the constrains on the system, and apply the relationships we derived in steps 3 and 4

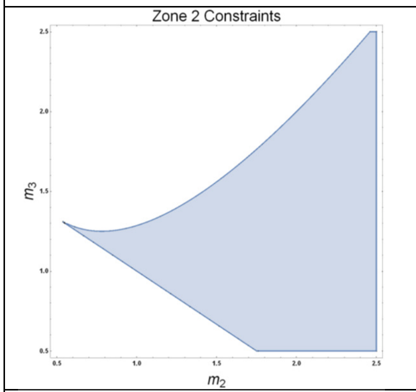
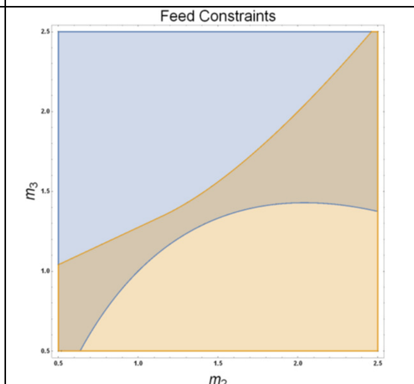
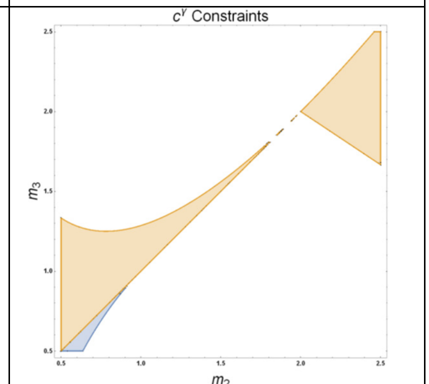
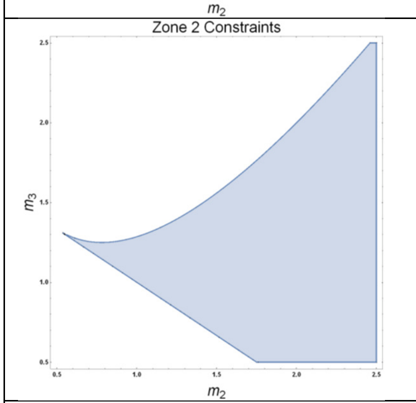
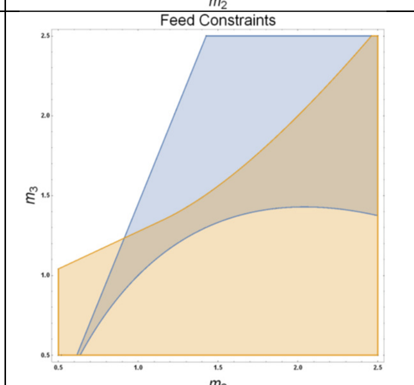
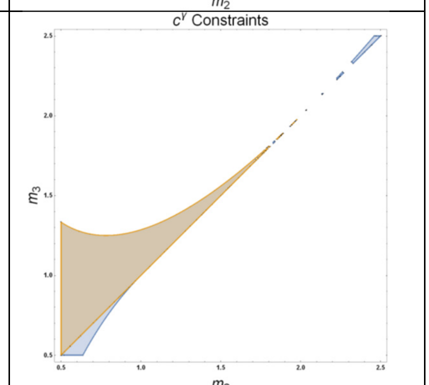
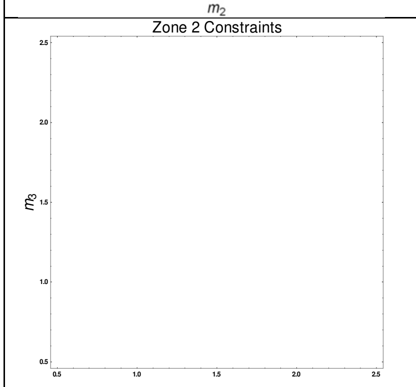
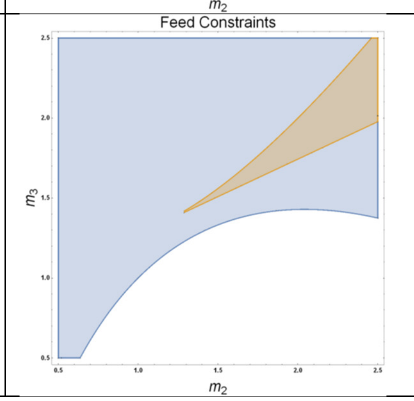
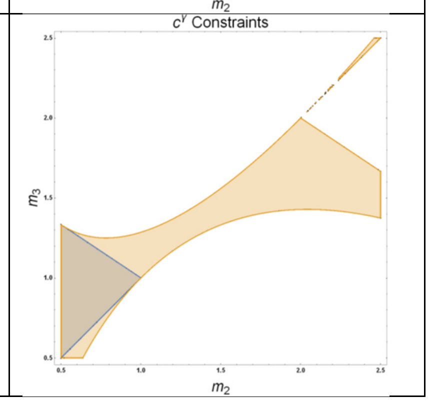
```
allConstraints={Simplify[zone2constraint/.balSolutions],
   feedConstraint/.balSolutions,
   cγConstraints/.cγSolutions};
```

Once we have these, we can plot them using Mathematica's RegionPlot function. First we can define our own custom plot function with

```
plotRegion[r_,label_]:=RegionPlot[r,{m₂,0.5,2.5},{m₃,0.5,2.5},
  PlotPoints->100,
  PlotLabel->label,
  FrameLabel->{m₂,m₃}
  ]
```

We can then assume some set of system parameters and plot the various regions. For convenience, we will use the same parameters as were chosen in Mazzotti 2006, so that we have a straightforward basis for comparison.

The results of our coding is shown in Figure 3.22. Each row in the table corresponds to a solution for c_2^{II} and c_1^{III} while each column represents one of the three constraints on the flow rate ratios. As we can see, the second row is the only solution where all three constraints overlap, and therefore represents our true solution.

Constraints from Zone 2	Constraints from the feed concentration	Positive real c^{Y} constraints
		
		
		

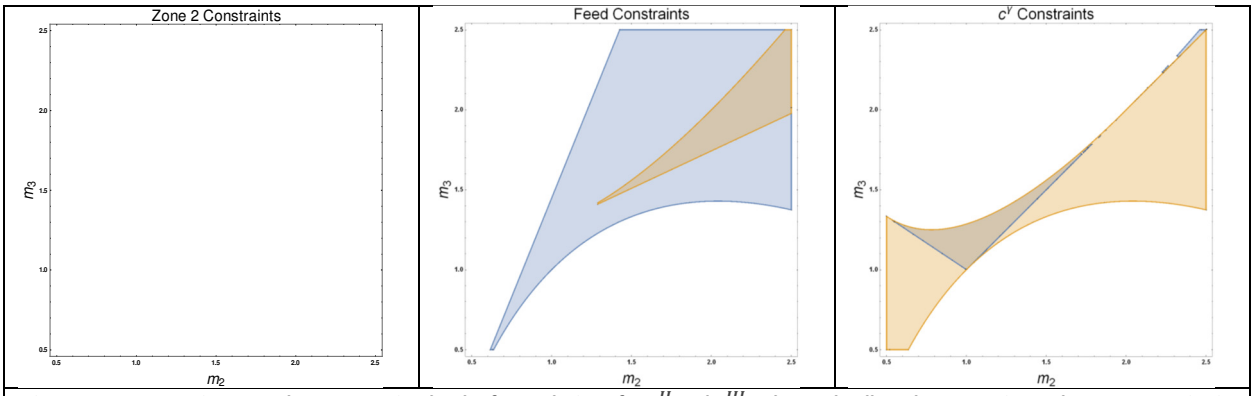


Figure 3.22: Constraints on column operation by the four solutions for c_2^{II} and c_1^{III}. Blue and yellow denote regions where a constraint is satisfied. When both colors are present, operation is only possible where the regions intersect.

Therefore, we can apply our plotting function to all three constraints at once, and get the final operating region:

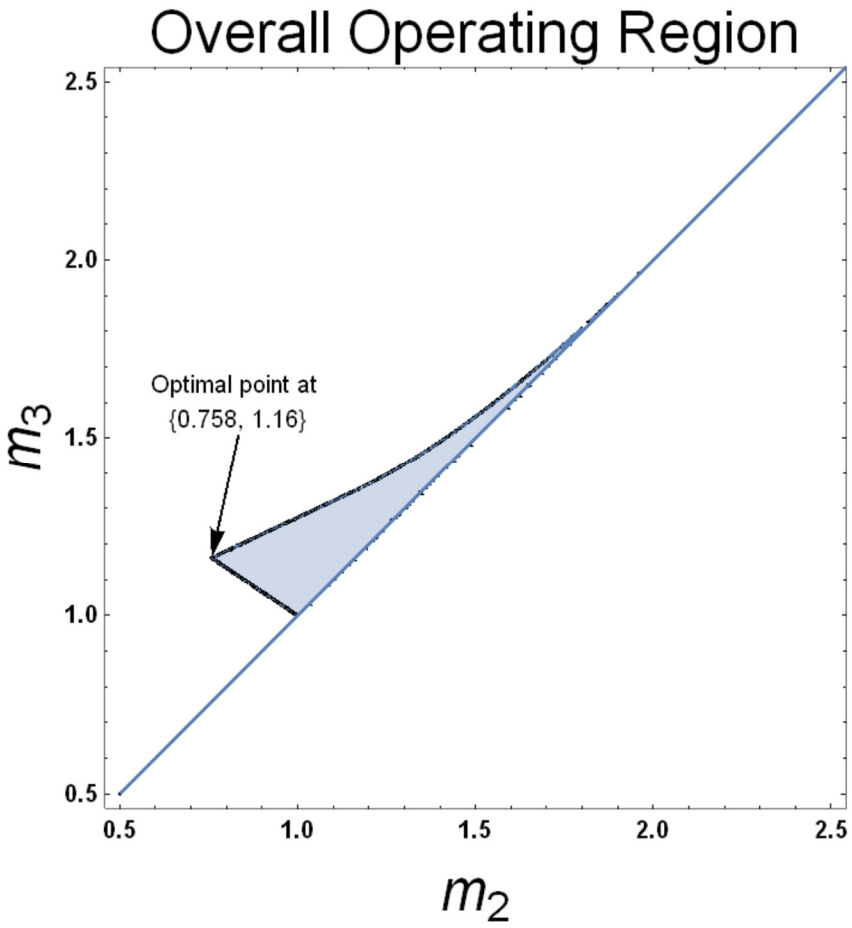


Figure 3.23: Region where all constraints are satisfied

With this result, we have completed the derivation of the triangle theory for the Langmuir isotherm. Given this diagram, it is easy to select the m_2, m_3 point which gives both the maximum feed flow rate (since the feed flow rate is proportional to $m_3 - m_2$) and complete separation (lies inside the blue region in Figure 3.23).

3.4 Standing Wave Design

Now that we have covered in detail the mathematics behind Triangle Theory, we can turn our attention to the alternative method known as "Standing Wave Design." Fortunately for us, Standing Wave Design leans heavily on the same mathematical principles as the Triangle Theory, so there is no need to reinvent the wheel. At the fundamental level, SWD is attempting to solve the same problem as Triangle Theory. Given some SMB system, how can we predict the optimal flow rates, or the flow rates that give complete separation? SWD only really diverges from the Triangle Theory at the very end of the analysis. Instead of using the relationships we described in section 3.3.6 to put analytical bounds on the m values, the authors of the SWD approach opted to employ an iterative solution approach.

3.4.1 Standing Wave Design in a Nonlinear Ideal System

We will start the explanation of SWD in the case of a Langmuir isotherm without significant mass transfer effects. This will allow us to easily see the parallels between the SWD approach and the ideas we developed in the Triangle Theory section.

The Standing Wave Design procedure for an ideal nonlinear system is shown below:

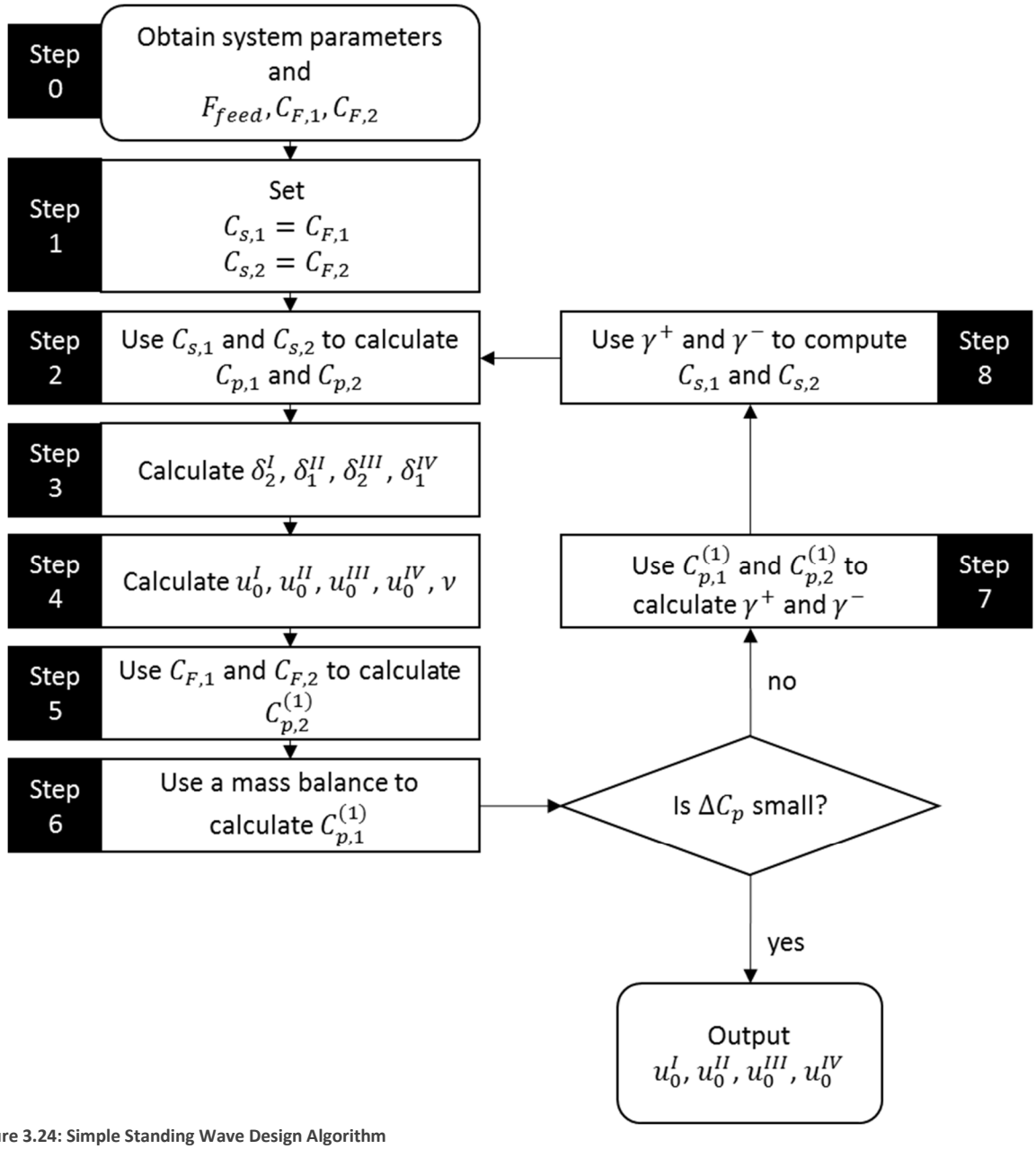


Figure 3.24: Simple Standing Wave Design Algorithm

Before proceeding, it is important to inventory the differences in notation and algorithm between this approach and the triangle theory approach discussed in section 3.3. When the SWD procedure discusses "plateau concentrations," they are talking about the steady-state concentrations reached in zones 2 and 3 that we discussed in the triangle theory derivation. Moreover, they combine the θ values

3.4.2 Notation Used in the Standing Wave Design Literature

Quantity	Symbol
Column Cross Section	A
Intercept of γ_+/γ_- Characteristic	A_+/A_-
First Langmuir Isotherm Parameter, Maximum Loading	a_i
Second Langmuir Isotherm Parameter	b_i
Concentration	C
Plateau Concentrations in Constant Regions	$C_{p,i}$
Dilution Concentration	$C_{s,i}$
Concentration of component i	c_i
Equilibrium liquid phase concentration	c_i^{eq}
Axial Dispersion	E_i
Volumetric flow rate of stream i	F_i
Henry's Constant for Component i	H_i
Mass Transfer Parameter	k_i
Column Length	L
Flow Rate Ratio in Zone k	m_k
Mass Flow Rate of the Liquid	$\dot{M}_L$
Mass Flow Rate of the Solid	$\dot{M}_s$
Solid phase loading	q
Volumetric Flow Rate of the Liquid	$\dot{Q}_L$
Volumetric Flow Rate of the Solid	$\dot{Q}_s$
Time	t
SMB Switching Time	t^*
Solid Linear Velocity	u_s
Liquid Interstitial Velocity in zone k	u_0^k
Target Yield of component i	Y_i
Liquid Linear Velocity	v
Dead Volume	V_D
Volume of Bed k	V_k
Column Axial Position	z
Retention Factor of component i in zone k	δ_i^k
Bulk density	ρ_b
Skeletal or Solid density	ρ_s
Envelope density	ρ_e
External or Inter-particle porosity	ε_i
Internal or Intra-particle porosity	ε_p
Total porosity	ε_t
Slope of the characteristic curves in the hodograph plane	$\gamma_+ \gamma_-$

3.4.2.1 Step 1

To begin the procedure, an initial guess of C_s is required, and various different methods for making this guess have been proposed[6,21,22]. In Simulated Moving Bed Chromatography for Chiral Separation[23], the authors propose that the feed concentration be used. Another proposed method[21] is to iteratively increase the initial guesses, starting from values that were much smaller than the feed concentrations. The reason these approaches work is because of the feed concentration limit mentioned in section

3.3.7.2. The feed state represents a boundary on the Zone 2 intermediate state, and this is shown visually in Figure 3.25.

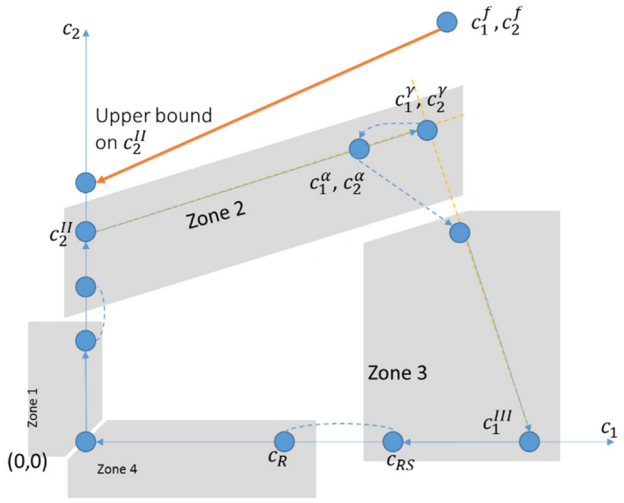


Figure 3.25: Feed State Imposes Upper Bound on c_2^{II}

Therefore, if we were trying to find some concentration state that satisfied our conditions, we could start from 0,0 and work upwards, or we could start from c^f and work inwards. It turns out that the feed concentration is always a better idea.

3.4.2.2 Step 2

This step introduces a procedure that is used here and again in in step 5. In this step, the roots of the equation

$$\frac{\partial q_2}{\partial C_1}\gamma^2 + \left(\frac{\partial q_2}{\partial C_2} - \frac{\partial q_1}{\partial C_1}\right)\gamma - \frac{\partial q_1}{\partial C_2} = 0 \tag{3.65}$$

are found and designated γ_+ and γ_-. This is the SWD nomenclature for the slopes of the characteristic curves in the (c_1, c_2) plane. We have already been able to calculate these in the triangle theory section, and there is nothing new in this formulation. The only difference is nomenclature, with $q_1$and q_2 being the isotherm equations.

In this step, $C_{s,1}$ and $C_{s,2}$ are used as the basis point for calculating the characteristic curves. As a reminder, these curves represent the transitions from the (c_{s1}, c_{s2}) state to the $(c_{p,1}, 0)$ and $(0, c_{p,2})$ states[2,24]. Therefore, finding the x and y intercepts of those lines allows us to come up with an estimate for $c_{p,1}$ and $c_{p,2}$. This procedure is conceptually identical to the first step of Exercise 2 in the triangle theory section, although care should be taken not to conflate the c^γ state from triangle theory with the C_s state from SWD, the C_s state is actually the SWD name for the c^β state. This is why the $c_{p,2}$ value is merely an estimate.

In the SWD literature, the intersections of the characteristic curves are found with the following procedure. First two parameters, A_+ and A_- are found with the equations

$$A_+ = \frac{(a_1 - a_2)}{a_2 b_1 + \frac{a_1 b_2}{\gamma_+}} \tag{3.66}$$

$$A_- = \frac{(a_1 - a_2)}{a_2 b_1 + \frac{a_1 b_2}{\gamma_-}} \tag{3.67}$$

Finally these equations update the C_p values:

$$C_{p,2} = -\frac{A_+}{\gamma_+} \tag{3.68}$$

$$C_{p,1} = A_- \tag{3.69}$$

As we have mentioned previously, these explicit equations are only possible because of the linearity of the Langmuir isotherm's characteristic curves. Regardless of the shape of the curves, however, the principle of finding the x and y intercepts of the two characteristic curves coming from the (c_{s1}, c_{s2}) point is consistent across isotherms, and the *Mathematica algorithms can find these intercepts even for nonlinear characteristics.*

3.4.2.3 Step 3

Now we begin to estimate wave velocities. In SWD, this is done by calculate δ values, which are referred to as retention factors for each zone. For a Langmuir isotherm, they are calculated with the equations[22]

$$\delta_2^I = \varepsilon_p + (1 - \varepsilon_p) a_2 \tag{3.70}$$

$$\delta_1^{II} = \varepsilon_p + (1 - \varepsilon_p) \frac{a_1}{1 + b_2 C_{p,2}} \tag{3.71}$$

$$\delta_2^{III} = \varepsilon_p + (1 - \varepsilon_p) \frac{a_2}{1 + b_1 C_{s,1} + b_2 C_{s,2}} \tag{3.72}$$

$$\delta_1^{IV} = \varepsilon_p + (1 - \varepsilon_p) \frac{a_1}{1 + b_1 C_{p,1}} \tag{3.73}$$

Retention factors can be derived in a very straightforward way for any isotherm[24]. Specifically, as we will see in the next step, they are a simple function of the eigenvalues of the isotherm matrix. In other words, we have already calculated these values by calculating θ_1 and θ_2 during our triangle theory derivation.

3.4.2.4 Step 4

To find the zone flow rates and interstitial velocity, we have to solve a system of 5 equations with 5 unknowns. Specifically, we have:

$$u_0^I = v\left(1 + \frac{1 - \varepsilon_b}{\varepsilon_b} \delta_2^I\right) \tag{3.74}$$

$$u_0^{II} = v(1 + \frac{1 - \varepsilon_b}{\varepsilon_b} \delta_1^{II}) \tag{3.75}$$

$$u_0^{III} = v(1 + \frac{1 - \varepsilon_b}{\varepsilon_b} \delta_2^{III}) \tag{3.76}$$

$$u_0^{IV} = v(1 + \frac{1 - \varepsilon_b}{\varepsilon_b} \delta_1^{IV}) \tag{3.77}$$

$$\frac{F_{feed}}{\varepsilon_b \pi r^2} = u_0^{III} - u_0^{II} \tag{3.78}$$

It is no accident that these equations resemble equation (3.54). Indeed, these equations expose the retention factors as simple functions of the eigenvalues. However, care should be taken, since in zones where shocks occur (e.g. zone 3 for the Langmuir isotherm) we still need to replace the proper eigenvalue equation with the difference equation from (3.62)

3.4.2.5 Step 5

In this step, we compute a new value for $C_{p,2}$ by estimating it from the feed concentration, using the same procedure as in Step 2. As we mentioned earlier in , the $C_{p,2}$ value calculated from the feed concentration represents an upper bound, and so we take that upper bound as our estimate here.

3.4.2.6 Step 6

In this step, we compute a new value for $C_{p,1}$ based on the mass balance. There is nothing too remarkable here, except that the consequences of our decisions in the previous step have been made clear. By allowing the contamination of the extract and raffinate streams, we can no longer ignore the concentrations of the extract and raffinate streams, as we did in the triangle theory derivation. Impurities mean that all the zones are coupled, and we can not get away with mathematically isolating zones 2 and 3 from zones 1 and 4.

$$C_{p,1} = \frac{C_{F,1} F_{feed} Y_1}{F_{raf}} \tag{3.79}$$

3.4.2.7 Tolerance Check

At this point, we can compare how much the $C_{p,1}$ and $C_{p,2}$ values have changed by taking the difference between the values found in step 2, and the values found in steps 5 and 6. If the change is small enough (i.e. our estimates were close enough), we can stop iterating, otherwise we continue to step 7

3.4.2.8 Step 7

Using our new estimates for the peak concentration, we calculate a new pair of γ values with a different method from step 2. These are once again the slopes of the transitions, except this time we are interested in the transition from $(C_{p,1}, C_{p,2})$ to $(C_{s,1}, C_{s,2})$ For a Langmuir system, the slopes are

$$\gamma_+ = \frac{a_2 - a_1 - a_1 b_2 C_{p,2}}{a_2 b_1 C_{p,2}} \tag{3.80}$$

$$\gamma_- = \frac{a_1 b_2 C_{p,1}}{a_1 - a_2 - a_2 b_1 C_{p,1}} \tag{3.81}$$

these can easily be derived by simplifying the eigenvectors when one component goes to zero.

3.4.2.9 Step 8

With our new γ values from step 7, we can now estimate a new pair of C_s values using the following equations, derived for the Langmuir isotherm. *This is the most difficult step for isotherms with non-linear characteristics,* because it involves finding the intersection of two curves, the γ_+ characteristic coming from $(0, C_{p,2})$ and the γ_- characteristic coming from $(C_{p,1}, 0)$. Fortunately, it is almost always possible to solve this numerically if analytic solutions are not forthcoming. Note however, that the intersection is only part of this equation. If we did simply calculate the intersection, then when we returned to step 2, the γ values calculated there would be identical, and lead us back to the exact same C_p state.

$$C_{s,1} = \frac{a_2 b_1 \gamma_+ \gamma_- (a_1 - a_2)}{a_1 a_2 b_1 b_2 (\gamma_+ + \gamma_-) + {a_1}^2 {b_2}^2 + {a_2}^2 {b_1}^2 \gamma_+ \gamma_-} \tag{3.82}$$

$$C_{s,2} = \frac{a_1 b_2 (a_2 - a_1)}{a_1 a_2 b_1 b_2 (\gamma_+ + \gamma_-) + {a_1}^2 {b_2}^2 + {a_2}^2 {b_1}^2 \gamma_+ \gamma_-} \tag{3.83}$$

From this point, we return to step 2.

3.4.3 Standing Wave Design in a System with Nonlinear Isotherm

So far, we have not really seen the power of the standing wave design system. SWD really shines in systems with significant mass transfer and dispersive effects. When those are present, the algorithm expands to include iterative estimates of the velocities, as shown in Figure 3.26.

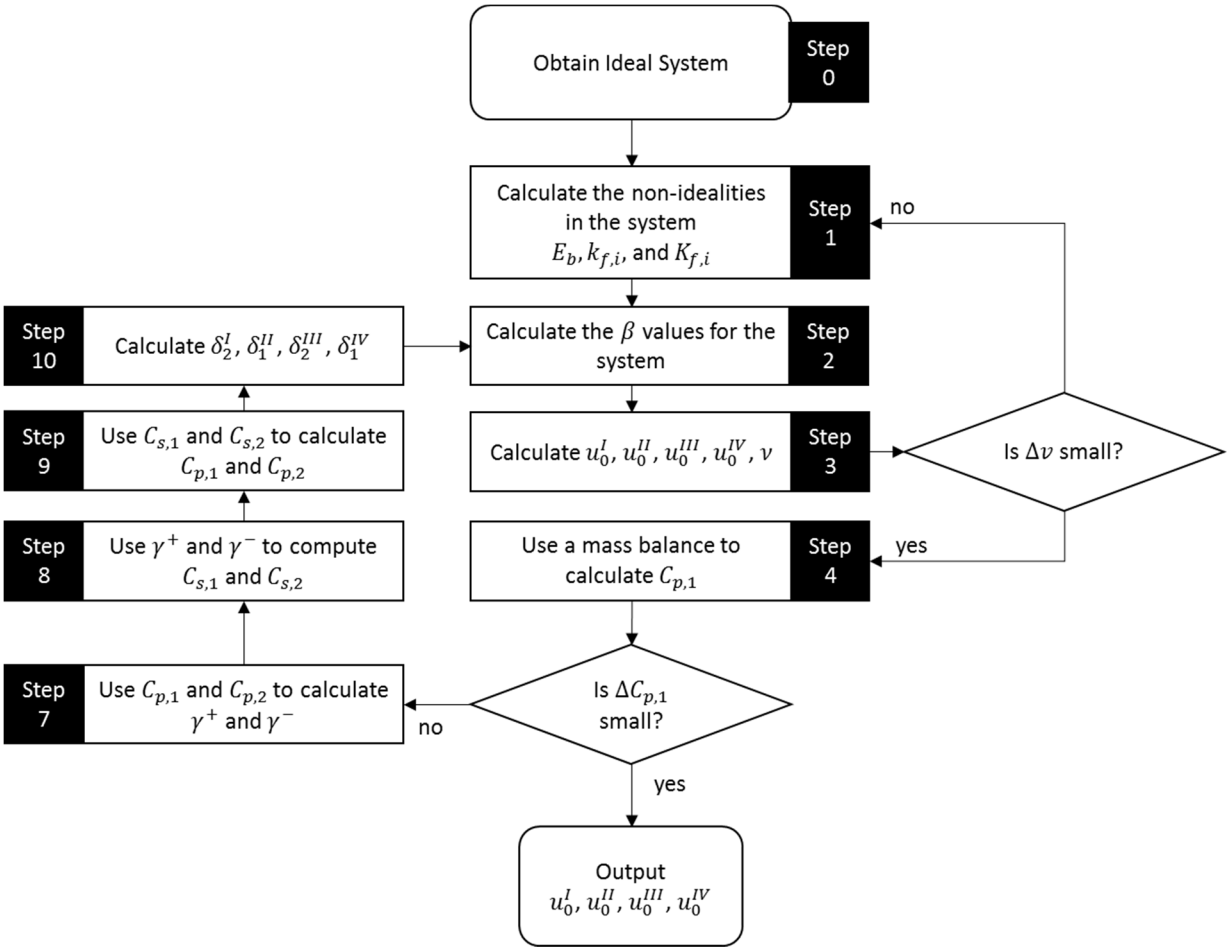


Figure 3.26: Standing Wave Design in a Nonlinear Nonideal System

In this expanded version of the algorithm, steps 4 through 10 are exactly the same as their ideal counterparts, i.e. the system without mass transfer resistance. We will therefore only cover steps 0-3 here.

3.4.3.1 Step 0

In this step, we carry out the SWD as though the system were ideal, to give us a starting estimate of the mass transfer resistance.

3.4.3.2 Step 1

It is assumed that the mass transfer resistances and dispersion coefficients may be functions of the fluid velocities. If that is the case, then in this step, we use the estimates of the fluid velocities to calculate them.

3.4.3.3 Step 2

This is the beginning of the key contribution of the standing wave design. What happens here is that instead of sticking with the negligible dispersion assumption that we used very early on, we now turn

our attention to a steady state analysis of the differential equations. Specifically, we attempt to solve the equation

$$(u_o - (1 + P\delta)v)\frac{dc_i}{dx} = \left(E_b + \frac{P\,v^2\,\delta^2}{K_f}\right)\frac{d^2c_i}{dx^2} \tag{3.84}$$

As we might expect, this equation has a solution that looks like an exponential decay in the x direction. Because of this, it is possible to write the solution in the form of

$$\frac{c_i|_{x=0}}{c_i|_{x=L}} = \exp\left[\frac{(1 + P\,\delta)v - u_0}{E_b + \frac{P\,v^2\,\delta^2}{K_f}}L\right] \tag{3.85}$$

Each zone has its own special exponential decay, but they can all be written in this form. Therefore, the velocity difference that we care about (i.e. $(1 + P\,\delta)v - u_0$) has a relationship with $\ln\left[\frac{c_i|_{x=0}}{c_i|_{x=L}}\right]$. This is where the β values come into play. Because the exponential decay functions only decrease along a column, the concentration ratio inside the natural log will end up being the ratio of the largest and smallest concentrations. This is where the β values come into play. By using the information we get from the characteristic transitions, we already have estimates of the smallest and largest concentrations in each column. The β values in each zone are therefore chosen to correspond to the component whose wave we want to have stand, and are defined as follows:

$$\beta_2^I = \ln\left(\frac{C_{E,2}}{C_2^I}\right) \tag{3.86}$$

$$\beta_1^{II} = \ln\left(\frac{C_{s,1}^*}{C_{E,1}}\right) \tag{3.87}$$

$$\beta_2^{III} = \ln\left(\frac{C_{s,2}}{C_{R,2}}\right) \tag{3.88}$$

$$\beta_1^{IV} = \ln\left(\frac{C_{R,1}}{C_2^I}\right) \tag{3.89}$$

3.4.3.4 Step 3

In this step, we leverage the β values together with (3.85) to generate the equations which allow us to calculate the superficial liquid velocities that make this new concentration wave—which takes into account nonidealities—stand where we want it to. Those equations are:

$$u_0^I = (1 + P\,\delta_2^I)v + \frac{\beta_2^I}{L^I}\left(E_b^I + \frac{P\,v^2\,(\delta_2^I)^2}{K_f^I}\right) \tag{3.90}$$

$$u_0^{II} = (1 + P\,\delta_1^{II})v + \frac{\beta_1^{II}}{L^{II}}\left(E_b^{II} + \frac{P\,v^2\,(\delta_1^{II})^2}{K_f^{II}}\right) \tag{3.91}$$

$$u_0^I = (1 + P\,\delta_2^{III})v + \frac{\beta_2^I}{L^{III}}\left(E_b^{III} + \frac{P\,v^2\,(\delta_2^{III})^2}{K_f^{III}}\right) \tag{3.92}$$

$$u_0^I = (1 + P\,\delta_1^{IV})v + \frac{\beta_1^{IV}}{L^{IV}}\left(E_b^{IV} + \frac{P\,v^2\,(\delta_1^{IV})^2}{K_f^{IV}}\right) \tag{3.93}$$

3.5 Remarks

Implementing the SWD algorithm in Mathematica is relatively straightforward. The most difficult parts have already been discussed in the Triangle Theory section, and so there is little more than algebra left

to distinguish the implementations. The Mathematica code for working with the different isotherms is provided in the Appendix, as described in Table 3.2.

Table 3.2: Locations of SWD Documents

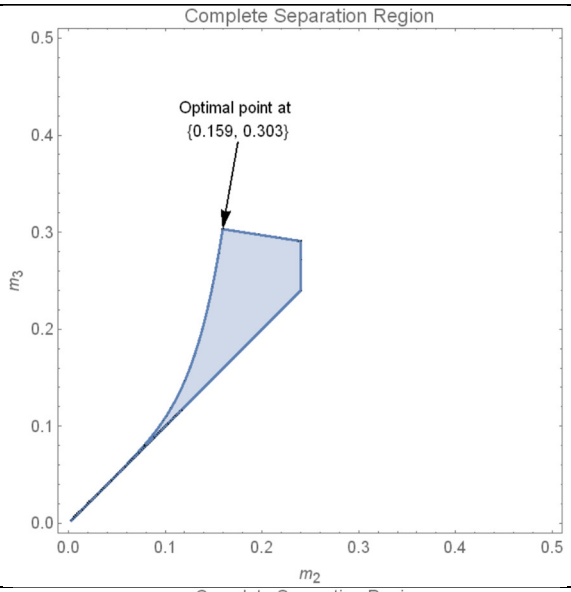
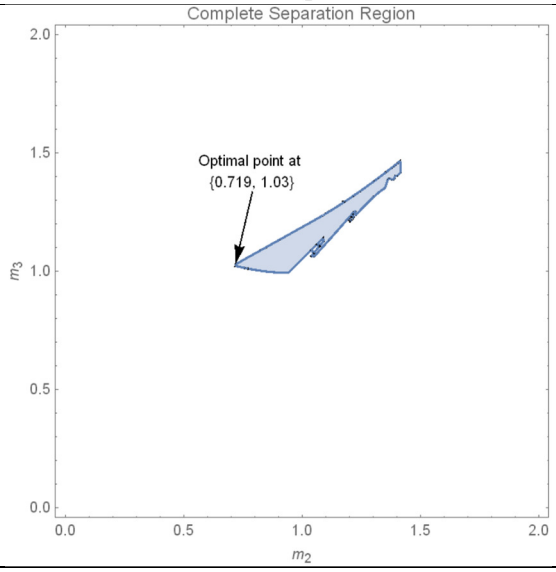
Isotherm	Location of Corresponding Mathematica Code
Freundlich Isotherm	Appendix B.1
Sips Isotherm	Appendix B.2
Toth Isotherm	Appendix B.3
Langmuir Isotherm (Ideal)	Appendix B.4
Langmuir Isotherm (Non-Ideal)	Appendix B.5

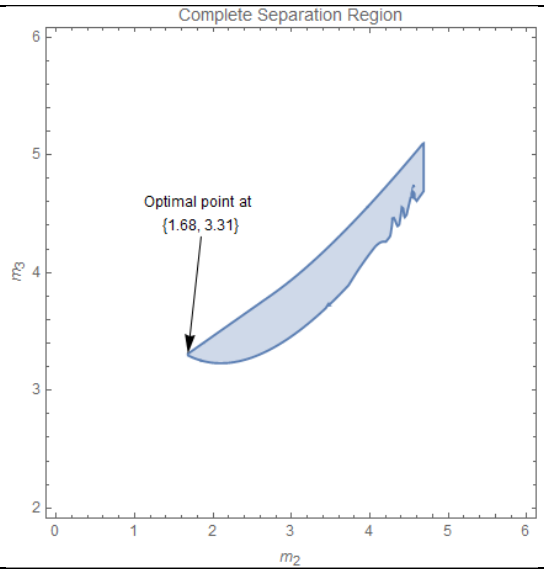
4 Application of Mathematica Tools

Following the same development procedure from section 3, it is possible to create Mathematica documents that can handle Triangle Theory and Standing Wave Design of isotherms whose design procedures have not yet been discussed in the literature. Specifically, we can derive algorithms for the various isotherms described in section 2.3: the Freundlich, Sips, and Toth isotherms. Not only do the Mathematica documents presented in the appendix allow for the calculation of operating regions not previously discussed in literature, they are easily extensible and can easily be modified to cover even more exotic or custom isotherms if required.

4.1 The Triangle Theory Method Applied to Non-Langmuir Isotherms

Table 4.1: Sample Operating Regions

Isotherm Form	Sample Operating Region	Parameters Used to Generate Region
Freundlich Isotherm Appendix A.2		$h_1 = 0.1$ $h_2 = 0.2$ $t = 1.3$ $c_1^F = c_2^F = 5g/L$
Sips Isotherm Appendix A.3		$a_1 = 1$ $a_2 = 2$ $b_1 = 0.1$ $b_2 = 0.2$ $t = 0.9$ $c_1^F = c_2^F = 5g/L$

Toth Isotherm Code Available in Appendix A.4		$b_1 = 0.07$ $b_2 = 0.2$ $t = 0.5$ $q_s = 71$ $c_1^F = c_2^F = 5g/L$

As we can see in Table 4.1, Mathematica is capable of calculating the operating region for these isotherms. The code used to generate these plots is provided in the Appendix. Appendix A.4 contains the code for a user interface that makes it easy to input the system values. Taken together, these documents provide a friendly GUI designed to make it easy for new users to apply the triangle theory to their design problems. In section 4.2, we will look at how this interface might be used to create a preliminary system design for a real-world separation.

4.2 Application of the User Interface for SMB Design

Suppose that we wished to design an SMB system for the separation of the enantiomers of 1-phenyl-1-propanol on cellulose tribenzoate. Kaczmarski et al[19] report that the best isotherm for describing the equilibrium of these species is the competitive Toth isotherm, despite their efforts to fit the separation to a Bi-Langmuir model. In other words, it appears that the cellulose tribenzoate adsorbent has a distribution of pores consistent with the Toth isotherm's assumptions, rather than the Langmuir or Bi-Langmuir assumptions. Due to the lack of good tools for performing Triangle Theory or Standing Wave Design, we may be tempted to use a Langmuir model for this adsorption purely out of convenience, and indeed some authors did exactly that, albeit in a supercritical phase not considered by Kaczmarski.[25]

Fortunately, the tools provided in this document do allow us to design an SMB system that is characterized by the competitive Toth isotherm, and so we may now use the isotherm which describes our system the best, instead of being forced to choose the isotherm that is compatible with the design methods we wish to use.

4.2.1 Workshop: Calculating the Triangle Theory Region for a System with a Toth Isotherm

Using the user interface generated in Mathematica, it is a simple matter to generate the triangle theory plots for this Toth isotherm. Kaczmarski et al.[25] report the isotherm parameters given in Table 4.2.

Table 4.2: Toth Isotherm Parameters from Kaczmarski et al.[25]

Parameter	Value
q_s	71
b_1	0.06
b_2	0.09
v	0.77

In addition to these parameters, we need to specify a feed concentration. Since the Kaczmarski paper derived the isotherm parameters from racemic mixtures with concentrations of less than $5g/L$ we will select a feed concentration of $c_1 = c_2 = 5g/L$ for our hypothetical SMB column.

4.2.1.1 Step 1: Calculate the Region of Complete Separation With Mathematica Tools

We can put these values directly into the user Mathematica interface to plot the operating region, as shown in Figure 4.1. The numbers can be directly entered into the input boxes, or the user can drag the sliders to adjust the values.

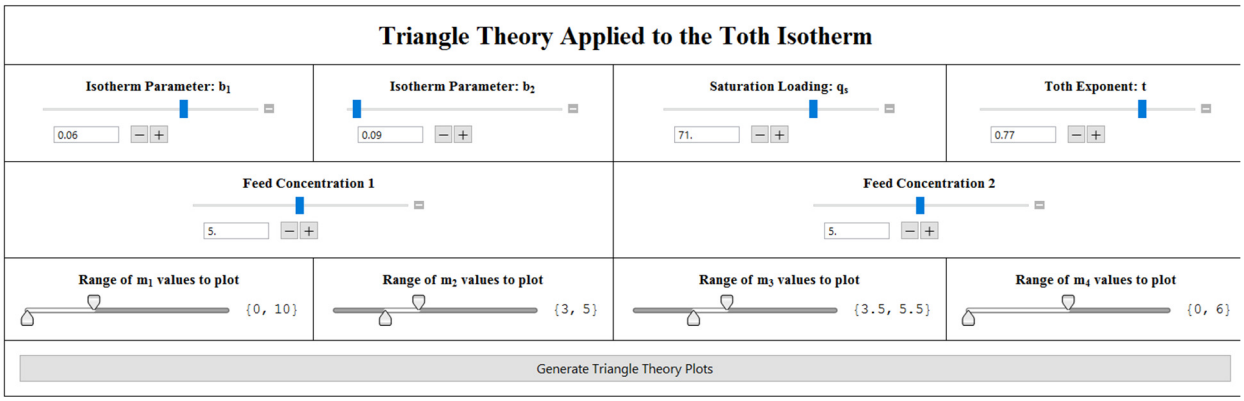


Figure 4.1: Inputting the values into the Mathematica User Interface

Once we click the button labeled “calculate triangle region” we will be presented with the following figure:

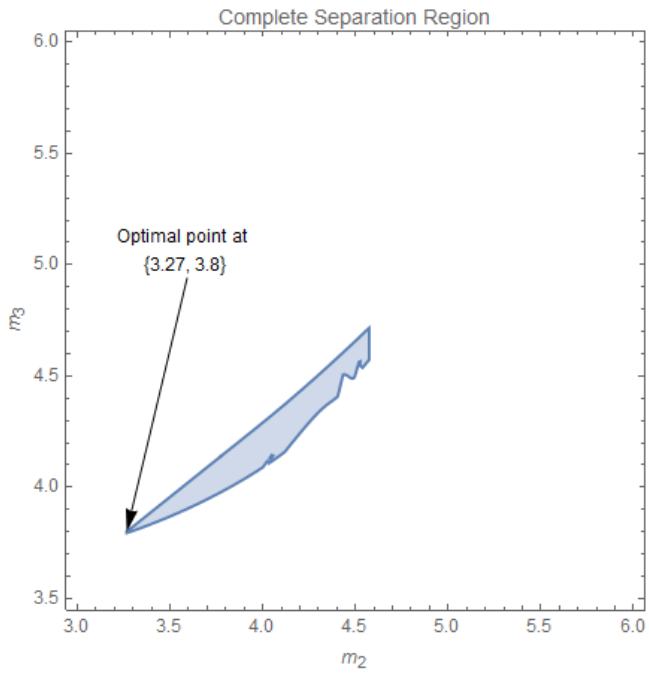


Figure 4.2: Optimum Operating Region in the m_2, m_3 plane

As we see, this is a very simple procedure for the user, and is accessible to everyone. Naturally, however, we have not yet completely designed our system. In order to finish the design of this system, we will need to also calculate the constraints in the m_4, m_1 region, so that we will know what our desorbent requirements will be. Fortunately, our user interface already does this for us. If we recall from section 3.3.7, the constraints on zones 1 and 4 are functions of all four flow rate ratios, m_1, m_2, m_3, m_4. Therefore, by choosing the optimal point from the m_2, m_3 plane, we can calculate the corresponding region in the m_4, m_1 plane and calculate the optimal conditions for zones 1 and 4. This region is shown in Figure 4.3, and is automatically generated using the optimum point from the m_2, m_3 region.

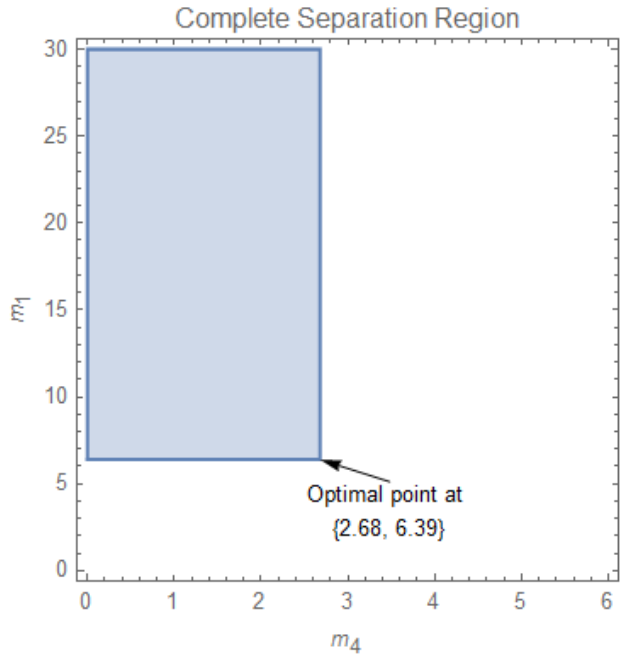


Figure 4.3: Optimum Operating Region in the m_4,m_1 plane

Therefore, the results of our Triangle Theory design is that the optimum operating conditions for our column can be calculated from the flow rate ratios in Table 4.3.

Table 4.3: Optimum Flow Rate Ratios

Flow Rate Ratio in Zone	Value
m_1	6.39
m_2	3.27
m_3	3.80
m_4	2.68

To convert these values into useful operating procedures, we follow this procedure.

4.2.1.2 Step 2: Choose a Pressure Drop Correlation

In SMB chromatography, it is common to use a linear pressure drop correlation. Specifically, the equation would have the form shown in (3.93) with ϕ being an experimentally determined parameter.

$$\frac{\Delta P}{l} = \phi \frac{q}{A} \tag{4.1}$$

Equation (3.93) tells us that the pressure drop over a given length is linearly related to the superficial velocity, which is the volumetric flow rate q, divided by the cross sectional area A. From this point, we solve the definition of the flow rate ration in equation (3.7) for the flow rate to get

$$q = \frac{v_{bed}\left(\varepsilon_p + \left(1-\varepsilon_p\right)m\right)}{t_{switch}} \tag{4.2}$$

Substituting this equation into our pressure drop correlation gives

$$\Delta P_{bed} = \frac{\phi\, l^2}{t_{switch}}\left(\varepsilon_p + \left(1-\varepsilon_p\right)m\right) \tag{4.3}$$

which tells us that the pressure drop in the bed is related to the flow rate ratio in the bed, as well as the switching interval. If we sum up all the beds, we will obtain the overall pressure drop in the column. Moreover, since we know that the each SMB zone has a constant flow rate ratio, the sum of equation (3.93) applied to each zone gives:

$$\Delta P_{max} = \frac{\phi\, l^2}{t_{switch}} \sum_{i=1}^{4} beds_i * \left(\varepsilon_p + \left(1-\varepsilon_p\right) m_i\right) \tag{4.4}$$

where $beds_i$ is the number of fixed beds in each zone.

4.2.1.3 Step 3: Choose System Geometry

For this hypothetical system, we will choose conventional values for a lab-scale system. In practice, the system dimensions may be chosen based on equipment availability, or they may be calculated to provide a target throughput. Table 4.4 lays out the choices for our hypothetical system.

Table 4.4: Column Geometry

Column Configuration	**1-2-2-1**	
Column Length (cm)	**L**	**15**
Column Diameter (cm)	**Dia**	**1.15**
Bed voidage (m^3 void /m^3 bed)	$\boldsymbol{\varepsilon_p}$	**0.68**
Maximum Pressure Drop	$\boldsymbol{\Delta P_{max}}$	**40 bar**
Pressure Drop Coefficient	$\boldsymbol{\phi}$	**0.1**

4.2.1.4 Step 4: Solve the Equations

By rearranging (3.93), we can calculate t_{switch}

$$t_{switch} = \frac{\phi\, l^2}{\Delta P_{max}} \sum_{i=1}^{4} beds_i * \left(\varepsilon_p + \left(1-\varepsilon_p\right) m_i\right) \tag{4.5}$$

Once we have found the switching interval, we can use it to solve the m-value equations for the inlet and outlet flow rates using the following equations:

$$\text{desorbent flow rate} = \frac{\text{vol}_{\text{bed}}\left(1-\epsilon_{\text{p}}\right)(\text{m}_1 - \text{m}_4)}{\text{tswitch}} \tag{4.6}$$

$$\text{extract flow rate} = \frac{\text{vol}_{\text{bed}}\left(1-\epsilon_{\text{p}}\right)(\text{m}_1 - \text{m}_2)}{\text{tswitch}} \tag{4.7}$$

$$\text{feed flow rate} = \frac{\text{vol}_{\text{bed}}\left(1-\epsilon_{\text{p}}\right)(\text{m}_2 - \text{m}_3)}{\text{tswitch}} \tag{4.8}$$

$$\text{raffinate flow rate} = \frac{\text{vol}_{\text{bed}}\left(1-\epsilon_{\text{p}}\right)(\text{m}_3 - \text{m}_4)}{\text{tswitch}} \tag{4.9}$$

Inserting the values from Table 4.3 and Table 4.4 allows to calculate the operating conditions for our hypothetical SMB column. Performing the calculations yields the results in Table 4.5:

Table 4.5: Operating Conditions from Triangle Theory

Quantity	Units	Value
t_{switch}	min	9.2
Desorbent flow rate	ml/min	1.21
Extract flow rate	ml/min	1.01
Feed flow rate	ml/min	0.17
Raffinate flow rate	ml/min	0.37

5 Optimization of SMB systems

5.1 Explanation of Optimization Variables and Solution Strategies

5.1.1 Analytic Solutions

As explained earlier, the Triangle Theory finds an explicit and potentially analytic solution for the optimum operating conditions in terms of a small set of system variables. Standing Wave Design also provides a quick method to calculate the optimum operating conditions given a certain set of assumptions and targets. However, these do not allow us to explicitly determine the optimum column configuration, or other system variables that are not defined in the TMB approximation. One large class of variables that falls into that category are those of so called "operational modes." Finding the optimum design in the space of the variables not present in the TMB approximation requires iterative methods.

5.1.2 Optimization Variables

In the big picture, optimization refers to an economic optimization. In practice, this generally involves a handful of performance indices. Specifically, those indices are:

- Product purity
- Product recovery
- Productivity (i.e. the ratio of product flowrate to the mass of the adsorbent)
- Desorbent consumption (i.e. the ratio of the desorbent waste to the product flowrate.)

5.1.3 Optimization in Aspen Chromatography

In order to optimize these variables in Aspen Chromatography, some care is required. Each of the variables requires that we take averages over a switching interval. However, Aspen provides us with several ways to take these averages. In our experience, the correct ways to define each of these variables is outlined below, and ACM code is provided.

5.1.3.1 Product Purity

For a component "A" and a stream "Stream," the purity "PurityA" can be calculated as below. "Total_Component" is a running total of the mass of component A that has passed through the stream. "Total_Mass" is a running total of the total mass of the components (excluding the solvent) that has passed through the stream. The following code should be placed in the "flowsheet constraints section"

```
PurityA as Fraction;
PurityA = Max(Min(Stream.Total_Component("A") /
Max(Stream.Total_Mass,0.0000001),1),0);
```

Naturally, this alone is not enough. This code will effectively take an average over the entire simulation history. To restrict the average to a single cycle, we need to also add the following to an ACM task:

```
Stream.Total_Component("A") : 0;
Stream.Total_Mass : 0;
```

And the task must be set to “Run Always” and once the values are reset, it should “Wait SwitchTime” before resetting the values once more. The Min and Max functions in the ACM code are to prevent over- and under-flows in the purity value when the total values are very close to zero.

5.1.3.2 Product Recovery

For a component “A,” a waste stream “Stream,” and a feed stream F, the recovery “RecoveryA” can be calculated as below. Note that we record the recovery as 1-%loss, this is to reduce the sensitivity to the “communication interval,” or the frequency that data is recorded during the aspen Simulation.

“Total_Component” is a running total of the mass of component A that has passed through the stream. “Total_Mass” is a running total of the total mass of the components (excluding the solvent) that has passed through the stream. The following code should be placed in the “flowsheet constraints section”

```
RecoveryA as Fraction;
RecoveryA = Min(1 – (Stream.Total_Component("A")) /
Max(F.Total_Component("A"),0.0000001),1);
```

As with the purity calculations, we must also reset the variables in these equations to 0 each switching interval.

```
Stream.Total_Component("A") : 0;
F.Total_Component("A") : 0;
```

5.1.3.3 Productivity

The productivity is easily calculated using the same techniques as the purity and recovery. The quantity of the adsorbent can be found in a variety of ways, but here we simply use the total column volume.

```
ProductivityA as MassFlow;
ProductivityA = (Stream.Total_Component("A") / SMB.Switch_Time) / (SMB.Hb *
SMB.Db^2 * 3.1415926);
```

5.1.3.4 Desorbent Consumption

This is simply tracked as a ratio of the desorbent to the feed.

```
DesorbentConsumption as MassFlow;
Desorbent.Total_Flow / Feed.Total_Flow;
```

5.1.4 Summary

Here we provide the sample code for calculating all of the above parameters, as implemented in an actual Aspen Simulation. In this example, the total volume of the column is hard-coded.

5.1.4.1 Flowsheet Constraints

```
PurityA as Fraction;
PurityA = Min(E.Total_Component("TB+") / Max(E.Total_Mass,0.0000001),1);

PurityB as Fraction;
PurityB = Min(R.Total_Component("TB-") /  Max(R.Total_Mass,0.0000001),1);

RecoveryA as Fraction;
RecoveryA = Min(1 - R.Total_Component("TB+") /
Max(F.Total_Component("TB+"),0.0000001),1);

RecoveryB as Fraction;
RecoveryB = Min(1 - E.Total_Component("TB-") /  Max(F.Total_Component("TB-
"),0.0000001),1);

ProductivityA as c_Flow_Mass;
ProductivityB as c_Flow_Mass;
ProductivityTotal as c_Flow_Mass;

ProductivityA = (E.Total_Component("TB+") /
SMB.Switch_Time)*60*1000/(6*2.492854) ;
ProductivityB = (R.Total_Component("TB-") /
SMB.Switch_Time)*60*1000/(6*2.492854) ;

ProductivityTotal = ProductivityA + ProductivityB;
```

5.1.4.2 Reset Task

```
E.Total_Component("TB+") : 0;
E.Total_Component("TB-") : 0;
E.Total_Mass : 0;

R.Total_Component("TB+") : 0;
R.Total_Component("TB-") : 0;
R.Total_Mass : 0;
```

```
F.Total_Component("TB+") : 0;
F.Total_Component("TB-") : 0;
F.Total_Mass : 0;

wait SMB.Switch_Time;
```

5.2 Distributed System

5.2.1 Motivation

In situations where the shortcut methods and analytic solutions are exhausted, we need to carry out optimizations on a full, rigorous simulation. Aspen Custom Modeler has a built-in optimization, which allows for some sophisticated optimization methods to be carried out on dynamic simulations. *However, the dynamic simulations are executed one at a time, and there is no way to "pick up where you left off" in the event of a simulation failure or software crash.*

Therefore, to more easily optimize these simulations, we develop a system to distribute the simulations across several computers and CPU cores. This will allow us to quickly and easily perform detailed optimizations as well as sensitivity analysis. Because of the program's origin in optimizing chromatographic separations, we chose to call the program "ChromRunner."

5.2.2 System Requirements

We designed the ChromRunner system to make better use of all the resources available to us. It was written as a WPF application in C# using the latest version of Visual Studio provided by Virginia Tech (VS 2015 Professional). In order to work properly, the computers that will be running the program must be able to access a central shared folder where the simulation files are stored, they must have read/write access to the database where simulation results are stored, and access to the Aspen License Server.

The program is written to use features of Aspen 8.x which do not exist in 7.x, and so it is not compatible with versions of Aspen prior to 8.0.

5.2.3 Use

This program has a relatively easy-to-use interface, built using the Windows Presentation Foundation (WPF) tools in Visual Studio. Below are some screenshots showing the features:

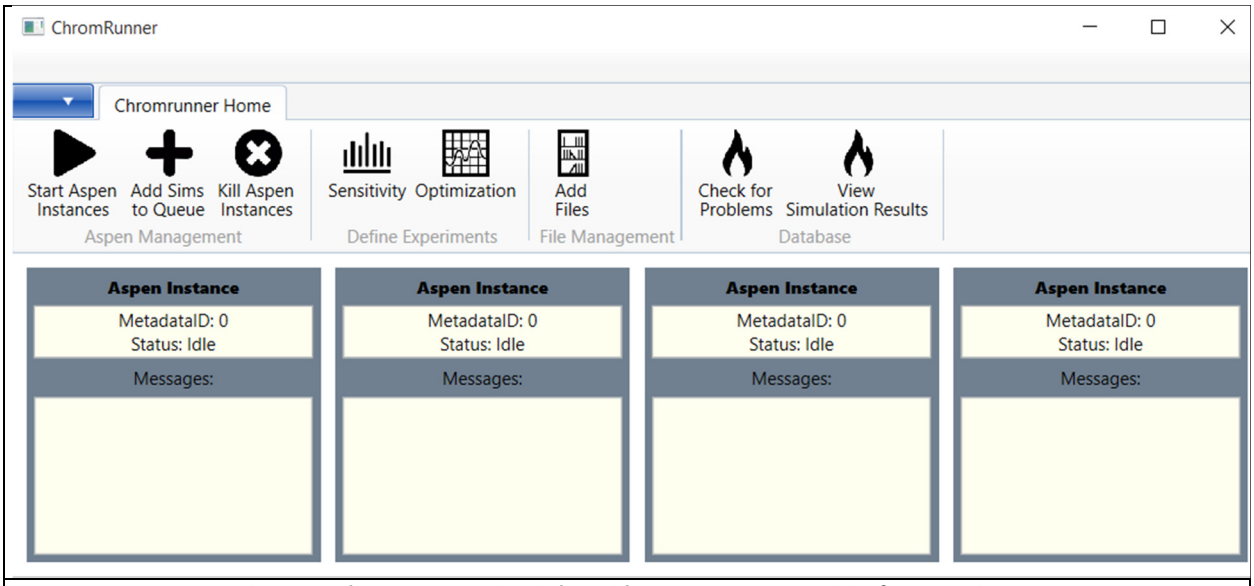


Figure 5.1: ChromRunner Main Window with 4 Aspen Instances Waiting for Instructions

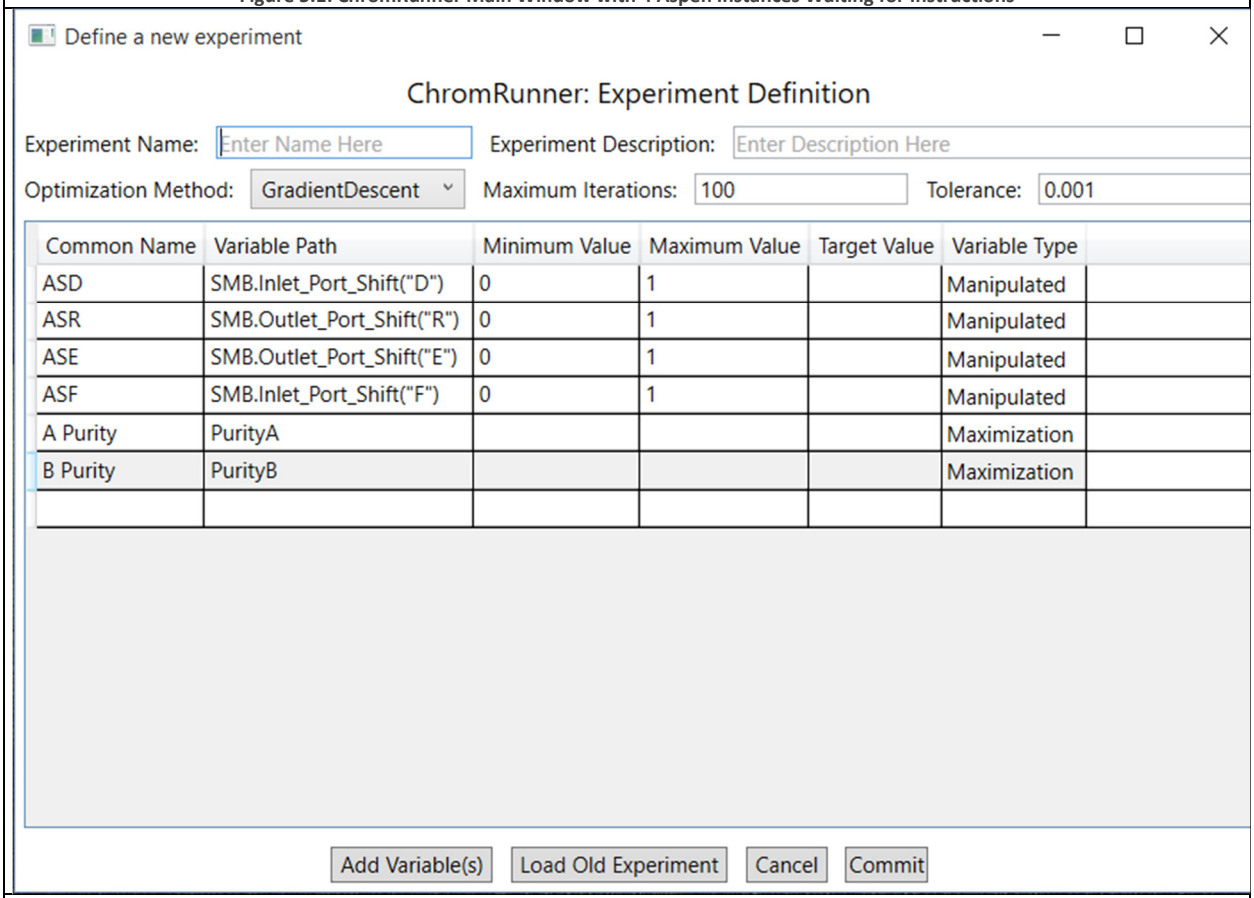


Figure 5.2: Defining an Optimization Experiment

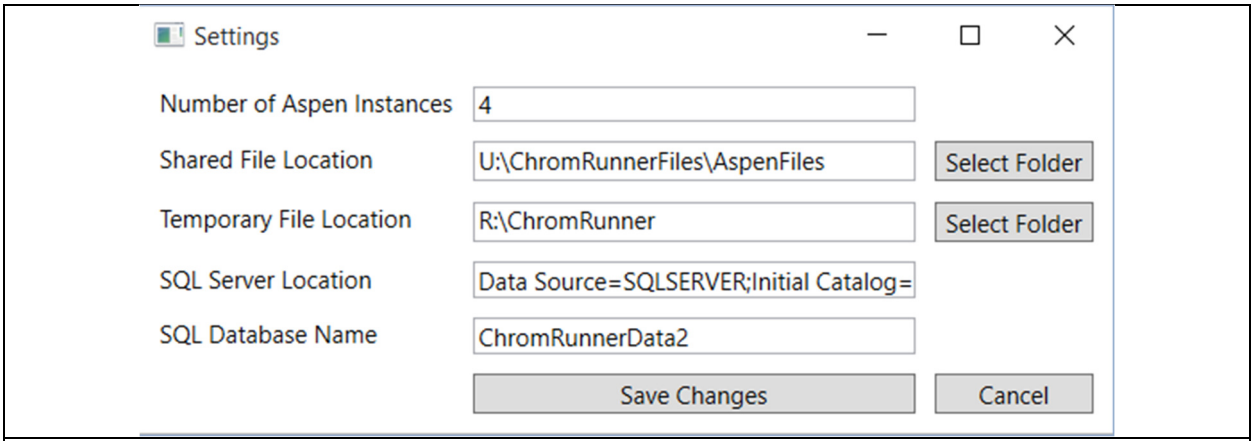


Figure 5.3: Configuration Options for ChromRunner

5.2.4 Code Structure

The code is grouped into several folders, according to its role. This is done in an effort to follow the "separation of concerns" principle, where sections of code with different concerns are prevented from depending too heavily on the decisions made in other parts of the program. We have decided that the major concerns of the program are:

- File input/output
  - This code deals with copying, deleting, and renaming files
  - It also keeps track of temporary files
- Experiments
  - This code deals with optimizations and sensitivity analysis.
  - There are some convenient math constructs (e.g. vectors) as well as all the optimization algorithms
- Database Related
  - This code deals with both the database structure and database Input/Output
  - The database structure is created by Entity Framework's "Code First," where the *database* structure is automatically determined from the *code* structure.
- UI Elements
  - This encompasses all the windows that the user can interact with, except for the main window which has some restrictions on where it can be stored
- Aspen Interactions
  - This encompasses all the code that interacts with Aspen through COM+
  - There are a variety of helpful methods that simplify opening files, preparing simulations, etc.
- Run Control

- This essentially describes the “glue” code that coordinates between the categories described so far
- It also contains the code that ensures that the correct number of instances of the Aspen Software are open

- Error Handlers
  - This contains the code related to managing errors that occur and logging them.

5.2.5 Specific Algorithms

To avoid including unnecessary and uninteresting sections of the program, we have restricted consideration to those classes which are unique to our system and give a general idea of how we have gone about solving the problem.

5.2.5.1 Database Structure

Figure 5.4 shows several database tables related to storing experiments. This demonstrates the conceptual distinction between experiment “Definitions” which contain information about a given experiment, and “Executions” which contain the information required to apply a “Definition” to an Aspen File.

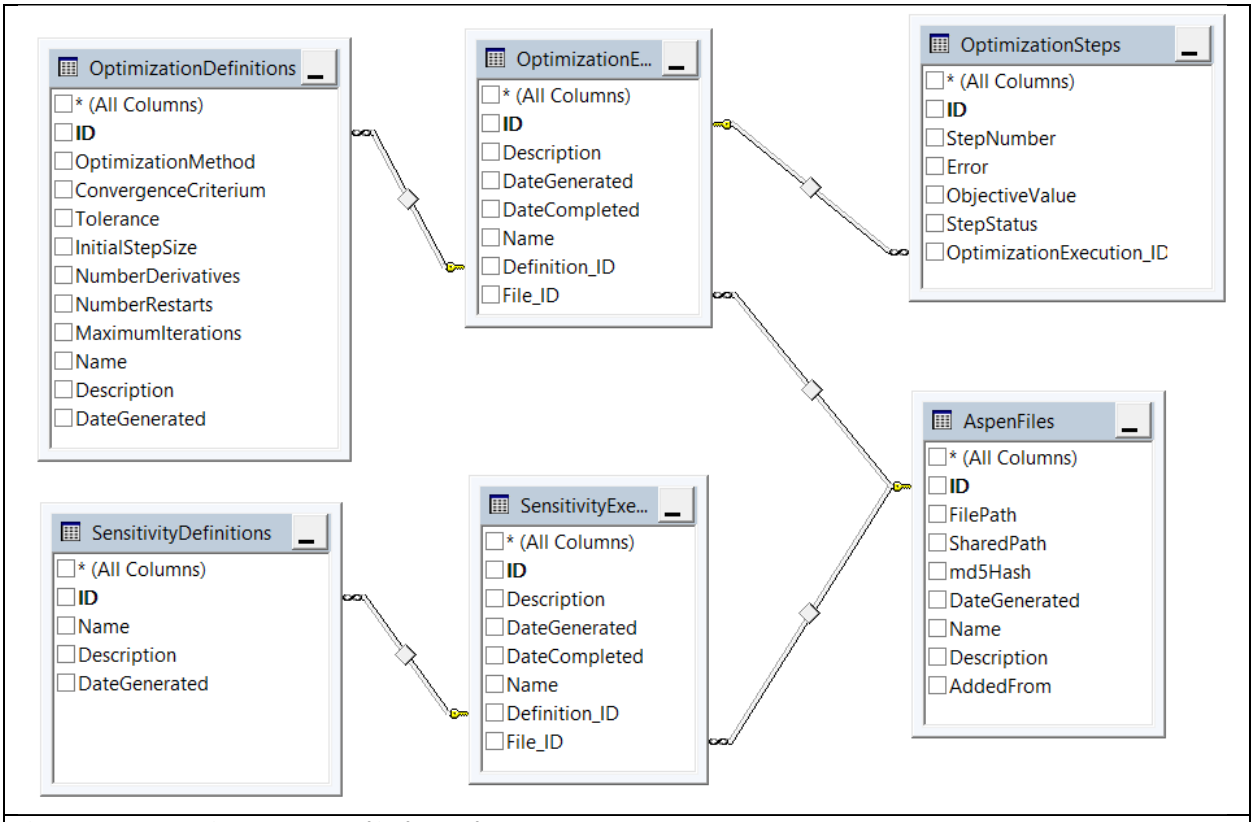


Figure 5.4: How Experiments Are Stored in the Database

Figure 5.5 shows the database tables related to storing the simulations that have yet to be run. Each entry in the “Metadatas” table represents an individual Aspen Simulation. It contains information about

the file to be run, which sensitivity or optimization it is a part of, and which variables need to be applied prior to the run and recorded after the run.

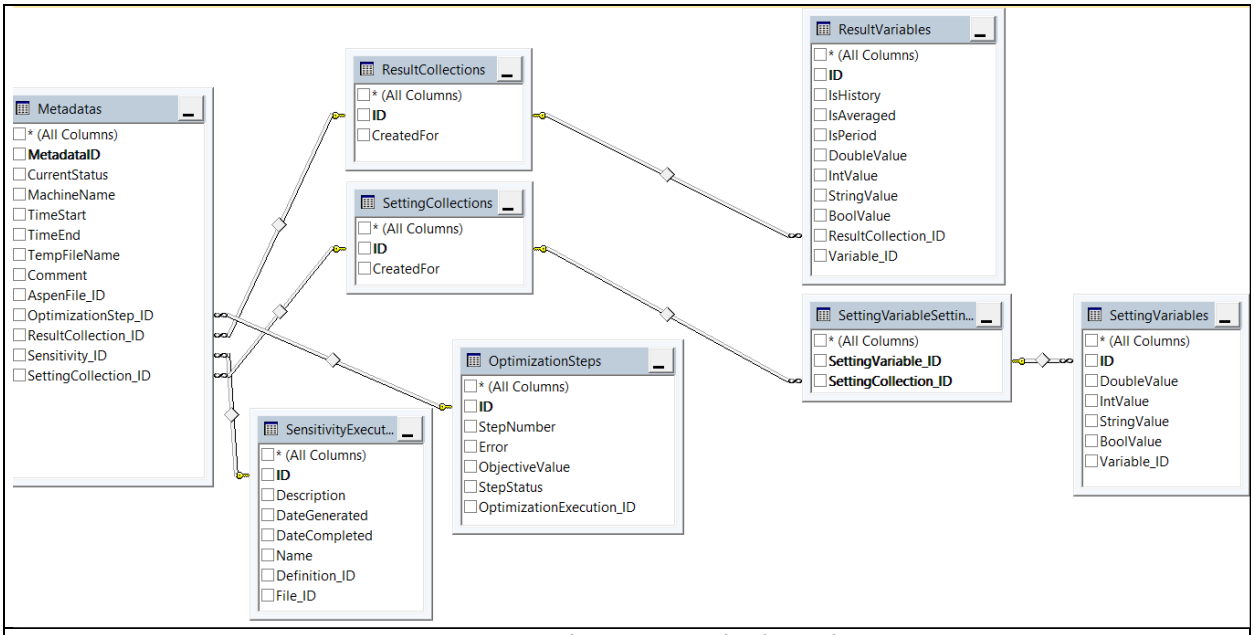


Figure 5.5: How Simulations Are Stored in the Database

5.2.5.2 Communication with Aspen

Talking to Aspen through the "Component Object Model" (COM+) interface is not a simple task. There are a number of nuanced system-related problems that can and do crop up and cause simulation results to become unavailable, or for Aspen to miss your instructions. Therefore, any part of the program which talks to Aspen should be made "bulletproof." We accomplished this with the following strategy.

First, any Aspen variable that gets referred to directly must be contained inside error-checking code. W accomplished this with the definition of an AspenVariable class as defined on the next page. The important features to note are that the class is "generic" which means that it can potentially deal with a variety of different data types (e.g. string, double, bool, etc.) Secondly, there is no explicit code to describe the relationship of the value (i.e. _Value) of this variable to Aspen. Instead, the method for getting the value from Aspen and setting the value in Aspen must be passed into this class as delegates.

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using AspenCustomModelerLibrary;

namespace ChromRunner6.Aspen_Interactions
{
    class AspenVariable<T> : IAspenVariable
    {
        private string _Path;
        private T _Value;
        private HistoryObject<T> _HistoryObject;
        private bool _IsSavableInDb;
        private bool _IsFlowsheet;
```

```
        public bool IsFlowsheet { get { return _IsFlowsheet; } }

        public delegate VariableStatus GetValueDelegate(AspenModelerSimulation
Instance, out T Value);
        public delegate VariableStatus SetValueDelegate(AspenModelerSimulation
Instance, T NewValue);
        public delegate VariableStatus GetHistoryDelegate(AspenModelerSimulation
Instance, out HistoryObject<T> Values);


        private GetValueDelegate _Getter;
        private SetValueDelegate _Setter;
        private GetHistoryDelegate _Hist;

        //public AspenVariable(PropertyInfo bool isFlowsheet)

        public AspenVariable(GetValueDelegate getter, SetValueDelegate setter, bool
isFlowsheet)
        {
            _IsFlowsheet = isFlowsheet;
            _Getter = getter;
            _Setter = setter;
            _IsSavableInDb = typeof(T) == typeof(double) || typeof(T) == typeof(string)
|| typeof(T) == typeof(int) || typeof(T) == typeof(bool) || typeof(T) == typeof(float);
        }
        public AspenVariable(GetValueDelegate getter, SetValueDelegate setter,
GetHistoryDelegate hist, bool isFlowsheet) : this(getter, setter, isFlowsheet)
        {
            _Hist = hist;
        }
        public string Path  //This is no longer a literal copy of the aspen path!
        {
            get { return _Path; }
            set { _Path = value; }
        }
        public T Value
        {
            get { return _Value; }
            set { _Value = value; }
        }
        public HistoryObject<T> History
        {
            get { return _HistoryObject; }
            set { _HistoryObject = value; }
        }
        public VariableStatus GetHistory(AspenModelerSimulation Instance)
        {
            if (!IsFlowsheet || _Hist == null)
                return VariableStatus.NoHistoryAvailable;

            return _Hist.Invoke(Instance, out _HistoryObject);
        }
        public VariableStatus UpdateValue(AspenModelerSimulation Instance)
        {
            return _Getter.Invoke(Instance, out _Value);
        }

        public VariableStatus SetValue(AspenModelerSimulation Instance)
```

```
        {
            return _Setter.Invoke(Instance, _Value);
        }
        public VariableStatus SetValue(T Value, AspenModelerSimulation Instance)
        {
            this._Value = Value;
            return SetValue(Instance);
        }
        public Type Type
        {
            get { return typeof(T); }
        }
    }
}
```

In conjunction with this class, we made a static helper class (called the VariableFactory) that helps me generate the delegate methods we need in order to create the AspenVariable classes we described previously. The most important part of this class is the "`CreateAspenVariable<T>`" method, which does exactly what you would expect. Note the `getter`, `setter`, and `HistGetter` variables. Each one of those variables defines the procedure for how to get/set the variable from Aspen. Most importantly, each operation is wrapped in a try – catch block which will catch the errors. Any errors that are thrown are immediately passed to the AspenErrorHandler which decides what action to take based on the error. In this way, every AspenVariable that is created through the VariableFactory is "bulletproof" automatically. It is therefore possible to use these variables in any other part of the program without worrying about exceptions being thrown. Failures can be detected easily with simple if statements.

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Reflection;
using System.Text;
using System.Threading.Tasks;
using AspenCustomModelerLibrary;
using static ChromRunner6.Aspen_Interactions.BulkVariableGenerator;
using System.Collections.Concurrent;

namespace ChromRunner6.Aspen_Interactions
{
    /// <summary>
    /// A factory for taking the dynamic output of aspen variables and converting them
into AspenVariables<> with the appropriate type
    /// They are returned as IAspenVariables
    /// </summary>
    static class VariableFactory
    {
        //In this method, the path must be immediately processable by the flowsheet
        public static IAspenVariable CreateAspenVariable(string Path,
AspenModelerSimulation Simulation)
        {
            dynamic AspenVariable = null;
            try
            {
                AspenVariable = Simulation.Flowsheet.Resolve(Path);
            }
            catch (Exception ex)
```

```
            {
                AspenExceptionHandler.HandleException(ex);
            }
            if (AspenVariable != null)
                return CreateAspenVariable(AspenVariable);
            else
                return new NullVariable();
        }
        public static IAspenVariable CreateAspenVariable(dynamic AspenVariable) //The
case list may not be exhaustive
        {
            string BaseType = "";
            try
            {
                BaseType = AspenVariable.BaseTypeName;
            }
            catch (Exception ex)
            {
                AspenExceptionHandler.HandleException(ex);
            }

            switch (BaseType as string)
            {
                case "IntegerParameter":
                    return CreateAspenVariable<int>(AspenVariable);
                case "IntegerSet":
                    return CreateAspenVariable<int[]>(AspenVariable);
                case "LogicalParameter":
                    return CreateAspenVariable<bool>(AspenVariable);
                case "StringParameter":
                case "ComponentListName":
                    return CreateAspenVariable<string>(AspenVariable);
                case "StringSet":
                    return CreateAspenVariable<string[]>(AspenVariable);
                case "RealParameter":
                case "RealVariable":
                    return CreateAspenVariable<double>(AspenVariable);
                default:
                    try
                    {
                        return new NullVariable(AspenVariable.GetPath());
                    }
                    catch(Exception ex)
                    {
                        AspenExceptionHandler.HandleException(ex);
                        return new NullVariable();
                    }
            }
        }
        //Once the type is known, this method creates the final IAspenVariable
        public static IAspenVariable CreateAspenVariable<T>(dynamic AspenVariable)
        {
            string Path = "";
            try
            {
                Path = AspenVariable.GetPath();
            }
            catch (Exception ex)
```

```
            {
                AspenExceptionHandler.HandleException(ex);
                return new NullVariable();
            }

            AspenVariable<T>.GetValueDelegate getter = (AspenModelerSimulation
Simulation, out T x) =>
            {
                try
                {
                    Simulation.Flowsheet.UpdateVariables(Path);
                    x = Simulation.Flowsheet.Resolve(Path).Value;
                    return VariableStatus.Normal;
                }
                catch (Exception ex)
                {
                    x = default(T);
                    return AspenExceptionHandler.HandleException(ex);
                }
            };
            AspenVariable<T>.SetValueDelegate setter = (AspenModelerSimulation
Simulation, T x) =>
            {
                try
                {
                    Simulation.Flowsheet.Resolve(Path).Value = x;
                    return VariableStatus.Normal;
                }
                catch (Exception ex)
                {
                    return AspenExceptionHandler.HandleException(ex);
                }
            };

            AspenVariable<T>.GetHistoryDelegate HistGetter = (AspenModelerSimulation
Simulation, out HistoryObject<T> x) =>
            {
                try
                {
                    dynamic hist = Simulation.Flowsheet.Resolve(Path).History;
                    x = new HistoryObject<T>(hist);
                    return VariableStatus.Normal;
                }
                catch (Exception err)
                {
                    x = null;
                    return AspenExceptionHandler.HandleException(err);
                }
            };

            return new AspenVariable<T>(getter, setter, HistGetter, true) { Path = Path
};
        }
        internal static IAspenVariable CreateNonAspenVariableFromList(int i,
AspenModelerSimulation sim)
        {
            try
            {
```

```
                dynamic val = ConstantPaths.AllDynamics[i].Invoke(sim);
                string type = ((object)val).GetType().Name.ToLowerInvariant();
                switch (type)
                {
                    case "double":
                        return CreateNonAspenVariableFromList<double>(i);
                    case "single":
                        return CreateNonAspenVariableFromList<float>(i);
                    case "int32":
                        return CreateNonAspenVariableFromList<int>(i);
                    case "string":
                        return CreateNonAspenVariableFromList<string>(i);
                    case "boolean":
                        return CreateNonAspenVariableFromList<bool>(i);
                    default:
                        return CreateNonAspenVariableFromList<dynamic>(i);
                }
            }
            catch (Exception ex)
            {
                string failed = ConstantPaths.AllPaths[i];
                AspenExceptionHandler.HandleException(ex);
                return new NullVariable(ConstantPaths.AllPaths[i]);
            }

        }
        internal static IAspenVariable CreateNonAspenVariableFromList<T>(int i)
        {
            AspenVariable<T>.GetValueDelegate getter = (AspenModelerSimulation
Simulation, out T x) =>
            {
                try
                {
                    x = ConstantPaths.AllDynamics[i].Invoke(Simulation);
                    return VariableStatus.Normal;
                }
                catch (Exception ex)
                {
                    x = default(T);
                    return AspenExceptionHandler.HandleException(ex);
                }
            };
            AspenVariable<T>.SetValueDelegate setter = (AspenModelerSimulation
Simulation, T x) =>
            {
                try
                {
                    ConstantPaths.AllSetters[i].Invoke(Simulation, x);
                    return VariableStatus.Normal;
                }
                catch (Exception ex)
                {
                    return AspenExceptionHandler.HandleException(ex);
                }
            };

            return new AspenVariable<T>(getter, setter, true) { Path =
ConstantPaths.AllPaths[i] };
```

```
        }
    }
}
```

5.2.6 Optimization Algorithms

There are a variety of optimization algorithms that we have considered for implementation in this system. Of the following, we have implementations for Gradient Descent, Coordinate Descent, and the Powell Method. The Powell method has been successfully tested, and was used in our published work[5]. We have not tested the other two descent methods as extensively, and so there may still be bugs present. The Nelder-Mead and Cuckoo optimizations are works in progress.

- Gradient Descent
    - Create a numerical estimate of the gradient; requires *n* simulations, where n is the number of manipulated variables
    - Search for an optimum in the gradient direction; either use a linear search method, or pick distances based on the magnitude of the gradient
- Cuckoo optimization
    - Belongs to the genetic algorithm family
    - Tends to converge faster than normal genetic algorithms
    - Has been used in the optimization of a variety of industrial designs[26]
- Nelder-Mead
    - Popular heuristic search
    - Not easily parallelized
- Coordinate descent
    - Optimize variables one at a time, iteratively
- Powell Method
    - Belongs to the coordinate descent family
    - After some iterations, combine the steps taken into a new direction

6 Applications

The ChromRunner system was used in the IE&C Research publication, "Simulation and Comparison of Operational Modes in Simulated Moving Bed (SMB) Chromatography." In this paper, we used ChromRunner's sensitivity analysis features to generate a number of plots like the one in Figure 6.14. Having the ability to automate simulations dramatically improved our ability to analyze the robustness of our system to changes in operating conditions. Without the automation provided by ChromRunner, the analysis carried out in the paper would not have been possible.

The following section borrows heavily from the author's publication "Simulation and Comparison of Operational Modes in Simulated Moving Bed (SMB) Chromatography.[5]"

6.1 SMB Operational Modes for Binary Separation

Simulated moving bed (SMB) chromatography[1,2,3] has attracted growing research interests and industrial applications in separating and purifying enantiomers, petrochemicals, pharmaceuticals and biochemicals with higher yield and lower solvent consumption. The success of SMB applications has led to the development of new methods for operating SMB units to further improve the separation efficiency. The literature contains a number of proposed changes to SMB operations[25,27,28], which we refer to as operational modes. The latest review[1] details the most recent SMB operational modes, applications and developments. In our work, we focus particularly from a practical point of view on several operational modes applied to the existing SMB units by switching ports or adjusting control valves. Specifically, this study considers:

(1) Varicol—switching the inlet and outlet ports asynchronously[29–33]
(2) PowerFeed or Partial Feeding—varying feed flow rates[9,29,34,35]
(3) ModiCon—varying feed concentrations[25,27,28]
(4) Combinations of the previous three operational modes[29]

Our work builds on the previous studies of these operational modes and contributes the following:

(1) Investigate two separation systems to seek heuristic guidelines for improving the performance of the SMB operational modes and their combinations;
(2) Combine two or three operational modes (Varicol, PowerFeed and ModiCon);
(3) Present time-dependent column concentration profiles to give dynamic insights on the difference of the basic SMB with PowerFeed and with ModiCon;
(4) Evaluate the product purity and recovery results through rigorous simulations to show not only the improved separation regions for ranges of product purity and recovery, but also the robustness of the operational modes to changing flow rate ratios, i.e. m values;
(5) Analyze and compare the purity and productivity performance in the Pareto plot across all the operational modes;
(6) Investigate the validity of our recommendations on operational modes with different separation factors through rigorous simulation and optimization.

In the interests of simplifying the design of SMB separation systems, we seek to identify heuristics that will guide the selection of operational modes across a variety of systems. To that end, we have employed a large-scale simulation of SMB systems using a distributed computing approach, which allows us to identify both the heuristics we seek, as well as physical insights into these systems.

Specifically, we have performed analyses on a binary and a ternary separation to find common recommendations between the systems. We design the analyses to accomplish three key objectives. First, we identify trends in the optimization of each operational mode's variables. We compare the sensitivity of purity and recovery to operational mode variables between the binary and ternary separations. Second, we quantify and compare the complete separation regions and the Pareto optimum plots[10,34,36,37] for the operational mode variables found in the first step. Third, we apply the Powell optimization method[38] to extend the investigation to more separation systems with different separation factors.

Having already covered the basics of SMB columns in section 2.9, we will now turn our attention to the various operational modes considered in our study.

6.2 Varicol

6.2.1 Description

The key difference between the Varicol operation and the basic SMB is an asynchronous shift of the inlet/outlet ports in a multicolumn system. The zone lengths are varied in time, but this variation is periodic so that the column recovers its initial position after a whole switching cycle. Thus, the operation is cyclic. The fact that the zone lengths vary over time means that the effective solid flow rate is not a meaningful concept. The change in the relative position of the ports means that we cannot say that each zone moves past one bed per switching time as we can with the basic SMB.

Figure 6.1 illustrates the asynchronous shifting of inlet/outlet ports in a Varicol process within a 6-bed SMB column. We denote the initial column configuration by χ = (1-2-2-1), and define the asynchronous shifting ratio (AS) as a fraction of the switching time t_{sw} at which the inlet or outlet port moves to its next port position. Specifically, AS_D, AS_E, AS_F and AS_R with values between 0 and 1 represent, respectively, the asynchronous shifting ratios for the desorbent-in port, extract-out port, feed-in port and raffinate-out port. For example, $AS_F = AS_E = 0.5$, together with $AS_D = AS_R = 1$ mean that at $t = 0.5 * t_{sw}$, both feed-in port and extract-out port switch to the next bed, and the column configuration becomes χ = (2-2-1-1). Then, at $t=t_{sw}$, desorbent-in port and raffinate-out port move to their next beds. At $t=t_{sw}$, all four ports have switched to their next beds, and the column configuration returns to χ = (1-2-2-1).

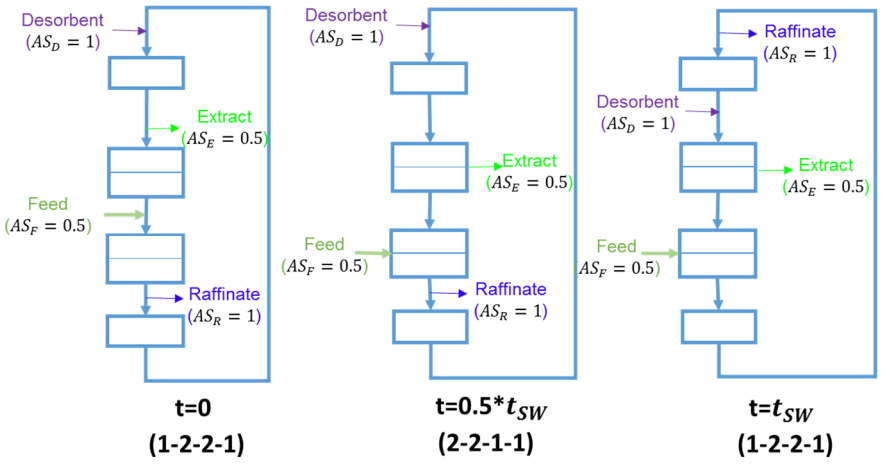


Figure 6.1 Diagram of the asynchronous shift of inlet/outlet ports in a Varicol process.

The effective number of beds is a useful parameter for describing a specific Varicol configuration. This number describes the effect of the asynchronous switching ratios on the length of a given zone. Given in equation form, we calculate the effective numbers of beds, Z_1 to Z_4, for a 4-zone Varicol system by:

$$Z_1 = N_1 + AS_D - AS_E \quad (6.1)$$
$$Z_2 = N_2 + AS_E - AS_F \quad (6.2)$$
$$Z_3 = N_3 + AS_F - AS_R \quad (6.3)$$
$$Z_4 = N_4 + AS_R - AS_D \quad (6.4)$$

where N_1, N_2, N_3 and N_4 are the bed number in the four zones, and Z_1, Z_2, Z_3 and Z_4 are the effective zone lengths. In this study, the Varicol operation is characterized by the four bed length parameters (Z_1-Z_2-Z_3-Z_4). We use these effective bed numbers to evaluate the effect on the product purity and recovery in the following section.

6.2.2 Varicol Modeling and Optimization

For the Varicol modeling, all the process and operating conditions are the same as the basic SMB, except for the new asynchronous port switching. The Varicol process has its additional degrees of freedom (DOF) shown in eq (6.5), because it adds four more adjustable parameters AS_D, AS_E, AS_F and AS_R into the system. This offers the potential for greater performance compared to the basic SMB system, and several studies have investigated this previously[30,31].

$$DOF_{Varicol} = DOF_{SMB} + 4 \quad (6.5)$$

We have outlined an effective optimization procedure using the rigorous models developed in Aspen Chromatography to demonstrate how the Varicol process can achieve superior separation performance over the basic SMB. We want to maximize the purity and recovery of the process by changing the Varicol port switching ratios AS_D, AS_E, AS_F and AS_R. This means that we need to maximize the lengths of the separation regions (zones 2 and 3), while making the non-separating regions as small as possible without causing impurity contamination. Figure 6.2 illustrates the strategy for the Varicol optimization.

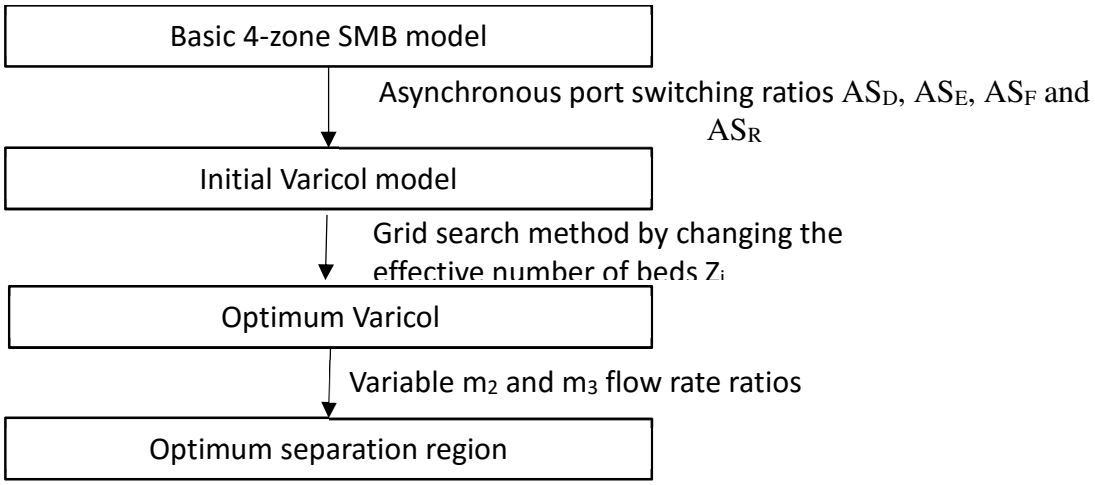


Figure 6.2 Varicol Optimization Strategy.

6.2.3 Varicol Results

The Varicol operational mode is not able to achieve greater than 99.9% purity in both components at the optimum point. However, it is possible to significantly improve on the base case in some of performance parameters, while only slightly sacrificing other separation performance. Figure 6.3 illustrates the general trends by looking at a contour plot of A and B purities in the (Z_1, Z_3) plane at Z_2=2.4. Solid lines outline regions of B purity, while dashed lines define regions of A purity. The figure shows that while there are regions where A (in blue) or B (in yellow) purity exceeds 99.9%, there is no overlap between the yellow and blue regions. We find that if we improve A purity to 99.9%, then B purity will drop, and this is observed at all Z_2. The results from the fact that zones 1 and 3 are primarily responsible for the isolation of component B, while zones 2 and 4 are primarily concerned with component A.

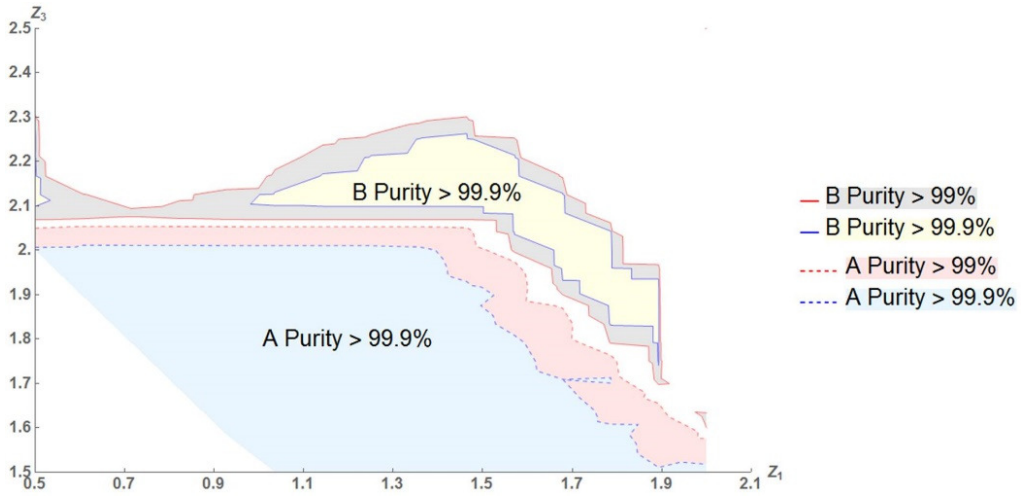


Figure 6.3 Contour plot of A and B purities in the (Z_1, Z_3) plane with Z_2 = 2.4.

We choose a Varicol model with $AS_D = 1, AS_F = 0.6, AS_E = 1, AS_R = 0.9$ as our optimum model because it results in high product purity and recovery. The optimum column configuration is (1-2.4-1.7-0.9). Column 4 of Table 6 summarizes the resulting Varicol performance parameter values.

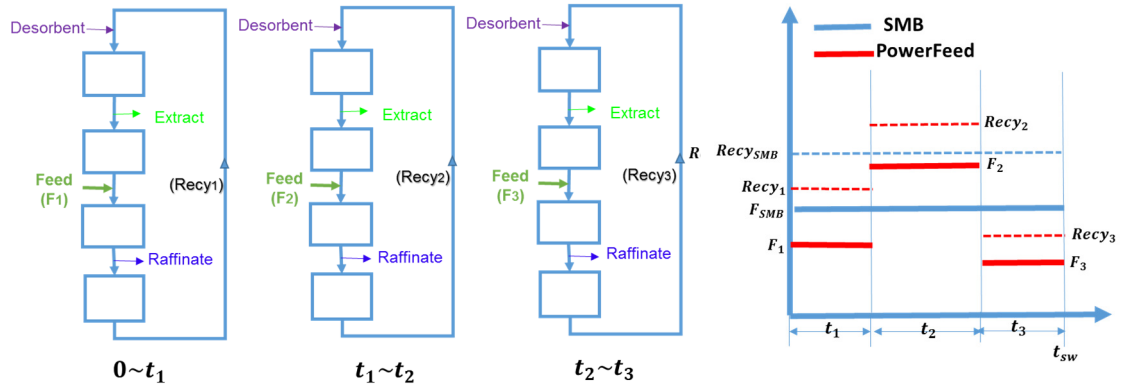
In this Varicol process, the average product purity of both A and B increases by 2.17%, and the average product recovery of both A and B increases by 1.45%, and solvent consumption decreases by 1.5% after the optimization, compared to the basic SMB model. This increase results from our ability to effectively fine-tune the zone lengths, allowing us to choose configurations that are part-way between two SMB configurations. However, the resulting optimum Varicol configuration (1-2.4-1.7-0.9) is actually very close to the original (1-2-2-1) SMB column configuration. Because Z_1 and Z_4 for the optimum Varicol are almost the same as those in the basic SMB, we conclude that a guideline to optimize a Varicol operation is to increase the length of zone 2, but decrease the length of zone 3.

6.3 PowerFeed

6.3.1 Description

Our second operational mode is a modified PowerFeed process. The difference between our work and the conventional PowerFeed[34,35] is that we also adjust the recycle flow rates within the subintervals, not only the feed flow rates. This is because in some cases, changing the feed flow rates may cause the system to exceed the maximum pressure drop. In our modified PowerFeed operation, we divide the switching time t_{sw} into three subintervals, and vary the feed flow rates (F_1, F_2 and F_3) from one subinterval to the next. In order to satisfy the maximum pressure drop criterion, eq 4, we vary the recycled flow rates ($Recy_1$, $Recy_2$ and $Recy_3$) during each operation cycle, as illustrated in Figure 6.4. This figure also compares the feed flow rate policies of the basic SMB and the PowerFeed operation in three switching subintervals. To compare the results of PowerFeed with the basic SMB model, we keep the average feed flow rate the same as the feed flow rate in our basic SMB model.

We note that the Partial Feeding[13,14] is a special case of the PowerFeed operation when the feed stream and the raffinate stream are active only during certain subintervals within a switching time.



(a) Feed and Recycle Flow Rate Shifting

(b) Feed and Recycle Flow Rate Policies for SMB and PowerFeed

Figure 6.4 (a) An illustration of variable feed and recycle flow rates within subintervals in the modified PowerFeed process. (b) A comparison of feed and recycle flow rates policies of the basic SMB and PowerFeed in three switching subintervals. In this modified PowerFeed with three switching subintervals, t= 0 ~ t_1, F =F_1, Recy=$Recy_1$; t = t_1~ t_2, F = F_2, Recy=$Recy_2$; t = t_2 ~ t_3, F = F_3, Recy=$Recy_3$; t = t_{sw}, F = F_1, Recy=$Recy_1$.

6.3.2 Modeling and Optimization

For the binary separation, we start with the same basic 4-zone SMB with a 1-2-2-1 column configuration. For our PowerFeed studies, we vary the feed flow rates in each cycle, and adjust the recycle flow rate to satisfy the maximum pressure drop criterion. First, we keep flow rate ratios m_i the same as those in

SMB, and we apply the analysis schemes shown in Figure 6.5 to evaluate this PowerFeed system with two or three subintervals, and get an optimum PowerFeed model. We also adjust feasible m_2 and m_3 flow rate ratios to make the (m_2-m_3) separation plane for providing some useful guideline of the PowerFeed operational mode. For our PowerFeed simulations, the following mass balance constraints apply:

$$\sum_{i=1}^{N_{sub}} F_i t_i = F_{SMB} t_{sw} \tag{6.6}$$

$$\sum_{i=1}^{N_{sub}} t_i = t_{sw} \tag{6.7}$$

In this way, we increase the degrees of freedom in PowerFeed because it adds six new parameters (t_1, t_2 and t_3, F_1, F_2 and F_3) to and subtract two constraints (eqs 6.6 and 6.7) from the system.

$$DOF_{PowerFeed} = DOF_{SMB} + 6 - 2 \tag{6.8}$$

Applying the maximum pressure drop criterion of eq 4, we find the following relationship between the maximum pressure drop and the feed and recycle flow rates:

$$\Delta P_{max} = 21.39 - 18.05 Q_F + 54.15 Q_{\text{recycle}} \tag{6.9}$$

We apply the following three steps to evaluate the PowerFeed separation performance. We analyze the effects of PowerFeed operation on the purity and recovery of products A and B, and report the key findings for each step.

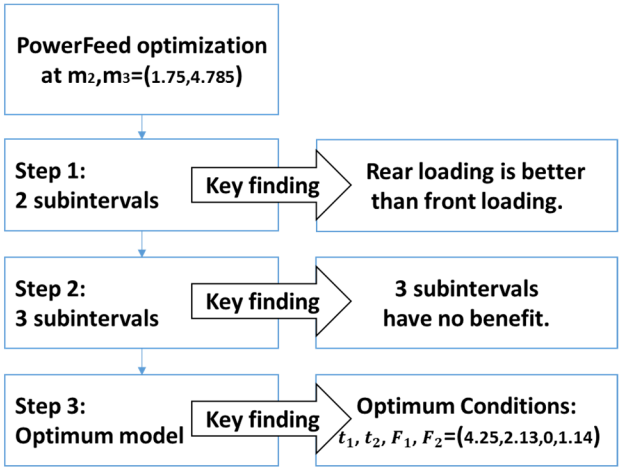
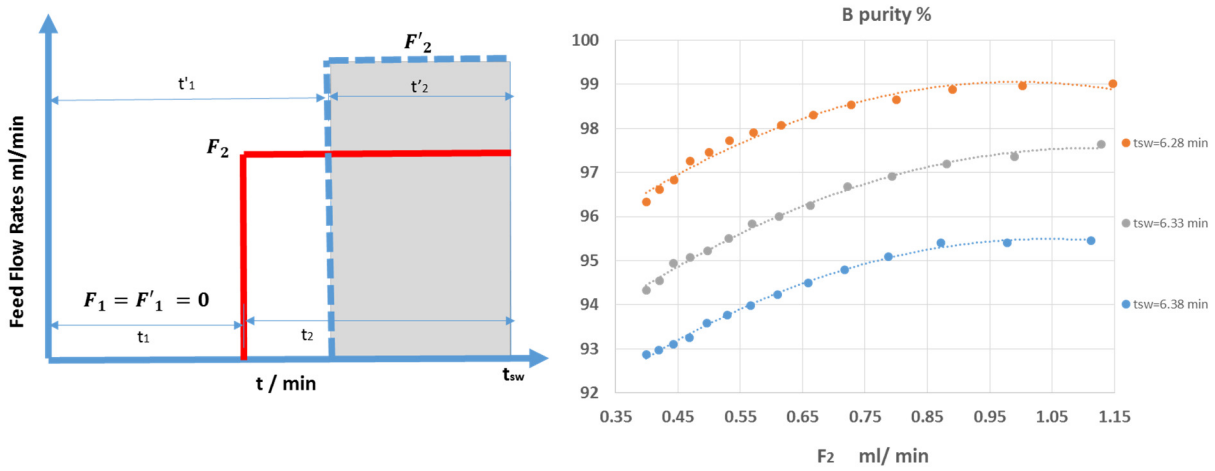


Figure 6.5 PowerFeed Optimization Steps.

In order to see if a front feed loading (larger F_1) or a rear feed loading (Larger F_2) gives a better separation performance, we evaluate the performance by changing the subinterval feed flow rates, and find that rear loading is better than front loading. We also evaluate the PowerFeed operation within three subinterval-switching times in step 2. We find that A purity, A recovery and B recovery essentially do not change throughout the simulation, and only B purity slightly improves compared to the SMB results. However, there is no benefit to the PowerFeed system with two subintervals. In step 3, we evaluate the partial feeding[21] with two subintervals to find a true optimum for the PowerFeed.

Specifically, we define the second subinterval feed flow rate F_2 as the most important parameter in this operational mode. We consider the B purity as the most important performance parameter, because A

purity and both A and B recoveries essentially do not change much. Figure 6.6(a) is a diagram of the strategy and trend conclusion for B purity in step 3. Figure 6.6(b) shows the effect of increasing the second subinterval feed flow rate F_2 on B purity for three different switching times. All the lines in Figure 6.6(b) show that the larger the subinterval with zero feeding, the higher is B purity. Therefore, the final optimum PowerFeed model corresponds to $F_1 = 0,\ F_2 = 1.14\ ml/min$ and $t_1 = 4.25\ min, t_2 = 2.13\ min.$



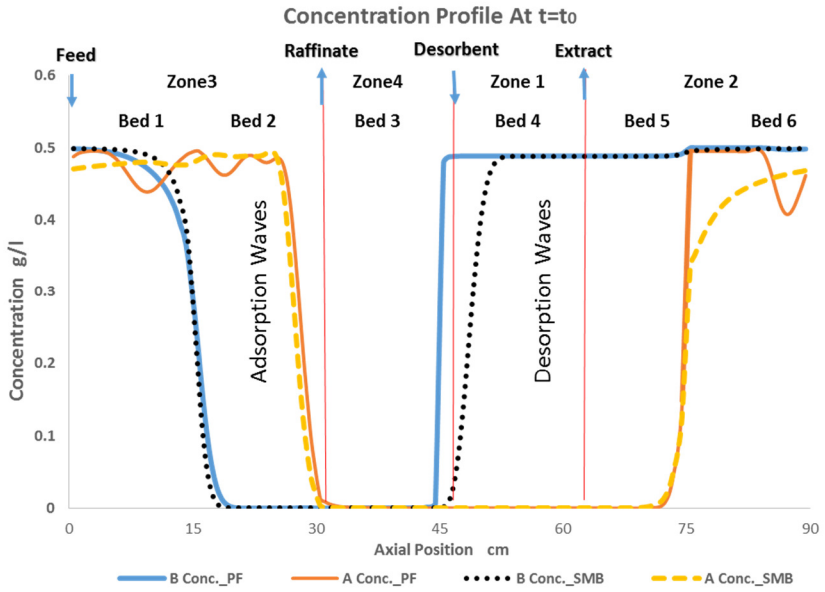
(a) Graphical Policy and Calculation Scheme (b) B Purity Simulation Results and Trend Conclusion

Figure 6.6 Diagrams and Results for PowerFeed operation.

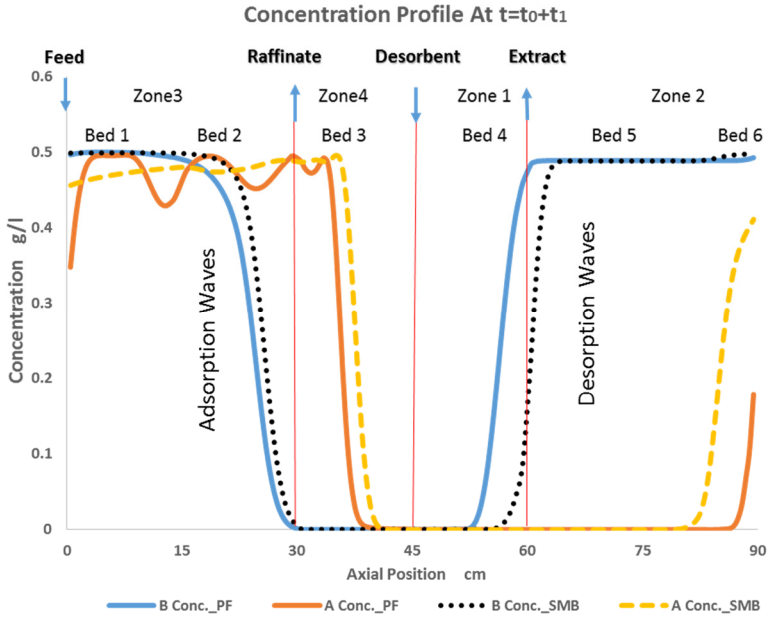
We see the reason for this trend when we compare the column concentration profiles of the optimum PowerFeed model and the basic SMB model. Figure 6.7 shows the steady-state concentration profiles at different points ($t = t_0, t = t_0 + t_1$ and$t = t_0 + t_{sw}$) within one switching subinterval for both PowerFeed and SMB. Solid lines represent concentration values from the PowerFeed model, while dashed lines refer to the SMB model. We find that the desorption waves in zones 1 and 2 are sharper in the PowerFeed model than those in the SMB model. In other words, the components are desorbed over a shorter distance in PowerFeed. This explains the improved B purity; the extract contains a high concentration of B for a longer period of time.

The separation performance of A is more complex. PowerFeed has two competing effects on the adsorption waves in zones 3 and 4. While the feed is shut off, the adsorption waves move through the column more slowly; this would ordinarily increase the purity of the raffinate. However, once the feed is turned on, the higher flow rate results in a broader B adsorption wave. This means that during part of the subinterval, i.e. the time between Figure 6.7(b) and (c), the raffinate will encounter a higher concentration of B which will adversely affect the purity. In combination, these effects cancel each other out and the overall purity of the raffinate remains constant.

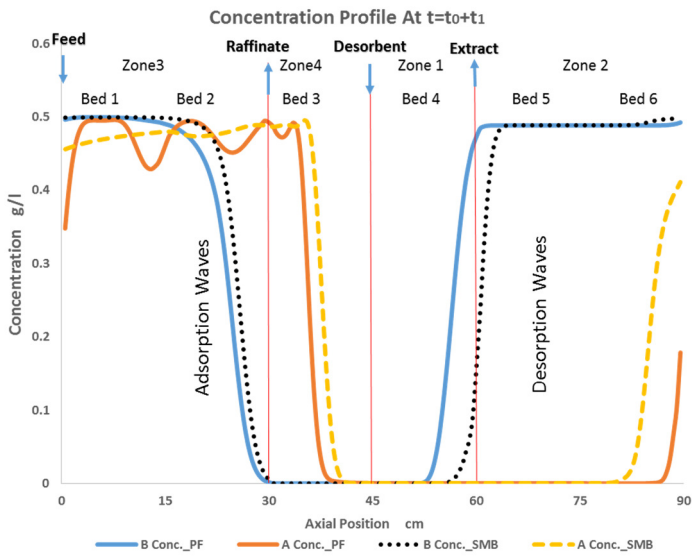
From our observation, B recovery decreases a little compared to that in SMB model, and A recovery stays at essentially 100%. From the adsorption waves at the end of the switching time in the blue line of Figure 6.7(c), we see that the raffinate stream loses some B. In PowerFeed, we find that B purity and A recovery have a balancing effect in the separation performance.



(a) Concentration Profiles at $t=t_0$.



(b) Concentration Profiles at $t= t_0+t_1$.



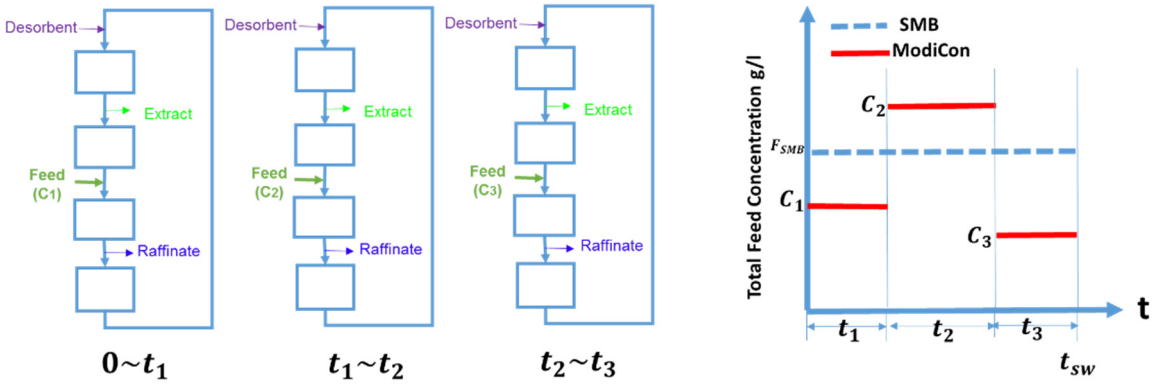
(c) Concentration Profiles at $t=t_0+t_{sw}$.

Figure 6.7 PowerFeed (PF) Concentration Profiles.

6.4 ModiCon

6.4.1 Description

ModiCon[15] varies the feed concentration (C_1, C_2, and C_3) of the SMB process within three switching subintervals, as illustrated in Figure 6.8. It keeps the feed flow rate and the column configuration unchanged as the basic SMB model.



(a) Feed Concentration Shift in the Feed Port

(b) Feed Concentration Policies for Basic SMB and ModiCon

Figure 6.8 (a) An illustration of variable feed concentrations within three subintervals in the ModiCon process. (b) A comparison of feed concentration policies of the basic SMB and ModiCon within three switching subintervals. In this ModiCon with three switching subintervals, t= 0 ~ t_1, C =C_1; t = t_1~ t_2, C = C_2; t = t_2 ~ t_3, C = C_3; t = t_{sw}, C = C_1.

6.4.2 Modeling and Optimization

We apply steps 1 to 3 in Figure 6.9 to evaluate this ModiCon system. We write the mass balance constraint, eq (6.10), and change the degrees of freedom for the basic SMB to that of ModiCon in eq

(6.11) by adding six new parameters (t_1, t_2, t_3 and C_1, C_2, C_3) and subtracting two constraints, eqs (6.6) and (6.10):

$$\sum_{i=1}^{N_{sub}} C_i t_i = C_{SMB} t_{sw} \tag{6.10}$$

$$DOF_{ModiCon} = DOF_{SMB} + 6 - 2 \tag{6.11}$$

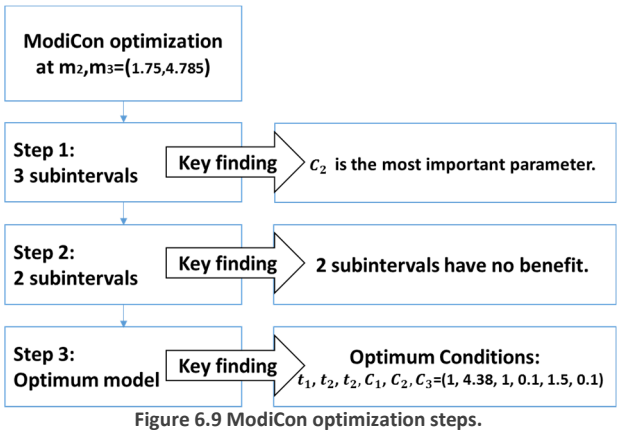
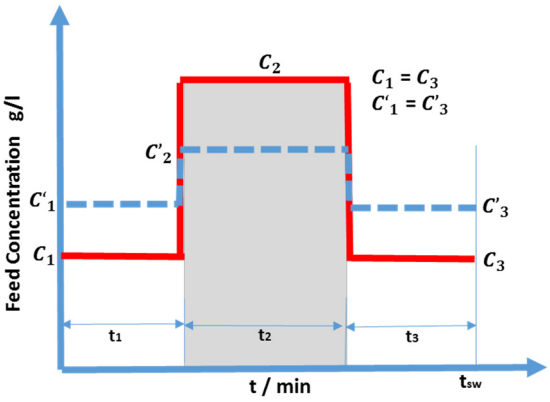


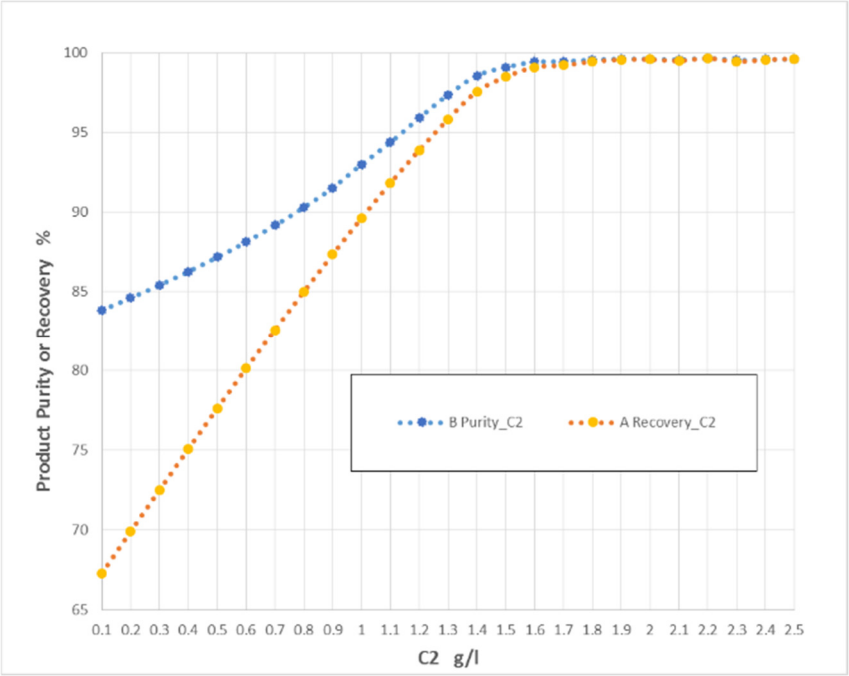
Figure 6.9 ModiCon optimization steps.

We analyze the ModiCon with three subintervals by varying subinterval feed concentrations. We find the second subinterval feed concentration as the most effective parameter in the system. We also find that ModiCon with 2 subintervals has no benefit than ModiCon with 3 subintervals. In step 3, we assume that $t_1 = t_3, t_1' = t_3', C_1 = C_3$, and increase C_2 from 0.1 g/l to 2.5 g/l to see the effect of the most important parameter C_2 of this system. In this way, we increase the second subinterval feed concentration and also enlarge the middle grey area in Figure 6.10(a). Figure 6.10(b) shows that the larger C_2 gives higher B purity and A recovery; and both B purity and A recovery remain constant after 1.5 g/l feed concentration. We also find that A purity and B recovery stay essentially constant at 100% while increasing C_2.

ModiCon achieves an improved separation performance with $t_1 = t_3$=1 min, t_2 =4.38 min, $C_1 = C_3$= 0.1 ml/min, C_2= 1.5 ml/min. The resulting B purity is 98.84%, which is 6.6% higher than that in the basic SMB process, and A recovery is 97.59%, a 9.4% increase from that in the basic SMB model. We observe diminishing returns as C_2 increases beyond 1.5 times the original concentration. Therefore, we select a C_2 value of 1.5 g/l as our optimum.



(a) Graphical Policy and Calculation Scheme

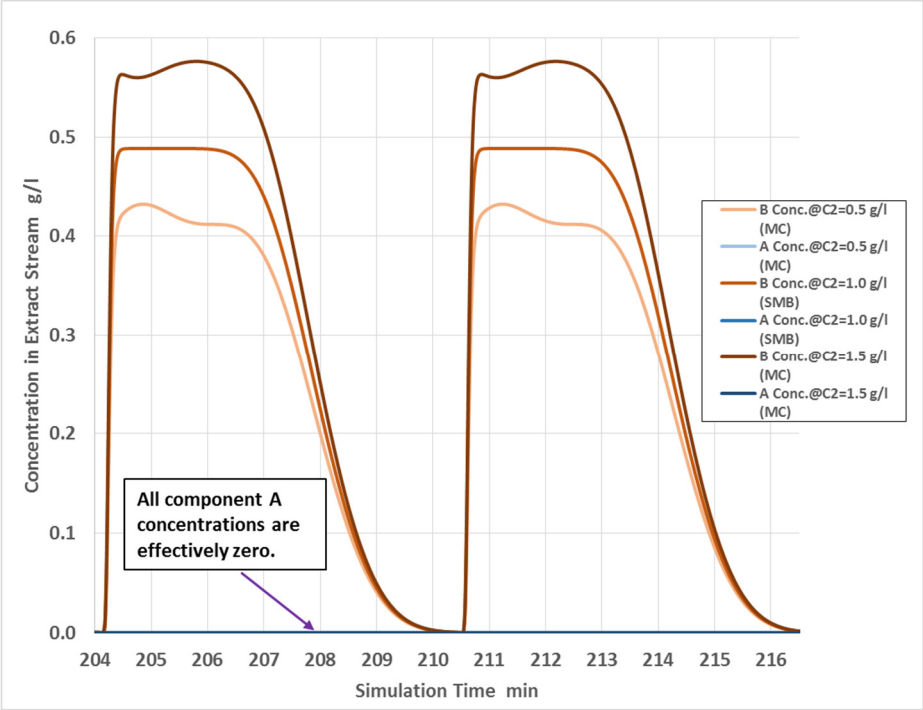


(b) Simulation Results and Trend Conclusion

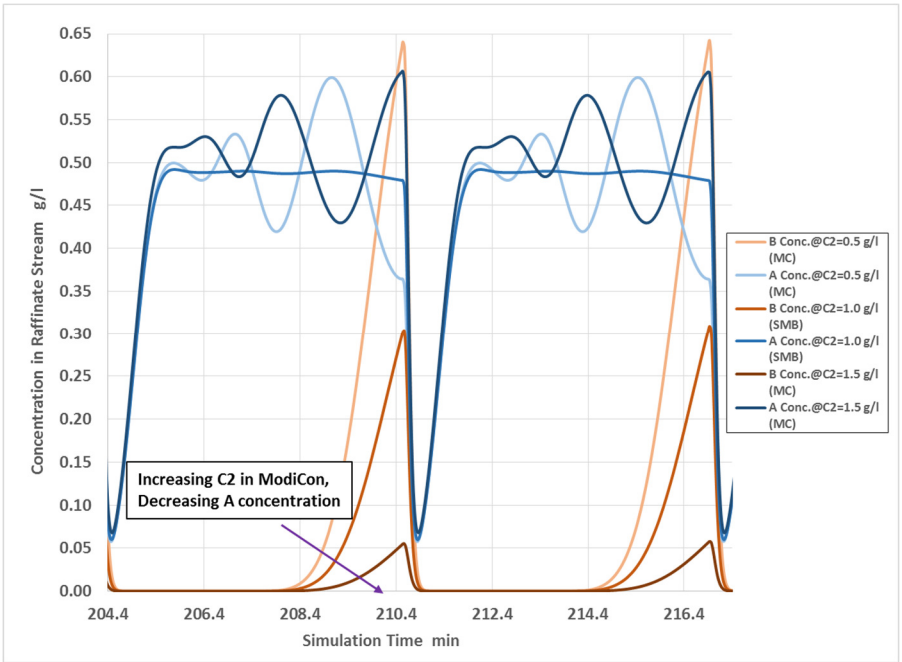
Figure 6.10 Final trend diagram for the ModiCon operation.

Figure 6.11 explains why ModiCon performs better than the basic SMB. Figure 6.11(a) shows that the concentration of B in the extract stream in both SMB and ModiCon systems is almost zero, and the concentration of A in ModiCon is clearly higher than that in the basic SMB, resulting in a higher A recovery. Figure 6.11(b) displays that the concentration of A in the raffinate for the ModiCon is lower and the concentration of B is higher than those in the basic SMB, which increases B purity and decreases A purity in ModiCon.

We analyze the product concentrations in extract and raffinate streams for different C_2 values. Figure 6.10 and Figure 6.11 both show that as we increase the C_2 values, the purity and recovery improve. Large C_2 values significantly reduce the quantity of A that is present in the raffinate which improves A recovery. It also increases the concentrations of B in the extract. We note that the resulting concentration of A is essentially zero, as in the basic SMB.



a) Extract concentrations over time (Starting after 20 cycles)



b) Raffinate concentrations over time

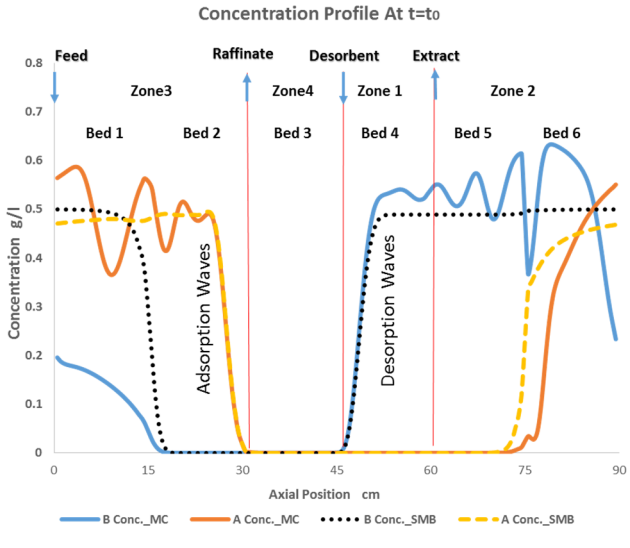
Figure 6.11 Product concentrations in SMB and ModiCon (MC) operations.

The column concentration profiles in Figure 6.12 show how modifying the subinterval feed concentrations in ModiCon achieves better separation performance. The figure shows the steady-state concentration profiles at different points within one switching interval ($t = t_0$, $t = t_0 + t_1$, $t = t_0 + t_2$ and $t = t_0 + t_{sw}$) for both ModiCon and SMB. By starting off with a lower feed concentration, the adsorption wave of component A moves more slowly and has a lower peak concentration. As a result,

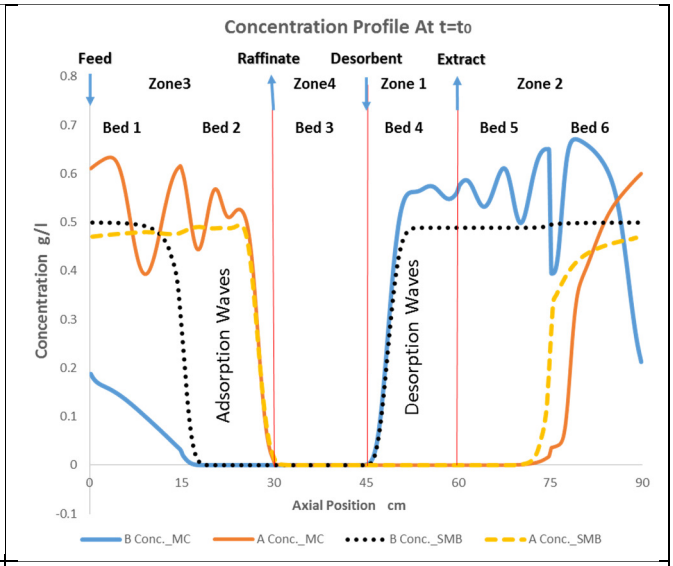
less component A reaches the raffinate stream over the course of the switching time. Since less component A is lost in the raffinate, there is more component A present in zone 1, which results in a higher concentration and recovery in the extract.

We find that the B concentration in zones 3 is lower in the ModiCon model than in the SMB model, because the components are accumulating in zone 1 and 2, which is good for extract product collection. In other words, the components are desorbed over a shorter distance in ModiCon. This explains the improved B purity; the extract contains a high concentration of B for a longer period of time.

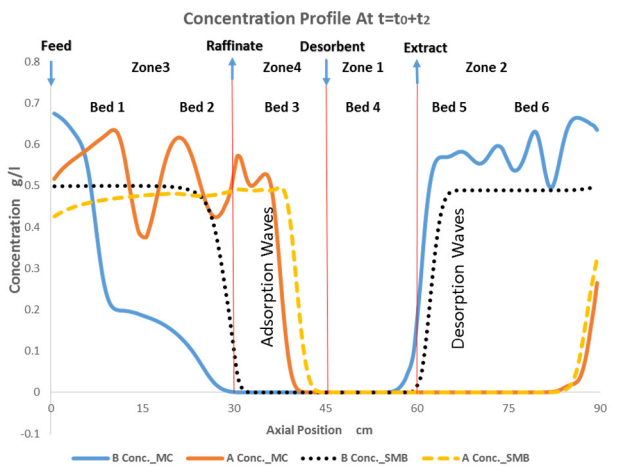
ModiCon also improves the separation performance of A. When the subinterval feed concentration in ModiCon increases, the adsorption waves in zone 3 and 4 in Figure 6.13(b) move through column more slowly, resulting in a higher concentration of B in the raffinate.



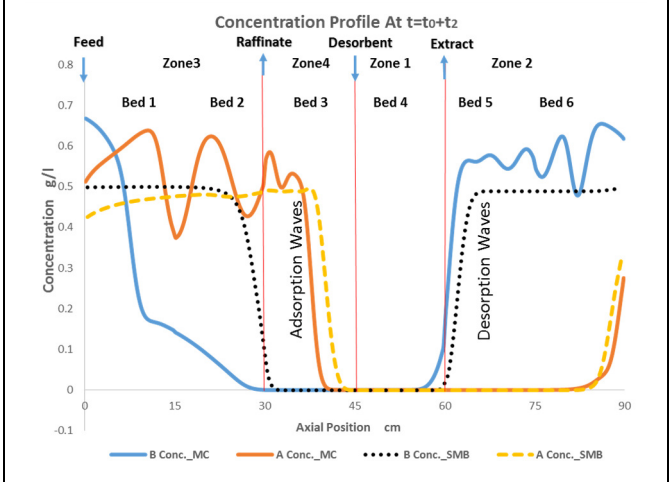
(a) Concentration Profiles at $t=t_0$ with 40 grid points.



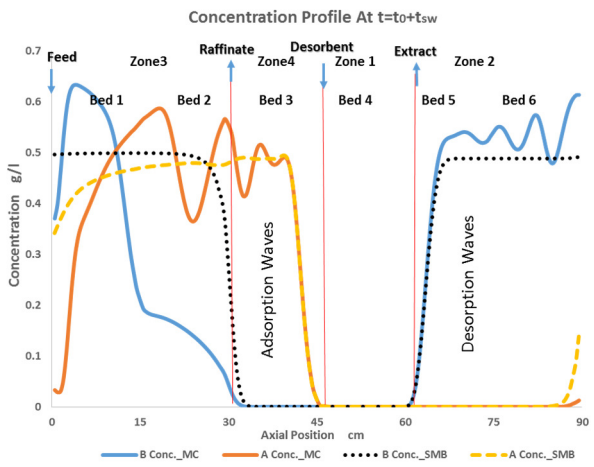
(b) Concentration Profiles at $t=t_0$ with 80 grid points.



(c) Concentration Profiles at $t=t_0+t_2$ with 40 grid points.

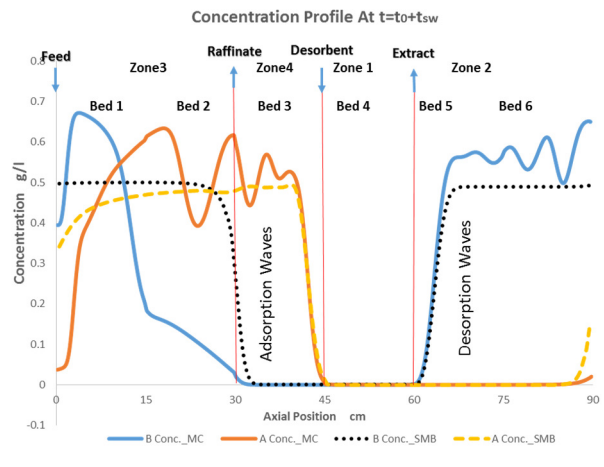


(d) Concentration Profiles at $t=t_0+t_2$ with 80 grid points.



(e) Concentration Profiles at $t=t_0+t_{sw}$ with 40 grid points.

Figure 6.12 Modicon (MC) Concentration Profiles with 40 grid points models



(f) Concentration Profiles at $t=t_0+t_{sw}$ with 80 grid points.

Figure 6.13 Modicon (MC) Concentration Profiles with 80 grid points models

6.5 Comparison of Separation Feasibility Regions of Basic Operational Modes

Figure 6.14 contains plots of the operating regions for all the investigated single operational modes. Each operational mode has two plots, one representing regions of purity, while the other represents regions of different recoveries. The difference between m_2 and m_3 is proportional to the feed flow rate, so the point that is farthest from the $m_2 = m_3$ line represents the maximum capacity of the column under each operational mode. We have circled and labeled this optimized point on each plot. Figure 6.14(a) represents the separation region with respect to product purity (the optimum point is also marked at m_2=1.55 and m_3=4.4.) Figure 6.14(b) is the m_2-m_3 plane related to product recovery, and the optimum point for both A and B recoveries higher than 99% is m_2=1.65 and m_3=4.4.

6.5.1 Varicol

The operating regions in Figure 6.14(c) and (d) correspond to the results of simulations which use the optimum Varicol configuration found in section 6.2 (i.e. Z_1-Z_2- Z_3-Z_4 = 1-2.4-1.7-0.9). For both the purity and recovery, the Varicol operational mode gives only small improvement over the base case. This result is consistent with previous work which found that the benefit of the Varicol operational mode decreases as the number of beds increases[17]. While Varicol is promising in 4- and 5-bed systems, it struggles to improve 6-bed systems.

We notice that the Varicol optimization involves a tradeoff. It may be possible to more significantly improve the operating region if we abandon our goal of maximizing the purity and recovery of both components to instead maximize one component at the expense of the other. This finding is consistent with other work which has found similar tradeoff when performing optimizations of Varicol systems[9].

6.5.2 PowerFeed

Figure 6.14(e) and (f) represent the operating regions using the optimum PowerFeed configuration found in section 6.3 (i.e. $F_1 = 0$, $F_2 = 1.14\ ml/min$ and $t_1 = 4.25\ min, t_2 = 2.13\ min.$). Like Varicol, the PowerFeed operational mode offers only small improvement over the base case, although it is slightly better than Varicol. Figure 6.14(e) shows that applying the PowerFeed enlarges the m_3 range,

which increases the feed handling capacity, compared to SMB and Varicol. If we choose to optimize for high purity of products, then we will improve our performance in the m_3 direction.

Other authors have found greater performance improvements from the PowerFeed mode, nearly as well as ModiCon[17]. However, their optimizations were performed on columns with less than 6 beds, and did not take place at constant m_2 and m_3. Therefore, there are two explanations for our different findings. First, like Varicol, the benefit from the PowerFeed mode decreases as the number of beds increases. Secondly, the optimum PowerFeed profile changes with m_2 and m_3 to a greater extent than the other operational modes. Therefore, the optimum we find at a particular m_2-m_3 point will not actually be optimum across the entire operating region.

We have the following guidelines for PowerFeed from our observations:

1. PowerFeed has very little benefit in the m_2 direction, similar to the Varicol operational mode. In this case, neither the Varicol nor PowerFeed improves the basic SMB operation.
2. Rear loading in PowerFeed has better separation performance than front loading. The best PowerFeed operation is that the feeding happens as late as possible during a switching period.
3. PowerFeed operational mode is highly sensitive to the m_2 parameter. Therefore, great care should be taken when operating a PowerFeed unit to keep the flow rate in zone 2 constant.

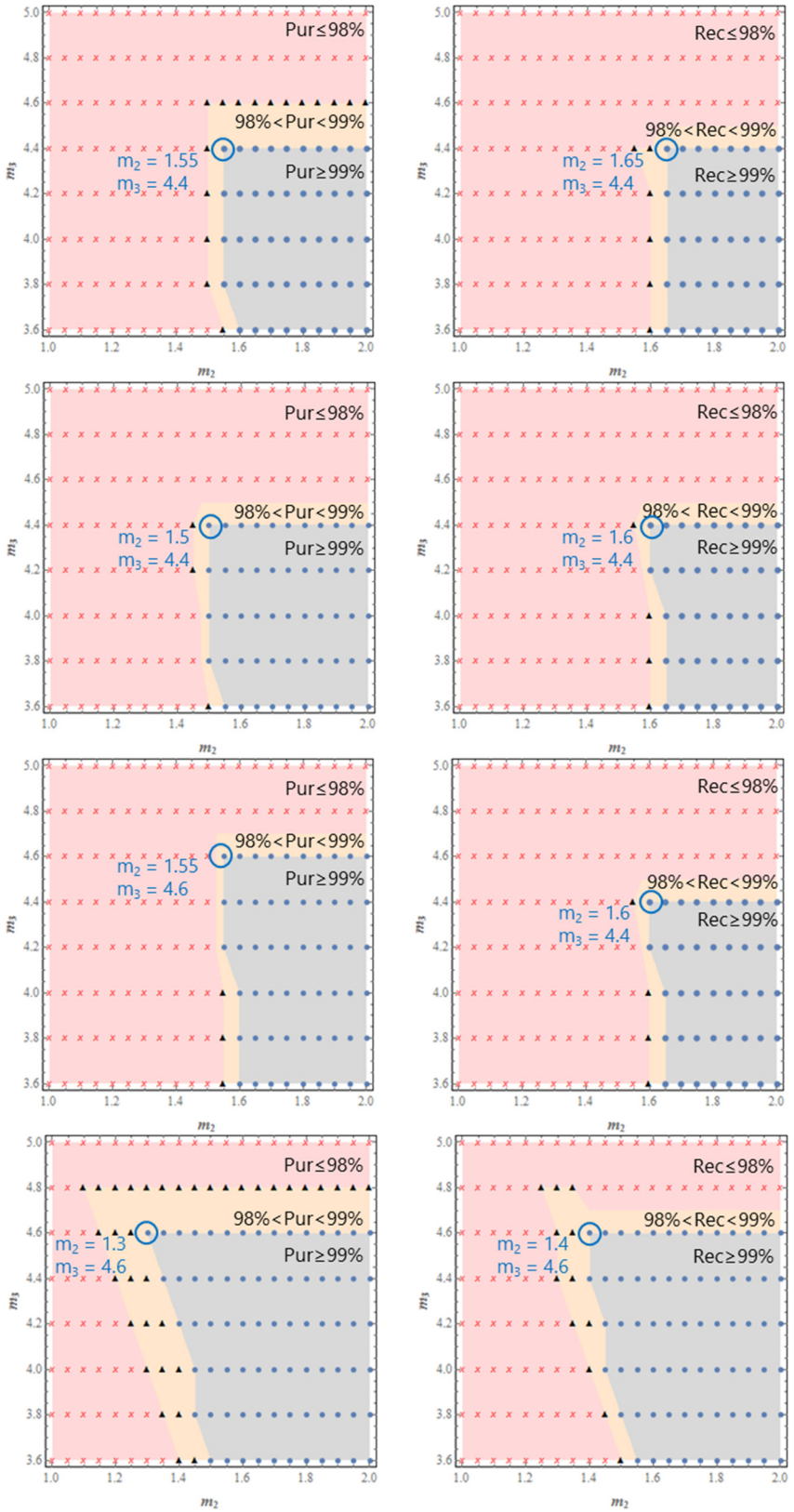
6.5.3 ModiCon

As with the previous two operational modes, the regions of complete separation for the optimized ModiCon process, Figure 6.14(g) and (h), show the purity and recovery effects of changing m_2 and m_3 values for the ModiCon operation. They compare favorably with the separation region plots for the basic SMB model in Figure 6.14(a) and (b). The grey region—the region of best performance—for the ModiCon is larger than that of the basic SMB for both purity and recovery.

Applying the ModiCon enlarges the m_3 range for product purity performance, and enlarges both the m_2 and m_3 directions for product recovery performance. The ModiCon extends the m_2 direction if we compare the m value plots in the PowerFeed operation, and also enlarge both m_2 and m_3 directions for product recovery performance. If we choose to optimize for high purity and recovery of products, ModiCon is a better choice than Varicol and PowerFeed.

We have the following guidelines for ModiCon from our observations:

1. By changing the subinterval feed concentrations in ModiCon, it is possible to significantly improve the separation performance of the SMB process, compared to Varicol and PowerFeed operational modes.
2. Middle loading in ModiCon with three subintervals has better separation performance than front loading or rear loading.
3. ModiCon operation is less sensitive to the m_2 and m_3 parameters than PowerFeed.



a) SMB Purity Regions

(b) SMB Recovery Regions

(c) Varicol Purity Regions

(d) Varicol Recovery Regions

(e) PowerFeed Purity Regions

(f) PowerFeed Recovery Regions

(g) Modicon Purity Regions

(h) Modicon Recovery Regions

Figure 6.14 Operating Region Comparison Plots for Single Operational Modes

6.6 Combined Operational Modes

In section 6.5, we find that each operational mode has unique advantages, and we want to explore whether combining these modes will provide a greater efficiency. For example, based on the previous optimization results for PowerFeed and ModiCon, it is apparent that ModiCon increases the product purity and recovery compared to the basic SMB case, while decreasing the solvent consumption. PowerFeed, on the other hand, performs similarly to the basic SMB case, while being more flexible. Therefore, we combine these two processes to see if the integrated system will perform better than either one individually.

We consider the combination of two operational modes, PowerFeed and ModiCon, and refer to it as PF&MC. We vary the feed flow rates (F_1, F_2 and F_3), recycled flow rates ($Recy_1$, $Recy_2$ and $Recy_3$) and feed concentrations (C_1, C_2 and C_3)[4, 16, 17]within given switching subintervals.

In the operational mode PF&MC, we use the following mass balance:

$$\sum_{i=1}^{N_{sub}} F_i C_i t_i = F_{SMB} C_{SMB} t_{sw} \tag{6.12}$$

We adopt essentially the same optimization steps for PF&MC as we have done for ModiCon. Equation 25 gives the updated degrees of freedom for PF&MC by adding nine new parameters (t_1, t_2, t_3, F_1, F_2, F_3 and C_1, C_2, C_3) and subtracting the two constraints:

$$DOF_{PF\&MC} = DOF_{SMB} + 9 - 2 \tag{6.13}$$

We also add Varicol to the PF&MC mode to create a V&PF&MC mode. The degrees of freedom for this mode is:

$$DOF_{V\&PF\&MC} = DOF_{SMB} + 9 - 2 + 4 \tag{6.14}$$

See equation (6.5) for Varicol for the addition of four new degrees of freedom to go from equation (6.12) to equation (6.13).

For both of these modes, we select an initial optimized model to generate the (m_2-m_3) separation plots. We find that the optimization of the combined modes proceeds similarly to the ModiCon optimization. Specifically, the operational mode performs best with a single peak of high concentration in the middle of the switching interval. Compared to the concentration changes, the changes in the feed flow rate within the switching time are modest. To demonstrate the importance of the central concentration peak, we set fixed values for sub-switching times and subinterval flow rates and increase the C_2 value from 1.48 g/l to 2.08 g/l. Figure 6.15 shows the sensitivity of B purity and A recovery to this second subinterval feed concentration. The result shows that the larger C_2, the higher B purity and A recovery are.

While the PF&MC optimization is very similar to ModiCon, the (m_2-m_3) separation plots in Figure 21(a) and (b) have enlarged the grey regions with purity and recovery greater than 99%. In other words, the changes in feed flow rate, while small compared to the changes in concentration, do allow improvement over ModiCon alone.

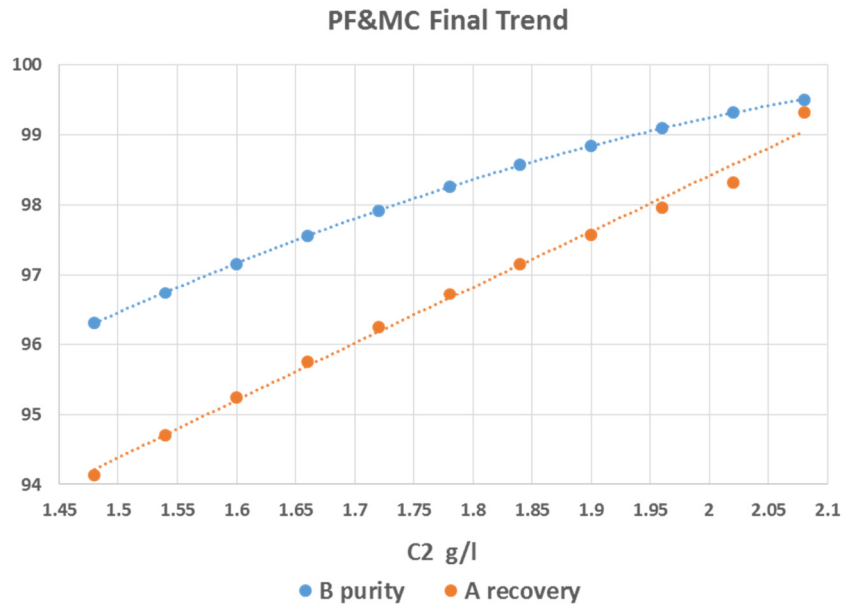
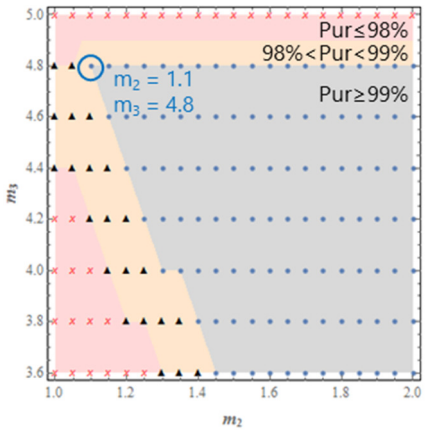
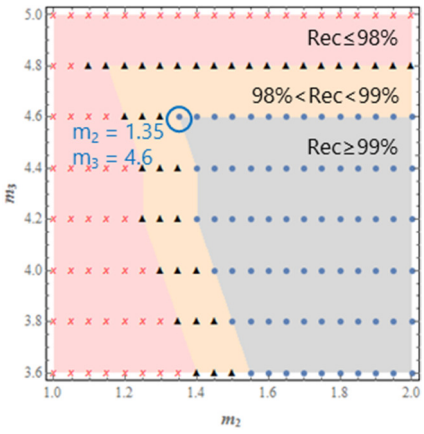


Figure 6.15 Final trend diagram for the hybrid PowerFeed and ModiCon (PF&MC) operation.

The optimization of the V&PF&MC mode starts from the optimized PF&MC mode and uses the same technique as in the Varicol optimization. It turns out that both combined modes are able to improve over the Modicon mode (the best single operational mode) and the V&PF&MC offers some small advantages in the region of complete recovery, but not purity. Therefore, adding Varicol into PF&MC offers little benefit over the PF&MC operation.

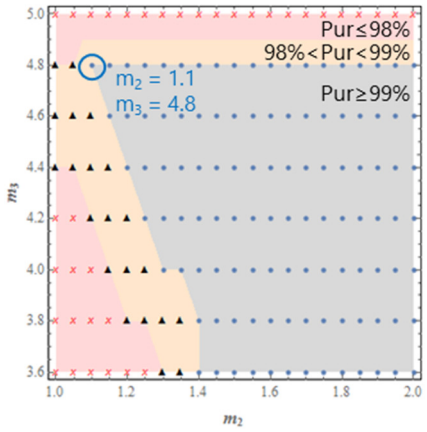
(a) PF+MC Purity Regions

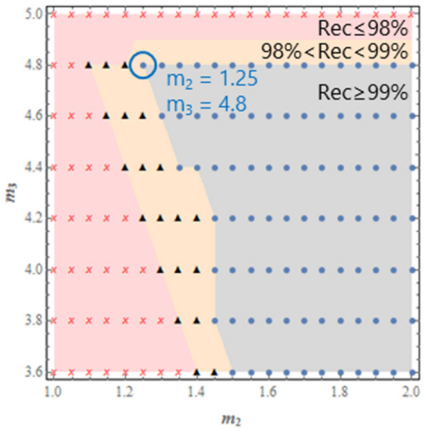




(b) PF+MC Recovery Regions

(c) V&PF&MC Purity Regions





(d) V&PF&MC Recovery Regions

Figure 6.16 Operating Region Comparison Plots for Combined Operational Modes

In section S3 of the Supporting Information, we give an application example to illustrate how to calculate the optimized points in the optimum separation regions for operational modes. In Section S4, Tables S1 and S2 list all the operating conditions and the optimum operating points for the binary separation of Tröger's base enantiomers.

6.7 Comparison of Sensitivities across All Operational Modes

To give some additional details to the comparisons made in sections 6.2 through 6.6, let us compare the sensitivity of the different operational modes side by side. Figure 6.17 through Figure 6.22 show how the different operational modes perform as m_2, m_3 and feed flow rate are varied. We express the change in feed flow rate as a deviation from the optimal feed flow rate. The optimum feed flow rate may be different for each operational mode.

Figures 22 and 23 show that the performance order is (Combined PF&MC/V&PF&MC > ModiCon > SMB/Varicol > PowerFeed) for A purity and B recovery results are same at varied m_2, but before m_2=1.6 and at fixed m_3 value of 4.4. Figure 6.19 and Figure 6.20 give B purity and A recovery results for all the operational modes at fixed m_2 and varied m_3. We see that after m_2 value close to 4.25, the performance of ModiCon, PF&MC and V&PF&MC drops steadily instead of increasing sharply like other operational modes. Figure 6.21 and Figure 6.22 show the effect of changing feed flow rate on both A and B purities. PowerFeed works the best in B purity, but the worse in A purity. As we change the feed rates, ModiCon, PF&MC and V&PF&MC give good and steady performance.

Thus, these plots show some interesting results. First, the PowerFeed operational mode is highly sensitive to the m_2 parameter, even more so than the basic SMB. Therefore, we should be cautious when operating a PowerFeed unit to keep the flow rate in zone 2 constant. The similarity of the Varicol to the SMB is expected because of the number of beds in the system, as discussed in section 6.2. The most important observation is the low sensitivity of operational modes involving ModiCon. All three scenarios involving ModiCon not only expand the operating region, *but also* show less sensitivity to m_2, m_3, and feed flow rate.

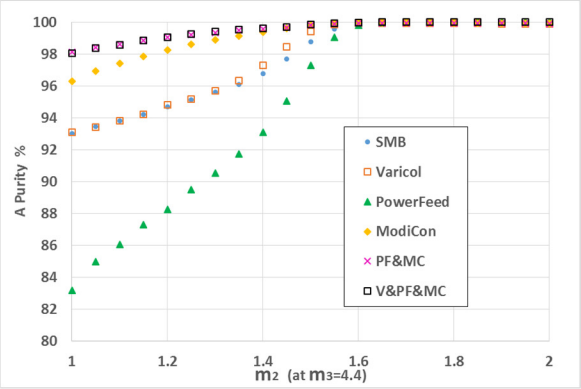


Figure 6.17 A purity effect of the m_2 value.

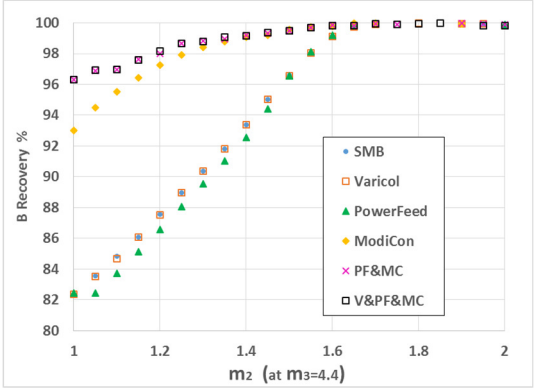


Figure 6.18 B recovery effect of the m_2 value.

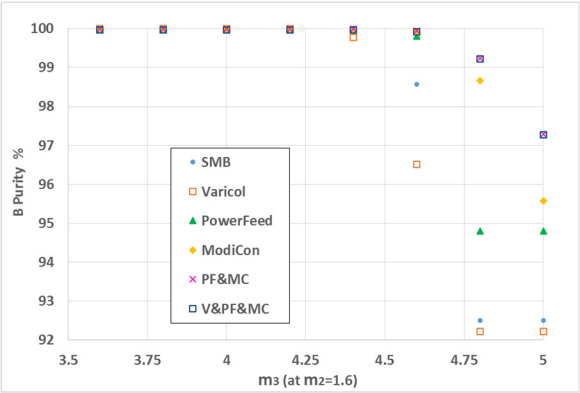


Figure 6.19 B purity effect of the m_3 value.

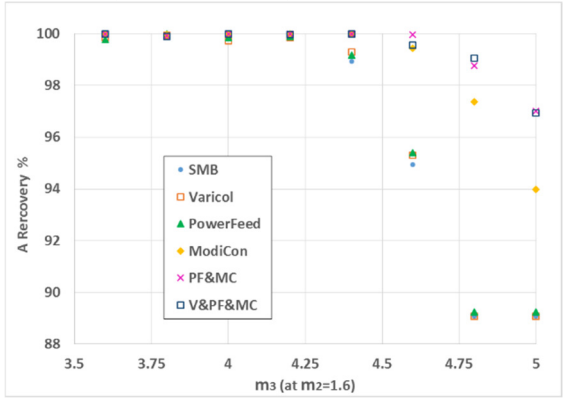


Figure 6.20 A recovery effect of the m_3 value.

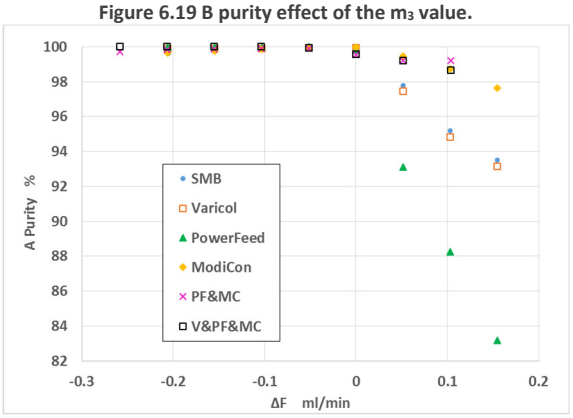


Figure 6.21 A purity effect on the change of feed flow rate.

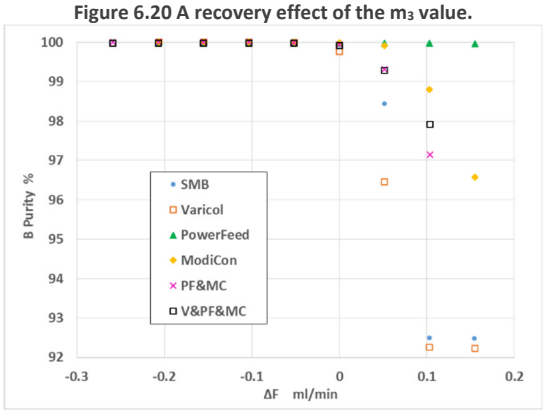


Figure 6.22 B purity effect on the change of feed flow rate.

6.8 Five-Zone SMB and Operational Modes for Ternary Separation

We apply all the operational modes we have studied for the 4-zone process for binary separations of Tröger's base enantiomers. We use the same optimization methods to find the optimized model for each operational mode: VARICOL, PowerFeed, ModiCon and combined operations. Afterwards, we examine the flow rate ratios by creating plots of separation performance plane to compare all the operational modes. In the 5-zone case, we plot the m_3-m_4 region, which is analogous to the m_2-m_3 region in a 4-zone system. The flow rate ratios m_j in a 5-zone system have precisely the same meaning as in the 4-zone system. They are the ratio of the net fluid flow rate through the zone in one switching interval to the quantity of solid adsorbent that the ports move past in one switching interval. Since the quantity of solid in each bed is constant, the difference in the m_3 and m_4 values is proportional to the feed flow rate. Therefore, we can select the best operating condition by selecting the point in an

operating region that is farthest from the $m_3 = m_4$ line (i.e. the point with the maximum feed flow rate that satisfies our purity requirements.)

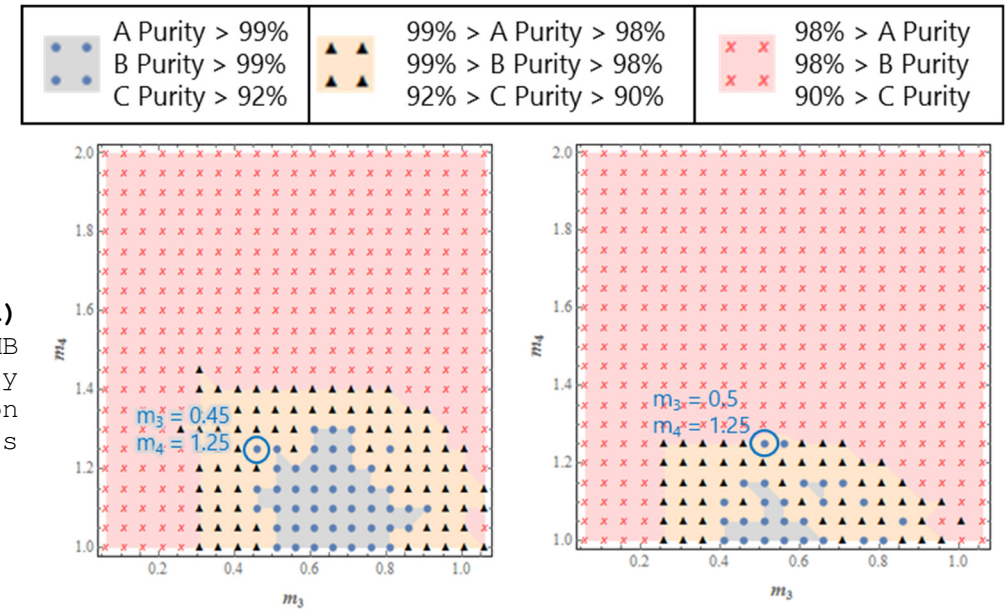
Table 8 lists the optimization results for all the operational modes in the ternary separation, while Figure 6.23 and Figure 6.24 show the optimum separation regions. We focus on evaluating the product purity performance, because operational modes affect primarily the product purity, but the product recovery in a 5-zone system.

Table 8. Operating conditions in the operational modes for ternary separation.

	Optimized Conditions for Operational Modes									
	Varicol		PowerFeed		ModiCon		PF&MC		V&PF&MC	
Optimum Operating Conditions	AS_D	0.2	t_1	9.4	t_1	9.4	t_1	7.078	C_1	0.15
	AS_F	0.5	t_2	1.6	t_2	1.6	t_2	1.573	C_2	2.605
	AS_{E1}	0.1	t_3	4.7	t_3	4.7	t_3	7.078	C_3	0.15
	AS_{E2}	0.4	F_1	0.1	C_1	0.15	F_1	0.506		
	AS_R	1	F_2	0.9	C_2	0.75	F_2	5.569	AS_D	0.6
			F_3	2.9	C_3	4.45	F_3	0.506	AS_F	0.9
			$Recy_1$	5.6			$Recy_1$	5.757	AS_{E1}	0.1
			$Recy_2$	6.0			$Recy_2$	8.289	AS_{E2}	0.6
			$Recy_3$	6.9			$Recy_3$	5.757	AS_R	1

6.9 Comparison of Separation Feasibility regions of Basic Operational Modes

The separation regions of the base case and three basic operational modes are shown in Figure 6.23. As in the 4-zone separation, we divide the simulations into three categories based on their separation performance and highlighted the simulation with the greatest feed flowrate.



(a) SMB Purity Regions

(b) Varicol Purity Regions

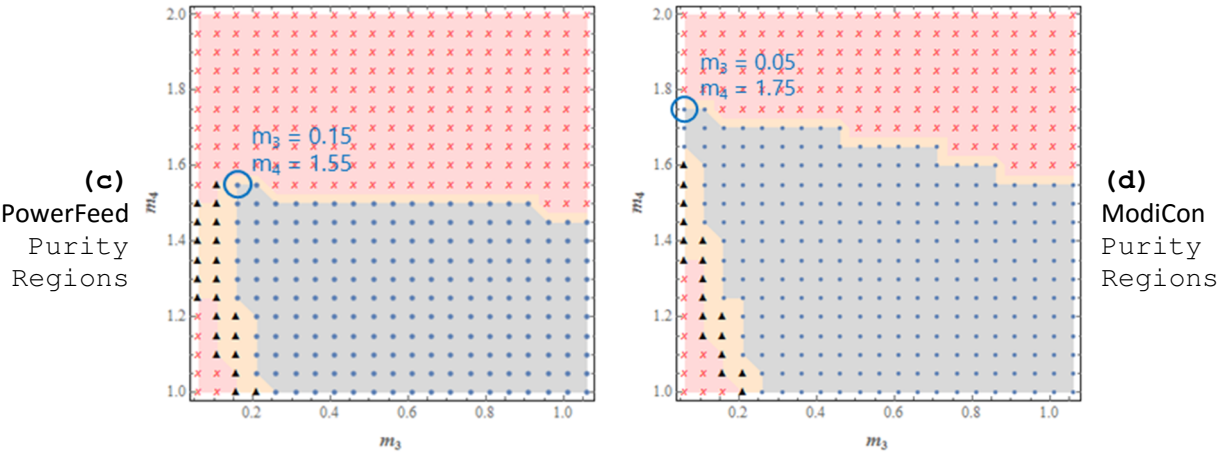


Figure 6.23 Operating Region Comparison Plots for Single Operational Modes for 5-Zone Ternary Separation.

6.9.1 Varicol Model

In the 5-zone model, we observe similar trends of product purity with regards to the zone length ratios as in the 4-zone system. In the 5-zone system, the most important parameter is the zone length of zone 5, which means that this system is at least somewhat limited by its ability to fully desorb component A from the solid in zone 5.

Figure 6.23(b) shows that like the 4-zone system, the Varicol mode by itself is unable to meaningfully expand the operating region. However, in the 5-zone system, the optimized Varicol model results in discontinuities in the region of complete separation. This indicates that the optimum Varicol model is more sensitive to the m values in the 5-zone system than in the 4-zone system.

6.9.2 PowerFeed Model

During PowerFeed operation, we vary the recycle flow rate to maintain a constant maximum pressure drop. Table 8 specifies the optimized PowerFeed model for the 5-zone ternary process.

Unlike the 4-zone system, PowerFeed performs quite well in the 5-zone system, as Figure 6.23(c) shows. It expands the operating region over the base case, and performs nearly as well as ModiCon. The 5-zone system still performs best with a rear-loading scheme, just like in the 4-zone system.

6.9.3 ModiCon Model

Once again, ModiCon gives the largest improvement in the feasible operating region, which means that the ModiCon column is able to provide both the highest productivity and lowest solvent consumption of the single operational modes for 5-zone ternary process. The ModiCon optimum point is similar to the 4-zone in that it prefers that the majority of the components be introduced in the middle of the switching time, rather than early feeding. Figure 6.23(d) shows the corresponding operation region.

6.10 Combined Operational Modes

Figure 6.24 shows that the combined operational modes do not expand the operating region nearly as well as PowerFeed or ModiCon on their own, although they do outperform the basic SMB system. Table 8 specifies the details of the combined mode settings. While the PowerFeed and ModiCon operational modes prefers late feeding, the combined system prefers a large spike of flowrate and concentration in the middle of the switching interval. The failure to outperform PowerFeed or ModiCon could be

explained by an increased sensitivity of the optimum to the m_3 and m_4 values, but it is also possible that the optimization did not find the true optimum.

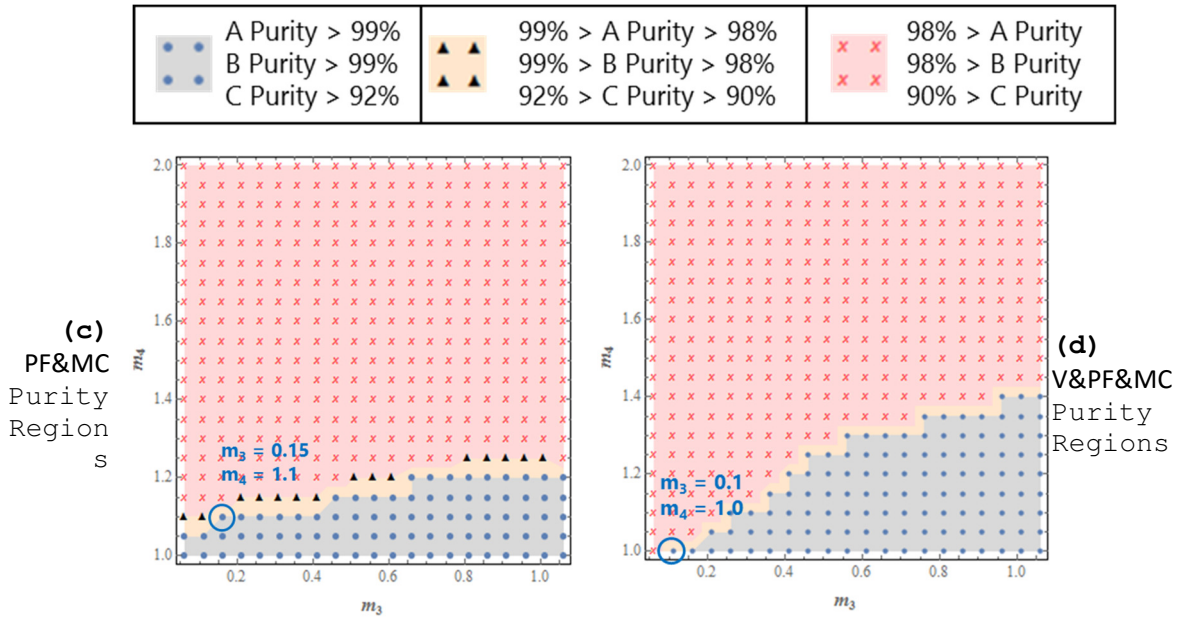


Figure 6.24 Operating Region Comparison Plots for Combined Operational Modes for 5-Zone Ternary Separation.

6.11 Findings

This work presents a comprehensive quantitative optimization of the product purity and recovery as well as desorbent consumptions of operational modes and their combinations for separating binary enantiomer and ternary amino acid mixtures. We consider the operational modes with the asynchronous shift of inlet/outlet ports ("Varicol"), variable feed flow rates ("PowerFeed" and "Partial feeding") and variable feed concentrations ("ModiCon") within time interval of inlet/outlet port switching, together with their combinations. We compare the performance of the operational modes with the basic 4-zone SMB for binary separations, and with the 5-zone SMB for ternary separations. Through simulation and optimization with Aspen Chromatography, we identify the optimum operating points and the optimum separation regions on the plane of key ratios of net liquid flow rate m_j through separation zone j in one switching interval to the quantity of solid adsorbent that the ports move past on one switching interval. We also compare the column concentration profiles at different times within a switching interval for several operational modes to provide physical insights of why one operational mode performs better or worse than the basic 4-zone SMB for binary separations.

Through optimization, we give qualitative guidelines for the selection of operating conditions for the binary separation of enantiomers. Varicol is not able to achieve greater than 99.9% purity in both components at the optimization point. However, it is possible to significantly improve on the basic 4-zone SMB in some of performance parameters, while only slightly sacrificing other performance parameters. This tradeoff is consistent with other work[9]. Like Varicol, PowerFeed gives only small improvement over the base case, although it is slightly better than Varicol. Rear feed loading within a

switching interval in PowerFeed performs better than front feed loading. ModiCon is the overall best single operational mode for both purity and recovery. Unlike PowerFeed, ModiCon performs best when we introduce the feed to the middle of the switching interval. The optimization of the combined modes performs similarly to ModiCon; specifically, it performs best with a single peak of high concentration in the middle of the switching interval.

In the optimum separation regions for binary separation, Varicol gives only small improvement over the basic 4-zone SMB. PowerFeed only improves performance in the m_3 direction. Overall, neither the Varicol nor PowerFeed gives significant improvements to the basic 4-zone SMB. PowerFeed is highly sensitive to the m_2 parameter, even more than the basic case. Therefore, we should be cautious when operating a PowerFeed unit to keep the flow rate in zone 2 constant. ModiCon is less sensitive to the m_2 and m_3 parameters than PowerFeed. ModiCon does not enlarge the operating region at the expense of robustness. Combining PowerFeed and ModiCon (PF&MC) performs better than ModiCon (the best single operational mode).

In our 5-zone SMB for ternary amino acid separation, we find some similar trends also some different ones. In Varicol, the most important parameter is the zone length of zone 5, suggesting that Varicol is limited by its ability to fully desorb component A from the solid in zone 5. The optimized Varicol model results in discontinuities in the region of complete separation, indicating that Varicol is more sensitive to the m values in the 5-zone system than in the 4-zone system. Unlike the 4-zone SMB, PowerFeed performs quite well in the 5-zone system. It expands the operating region over the base case, and performs nearly as well as ModiCon. The 5-zone PowerFeed system performs best with a rear feed loading scheme, just like in the 4-zone system. Once again, ModiCon gives the largest improvement in feasible operating region, and it is able to achieve both the highest productivity and lowest solvent consumption of the single operational modes in a 5-zone SMB. However, unlike the 4-zone system, the 5-zone ModiCon performs best when we introduce the majority of the components late in the switching interval, which is similar to the PowerFeed. The combined operational modes do not expand the operating region nearly as well as PowerFeed or ModiCon on their own, although they do outperform the basic SMB system.

7 Concluding Remarks

In this work, we have attempted to bridge the gap that has grown between two diverging sets of ideas regarding the design of SMB systems. While we successfully derived both methods in the language of Mathematica, and created usable tools for education, there is still more to do. If these tools are packaged and delivered through a textbook and via our website, they may have an even greater impact.

Computer aided process simulation is a field that is ripe for revolutionary ideas. In this dissertation, we have presented a modest proposal for a structure that allows older process simulation software to take advantage of the trends in modern computer hardware. While the approach taken here was inspired by and designed around the Aspen Custom Modeler suite of products, there is no reason that it could not be applied to any legacy simulation software.

Finally, we have shown via our operational modes research that there is significant research value to having a system that can automate and distribute important simulation tasks such as optimization and sensitivity analysis. Because of the additional efficiency afforded by our distributed system, we were

able to investigate much larger numeric problems than we could by hand. It is highly likely that this distributed approach will find many applications in process screening and optimization.

8 Literature Cited

(1) Aniceto, J. P. S.; Silva, C. M. Simulated Moving Bed Strategies and Designs: From Established Systems to the Latest Developments. *Sep. Purif. Rev.* **2013**, 131023080534003.

(2) Xie, Y.; Wu, D.; Ma, Z.; Wang, N.-H. L. Extended Standing Wave Design Method for Simulated Moving Bed Chromatography: Linear Systems. *Ind. Eng. Chem. Res.* **2000**, *39* (6), 1993.

(3) Helfferich, F. G.; Carr, P. W. Non-Linear Waves in Chromatography. *J. Chromatogr. A* **1993**, *629* (2), 97.

(4) Mazzotti, M. Design of Simulated Moving Bed Separations: Generalized Langmuir Isotherm. *Ind. Eng. Chem. Res.* **2006**, *45* (18), 6311.

(5) Yu, Y.; Wood, K. R.; Liu, Y. A. Simulation and Comparison of Operational Modes in Simulated Moving Bed Chromatography. *Ind. Eng. Chem. Res.* **2015**.

(6) Ma, Z.; Wang, N.-H. L. Standing Wave Analysis of SMB Chromatography: Linear Systems. *AIChE J.* **1997**, *43* (10), 2488.

(7) Azevedo, D.; Rodrigues, A. E. Design of a Simulated Moving Bed in the Presence of Mass-Transfer Resistances. *AIChE J.* **1999**, *45* (5), 956.

(8) Azevedo, D.; Rodrigues, A. E. Design Methodology and Operation of a Simulated Moving Bed Reactor for the Inversion of Sucrose and Glucose–fructose Separation. *Chem. Eng. J.* **2001**, *82* (1), 95.

(9) Kawajiri, Y.; Biegler, L. T. Optimization Strategies for Simulated Moving Bed and PowerFeed Processes. *AIChE J.* **2006**, *52* (4), 1343.

(10) Sreedhar, B.; Kawajiri, Y. Multi-Column Chromatographic Process Development Using Simulated Moving Bed Superstructure and Simultaneous Optimization – Model Correction Framework. *Chem. Eng. Sci.* **2014**, *116*, 428.

(11) Jin, W.; Wankat, P. C. Thermal Operation of Four-Zone Simulated Moving Beds. *Ind. Eng. Chem. Res.* **2007**, *46*, 7208.

(12) Lee, J. W.; Wankat, P. C. Design of Pseudo-Simulated Moving Bed Process with Multi-Objective Optimization for the Separation of a Ternary Mixture: Linear Isotherms. *J. Chromatogr. A* **2010**, *1217* (20), 3418.

(13) Hur, J. S.; Wankat, P. C. New Design of Simulated Moving Bed (SMB) for Ternary Separations. *Ind. Eng. Chem. Res.* **2005**, *44* (6), 1906.

(14) Langmuir, I. THE ADSORPTION OF GASES ON PLANE SURFACES OF GLASS, MICA AND PLATINUM. *J. Am. Chem. Soc.* **1918**, *40* (9), 1361.

(15) Brunauer, S.; Emmett, P. H.; Teller, E. Adsorption of Gases in Multimolecular Layers. *J. Am. Chem. Soc.* **1938**, *60* (2), 309.

(16) Ebadi, A.; Soltan Mohammadzadeh, J. S.; Khudiev, A. What Is the Correct Form of BET Isotherm for Modeling Liquid Phase Adsorption? *Adsorption* **2009**, *15* (1), 65.

(17) Freundlich, H. *Über Die Adsorption in Lösungen*; W. Engelmann, 1906.

(18) Digiano, F. A. Mathematical Models of Competitive Adsorption: Successes, Failures and Future Applications.; Pittsburgh, Pa., 1978.

(19) Kaczmarski, K.; Cavazzini, A.; Szabelski, P.; Zhou, D.; Liu, X.; Guiochon, G. Application of the General Rate Model and the Generalized Maxwell–Stefan Equation to the Study of the Mass Transfer Kinetics of a Pair of Enantiomers. *J. Chromatogr. A* **2002**, *962* (1), 57.
(20) Broughton, D. B.; Gerhold, C. G. Continuous Sorption Process Employing Fixed Bed of Sorbent and Moving Inlets and Outlets. US2985589 A, May 23, 1961.
(21) Mun, S.; Xie, Y.; Wang, N.-H. L. Robust Pinched-Wave Design of a Size-Exclusion Simulated Moving-Bed Process for Insulin Purification. *Ind. Eng. Chem. Res.* **2003**, *42* (13), 3129.
(22) Lee, K. B.; Chin, C. Y.; Xie, Y.; Cox, G. B.; Wang, N.-H. L. Standing-Wave Design of a Simulated Moving Bed under a Pressure Limit for Enantioseparation of Phenylpropanolamine. *Ind. Eng. Chem. Res.* **2005**, *44* (9), 3249.
(23) Lee, K. B.; Wang, N.-H. L. *Simulated Moving Bed Chromatography for Chiral Separation: Design, Experiment, Optimization*; Lap Lambert Academic Publishing GmbH KG, 2013.
(24) Ma, Z.; Katti, A.; Lin, B.; Guiochon, G. Simple Wave Effects in Two-Component Nonlinear Liquid Chromatography: Application to the Measurement of Competitive Adsorption Isotherms. *J. Phys. Chem.* **1990**, *94* (17), 6911.
(25) Rajendran, A.; Peper, S.; Johannsen, M.; Mazzotti, M.; Morbidelli, M.; Brunner, G. Enantioseparation of 1-Phenyl-1-Propanol by Supercritical Fluid-Simulated Moving Bed Chromatography. *J. Chromatogr. A* **2005**, *1092* (1), 55.
(26) Walton, S.; Hassan, O.; Morgan, K. Selected Engineering Applications of Gradient Free Optimisation Using Cuckoo Search and Proper Orthogonal Decomposition. *Arch. Comput. Methods Eng.* **2013**, *20* (2), 123.
(27) Sá Gomes, P.; Minceva, M.; Rodrigues, A. E. Simulated Moving Bed Technology: Old and New. *Adsorption* **2006**, *12*, 375.
(28) Juza, M.; Mazzotti, M.; Morbidelli, M. Simulated Moving-Bed Chromatography and Its Application to Chirotechnology. *Trends Biotechnol.* **2000**, *18* (3), 108.
(29) Zhang, Z.; Mazzotti, M.; Morbidelli, M. Continuous Chromatographic Processes with a Small Number of Columns: Comparison of Simulated Moving Bed with Varicol, PowerFeed, and ModiCon. *Korean J. Chem. Eng.* **2004**, *21* (2), 454.
(30) Ludemann-Hombourger, O.; Nicoud, R. M.; Bailly, M. The “VARICOL” Process: A New Multicolumn Continuous Chromatographic Process. *Sep. Sci. Technol.* **2000**, *35* (12), 1829.
(31) Ludemann-Hombourger, O.; Pigorini, G.; Nicoud, R. M.; Ross, D. S.; Terfloth, G. Application of the “VARICOL” Process to the Separation of the Isomers of the SB-553261 Racemate. *J. Chromatogr. A* **2002**, *947* (1), 59.
(32) Pais, L. S.; Rodrigues, A. E. Design of Simulated Moving Bed and Varicol Processes for Preparative Separations with a Low Number of Columns. *J. Chromatogr. A* **2003**, *1006* (1), 33.
(33) Toumi, A.; Engell, S.; Ludemann-Hombourger, O.; Nicoud, R. M.; Bailly, M. Optimization of Simulated Moving Bed and Varicol Processes. *J. Chromatogr. A* **2003**, *1006* (1), 15.
(34) Zhang, Z.; Mazzotti, M.; Morbidelli, M. PowerFeed Operation of Simulated Moving Bed Units: Changing Flow-Rates during the Switching Interval. *J. Chromatogr. A* **2003**, *1006* (1), 87.

(35) Zhang, Z.; Morbidelli, M.; Mazzotti, M. Experimental Assessment of PowerFeed Chromatography. *AIChE J.* **2004**, *50* (3), 625.
(36) Lee, K. B.; Kasat, R. B.; Cox, G. B.; Wang, N.-H. L. Simulated Moving Bed Multiobjective Optimization Using Standing Wave Design and Genetic Algorithm. *AIChE J.* **2008**, *54* (11), 2852.
(37) Amanullah, M.; Mazzotti, M. Optimization of a Hybrid Chromatography-Crystallization Process for the Separation of Tröger's Base Enantiomers. *J. Chromatogr. A* **2006**, *1107* (1–2), 36.
(38) Powell, M. J. D. An Efficient Method for Finding the Minimum of a Function of Several Variables without Calculating Derivatives. *Comput. J.* **1964**, *7* (2), 155.

9 Appendix

Appendix A.1

Triangle Theory - Langmuir

Notation

```
<< Notation`
```

```
Symbolize[ ϵ* ]
```

```
Symbolize[ ϵb ]
```

```
Symbolize[ ϵp ]
```

```
Symbolize[ v* ]
```

```
Symbolize[ Vs ]
```

```
Symbolize[ Vl ]
```

```
Symbolize[ css ]
```

```
Symbolize[ ca ]
```

```
Symbolize[ cA ]
```

```
Symbolize[ cb ]
```

```
Symbolize[ cB ]
```

```
Symbolize[ nss ]
```

```
Symbolize[ na ]
```

```
Symbolize[ nA ]
```

```
Symbolize[ nb ]
```

```
Symbolize[ nB ]
```

```
Notation[ n_a_,b_ ⟺ D[n_a_, c_b_, NonConstants → {n_1, n_2}] ]
Notation[ n_i_^ss ⟺ n^ss_i_ ]
Notation[ n_i_^a ⟺ n^a_i_ ]
Notation[ n_i_^A ⟺ n^A_i_ ]
Notation[ n_i_^b ⟺ n^b_i_ ]
Notation[ n_i_^B ⟺ n^B_i_ ]
Notation[ c_i_^ss ⟺ c^ss_i_ ]
Notation[ c_i_^a ⟺ c^a_i_ ]
Notation[ c_i_^A ⟺ c^A_i_ ]
Notation[ c_i_^b ⟺ c^b_i_ ]
Notation[ c_i_^B ⟺ c^B_i_ ]

AddInputAlias["eb" → ϵ_b ]
AddInputAlias["ep" → ϵ_p ]
AddInputAlias["phase" → ϵ_p / (1 - ϵ_p) ]
AddInputAlias["ns" → n_i^ss ]
AddInputAlias["cs" → c_i^ss ]
```

Step 1: Definition of the Problem

```
f = Table[(c_i + v* n_i) ψ, {i, 1, 2}]
g = Table[((1 - μ ϵ_p) c_i - μ (1 - ϵ_p) n_i), {i, 1, 2}]
isotherm = Table[a_i c_i / (1 + b_1 c_1 + b_2 c_2), {i, 1, 2}]
```

$\{\psi\ (c_1 + v^*\ n_1),\ \psi\ (c_2 + v^*\ n_2)\}$

$\{(1 - \epsilon_p\ \mu)\ c_1 - (1 - \epsilon_p)\ \mu\ n_1,\ (1 - \epsilon_p\ \mu)\ c_2 - (1 - \epsilon_p)\ \mu\ n_2\}$

$\left\{\frac{a_1\ c_1}{1 + b_1\ c_1 + b_2\ c_2},\ \frac{a_2\ c_2}{1 + b_1\ c_1 + b_2\ c_2}\right\}$

```
jacobian[x_] := D[x, {{c_1, c_2}}, NonConstants → {n_1, n_2}]
```

Step 2: Calculate the Various Eigenvalues and Eigenvectors

Actual Calculations

```
jacobianF = Simplify@jacobian[f]
jacobianG = Simplify@jacobian[g]
jacobianIso = Simplify@jacobian[isotherm]
jacobianH = Simplify[Inverse[jacobianG].jacobianF]

eigensFn = Simplify@Eigensystem[jacobianF]
eigensGn = Simplify@Eigensystem[jacobianG]
eigensIso = Simplify@Eigensystem[jacobianIso]
eigensHn = Simplify@Eigensystem[jacobianH]

nReplacement =
 Thread /@ Thread[Table[n_{i,j}, {i, 1, 2}, {j, 1, 2}] → jacobianIso] // Flatten

eigensF = Simplify[eigensFn /. nReplacement, Assumptions → 1 + b1 c1 + b2 c2 > 0]
eigensG = Simplify[eigensGn /. nReplacement, Assumptions → 1 + b1 c1 + b2 c2 > 0]
eigensH = Simplify[eigensHn /. nReplacement, Assumptions → 1 + b1 c1 + b2 c2 > 0]
```

Interesting Properties

The eigenvectors of the F, G, H, and isotherm matricies are the same.

```
eigensHn[[2]] == eigensFn[[2]] == eigensGn[[2]]
Simplify[Thread /@ Thread[eigensH[[2]] == eigensIso[[2]]]]
```

```
True
```

```
{{True, True}, {True, True}}
```

The eigenvalues of the matricies are related to the isotherm eigenvalues, denoted as θ

```
θEquations = Thread[eigensIso[[1]] == {θ1, θ2}]
```

F matrix's eigenvalues denoted as ϕ

```
ϕEquations = Thread[eigensF[[1]] == {ϕ1, ϕ2}];
Eliminate[Flatten[{θEquations, ϕEquations}], {a1, a2}];
Simplify[%, Assumptions → 1 + b1 c1 + b2 c2 > 0] /. And → List
ϕSolutions = Solve[%, {ϕ1, ϕ2}] // Simplify
```

$$\{\psi + v^* \psi \theta_1 == \phi_1,\ \psi + v^* \psi \theta_2 == \phi_2,\ \theta_1 (-\psi + \phi_2) == \theta_2 (-\psi + \phi_1)\}$$

$$\{\{\phi_1 \to \psi + v^* \psi \theta_1,\ \phi_2 \to \psi + v^* \psi \theta_2\}\}$$

G matrix's eigenvalues denoted as γ

```
γEquations = Thread[eigensG[[1]] == {γ1, γ2}];
Eliminate[Flatten[{θEquations, γEquations}], {a1, a2}];
Simplify[%, Assumptions → 1 + b1 c1 + b2 c2 > 0] /. And → List
Solve[%, {γ1, γ2}] // Simplify
```

$$\{1 + (-1 + \epsilon_p)\ \mu\ \theta_1 == \epsilon_p\ \mu + \gamma_1,\ 1 + (-1 + \epsilon_p)\ \mu\ \theta_2 == \epsilon_p\ \mu + \gamma_2\}$$

$$\{\{\gamma_1 \to 1 - \epsilon_p\ \mu + (-1 + \epsilon_p)\ \mu\ \theta_1,\ \gamma_2 \to 1 - \epsilon_p\ \mu + (-1 + \epsilon_p)\ \mu\ \theta_2\}\}$$

H matrix's eigenvalues denoted as η

Note the replacement rule, and extra elimination steps. For some reason, the η equations put the square root in a different form, which can make finding the solution more difficult for Mathematica. The elimination is also performed in two steps, to speed up computation.

```
replacement1 =
 √(a2^2 b2^2 (1 + b1 c1)^2 + a1^2 b1^2 (1 + b2 c2)^2 + 2 a1 a2 b1 b2 (-1 - b2 c2 + b1 c1 (-1 + b2 c2))) →
  √(-4 a1 a2 (1 + b1 c1 + b2 c2) + (a2 (1 + b1 c1) + a1 (1 + b2 c2))^2)
replacement2 = √(a2^2 (1 + b1 c1)^2 + a1^2 (1 + b2 c2)^2 + 2 a1 a2 (-1 - b2 c2 + b1 c1 (-1 + b2 c2))) →
  √(-4 a1 a2 (1 + b1 c1 + b2 c2) + (a2 (1 + b1 c1) + a1 (1 + b2 c2))^2)
ηEquations = Thread[eigensH[[1]] == {η1, η2}];
Simplify[ηEquations /. replacement1 /. replacement2];
Flatten[{θEquations, %}];
{Simplify[Eliminate[%[[{1, 3}]], a1], Assumptions → 1 + b1 c1 + b2 c2 > 0],
 Simplify[Eliminate[%[[{2, 4}]], a2], Assumptions → 1 + b1 c1 + b2 c2 > 0]}
ηsolutions = Solve[%, {η1, η2}] // Simplify
```

$$\{\psi + v^* \psi \theta_1 + \eta_1 (-1 + \epsilon_p \mu + (\mu - \epsilon_p \mu) \theta_1) == 0,\ \psi + v^* \psi \theta_2 + \eta_2 (-1 + \epsilon_p \mu + (\mu - \epsilon_p \mu) \theta_2) == 0\}$$

$$\left\{\left\{\eta_1 \to \frac{\psi + v^* \psi \theta_1}{1 - \epsilon_p \mu + (-1 + \epsilon_p) \mu \theta_1},\ \eta_2 \to \frac{\psi + v^* \psi \theta_2}{1 - \epsilon_p \mu + (-1 + \epsilon_p) \mu \theta_2}\right\}\right\}$$

Step 3: Analyze the Eigenvectors

The Eigenvectors can be Problematic at the Origin

Calculate Vector Directions at the Origin

```
vectors = eigensIso[[2]];

vectorDirections = Simplify[Normalize /@ vectors];

type1 = Limit[Abs[#], c2 → 0, Assumptions → {a1 < a2, b1 ∈ Reals}] & /@
   (vectorDirections /. c1 → 0)
type2 = Limit[Abs[#], c2 → 0, Assumptions → {a1 > a2, b1 ∈ Reals}] & /@
   (vectorDirections /. c1 → 0)
```

```
{{1, 0}, {0, 1}}
```

```
{{0, 1}, {1, 0}}
```

So as we see, the vectors at the origin point along the axes. Which vector is vertical (and therefore has infinite slope) depends on which component is more strongly adsorbed

Nice Pictures

Use Origin Information to Inform Choice Between $\frac{\partial c_1}{\partial c_2}$ or $\frac{\partial c_2}{\partial c_1}$

Assume that $a_1 < a_2$

```
slopeEq1 = vectors[[1, 2]] / vectors[[1, 1]] // Simplify
slopeEq2 = vectors[[2, 1]] / vectors[[2, 2]] // Simplify
```

$$(2\, a_2\, b_1\, c_2) \Big/ \left(-a_1 + a_2 + a_2\, b_1\, c_1 - a_1\, b_2\, c_2 + \sqrt{\left(-4\, a_1\, a_2\, (1 + b_1\, c_1 + b_2\, c_2) + (a_2\, (1 + b_1\, c_1) + a_1\, (1 + b_2\, c_2))^2\right)}\right)$$

$$-\frac{1}{2\, a_2\, b_1\, c_2} \left(a_1 - a_2 - a_2\, b_1\, c_1 + a_1\, b_2\, c_2 + \sqrt{\left(-4\, a_1\, a_2\, (1 + b_1\, c_1 + b_2\, c_2) + (a_2\, (1 + b_1\, c_1) + a_1\, (1 + b_2\, c_2))^2\right)}\right)$$

Prove that the vectors are linear

Calculate the slopes at the initial condition ($c0_1$,$c0_2$)

```
initialSlope1 = slopeEq1 /. {c1 → c01, c2 → c02}
initialSlope2 = slopeEq2 /. {c1 → c01, c2 → c02}

line1 = Simplify[c2 == c1 * initialSlope1 + (c02 - c01 * initialSlope1)]
line2 = Simplify[c1 == c2 * initialSlope2 + (c01 - c02 * initialSlope2)]
```

Ensure that the equations can work at the origin

```
line1 /. {c02 → 0}
Simplify[line2 /. {c01 → 0}, Assumptions → {a2 - a1 (1 + b2 c02) > 0}]
```

$c_2 == 0$

$c_1 == 0$

Step 4: Calculate the Intersection of the Lines

Make the line2 equation piecewise, for performance

```
line2simpler = { line2   c01 ≠ 0
               { c1 == 0 True
```

Make a function that calculates line1 and line2 at a particular point

```
linesFrom[c01_, c02_] := Simplify[{ line1, line2simpler} /. {c01 → c01, c02 → c02},
  Assumptions → { { c01 ≠ 0   ! NumericQ[c01] } ]
                 { {}         True            }
```

Calculate the lines at two different conditions

```
allIntersections[eq1_, eq2_] :=
 Outer[First[Simplify[Solve[{#1, #2}, {c1, c2}]]] &, eq1, eq2 ]
relevantIntersections[eq1_, eq2_] :=
 Extract[allIntersections[eq1, eq2], {{1, 2}, {2, 1}}]
relevantIntersections2[pt1_, pt2_] :=
 Extract[allIntersections[linesFrom @@ pt1, linesFrom @@ pt2], {{1, 2}, {2, 1}}]

relevantIntersections[linesFrom @@ {0, cI2}, linesFrom @@ {cF1, cF2}]
```

Use the line equations to generate some plots

Make the functions which will create the plots

```
plotFunction[color_, func_, max_] :=
  ContourPlot[Evaluate[func], {c1, -.1, max}, {c2, -.1, max},
   PlotRange → {0, max},
   ContourStyle → {color, Directive[Dashed, Thick, color]},
   FrameLabel → {Style["c1", Medium], Style["c2", Medium]},
   PlotLabel → "Transitions in the c1 c2 plane",
   PlotLegends →
    {"First Eigenvector Transition", "Second Eigenvector Transition"}
  ];

fullPlot[func_, initial_, final_, a1_, a2_, b1_, b2_] :=
 Block[{max = Max[initial ~Join~ final] * 1.5,
   fnc1, fnc2, lbl1, lbl2, gph1, gph2, gph3, gph4},
  fnc1 = func @@ initial /. {a1 → a1, a2 → a2, b1 → b1, b2 → b2};
  fnc2 = func @@ final /. {a1 → a1, a2 → a2, b1 → b1, b2 → b2};
  lbl1 = {c1, c2} /. Solve[{fnc1[[1]], fnc2[[2]]}] // First;
  lbl2 = {c1, c2} /. Solve[{fnc1[[2]], fnc2[[1]]}] // First;

  gph1 = Graphics[{Text["Intermediate State 1", lbl1, {1.4, -1.3}],
     PointSize[Large], Point[lbl1]}];
  gph2 = Graphics[{Text["Intermediate State 2", lbl2, {-1.1, -1.1}],
     PointSize[Large], Point[lbl2]}];
  gph3 = Graphics[Text["Initial State", initial, {-1.1, -1.1}]];
  gph4 = Graphics[Text["Final State", final, {1.1, -1.1}]];

  Show[{
    plotFunction[ColorData[97][1], fnc1, max],
    plotFunction[ColorData[97][2], fnc2, max],
    gph1,
    gph2,
    gph3,
    gph4
   }]
 ]
```

Step 5: Calculate the Eigenvalues Along the Curves

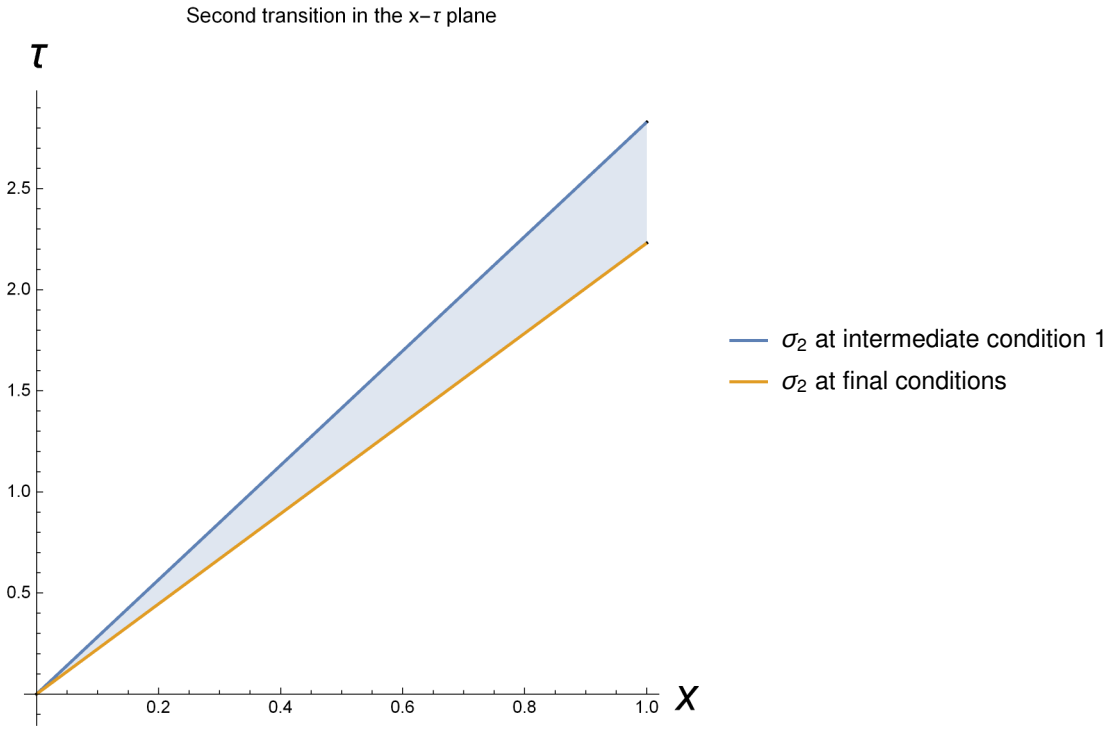
```
eigenvalues = eigensIso[[1]]
line1Replacement = Simplify@First@Solve[line1, c2]
line2Replacement = Simplify@First@Solve[line2simpler, c1]

eigenvalueEquation1 = Simplify[eigenvalues[[1]] /. line1Replacement]
eigenvalueEquation2 = Simplify[eigenvalues[[2]] /. line2Replacement]
```

Eigenvalue Plots

Make the Eigenvalue Plots $a_1 < a_2$

```
vectPlot[initial_, final_, inter1_, inter2_, a1_, a2_, b1_, b2_] :=
 Block[{fnc1, fnc2, plotslope1, plotslope2, gph1, gph2, gph3, gph4},
  fnc1 = eigenvalueEquation1 /. Thread[{c0₁, c0₂} → initial] /.
    {a₁ → a1, a₂ → a2, b₁ → b1, b₂ → b2};
  fnc2 = Simplify[eigenvalueEquation2 /. c0₁ → final[[1]],
      Assumptions → {a₂ - a₁ (1 + b₂ c0₂) > 0}] /.
    c0₂ → final[[2]] /. {a₁ → a1, a₂ → a2, b₁ → b1, b₂ → b2};
  plotslope1 = fnc1 /. {Thread[{c₁, c₂} → initial], Thread[{c₁, c₂} → inter1]};
  plotslope2 = fnc2 /. {Thread[{c₁, c₂} → inter1], Thread[{c₁, c₂} → final]};

  Print[plotslope1];
  Print[plotslope2];

  Plot[Evaluate[x * (1 + (1 - 0.4)/0.4 Flatten[{plotslope1}])],
   {x, 0, 1}, AxesLabel → (Style[#, Large] & /@ {x, τ}),
   Filling → {1 → {2}, 3 → {4}}, PlotRange → Full, AspectRatio → 1,
   PlotLabel → "First transition in the x-τ plane",
   PlotLegends → {"σ₁ at initial conditions", "σ₁ at intermediate condition 1"}

  ];
  Plot[Evaluate[x * (1 + (1 - 0.4)/0.4 Flatten[{plotslope2}])],
   {x, 0, 1}, AxesLabel → (Style[#, Large] & /@ {x, τ}),
   Filling → {1 → {2}, 3 → {4}}, PlotRange → Full, AspectRatio → 1,
   PlotLabel → "Second transition in the x-τ plane",
   PlotLegends → {"σ₂ at intermediate condition 1", "σ₂ at final conditions"}

  ]
 ]

vectPlot[{0, 0}, {5, 5}, {6.5, 0}, {2.2, 0}, 1, 2, 0.1, 0.1]
```

```
{1., 0.367309}

{1.21922, 0.820194}
```

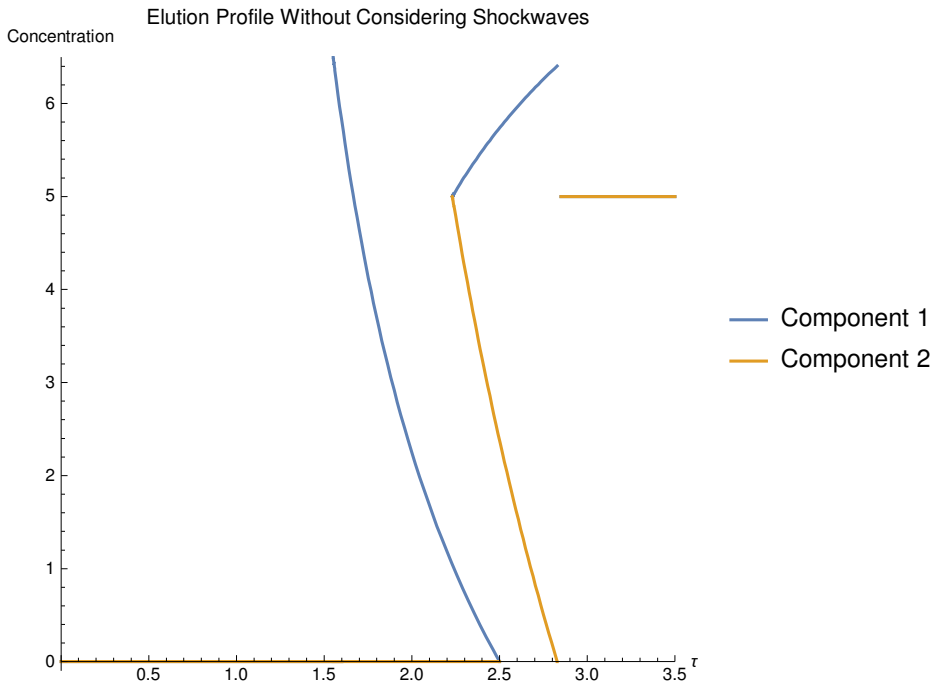


```
fullPlotExtra[func_, initial_, final_, a1_, a2_, b1_, b2_] :=
 Block[{max = Max[initial ~Join~ final] * 1.5,
   fnc1, fnc2, lbl1, lbl2, gph1, gph2, gph3, gph4, slopes, plt1},
  fnc1 = func @@ initial /. {a1 → a1, a2 → a2, b1 → b1, b2 → b2};
  fnc2 = func @@ final /. {a1 → a1, a2 → a2, b1 → b1, b2 → b2};
  lbl1 = {c1, c2} /. Solve[{fnc1[[1]], fnc2[[2]]}] // First;
  lbl2 = {c1, c2} /. Solve[{fnc1[[2]], fnc2[[1]]}] // First;

  slopes = vectPlot[initial, lbl1, lbl2, final, 1, 2, 0.1, 0.1];
  plt1 =
   Plot[Evaluate[x * (1 + (1 - 0.4)/0.4 Flatten[slopes])], {x, 0, 1}, AxesLabel → {x, τ},
    Filling → {1 → {2}, 3 → {4}}, PlotRange → Full, AspectRatio → 1, Frame -> True,
    FrameTicks → {{Range[11] - 1, Automatic}, {Range[10] / 10 // N, Automatic}},
    FrameLabel → {"x", "τ"}, PlotLabel → "Transitions in the x,τ plane"];

  gph1 = Graphics[{Text["Intermediate State 2", lbl1, {-0.5, -1.1}],
     PointSize[Large], Point[lbl1]}];
  gph2 = Graphics[{Text["Intermediate State 1", lbl2, {-1.1, -1.1}],
     PointSize[Large], Point[lbl2]}];
  gph3 = Graphics[Text["Initial State", initial, {-1.1, -1.1}]];
  gph4 = Graphics[Text["Final State", final, {1.1, -1.1}]];

  Grid[{{Show[{
       plotFunction[ColorData[97][1], fnc1, max],
       plotFunction[ColorData[97][2], fnc2, max],
       gph1,
       gph2,
       gph3,
       gph4
      }],
     Show[plt1]
    }}]
 ]
```

Full Profile

```
τprofile1 = 1 * ϕ1 /. ϕSolutions /. Θ1 → eigenvalueEquation1 // First;
τprofile2 = 1 * ϕ2 /. ϕSolutions /. Θ2 → eigenvalueEquation2 // First;

parametricCurve1 = {{τprofile1, c1}, {τprofile1, (c2 /. line1Replacement)}}
parametricCurve2 = {{τprofile2, c1 /. line2Replacement}, {τprofile2, c2}}
```

```
Block[{a1 = 1, a2 = 2, b1 = 0.1, b2 = 0.1, ψ1 = 1, v = (1 - 0.4)/0.4,
  initial = {0, 0}, final = {5, 5}, eq1, eq2, plt1, plt2, plt3, plt4},
 eq1 = parametricCurve1
    /. {a1 → a1, a2 → a2, b1 → b1, b2 → b2, ψ → ψ1, v* → v}
   /. Thread[{c01, c02} → initial];
 eq2 = parametricCurve2
    /. {a1 → a1, a2 → a2, b1 → b1, b2 → b2, ψ → ψ1, v* → v}
   /. Thread[{c01, c02} → final];

 plt1 = ParametricPlot[Evaluate[eq1], {c1, 0, 6.5},
   AspectRatio → 1, PlotRange → {{0, 3.5}, {-0.1, 6.5}},
   PlotLabel → "Elution Profile Without Considering Shockwaves",
   AxesLabel → {"τ", "Concentration"},
   PlotLegends → {"Component 1", "Component 2"}
  ];
 plt2 = ParametricPlot[Evaluate[eq2],
   {c2, 0, 5}, AspectRatio → 1, PlotRange → {{0, 3.5}, {0, 6.5}}];
 plt3 = Plot[{0, 0}, {τ, 0, 1.6}];
 plt4 = Plot[{5, 5}, {τ, 2.85, 3.5}];
 Show[plt1, plt2, plt3, plt4]
]
```



Shockwaves

$$\theta_{\text{shocks}} = \frac{(\text{isotherm} /. \{c_1 \to cF_1,\ c_2 \to cF_2\}) - (\text{isotherm} /. \{c_1 \to c0_1,\ c_2 \to c0_2\})}{\{cF_1 - c0_1,\ cF_2 - c0_2\}}$$

$$\left\{\frac{-\frac{a_1\, c0_1}{1+b_1\, c0_1+b_2\, c0_2}+\frac{a_1\, cF_1}{1+b_1\, cF_1+b_2\, cF_2}}{-c0_1+cF_1},\ \frac{-\frac{a_2\, c0_2}{1+b_1\, c0_1+b_2\, c0_2}+\frac{a_2\, cF_2}{1+b_1\, cF_1+b_2\, cF_2}}{-c0_2+cF_2}\right\}$$

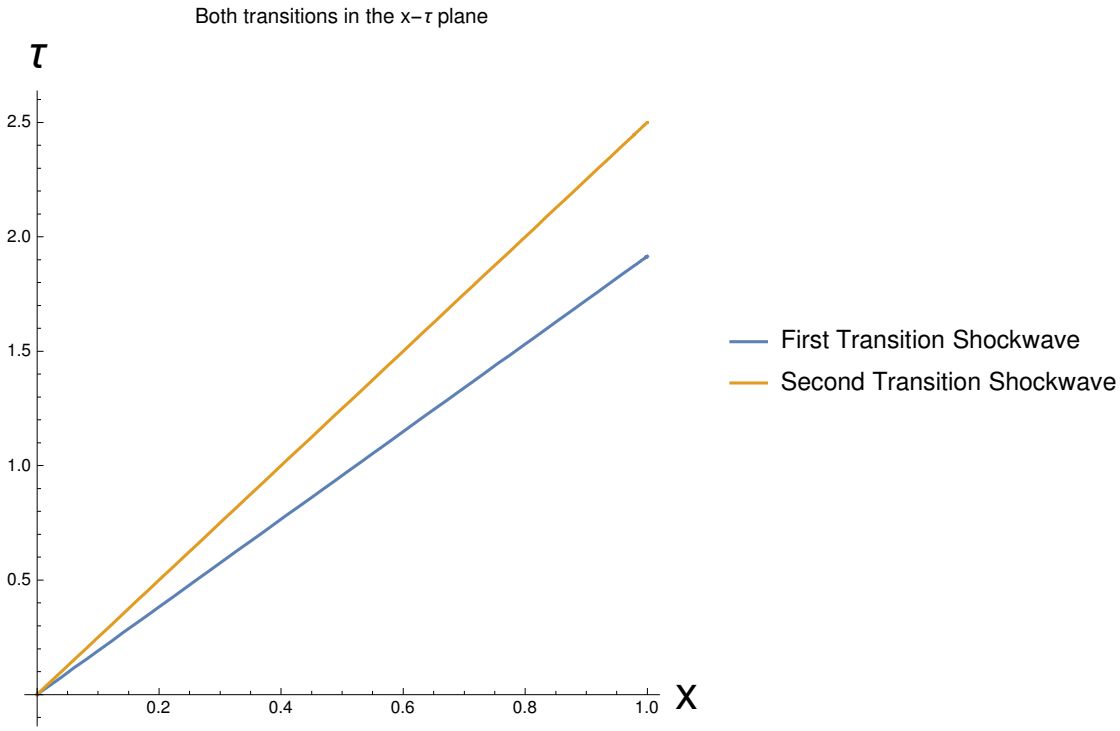
$$\tau\text{profile1shock} = \phi_1 /. \phi\text{Solutions} /. \theta_1 \to \theta_{\text{shocks}}[[1]]$$

$$\tau\text{profile2shock} = \phi_2 /. \phi\text{Solutions} /. \theta_2 \to \theta_{\text{shocks}}[[2]]$$

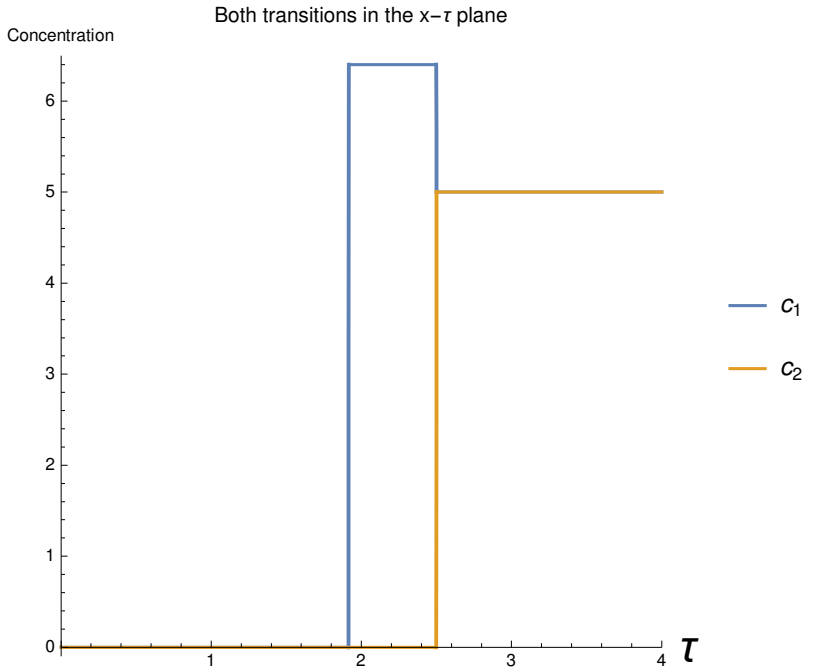
$$\left\{\psi + \frac{v^*\,\psi\left(-\frac{a_1\, c0_1}{1+b_1\, c0_1+b_2\, c0_2}+\frac{a_1\, cF_1}{1+b_1\, cF_1+b_2\, cF_2}\right)}{-c0_1+cF_1}\right\}$$

$$\left\{\psi + \frac{v^*\,\psi\left(-\frac{a_2\, c0_2}{1+b_1\, c0_1+b_2\, c0_2}+\frac{a_2\, cF_2}{1+b_1\, cF_1+b_2\, cF_2}\right)}{-c0_2+cF_2}\right\}$$

```
Block[{a1 = 1, a2 = 2, b1 = 0.1, b2 = 0.1, ψ1 = 1, v = (1 - 0.4)/0.4,
  initial = {0, 0}, inter1 = {6.4, 0}, final = {5, 5}, eq1, eq2},

 eq1 = τprofile1shock
      /. {a1 → a1, a2 → a2, b1 → b1, b2 → b2, ψ → ψ1, v* → v}
     /. Thread[{c01, c02} → initial]
    /. Thread[{cF1, cF2} → inter1];
 eq2 = τprofile2shock
      /. {a1 → a1, a2 → a2, b1 → b1, b2 → b2, ψ → ψ1, v* → v}
     /. Thread[{c01, c02} → inter1]
    /. Thread[{cF1, cF2} → final];
 Plot[{x * eq1, x * eq2}, {x, 0, 1}, PlotLegends →
   {"First Transition Shockwave", "Second Transition Shockwave"}, AspectRatio → 1,
  AxesLabel → {Style["x", Large], Style["τ", Large]},
  PlotLabel → "Both transitions in the x-τ plane"
 ]
]
```



```
tempPlot2 = Block[{a1 = 1, a2 = 2, b1 = 0.1, b2 = 0.1, ψ1 = 1,
    v = (1 - 0.4)/0.4, initial = {0, 0}, inter1 = {6.4, 0}, final = {5, 5}, eq1, eq2},
  eq1 = First[τprofile1shock]
      /. {a1 → a1, a2 → a2, b1 → b1, b2 → b2, ψ → ψ1, v* → v}
     /. Thread[{c01, c02} → initial]
    /. Thread[{cF1, cF2} → inter1];
  eq2 = First[τprofile2shock]
      /. {a1 → a1, a2 → a2, b1 → b1, b2 → b2, ψ → ψ1, v* → v}
     /. Thread[{c01, c02} → inter1]
    /. Thread[{cF1, cF2} → final];
  Show[Plot[Piecewise[{{0, τ ≤ eq1}, {6.4, eq1 < τ < eq2}, {5, τ ≥ eq2}}], {τ, 0, 4},
    ExclusionsStyle → Directive[Thick, ColorData[97][1]],
    PlotStyle → ColorData[97][1],
    PlotRange → {{0, 4}, {-0.1, 6.5}},
    PlotLegends → {"c1", "c2"}, AspectRatio → 1,
    AxesLabel → {Style["τ", Large], "Concentration"},
    PlotLabel → "Both transitions in the x-τ plane"],
   Plot[Piecewise[{{0, τ ≤ eq1}, {0, eq1 < τ < eq2}, {5, τ ≥ eq2}}], {τ, 0, 4},
    ExclusionsStyle → Directive[Thick, ColorData[97][2]],
    PlotStyle → ColorData[97][2],
    PlotRange → {{0, 4}, {-0.1, 6.5}},
    PlotLegends → {"c2"}, AspectRatio → 1,
    AxesLabel → {Style["τ", Large], "Concentration"},
    PlotLabel → "Both transitions in the x-τ plane"]
  ]
 ]
```



Step 6: Combine the Differential Equations With Mass Balances

```
state[b][A_, B_] := m δB ≤ Min[ω1 @@ A, ω1 @@ B];
state[i][A_, B_] := Max[ω1 @@ A, ω1 @@ B] ≤ m δi ≤ Min[ω2 @@ A, ω2 @@ B];
state[a][A_, B_] := Max[ω2 @@ A, ω2 @@ B] ≤ m δA;

thetas = eigensIso[[1]];
```

Describe the various constraints

Zone II

```
zone2state = {c1 → 0, c2 → cII2}
zone2thetas = Simplify[thetas /. zone2state]
zone2constraint = {zone2thetas[[1]] ∈ Reals &&
    zone2thetas[[2]] ∈ Reals && zone2thetas[[1]] ≤ m2 ≤ zone2thetas[[2]]}
```

Zone III

```
zone3states = {{cF1 → cIII1, cF2 → 0}, {c01 → cβ1, c02 → cβ2}}
zone3theta = θshocks[[1]] /. Flatten[zone3states]
zone3constraint = {zone3theta ∈ Reals && m3 ≤ zone3theta}
```

Feed concentration constraint

The steady state concentrations cannot exceed the concentrations predicted by

$(c\gamma_1 == cF_1,\ c\gamma_2 == cF_2)\ cIII_1 \le c_1$

```
relevantIntersections2[{0, 0}, {cF1, cF2}]
feedConstraint = Simplify[cII2 ≤ c2] /. Diagonal[%]
```

cγ constraints

```
cγConstraints = {cγ1 ∈ Reals && cγ1 > 0, cγ2 ∈ Reals && cγ2 > 0}
```

$\{c\gamma_1 \in \text{Reals} \&\& c\gamma_1 > 0,\ c\gamma_2 \in \text{Reals} \&\& c\gamma_2 > 0\}$

Calculate the various relationships

Calculate the steady state concentrations from the intersection of the eigenvectors

```
intersectionSolutions =
 {cIII1 → c1, cII2 → c2} /. Diagonal@relevantIntersections2[{0, 0}, {cγ1, cγ2}]
```

Calculate the steady state concentrations from the mass balances

$$\text{balanceEq1} = -(m_3 - m_2)\ cF_2 == m_2\ cII_2 - \frac{a_2\ cII_2}{1 + b_2\ cII_2};$$

$$\text{balanceEq2} = (m_3 - m_2)\ cF_1 == m_2\ cIII_1 - \frac{a_1\ cIII_1}{1 + b_1\ cIII_1};$$

```
balSolutions = Simplify@Solve[{balanceEq1, balanceEq2}, {cII2, cIII1}]
```

Combine the two steady state concentration relationships

```
combinedEquations = {
     (cIII1 /. intersectionSolutions) == cIII1,
     (cII2 /. intersectionSolutions) == cII2
    } /. balSolutions;

simplifiedEquations =
  FullSimplify[combinedEquations, Assumptions → {a1 < a2, a2 > 0}];

cγSolutions = First@Simplify@Solve[#, {cγ1, cγ2}] & /@ simplifiedEquations;
```

Calculate the beta values

$$c\beta rules = \left\{c\beta_1 \to \frac{(m_3 - m_2)\, cF_1 + isotherm[[1]] + \frac{\epsilon_p}{1-\epsilon_p}\, c\gamma_1}{m_3 + \frac{\epsilon_p}{(1-\epsilon_p)}} /. \{c_1 \to c\gamma_1,\ c_2 \to c\gamma_2\},\right.$$

$$\left. c\beta_2 \to \frac{isotherm[[2]] + \frac{\epsilon_p}{1-\epsilon_p}\, c\gamma_2}{m_3 + \frac{\epsilon_p}{(1-\epsilon_p)}} /. \{c_1 \to c\gamma_1,\ c_2 \to c\gamma_2\}\right\} // Simplify$$

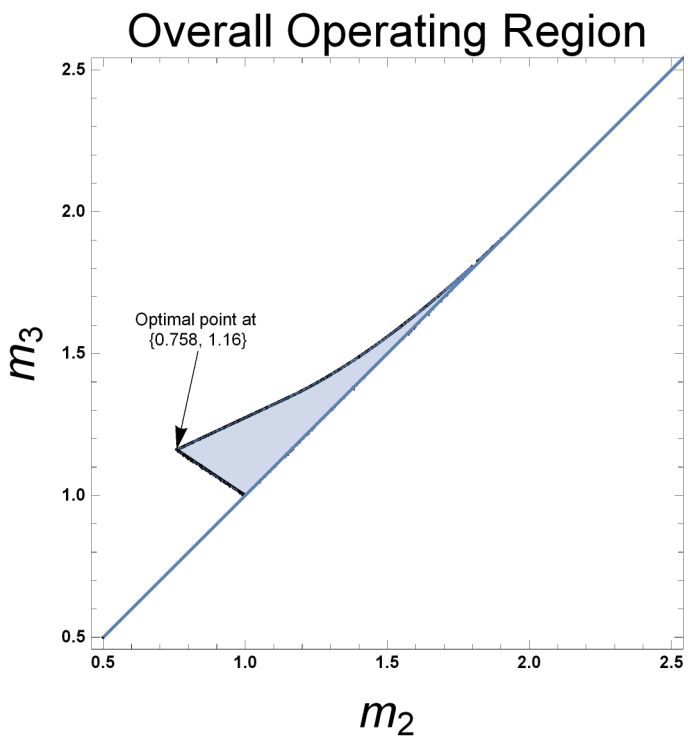
$$\left\{c\beta_1 \to \frac{c\gamma_1 \left(\frac{\epsilon_p}{1-\epsilon_p} + \frac{a_1}{1+b_1\, c\gamma_1 + b_2\, c\gamma_2}\right) + cF_1\ (-m_2 + m_3)}{\frac{\epsilon_p}{1-\epsilon_p} + m_3},\ c\beta_2 \to \frac{c\gamma_2 \left(\frac{\epsilon_p}{1-\epsilon_p} + \frac{a_2}{1+b_1\, c\gamma_1 + b_2\, c\gamma_2}\right)}{\frac{\epsilon_p}{1-\epsilon_p} + m_3}\right\}$$

Apply the relationships to the constraints

```
allConstraints = {Simplify[zone2constraint /. balSolutions],
    zone3constraint /. cβrules /. Join[cγSolutions, balSolutions, 2],
    feedConstraint /. balSolutions,
    cγConstraints /. cγSolutions};

allConstraints = {Simplify[zone2constraint /. balSolutions],
    feedConstraint /. balSolutions,
    cγConstraints /. cγSolutions};

plotRegion[r_, label_] := Show[
  RegionPlot[r, {m2, 0.5, 2.5}, {m3, 0.5, 2.5},
   MaxRecursion → 9,
   PlotPoints → 90,
   PlotLabel → Style[label, Large, Black],
   FrameLabel → {Style[m2, Large, Black], Style[m3, Large, Black]},
   FrameTicksStyle → Directive[Bold, Black]
  ], ImageSize → Medium]

Block[{r = {a1 → 1, a2 → 2, b1 → 0.1, b2 → 0.2, cF1 → 3, cF2 → 3}, equation, obj},
 LaunchKernels[4];
 equation = Simplify[allConstraints /. r];
 DistributeDefinitions[equation, plotRegion];
 obj = {ParallelSubmit[plotRegion[#, "Zone 2 Constraints"] & /@ equation[[1]]],
   ParallelSubmit[plotRegion[#, "Feed Constraints"] & /@ equation[[2]]],
   ParallelSubmit[plotRegion[#, "cγ Constraints"] & /@ equation[[3]]],
   ParallelSubmit[plotRegion[#, "Overall Operating Region"] &@
     (And @@ Flatten[allConstraints[[1 ;; 3, 2]]])]};
 WaitAll[obj]
]
CloseKernels[];
```

```
Block[
 {r = {a1 → 1, a2 → 2, b1 → 0.1, b2 → 0.2, cF1 → 3, cF2 → 3}, equation, eq2, eq3, eq4, eq5},
 eq2 = Simplify[Flatten[allConstraints[[1 ;; 3, 2]] /. r /. And → List]];
 eq3 = Cases[eq2, Element[x_, _]];
 eq4 = Complement[eq2, eq3];
 FindMaximum[{m3 - m2, eq4}, {{m3, 1.1}, {m2, 1}}]
]

Show[overallReg,
 Graphics[{
   Arrow[{{m2 * 1.1, m3 * 1.3}, {m2, m3}}] /. %[[2]] ,
   Text["Optimal point at \n" <> ToString[NumberForm[{m2, m3}, 3] /. %[[2]]]
    , {m2 * 1.1, m3 * 1.3} /. %[[2]] , {0, -1}
   ]}
 ]
]
```

```
{0.403059, {m3 → 1.16122, m2 → 0.758165}}
```



Appendix A.2

Triangle Theory - Freundlich

Notation

```
<< Notation`
```

```
Symbolize[ ϵ* ]
```

```
Symbolize[ ϵb ]
```

```
Symbolize[ ϵp ]
```

```
Symbolize[ v* ]
```

```
Symbolize[ Vs ]
```

```
Symbolize[ Vl ]
```

```
Symbolize[ css ]
```

```
Symbolize[ ca ]
```

```
Symbolize[ cA ]
```

```
Symbolize[ cb ]
```

```
Symbolize[ cB ]
```

```
Symbolize[ nss ]
```

```
Symbolize[ na ]
```

```
Symbolize[ nA ]
```

```
Symbolize[ nb ]
```

```
Symbolize[ nB ]
```

```
Symbolize[ qs ]
```

```
Notation[ n_a_,b_ ⟺ D[n_a_, c_b_, NonConstants → {n1, n2}] ]
Notation[ n^ss_i_ ⟺ n^ss_i_ ]
Notation[ n^a_i_ ⟺ n^a_i_ ]
Notation[ n^A_i_ ⟺ n^A_i_ ]
Notation[ n^b_i_ ⟺ n^b_i_ ]
Notation[ n^B_i_ ⟺ n^B_i_ ]
Notation[ c^ss_i_ ⟺ c^ss_i_ ]
Notation[ c^a_i_ ⟺ c^a_i_ ]
Notation[ c^A_i_ ⟺ c^A_i_ ]
Notation[ c^b_i_ ⟺ c^b_i_ ]
Notation[ c^B_i_ ⟺ c^B_i_ ]

AddInputAlias["eb" → ϵb ]
AddInputAlias["ep" → ϵp ]
AddInputAlias["phase" → ϵp / (1 - ϵp) ]
AddInputAlias["ns" → n_i^ss ]
AddInputAlias["cs" → c_i^ss ]
```

Step 1: Definition of the Problem

```
f = Table[(ci + v* ni) ψ, {i, 1, 2}]
g = Table[((1 - μ ϵp) ci - μ (1 - ϵp) ni), {i, 1, 2}]
isotherm = Simplify@Table[ bi^(1/t) ci / (b1^(1/t) c1 + b2^(1/t) c2)^(1-t), {i, 1, 2}]
```

```
{ψ (c1 + v* n1), ψ (c2 + v* n2)}
{(1 - ϵp μ) c1 - (1 - ϵp) μ n1, (1 - ϵp μ) c2 - (1 - ϵp) μ n2}
{b1^(1/t) c1 (b1^(1/t) c1 + b2^(1/t) c2)^(-1+t), b2^(1/t) c2 (b1^(1/t) c1 + b2^(1/t) c2)^(-1+t)}
```

```
jacobian[x_] := D[x, {{c1, c2}}, NonConstants → {n1, n2}]
```

Step 2: Calculate the Various Eigenvalues and Eigenvectors

Actual Calculations

```
jacobianF = Simplify@jacobian[f]
jacobianG = Simplify@jacobian[g]
jacobianIso = Simplify@jacobian[isotherm]
jacobianH = Simplify[Inverse[jacobianG].jacobianF]

eigensFn = Simplify@Eigensystem[jacobianF]
eigensGn = Simplify@Eigensystem[jacobianG]
eigensIso = Simplify@Eigensystem[jacobianIso]
eigensHn = Simplify@Eigensystem[jacobianH]

nReplacement =
 Thread /@ Thread[Table[n_{i,j}, {i, 1, 2}, {j, 1, 2}] → jacobianIso] // Flatten

eigensF = Simplify[eigensFn /. nReplacement, Assumptions → 1 + b1 c1 + b2 c2 > 0]
eigensG = Simplify[eigensGn /. nReplacement, Assumptions → 1 + b1 c1 + b2 c2 > 0]
eigensH = Simplify[eigensHn /. nReplacement, Assumptions → 1 + b1 c1 + b2 c2 > 0]
```

Interesting Properties

The eigenvectors of the F, G, H, and isotherm matricies are the same.

```
eigensHn[[2]] == eigensFn[[2]] == eigensGn[[2]]
FullSimplify[Thread /@ Thread[eigensH[[2]] == eigensIso[[2]]],
 Assumptions → {b1 > 0, b2 > 0, b2 > b1, c1 > 0, c2 > 0, t > 0}]
```

```
True
```

```
{{True, True}, {True, True}}
```

Step 3: Analyze the Eigenvectors

The Eigenvectors can be Problematic at the Origin

Calculate Vector Directions at the Origin

no need, when $b_1 < b_2$ the directions are {1,0} and {0,1} just like lang

So as we see, the vectors at the origin point along the axes. Which vector is vertical (and therefore has infinite slope) depends on which component is more strongly adsorbed

```
vectors = eigensIso[[2]]
```

Use Origin Information to Inform Choice Between $\frac{\partial c_1}{\partial c_2}$ or $\frac{\partial c_2}{\partial c_1}$

Assume that $b_1 < b_2$

```
slopeEq1 = FullSimplify[vectors[[1, 2]] / vectors[[1, 1]],
  Assumptions → {b1 > 0, b2 > 0, b2 > b1, c1 > 0, c2 > 0, t > 1}]
slopeEq2 = FullSimplify[vectors[[2, 1]] / vectors[[2, 2]],
  Assumptions → {b1 > 0, b2 > 0, b2 > b1, c1 > 0, c2 > 0, t > 1}]
```

Prove that the vectors are nonlinear

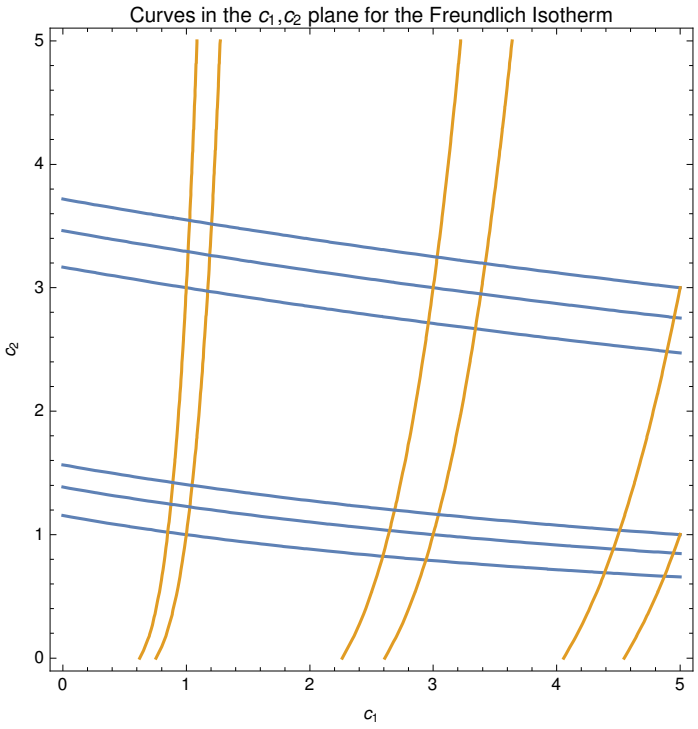
Step 4: Calculate the Intersection of the Lines

Calculate the curves with NDSolve

```
c2'[c1] == funcSlopes1
c1'[c2] == funcSlopes2

linear[b1_, b2_, c01_, c02_, t0_] :=
 Block[{r = {c2[c1] → c2, c1[c2] → c1, b1 → b1, b2 → b2, t → t0, c1 → c01, c2 → c02}, eq},
  Simplify[linetest //. r]
 ]

curve1[b1_, b2_, c01_, c02_, t0_, c1start_, c1end_] :=
 Block[{r = {b1 → b1, b2 → b2, t → t0, c1 → c1, c2 → c2}, eq},
  eq = c2'[c1] == funcSlopes1 /. r;
  NDSolve[{eq, c2[c01] == c02}, c2[c1], {c1, c1start, c1end}] /. {c1 → c1, c2 → c2}
 ]
curve2[b1_, b2_, c01_, c02_, t0_, c2start_, c2end_] :=
 Block[{r = {b1 → b1, b2 → b2, t → t0, c1 → c1, c2 → c2}, eq},
  eq = c1'[c2] == funcSlopes2 /. r;
  NDSolve[{eq, c1[c02] == c01}, c1[c2], {c2, c2start, c2end}] /. {c1 → c1, c2 → c2}
 ]
```

View Some Results

```
Table[
  Flatten[{curve1[.1, .2, ca, cb, 1.2, 0, 5], curve2[.1, .2, ca, cb, 1.2, -0.001, 5]}],
  {ca, 1, 5, 2}, {cb, 1, 3, 2}];
{c2 == c2[c1], c1[c2] == c1} /. %;
Show[Map[
  ContourPlot[Evaluate[#], {c1, 0, 5}, {c2, 0, 5},
    PlotLabel → "Curves in the c1,c2 plane for the Freundlich Isotherm",
    FrameLabel → {c1, c2}] &
  , %, {2}
 ]]
```



Make the line2 equation piecewise, for performance

```
curve2simpler[b1_, b2_, c01_, c02_, t0_, c1start_, c1end_] :=
 Piecewise[{{curve2[b1, b2, c01, c02, t0, c1start, c1end], c01 ≠ 0 && c02 ≠ 0},
   {curve2[b1, b2, c01, -0.001, t0, c1start, c1end], c01 ≠ 0 && c02 == 0},
   {{c1[c2] → 0}}, True}]
```

Calculate the c_1 and c_2 intercepts of the curves from a point

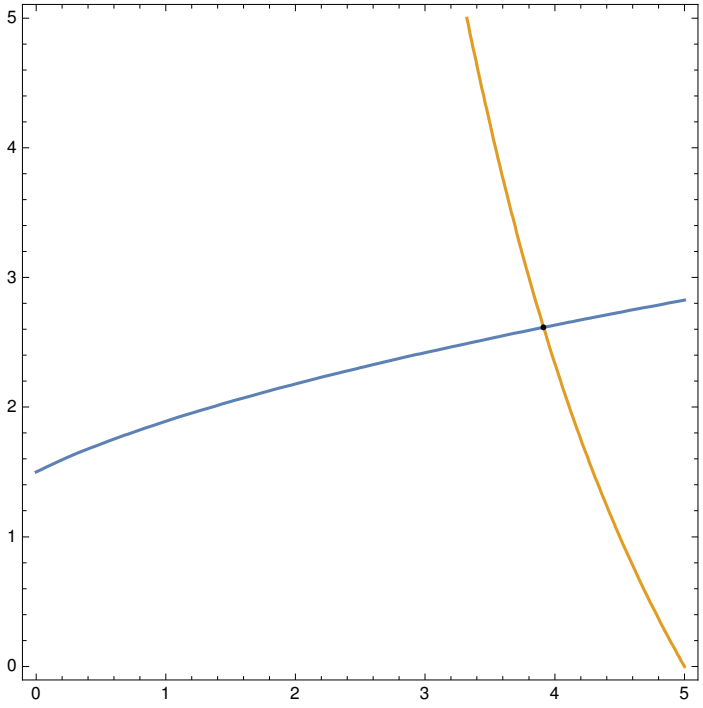
```
curve1intercept[b1_, b2_, c01_, c02_, t0_] :=
 Block[{r = {b1 → b1, b2 → b2, t → t0, c1 → c1, c2 → c2}, eq, intercept, curve},
  eq = c2'[c1] == funcSlopes1 /. r;
  curve = First@NDSolve[{eq, c2[c01] == c02}, c2[c1], {c1, c01, 0},
      Method → {"EventLocator", "Event" → c1, "EventAction" :> (intercept = c2[c1])}
     ] /. {c1 → c1, c2 → c2};
  {{0, intercept}, First@curve}
 ]
curve2intercept[b1_, b2_, c01_, c02_, t0_] :=
 Block[{r = {b1 → b1, b2 → b2, t → t0, c1 → c1, c2 → c2}, eq, intercept, curve},
  eq = c1'[c2] == funcSlopes2 /. r;
  curve = First@NDSolve[{eq, c1[c02] == c01}, c1[c2], {c2, c02, -0.00001},
      Method → {"EventLocator", "Event" → c2, "EventAction" :> (intercept = c1[c2])}
     ] /. {c1 → c1, c2 → c2};
  {{intercept, 0}, First@curve}
 ]
axesIntercepts[b1_, b2_, c01_, c02_, t0_] :=
 Transpose@Through[{curve1intercept, curve2intercept}[b1, b2, c01, c02, t0]]

axesIntercepts[.1, .2, 5, 5, .5]
```

Calculate the intersections of curves from two points on the c_1 c_2 axes

```
findIntersect[b1_, b2_, c01_, c02_, t0_] :=
 Block[{r = {b1 → b1, b2 → b2, t → t0, c1 → c1, c2 → c2},
   eq1, eq2, curve1, curve2, tempfunc, intercept},
  eq1 = c2'[c1] == funcSlopes1 /. r;
  curve1 = NDSolve[{eq1, c2[0] == c02}, c2[c1], {c1, 0, c01}];

  tempfunc = Function[{c1}, Evaluate[c2[c1] /. curve1]];

  eq2 = c1'[c2] == funcSlopes2 /. r;
  curve2 =
   NDSolve[{eq2, c1[-0.00001] == c01}, c1[c2], {c2, -0.00001, (tempfunc[c01])},
    Method → {"EventLocator", "Event" → tempfunc[c1[c2]] - c2,
      "EventAction" :↦ (intercept = {c1[c2], c2})}
   ];

  {intercept, Flatten[{curve1, curve2}]} /. {c1 → c1, c2 → c2}
 ]
```

```
findIntersect[.1, .2, 5, 1.5, .5]
ContourPlot[
 Evaluate[{c2 == c2[c1], c1 == c1[c2]} /. %[[2]] ],
 {c1, 0, 5}, {c2, 0, 5},
 Epilog → {Point[%[[1]]]}
]
```

```
{{3.91324, 2.61463}, {c2[c1] → InterpolatingFunction[ ⊞ Domain: {{0., 5.}}
                                                      Output: scalar ][c1],
  c1[c2] → InterpolatingFunction[ ⊞ Domain: {{-0.00001, 2.82}}
                                   Output: scalar ][c2]}}
```



Intercets->intercepts gets you back where you started

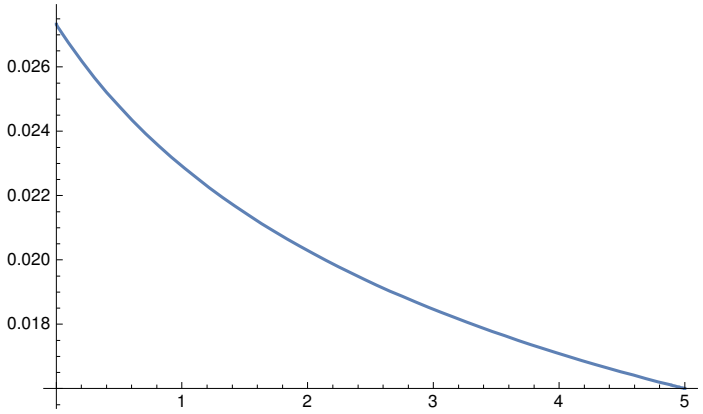
```
findIntersect[.1, .2, 5, 1.5, .5]
axesIntercepts[.1, .2, %[[1, 1]], %[[1, 2]], .5]
```

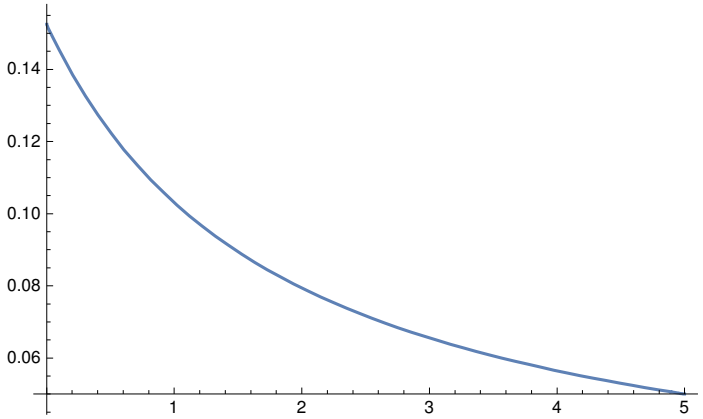
Step 5: Calculate the Eigenvalues Along the Curves

```
eigenvalues = eigensIso[[1]]
```

```
eigenvalueCurve1[transitionFunc_, b1_, b2_, t0_] :=
 eigenvalues[[1]] /. {c2 → ( c2[c1] /. transitionFunc), b1 → b1, b2 → b2, t → t0}
eigenvalueCurve2[transitionFunc_, b1_, b2_, t0_] :=
 eigenvalues[[2]] /. {c1 → ( c1[c2] /. transitionFunc), b1 → b1, b2 → b2, t → t0}
```

```
axesIntercepts[.1, .2, 5, 5, .5];
eigenvalueCurve1[%[[2, 1]], .1, .2, .5, 71];
Plot[Evaluate[%], {c1, 0, 5}]
eigenvalueCurve2[%%%[[2, 2]], .1, .2, .5, 71];
Plot[Evaluate[%], {c2, 0, 5}]
```






```
axesIntercepts[.1, .2, 5, 5, .5]
```

```
axesIntercepts[.1, .2, 5, 5, .5][[2, 2]];
eigenvalueCurve2[%, .1, .2, .5, 71];
ParametricPlot[Evaluate[{{%, c1[c2] /. %%}, {%, c2}}],
 {c2, 0, 5}, AspectRatio → 1, PlotRange → Full]
```

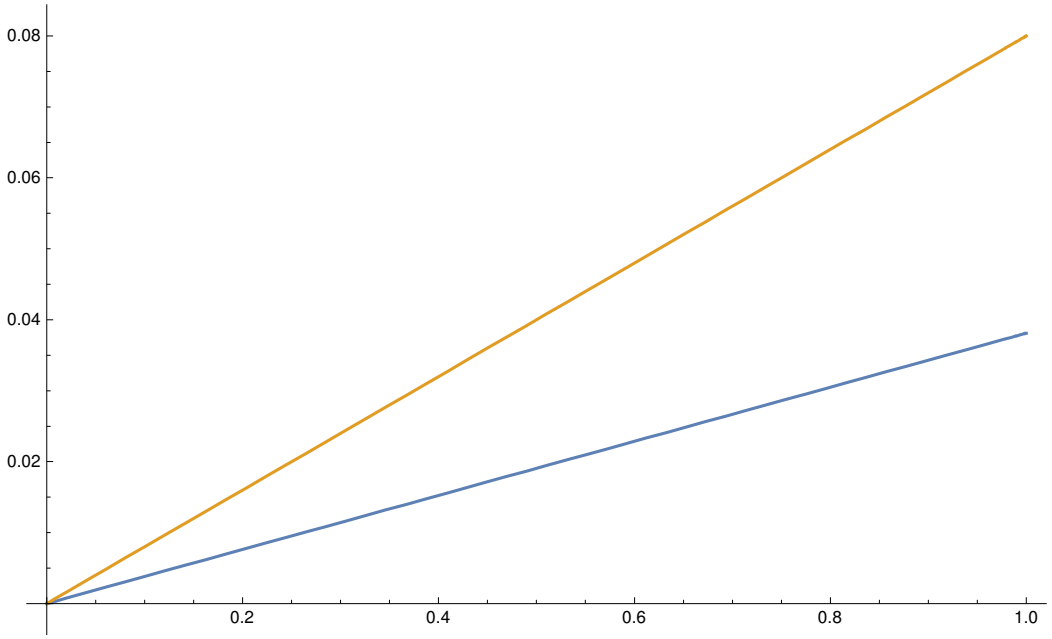
```
eigenvalueCurve1[c2[c1] → 0, .1, .2, .5, 71];
ParametricPlot[Evaluate[{{%, c1}, {%, 0}}],
 {c1, 0, 7.992}, AspectRatio → 1, PlotRange → Full]
```

Shockwaves

```
θshocks[init_, fin_, b1_, b2_, t0_] :=
  (isotherm /. Thread[{c1, c2} → fin]) - (isotherm /. Thread[{c1, c2} → init])
  ──────────────────────────────────────────────────────────────────────────── /.
                                    fin - init
   {b1 → b1, b2 → b2, t → t0, qs → qs}

axesIntercepts[.1, .2, 5, 5, .5];
{θshocks[{0, 0}, %[[1, 2]], .1, .2, .5, 71][[1]],
 θshocks[{5, 5}, %[[1, 2]], .1, .2, .5, 71][[2]]}
Plot[Evaluate[% * τ], {τ, 0, 1}]
```

Power::infy : Infinite expression $\frac{1}{0}$ encountered. ≫

```
{0.038119, 0.08}
```



Shockwave Plot

```
Block[{inter = axesIntercepts[.05, .1, 5, 5, .9], shocks},
 shocks =
  {θshocks[{0, 0}, inter[[1, 2]], .05, .1, .9][[1]],
   θshocks[{5, 5}, inter[[1, 2]], .05, .1, .9][[2]]};
 Show[
  Plot[Evaluate[
    (⎧ 0                 τ < shocks[[1]]
     ⎨ inter[[1, 2, 1]]  shocks[[1]] ≤ τ ≤ shocks[[2]]
     ⎩ 5                 True                          )
   ], {τ, 0, 10},
   ExclusionsStyle →
    {Directive[Thick, ColorData[97][1]], Directive[Thick, ColorData[97][2]]},
   PlotLegends → {"c₁"}, AspectRatio → 1,
   AxesLabel → {Style["τ", Large], "Concentration"},
   PlotLabel → "Both transitions in the x-τ plane"
  ],
  Plot[Evaluate[
    (⎧ 0                 τ < shocks[[1]]
     ⎨ inter[[1, 2, 2]]  shocks[[1]] ≤ τ ≤ shocks[[2]]
     ⎩ 5                 True                          )
   ], {τ, 0, 10},
   ExclusionsStyle → {Directive[Thick, Dashed, ColorData[97][2]]},
   PlotStyle → {Directive[Thick, Dashed, ColorData[97][2]]},
   PlotLegends → {"c₂"}, AspectRatio → 1,
   AxesLabel → {Style["τ", Large], "Concentration"},
   PlotLabel → "Both transitions in the x-τ plane"
  ]
 ]
]
```

Step 6: Combine the Differential Equations With Mass Balances

Describe the various constraints

Zone II

```
zone2state = {c1 → 0, c2 → cII2};
zone2thetas = FullSimplify[eigenvalues /. zone2state,
    Assumptions → {b1 > 0, b2 > 0, b2 > b1, c1 > 0, c2 > 0, t > 1}];
zone2constraint[b1_, b2_, c2_, t0_, m2_] := zone2thetas[[1]] ∈ Reals &&
    zone2thetas[[2]] ∈ Reals && zone2thetas[[1]] ≤ m2 ≤ zone2thetas[[2]] /.
  {b1 → b1, b2 → b2, t → t0, cII2 → c2, m2 → m2}
```

Zone III

```
zone3theta = FullSimplify[θshocks[{cIII1, 0}, {cβ1, cβ2}, b1, b2, t, qs],
    Assumptions → {b1 > 0, b2 > 0, b2 > b1, c1 ≥ 0, c2 ≥ 0, t > 1}];

zone3constraint[b1_, b2_, c3_, cb1_, cb2_, t0_] :=
 Evaluate[{zone3theta[[1]] ∈ Reals && m3 ≤ zone3theta[[1]]} /.
    {b1 → b1, b2 → b2, t → t0, cIII1 → c3, cb1 → cβ1, cb2 → cβ2}]
```

Feed concentration constraint

The steady state concentrations cannot exceed the concentrations predicted by $(c\gamma_1 == cF_1,\ c\gamma_2 == cF_2)\ cIII_1 \le c_1$

```
feedConstraint[b1_, b2_, cf1_, cf2_, t0_, c2_] :=
 Simplify[c2 ≤ c2] /. (c2 → axesIntercepts[b1, b2, cf1, cf2, t0][[1, 1, 2]])
```

cγ constraints

```
cγConstraints = {cγ1 ∈ Reals && cγ1 > 0, cγ2 ∈ Reals && cγ2 > 0}
```

$\{c\gamma_1 \in \text{Reals} \&\& c\gamma_1 > 0,\ c\gamma_2 \in \text{Reals} \&\& c\gamma_2 > 0\}$

Calculate the various relationships

Calculate the steady state concentrations from the intersection of the eigenvectors

Calculate the steady state concentrations from the mass balances

```
balanceEq1 = Simplify[-(m3 - m2) cF2 == m2 cII2 - (isotherm[[2]] /. {c1 → 0, c2 → cII2})]
balanceEq2 = Simplify[(m3 - m2) cF1 == m3 cIII1 - (isotherm[[1]] /. {c1 → cIII1, c2 → 0})]

balSolutions[b1_, b2_, cf1_, cf2_, t0_, m2_, m3_] :=
  Flatten[{balanceEq1, balanceEq2} /.
    {m2 → m2, m3 → m3, b1 → b1, cF1 → cf1, cF2 → cf2, b2 → b2, t → t0}];

Simplify[balSolutions[0.1, 0.2, 5, 5, 0.5, m2, m3], Assumptions → {m2 < m3}]
```

$$\left\{0.2\ cII_2^{0.5} + 5\ m_2 - 5\ m_3 == cII_2\ m_2,\ 0.1\ cIII_1^{0.5} - 5\ m_2 + 5\ m_3 == cIII_1\ m_3\right\}$$

```
Simplify[balSolutions[0.1, 0.2, 5, 5, 0.9, m2, m3], Assumptions → {m2 < m3}][[1]]
RegionPlot[
 Block[{m2 = mm, m3 = mmm, eq1, eq2},
  eq1 = Simplify[% /. {m2 → m2, m3 → m3}];
  eq2 = Solve[eq1] // Quiet;
  Or @@ (cII2 ∈ Reals /. eq2) && m3 ≥ m2], {mm, 0, 9}, {mmm, 0, 9},
 PlotLabel → "positive cII2 region",
 FrameLabel → {m2, m3}
]
```

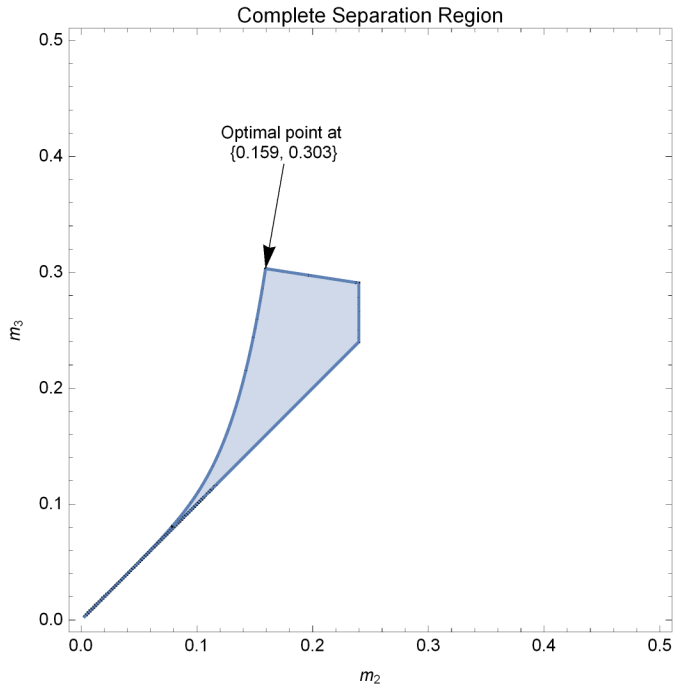
$$0.2\ cII_2^{0.9} + 5\ m_2 - 5\ m_3 == cII_2\ m_2$$

```
$Aborted
```

Combine the two steady state concentration relationships

```
test =.
test[b1_, b2_, cf1_, cf2_, t0_, m2_?NumericQ, m3_?NumericQ] :=
 And[
  m2 ≤ m3,
  Block[{eq2, c2ss, bal = balSolutions[b1, b2, cf1, cf2, t0, m2, m3]},
   eq2 = FindRoot[bal[[1]] , {cII₂, 1}] // Quiet;
   And[
    Or @@ (cII₂ ∈ Reals /. eq2),
    (Abs[bal[[1, 1]] - bal[[1, 2]]] /. eq2) < 0.001,
    (c2ss = cII₂ /. eq2);
    feedConstraint[b1, b2, cf1, cf2, t0, c2ss],
    zone2constraint[b1, b2, c2ss, t0, m2] // Quiet
   ]
  ]
 ]

sepRegion[b1_, b2_, cf1_, cf2_, t0_, m2min_, m2max_, m3min_, m3max_] :=
 Block[{plot1, nums, plot2},
  plot1 = RegionPlot[
      test[b1, b2, cf1, cf2, t0, m2, m3]
      , {m2, m2min, m2max}, {m3, m3min, m3max},
      PlotLabel → "Complete Separation Region",
      FrameLabel → {m₂, m₃},
      PlotPoints → 80,
      MaxRecursion → 2
     ] // Quiet;

  nums =
   DeleteDuplicates@Flatten[Cases[Normal@plot1, Polygon[x_] :→ x, Infinity], 1];
  max = First@MaximalBy[nums, (#[[2]] - #[[1]] &)];

  Show[plot1, Graphics[{
      Arrow[{max * {1.1, 1.3}, max}],
      Text["Optimal point at \n" <> ToString[NumberForm[max, 3]]
       , max * {1.1, 1.3}, {0, -1}
      ]}]]
 ]
```

```
sepRegion[0.1, 0.2, 5, 5, 1.2, 0, 0.5, 0, 0.5]
```



Calculate the zones 1 and 4 constraints

In zone 1, $m_1 > \theta_2[0, 0]$, so lets calculate that limit!

```
FullSimplify[eigenvalues, Assumptions → {b2 > 0, c1 == 0, c2 > 0, b1 > 0, b2 > b1, t > 1}]
θzero = Limit[%, c2 → 0, Assumptions → {b1 < b2, t ≥ 1}]
```

In zone 4, $m_4 < \theta_1[c_R, 0]$

```
FullSimplify[eigenvalues, Assumptions → {b2 > 0, c2 == 0, c1 ≥ 0, b1 > 0, b2 > b1, t > 1}]
θraff = %[[1]] /. c1 → cR1
```

$$\left\{\begin{pmatrix} t\, b_1\, c_1^{-1+t} & t\, b_1^{\frac{1}{t}} \le b_2^{\frac{1}{t}} \\ b_1^{\frac{-1+t}{t}}\, b_2^{\frac{1}{t}}\, c_1^{-1+t} & \text{True} \end{pmatrix}, \begin{pmatrix} t\, b_1\, c_1^{-1+t} & t\, b_1^{\frac{1}{t}} > b_2^{\frac{1}{t}} \\ b_1^{\frac{-1+t}{t}}\, b_2^{\frac{1}{t}}\, c_1^{-1+t} & \text{True} \end{pmatrix}\right\}$$

$$\begin{pmatrix} t\, b_1\, cR_1^{-1+t} & t\, b_1^{\frac{1}{t}} \le b_2^{\frac{1}{t}} \\ b_1^{\frac{-1+t}{t}}\, b_2^{\frac{1}{t}}\, cR_1^{-1+t} & \text{True} \end{pmatrix}$$

```
zone4MassBal = cR1 → (m3 - m2) cF1 / (m3 - m4)
```

$$cR_1 \to \frac{cF_1\ (-m_2 + m_3)}{m_3 - m_4}$$

```
zone4constraint = FullSimplify[m4 ≤ θraff /. zone4MassBal,
  Assumptions → {b2 > 0, c1 ≥ 0, b1 > 0, b2 > b1, t > 1}]
```

$$m_4 \le \begin{cases} t\, b_1 \left(\frac{cF_1\,(-m_2+m_3)}{m_3-m_4}\right)^{-1+t} & t\, b_1^{\frac{1}{t}} \le b_2^{\frac{1}{t}} \\ b_1^{\frac{-1+t}{t}}\, b_2^{\frac{1}{t}} \left(\frac{cF_1\,(-m_2+m_3)}{m_3-m_4}\right)^{-1+t} & \text{True} \end{cases}$$

```
region2test[b1_, b2_, cf1_, cf2_, t0_, m2_, m3_, m1_, m4_] := With[
  {r = {b1 → b1, b2 → b2, cF1 → cf1, cF2 → cf2, t → t0, m1 → m1, m2 → m2, m3 → m3, m4 → m4}},
  Simplify@And[
    zone1constraint /. r,
    zone4constraint /. r
   ]
 ]
```

```
region2test[0.06, 0.09, 5, 5, 1.3, 0.06, 0.14, m1, m4]
```

$$m1 > 0 \;\&\&\; m4 \le 0.0592533 \left(-\frac{1.}{-0.14+m4}\right)^{0.3}$$

```
separationRegion2[b1_, b2_, cf1_, cf2_,
  t0_, m2_, m3_, m1min_, m1max_, m4min_, m4max_] :=
 Block[{eq, plot1, nums},
  eq = region2test[b1, b2, cf1, cf2, t0, m2, m3, m1, m4];
  plot1 = RegionPlot[
      eq
      , {m4, m4min, m4max}, {m1, m1min, m1max},
      PlotLabel → "Complete Separation Region",
      FrameLabel → {m4, m1},
      PlotPoints → 80,
      MaxRecursion → 2
     ] // Quiet;
  nums =
   DeleteDuplicates@Flatten[Cases[Normal@plot1, Polygon[x_] :→ x, Infinity], 1];
  max2 = Chop[#, 10^-6] &@First@MaximalBy[nums, (#[[1]] - #[[2]] &)];

  Show[plot1, Graphics[{
      Arrow[{max2 * {1.4, 0.8} + {1, 1}, max2}],
      Text["Optimal point at \n" <> ToString[NumberForm[max2, 3]]
       , max2 * {1.4, 0.8} + {1, 1}, {0, -1}
      ]}]]
 ]
```

Appendix A.3

Triangle Theory - Sips

Notation

```
<< Notation`
```

```
Symbolize[ ϵ* ]
```

```
Symbolize[ ϵb ]
```

```
Symbolize[ ϵp ]
```

```
Symbolize[ v* ]
```

```
Symbolize[ Vs ]
```

```
Symbolize[ Vl ]
```

```
Symbolize[ css ]
```

```
Symbolize[ ca ]
```

```
Symbolize[ cA ]
```

```
Symbolize[ cb ]
```

```
Symbolize[ cB ]
```

```
Symbolize[ nss ]
```

```
Symbolize[ na ]
```

```
Symbolize[ nA ]
```

```
Symbolize[ nb ]
```

```
Symbolize[ nB ]
```

```
Symbolize[ qs ]
```

```
Notation[ n_{a_,b_} ⟺ D[n_{a_}, c_{b_}, NonConstants → {n_1, n_2}] ]
Notation[ n^{ss}_{i_} ⟺ n^{ss}_{i_} ]
Notation[ n^{a}_{i_} ⟺ n^{a}_{i_} ]
Notation[ n^{A}_{i_} ⟺ n^{A}_{i_} ]
Notation[ n^{b}_{i_} ⟺ n^{b}_{i_} ]
Notation[ n^{B}_{i_} ⟺ n^{B}_{i_} ]
Notation[ c^{ss}_{i_} ⟺ c^{ss}_{i_} ]
Notation[ c^{a}_{i_} ⟺ c^{a}_{i_} ]
Notation[ c^{A}_{i_} ⟺ c^{A}_{i_} ]
Notation[ c^{b}_{i_} ⟺ c^{b}_{i_} ]
Notation[ c^{B}_{i_} ⟺ c^{B}_{i_} ]

AddInputAlias["eb" → ϵ_b ]
AddInputAlias["ep" → ϵ_p ]
AddInputAlias["phase" → ϵ_p / (1 - ϵ_p) ]
AddInputAlias["ns" → n^{ss}_i ]
AddInputAlias["cs" → c^{ss}_i ]
```

Step 1: Definition of the Problem

```
f = Table[(c_i + v* n_i) ψ, {i, 1, 2}]
g = Table[((1 - μ ϵ_p) c_i - μ (1 - ϵ_p) n_i), {i, 1, 2}]
isotherm = Simplify@Table[ a_i c_i / (1 + b_1 c_1^t + b_2 c_2^t), {i, 1, 2}]
```

$\{\psi\ (c_1 + v^*\ n_1),\ \psi\ (c_2 + v^*\ n_2)\}$

$\{(1 - \epsilon_p\ \mu)\ c_1 - (1 - \epsilon_p)\ \mu\ n_1,\ (1 - \epsilon_p\ \mu)\ c_2 - (1 - \epsilon_p)\ \mu\ n_2\}$

$\left\{\frac{a_1\ c_1}{1 + b_1\ c_1^t + b_2\ c_2^t},\ \frac{a_2\ c_2}{1 + b_1\ c_1^t + b_2\ c_2^t}\right\}$

```
jacobian[x_] := D[x, {{c_1, c_2}}, NonConstants → {n_1, n_2}]
```

Step 2: Calculate the Various Eigenvalues and Eigenvectors

Actual Calculations

Interesting Properties

The eigenvectors of the F, G, H, and isotherm matricies are the same.

```
eigensHn[[2]] == eigensFn[[2]] == eigensGn[[2]]
FullSimplify[Thread /@ Thread[eigensH[[2]] == eigensIso[[2]]],
 Assumptions → {b1 > 0, b2 > 0, a2 > a1, c1 > 0, c2 > 0, t > 0}]
```

```
True
```

```
{{True, True}, {True, True}}
```

Step 3: Analyze the Eigenvectors

The Eigenvectors can be Problematic at the Origin

Calculate Vector Directions at the Origin

no need, when $b_1 < b_2$ the directions are {1,0} and {0,1} just like lang

So as we see, the vectors at the origin point along the axes. Which vector is vertical (and therefore has infinite slope) depends on which component is more strongly adsorbed

```
vectors = eigensIso[[2]]
```

$$\left\{\left\{\frac{1}{2\,t\,a_2\,b_1\,c_2^2}\,c_1^{-t}\right.\right.$$
$$\left(-a_1\,c_1\,c_2+a_2\,c_1\,c_2-a_1\,b_1\,c_1^{1+t}\,c_2+t\,a_1\,b_1\,c_1^{1+t}\,c_2+a_2\,b_1\,c_1^{1+t}\,c_2-a_1\,b_2\,c_1\,c_2^{1+t}+a_2\,b_2\,c_1\,c_2^{1+t}-\right.$$
$$t\,a_2\,b_2\,c_1\,c_2^{1+t}+\sqrt{\left(c_1^2\,c_2^2\left(4\,a_1\,a_2\,\left(1+b_1\,c_1^t+b_2\,c_2^t\right)\left(-1+(-1+t)\,b_1\,c_1^t+(-1+t)\,b_2\,c_2^t\right)+\right.\right.}$$
$$\left.\left.\left.\left.\left(a_1\left(-1+(-1+t)\,b_1\,c_1^t-b_2\,c_2^t\right)-a_2\left(1+b_1\,c_1^t-(-1+t)\,b_2\,c_2^t\right)\right)^2\right)\right),\,1\right\},$$
$$\left\{\frac{1}{2\,t\,a_2\,b_1\,c_2^2}\,c_1^{-t}\left(-a_1\,c_1\,c_2+a_2\,c_1\,c_2-a_1\,b_1\,c_1^{1+t}\,c_2+t\,a_1\,b_1\,c_1^{1+t}\,c_2+a_2\,b_1\,c_1^{1+t}\,c_2-\right.\right.$$
$$a_1\,b_2\,c_1\,c_2^{1+t}+a_2\,b_2\,c_1\,c_2^{1+t}-t\,a_2\,b_2\,c_1\,c_2^{1+t}-$$
$$\sqrt{\left(c_1^2\,c_2^2\left(4\,a_1\,a_2\,\left(1+b_1\,c_1^t+b_2\,c_2^t\right)\left(-1+(-1+t)\,b_1\,c_1^t+(-1+t)\,b_2\,c_2^t\right)+\right.\right.}$$
$$\left.\left.\left.\left.\left.\left(a_1\left(-1+(-1+t)\,b_1\,c_1^t-b_2\,c_2^t\right)-a_2\left(1+b_1\,c_1^t-(-1+t)\,b_2\,c_2^t\right)\right)^2\right)\right),\,1\right\}\right\}$$

Use Origin Information to Inform Choice Between $\frac{\partial c_1}{\partial c_2}$ or $\frac{\partial c_2}{\partial c_1}$

Assume that $b_1 < b_2$

```
slopeEq1 = FullSimplify[vectors[[1, 2]] / vectors[[1, 1]],
  Assumptions → {a1 > 0, a2 > 0, b1 > 0, b2 > 0, a2 > a1, c1 ≥ 0, c2 ≥ 0, t > 0}]
slopeEq2 = FullSimplify[vectors[[2, 1]] / vectors[[2, 2]],
  Assumptions → {a1 > 0, a2 > 0, b1 > 0, b2 > 0, a2 > a1, c1 ≥ 0, c2 ≥ 0, t > 0}]
```

Prove that the vectors are nonlinear

```
funcSlopes1 = slopeEq1 /. c2 → c2[c1];
funcSlopes2 = slopeEq2 /. c1 → c1[c2];

secondDerivative1 = Simplify[∂c1 funcSlopes1,
   Assumptions → {a1 > 0, a2 > 0, b1 > 0, b2 > 0, a2 > a1, c1 ≥ 0, c2 ≥ 0, t > 0}];
secondDerivative2 = Simplify[∂c2 funcSlopes2,
   Assumptions → {a1 > 0, a2 > 0, b1 > 0, b2 > 0, a2 > a1, c1 ≥ 0, c2 ≥ 0, t > 0}];

linetest = {Simplify[secondDerivative1 /. c2'[c1] → funcSlopes1,
   Assumptions → {a1 > 0, a2 > 0, b1 > 0, b2 > 0, a2 > a1, c1 ≥ 0, c2 ≥ 0, t > 0}],
  Simplify[secondDerivative2 /. c1'[c2] → funcSlopes2,
   Assumptions → {a1 > 0, a2 > 0, b1 > 0, b2 > 0, a2 > a1, c1 ≥ 0, c2 ≥ 0, t > 0}]}
```

Step 4: Calculate the Intersection of the Lines

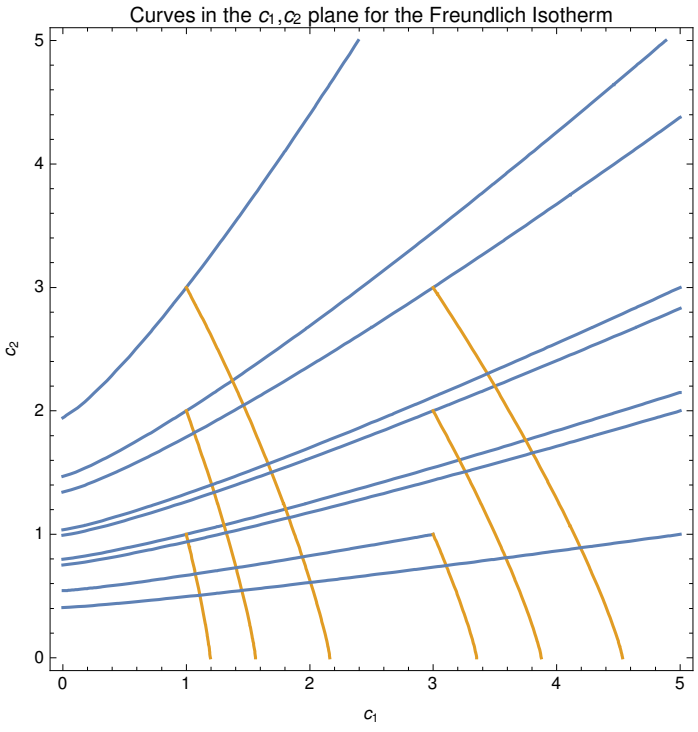
Calculate the curves with NDSolve

```
c2'[c1] == funcSlopes1
c1'[c2] == funcSlopes2

linear[a1_, a2_, b1_, b2_, c01_, c02_, t0_] :=
 Block[{r = {c2[c1] → c2, c1[c2] → c1, a1 → a1,
     a2 → a2, b1 → b1, b2 → b2, t → t0, c1 → c01, c2 → c02}, eq},
  Simplify[linetest //. r]
 ]
```

```
curve1[a1_, a2_, b1_, b2_, c01_, c02_, t0_, c1start_, c1end_] :=
 Block[{r = {a1 → a1, a2 → a2, b1 → b1, b2 → b2, t → t0, c1 → c1, c2 → c2}, eq},
  eq = c2'[c1] == funcSlopes1 /. r;
  NDSolve[{eq, c2[c01] == c02}, c2[c1], {c1, c1start, c1end}] /. {c1 → c1, c2 → c2}
 ]
curve2[a1_, a2_, b1_, b2_, c01_, c02_, t0_, c2start_, c2end_] :=
 Block[{r = {a1 → a1, a2 → a2, b1 → b1, b2 → b2, t → t0, c1 → c1, c2 → c2}, eq},
  eq = c1'[c2] == funcSlopes2 /. r;
  NDSolve[{eq, c1[c02] == c01}, c1[c2], {c2, c2start, c2end}] /. {c1 → c1, c2 → c2}
 ]
```

View Some Results

```
Table[Flatten[{curve1[1, 2, .1, .2, ca, cb, 1.2, 0, 5],
    curve2[1, 2, .1, .2, ca, cb, 1.2, -0.001, 5]}], {ca, 1, 5, 2}, {cb, 1, 3, 1}];
{c2 == c2[c1], c1[c2] == c1} /. %;
Show[Map[
  ContourPlot[Evaluate[#], {c1, 0, 5}, {c2, 0, 5},
    PlotLabel → "Curves in the c1,c2 plane for the Freundlich Isotherm",
    FrameLabel → {c1, c2}] &
  , %, {2}
 ]]
```



Make the line2 equation piecewise, for performance

```
curve2simpler[b1_, b2_, c01_, c02_, t0_, c1start_, c1end_] :=
 Piecewise[{{curve2[b1, b2, c01, c02, t0, c1start, c1end], c01 ≠ 0 && c02 ≠ 0},
   {curve2[b1, b2, c01, -0.001, t0, c1start, c1end], c01 ≠ 0 && c02 == 0},
   {{{c1[c2] → 0}}, True}}]
```

Calculate the c_1 and c_2 intercepts of the curves from a point

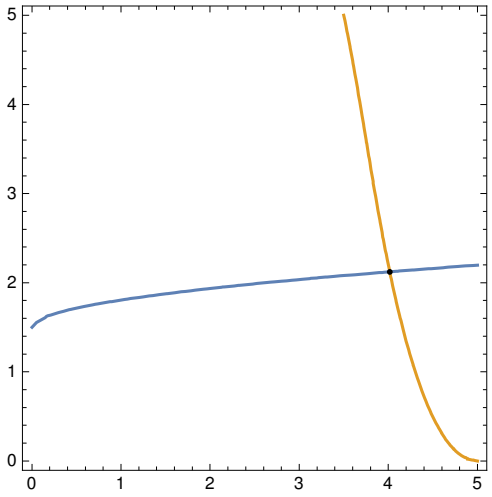
```
curve1intercept[a1_, a2_, b1_, b2_, c01_, c02_, t0_] :=
 Block[{r = {a1 → a1, a2 → a2, b1 → b1, b2 → b2, t → t0, c1 → c1, c2 → c2},
   eq, intercept, curve},
  eq = c2'[c1] == funcSlopes1 /. r;
  curve = First@NDSolve[{eq, c2[c01] == c02}, c2[c1], {c1, c01, 10^-5},
      Method → {"EventLocator", "Event" → c1 - 10^-5, "EventAction" :> (intercept = c2[c1])}
     ] /. {c1 → c1, c2 → c2};
  {{0, intercept}, First@curve}
 ]
curve2intercept[a1_, a2_, b1_, b2_, c01_, c02_, t0_] :=
 Block[{r = {a1 → a1, a2 → a2, b1 → b1, b2 → b2, t → t0, c1 → c1, c2 → c2},
   eq, intercept, curve},
  eq = c1'[c2] == funcSlopes2 /. r;
  curve = First@NDSolve[{eq, c1[c02] == c01}, c1[c2], {c2, c02, 10^-5},
      Method → {"EventLocator", "Event" → c2 - 10^-5, "EventAction" :> (intercept = c1[c2])}
     ] /. {c1 → c1, c2 → c2};
  {{intercept, 0}, First@curve}
 ]
axesIntercepts[a1_, a2_, b1_, b2_, c01_, c02_, t0_] :=
 Transpose@Through[{curve1intercept, curve2intercept}[a1, a2, b1, b2, c01, c02, t0]]
```

Calculate the intersections of curves from two points on the c_1 c_2 axes

```
findIntersect[a1_, a2_, b1_, b2_, c01_, c02_, t0_] :=
 Block[{r = {a1 → a1, a2 → a2, b1 → b1, b2 → b2, t → t0, c1 → c1, c2 → c2},
   eq1, eq2, curve1, curve2, tempfunc, intercept},
  eq1 = c2 '[c1] == funcSlopes1 /. r;
  curve1 = NDSolve[{eq1, c2[10^-9] == c02}, c2[c1], {c1, 10^-9, c01}];

  tempfunc = Function[{c1}, Evaluate[c2[c1] /. curve1]];

  eq2 = c1 '[c2] == funcSlopes2 /. r;
  curve2 = NDSolve[{eq2, c1[10^-9] == c01}, c1[c2], {c2, 10^-9, (tempfunc[c01])}},
    Method → {"EventLocator",
      "Event" → tempfunc[c1[c2]] - c2, "EventAction" :↦ (intercept = {c1[c2], c2})}
   ];

  {intercept, Flatten[{curve1, curve2}]} /. {c1 → c1, c2 → c2}
 ]
```

```
findIntersect[1, 2, .1, .2, 5, 1.5, .5]
ContourPlot[
 Evaluate[{c2 == c2[c1], c1 == c1[c2]} /. %[[2]] ],
 {c1, 0, 5}, {c2, 0, 5},
 Epilog → {Point[%[[1]]]}
]
```

```
{{4.01567, 2.12272},
 {c2[c1] → InterpolatingFunction[ ⊞ Domain: {{1.×10^-9, 5.}}
                                  Output: scalar ][c1],
  c1[c2] → InterpolatingFunction[ ⊞ Domain: {{1.×10^-9, 2.2}}
                                  Output: scalar ][c2]}}
```



Intercets->intercepts gets you back where you started

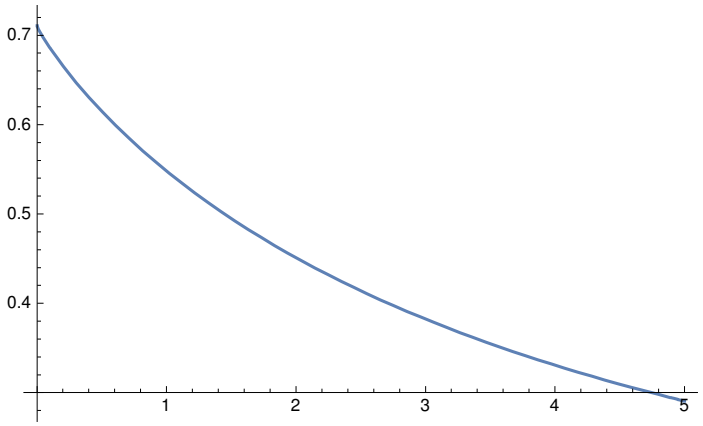
```
findIntersect[1, 2, .1, .2, 5, 1.5, .5]
axesIntercepts[1, 2, .1, .2, %[[1, 1]], %[[1, 2]], .5]
```

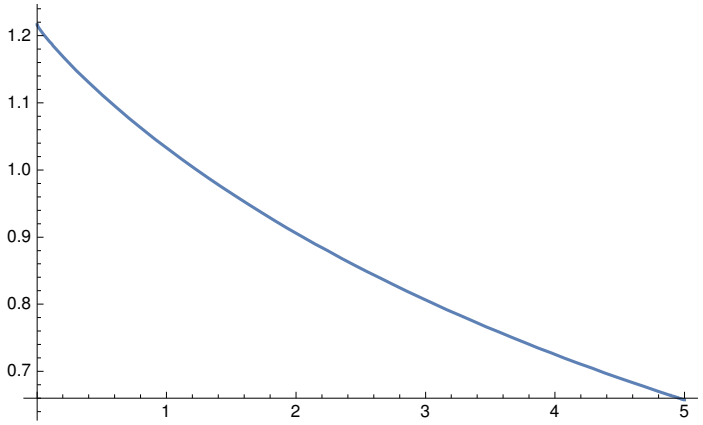
Step 5: Calculate the Eigenvalues Along the Curves

```
eigenvalues = eigensIso[[1]]
```

```
eigenvalueCurve1[transitionFunc_, a1_, a2_, b1_, b2_, t0_] := eigenvalues[[1]] /.
  {c2 → ( c2[c1] /. transitionFunc), a1 → a1, a2 → a2, b1 → b1, b2 → b2, t → t0}
eigenvalueCurve2[transitionFunc_, a1_, a2_, b1_, b2_, t0_] := eigenvalues[[2]] /.
  {c1 → ( c1[c2] /. transitionFunc), a1 → a1, a2 → a2, b1 → b1, b2 → b2, t → t0}
```

```
axesIntercepts[1, 2, .1, .2, 5, 5, 0.9];
eigenvalueCurve1[%[[2, 1]], 1, 2, .1, .2, 0.9];
Plot[Evaluate[%], {c1, 0, 5}]
eigenvalueCurve2[%%%[[2, 2]], 1, 2, .1, .2, 0.9];
Plot[Evaluate[%], {c2, 0, 5}]
```






```
axesIntercepts[1, 2, .1, .2, 5, 5, .9]
```

```
axesIntercepts[1, 2, .1, .2, 5, 5, .9][[2, 2]];
eigenvalueCurve2[%, 1, 2, .1, .2, .9];
ParametricPlot[Evaluate[{{%, c1[c2] /. %%}, {%, c2}}],
 {c2, 0, 5}, AspectRatio → 1, PlotRange → Full]
```

```
eigenvalueCurve1[c2[c1] → 10^-6, 1, 2, .1, .2, .9];
ParametricPlot[Evaluate[{{%, c1}, {%, 0}}],
 {c1, 0, 7.992}, AspectRatio → 1, PlotRange → Full]
```

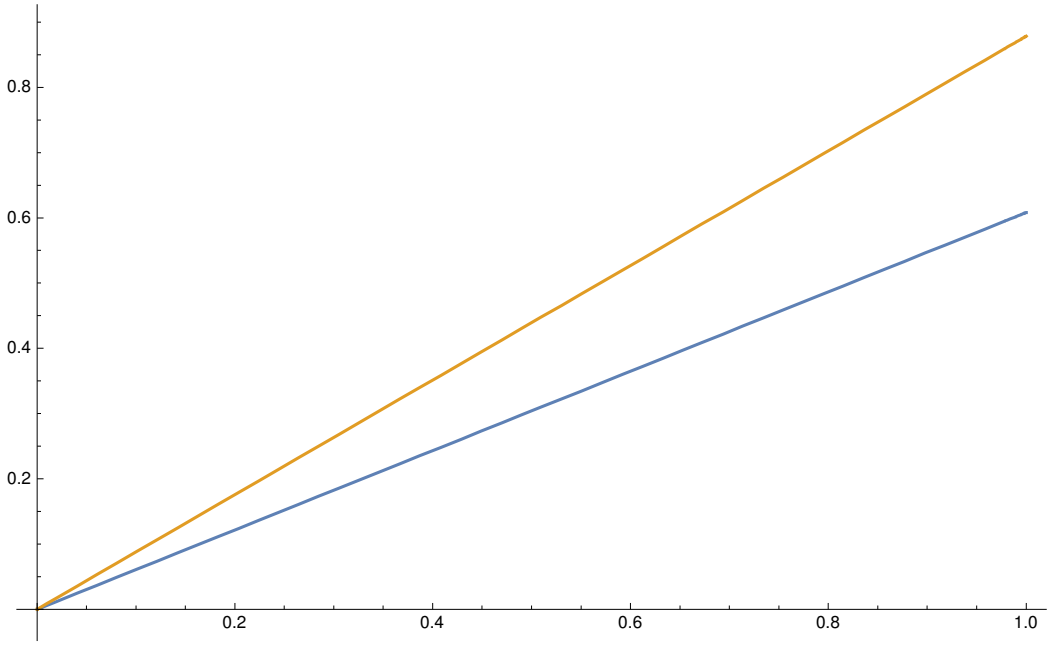
Shockwaves

```
θshocks[init_, fin_, a1_, a2_, b1_, b2_, t0_] :=
 (isotherm /. Thread[{c1, c2} → fin]) - (isotherm /. Thread[{c1, c2} → init])
 ------------------------------------------------------------------------------ /.
                                  fin - init
  {a1 → a1, a2 → a2, b1 → b1, b2 → b2, t → t0}

axesIntercepts[1, 2, .1, .2, 5, 5, .9]
{θshocks[{0, 0}, %[[1, 2]], 1, 2, .1, .2, .9][[1]],
 θshocks[{5, 5}, %[[1, 2]], 1, 2, .1, .2, .9][[2]]}
Plot[Evaluate[% * τ], {τ, 0, 1}]
```

```
{{{0, 2.20529}, {7.92553, 0}},
 {c2[c1] → InterpolatingFunction[ ⊞ Domain: {{0.00001, 5.}}
                                  Output: scalar ][c1],
  c1[c2] → InterpolatingFunction[ ⊞ Domain: {{0.00001, 5.}}
                                  Output: scalar ][c2]}}
```

Power::infy : Infinite expression $\frac{1}{0}$ encountered. >>

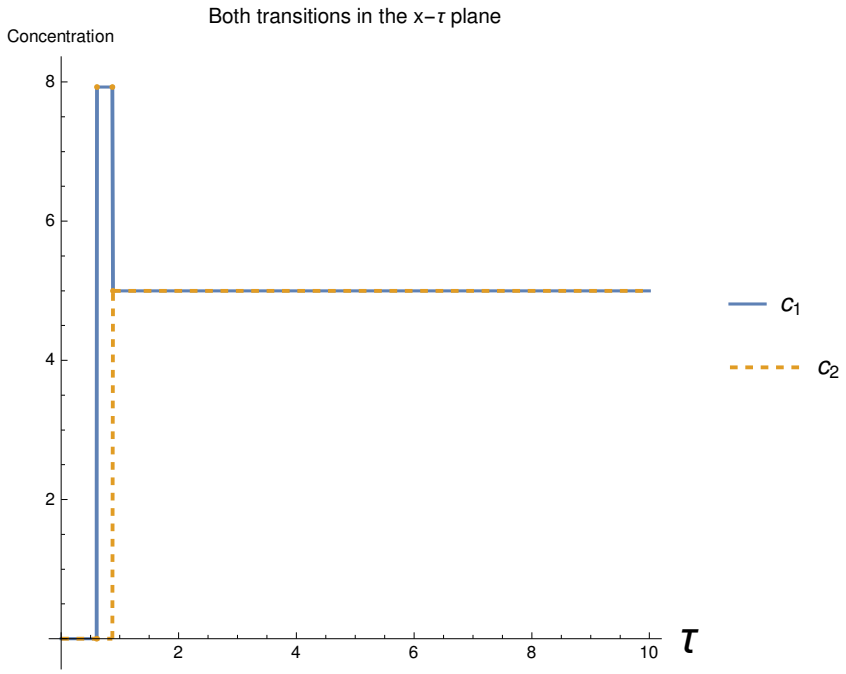
```
{0.608141, 0.878345}
```



Shockwave Plot

```
Block[{inter = axesIntercepts[1, 2, .1, .2, 5, 5, .9], shocks},
 shocks =
  {θshocks[{0, 0}, inter[[1, 2]], 1, 2, .1, .2, .9][[1]],
   θshocks[{5, 5}, inter[[1, 2]], 1, 2, .1, .2, .9][[2]]};
 Show[
  Plot[Evaluate[
    ({ 0                 τ < shocks[[1]]
     { inter[[1, 2, 1]]  shocks[[1]] ≤ τ ≤ shocks[[2]]
     { 5                 True
   ], {τ, 0, 10},
   ExclusionsStyle →
    {Directive[Thick, ColorData[97][1]], Directive[Thick, ColorData[97][2]]},
   PlotLegends → {"c₁"}, AspectRatio → 1,
   AxesLabel → {Style["τ", Large], "Concentration"},
   PlotLabel → "Both transitions in the x-τ plane"
  ],
  Plot[Evaluate[
    ({ 0                 τ < shocks[[1]]
     { inter[[1, 2, 2]]  shocks[[1]] ≤ τ ≤ shocks[[2]]
     { 5                 True
   ], {τ, 0, 10},
   ExclusionsStyle → {Directive[Thick, Dashed, ColorData[97][2]]},
   PlotStyle → {Directive[Thick, Dashed, ColorData[97][2]]},
   PlotLegends → {"c₂"}, AspectRatio → 1,
   AxesLabel → {Style["τ", Large], "Concentration"},
   PlotLabel → "Both transitions in the x-τ plane"
  ]
 ]
]
```

Power::infy : Infinite expression $\frac{1}{0}$ encountered. >>



Step 6: Combine the Differential Equations With Mass Balances

Describe the various constraints

Zone II

```
zone2state = {c1 → 0, c2 → cII2};
zone2thetas = Limit[eigenvalues, c1 → 0,
    Assumptions → {a1 > 0, a2 > 0, b1 > 0, b2 > 0, a2 > a1, c2 > 0, t > 0}] /. c2 → cII2;
zone2constraint[a1_, a2_, b1_, b2_, c2_, t0_, m2_] := zone2thetas[[1]] ∈ Reals &&
   zone2thetas[[2]] ∈ Reals && zone2thetas[[1]] ≤ m2 ≤ zone2thetas[[2]] /.
  {a1 → a1, a2 → a2, b1 → b1, b2 → b2, t → t0, cII2 → c2, m2 → m2}
```

Zone III

```
zone3theta = FullSimplify[θshocks[{cIII1, 0}, {cβ1, cβ2}, a1, a2, b1, b2, t],
   Assumptions → {a1 > 0, a2 > 0, b1 > 0, b2 > 0, a2 > a1, c1 > 0, c2 > 0, t > 0}];

zone3constraint[a1_, a2_, b1_, b2_, c3_, cb1_, cb2_, t0_] :=
 Evaluate[{zone3theta[[1]] ∈ Reals && m3 ≤ zone3theta[[1]]} /.
   {a1 → a1, a2 → a2, b1 → b1, b2 → b2, t → t0, cIII1 → c3, cb1 → cβ1, cb2 → cβ2}]
```

```
zone3constraint[1, 2, 0.1, 0.2, cIII1, cβ1, cβ2, 0.9]
```

$$\left\{\frac{-\frac{cIII_1}{1+0.1\,cIII_1^{0.9}}+\frac{c\beta_1}{1+0.1\,c\beta_1^{0.9}+0.2\,c\beta_2^{0.9}}}{-cIII_1+c\beta_1}\in \text{Reals}\;\&\&\; m_3\le\frac{-\frac{cIII_1}{1+0.1\,cIII_1^{0.9}}+\frac{c\beta_1}{1+0.1\,c\beta_1^{0.9}+0.2\,c\beta_2^{0.9}}}{-cIII_1+c\beta_1}\right\}$$

Feed concentration constraint

The steady state concentrations cannot exceed the concentrations predicted by

$(c\gamma_1 == cF_1,\ c\gamma_2 == cF_2)\ cIII_1 \le c_1$

```
feedConstraint[a1_, a2_, b1_, b2_, cf1_, cf2_, t0_, c2_] :=
 Simplify[c2 ≤ c2] /. (c2 → axesIntercepts[a1, a2 , b1, b2, cf1, cf2, t0][[1, 1, 2]])
```

cγ constraints

```
cγConstraints = {cγ1 ∈ Reals && cγ1 > 0, cγ2 ∈ Reals && cγ2 > 0}
```

$\{c\gamma_1 \in \text{Reals}\;\&\&\; c\gamma_1 > 0,\ c\gamma_2 \in \text{Reals}\;\&\&\; c\gamma_2 > 0\}$

Calculate the various relationships

Calculate the steady state concentrations from the intersection of the eigenvectors

Calculate the steady state concentrations from the mass balances

```
balanceEq1 = Simplify[
   -(m3 - m2) cF2 == m2 cII2 - (isotherm[[2]] /. {c1 → 0, c2 → cII2}), Assumptions → {t > 0}]
balanceEq2 = Simplify[(m3 - m2) cF1 == m3 cIII1 - (isotherm[[1]] /. {c1 → cIII1, c2 → 0}),
   Assumptions → {t > 0}]
```

$$cF_2\,(m_2-m_3) == cII_2\left(-\frac{a_2}{1+b_2\,cII_2^t}+m_2\right)$$

$$cF_1\,(-m_2+m_3) == cIII_1\left(-\frac{a_1}{1+b_1\,cIII_1^t}+m_3\right)$$

```
balSolutions[a1_, a2_, b1_, b2_, cf1_, cf2_, t0_, m2_, m3_] :=
   Flatten[{balanceEq1, balanceEq2} /.
     {m2 → m2, m3 → m3, a1 → a1, a2 → a2, b1 → b1, cF1 → cf1, cF2 → cf2, b2 → b2, t → t0}];

Simplify[balSolutions[1, 2, 0.1, 0.2, 5, 5, 0.9, m2, m3], Assumptions → {m2 < m3}]
```

$$\left\{5\,(m_2-m_3) == cII_2\left(-\frac{2}{1+0.2\,cII_2^{0.9}}+m_2\right),\ 5\,(-m_2+m_3) == cIII_1\left(-\frac{1}{1+0.1\,cIII_1^{0.9}}+m_3\right)\right\}$$

Combine the two steady state concentration relationships

```
test =.
test[a1_, a2_, b1_, b2_, cf1_, cf2_, t0_, m2_?NumericQ, m3_?NumericQ] :=
 And[
  m2 ≤ m3,
  Block[{eq2, c2ss, bal = balSolutions[a1, a2, b1, b2, cf1, cf2, t0, m2, m3]},
   eq2 = FindRoot[bal[[1]] , {cII₂, 1}] // Quiet;
   And[
    Or @@ (cII₂ ∈ Reals /. eq2),
    (Abs[bal[[1, 1]] - bal[[1, 2]]] /. eq2) < 0.001,
    (c2ss = cII₂ /. eq2);
    feedConstraint[a1, a2, b1, b2, cf1, cf2, t0, c2ss],
    zone2constraint[a1, a2, b1, b2, c2ss, t0, m2] // Quiet
   ]
  ]
 ]

sepRegion[a1_, a2_, b1_, b2_, cf1_, cf2_, t0_, m2min_, m2max_, m3min_, m3max_] :=
 Block[{plot1, nums, plot2},
  plot1 = RegionPlot[
      test[a1, a2, b1, b2, cf1, cf2, t0, m2, m3]
      , {m2, m2min, m2max}, {m3, m3min, m3max},
      PlotLabel → "Complete Separation Region",
      FrameLabel → {m₂, m₃},
      PlotPoints → 80,
      MaxRecursion → 2
     ] // Quiet;

  nums =
   DeleteDuplicates@Flatten[Cases[Normal@plot1, Polygon[x_] :→ x, Infinity], 1];
  max = First@MaximalBy[nums, (#[[2]] - #[[1]] &)];

  Show[plot1, Graphics[{
      Arrow[{max * {1.1, 1.3}, max}],
      Text["Optimal point at \n" <> ToString[NumberForm[max, 3]]
       , max * {1.1, 1.3}, {0, -1}
      ]}]]
 ]
```

Calculate the zones 1 and 4 constraints

In zone 1, $m_1 > \theta_2[0, 0]$, so lets calculate that limit!

```
FullSimplify[eigenvalues,
 Assumptions → {a1 > 0, a2 > 0, b2 > 0, c1 > 0, c2 > 0, b1 > 0, a2 > a1, t > 1}]
Limit[%, c2 → 0, Assumptions → {a1 < a2, t > 0}]
θzero = Limit[%, c1 → 0, Assumptions → {a1 < a2, t > 0}]
```

```
zone1constraint = m1 > θzero[[2]]
```

$$m_1 > a_2$$

In zone 4, $m_4 < \theta_1[c_R, 0]$

```
FullSimplify[eigenvalues, Assumptions → {b2 > 0, c2 ≥ 0, c1 ≥ 0, b1 > 0, b2 > b1, t > 1}]
Limit[%, c2 → 0, Assumptions → {a1 < a2, t > 0}]
θraff = %[[1]] /. c1 → cR1
```

$$\text{zone4MassBal} = cR_1 \to \frac{(m_3 - m_2)\, cF_1}{(m_3 - m_4)}$$

$$cR_1 \to \frac{cF_1\ (-m_2 + m_3)}{m_3 - m_4}$$

```
zone4constraint = FullSimplify[m4 ≤ θraff /. zone4MassBal,
  Assumptions → {b2 > 0, c1 ≥ 0, b1 > 0, a2 > a1, t > 0}]
```

```
region2test[a1_, a2_, b1_, b2_, cf1_, cf2_, t0_, m2_, m3_, m1_, m4_] := With[
  {r = {a1 → a1, a2 → a2, b1 → b1, b2 → b2,
     cF1 → cf1, cF2 → cf2, t → t0, m1 → m1, m2 → m2, m3 → m3, m4 → m4}},
  Simplify@And[
    zone1constraint /. r,
    zone4constraint /. r
   ]
 ]
```

```
region2test[1, 2, 0.1, 0.2, 5, 5, 0.9, 0.72, 1, m1, m4]
```

$$m1 > 2 \;\&\&\; m4 \le -\frac{-3. + \sqrt{\left(1 + 0.257199 \left(\frac{1}{1-m4}\right)^{0.9}\right)^2 - 0.284272 \left(\frac{1}{1-m4}\right)^{0.9}}}{2\left(1 + 0.135368 \left(\frac{1}{1-m4}\right)^{0.9}\right)^2}$$

```
separationRegion2[a1_, a2_, b1_, b2_, cf1_,
  cf2_, t0_, m2_, m3_, m1min_, m1max_, m4min_, m4max_] :=
 Block[{eq, plot1, nums},
  eq = region2test[a1, a2, b1, b2, cf1, cf2, t0, m2, m3, m1, m4];
  plot1 = RegionPlot[
      eq
      , {m4, m4min, m4max}, {m1, m1min, m1max},
      PlotLabel → "Complete Separation Region",
      FrameLabel → {m4, m1},
      PlotPoints → 80,
      MaxRecursion → 2
     ] // Quiet;
  nums =
   DeleteDuplicates@Flatten[Cases[Normal@plot1, Polygon[x_] :→ x, Infinity], 1];
  max2 = Chop[#, 10^-6] &@First@MaximalBy[nums, (#[[1]] - #[[2]] &)];

  Show[plot1, Graphics[{
      Arrow[{max2 * {1.4, 0.8} + {1, 1}, max2}],
      Text["Optimal point at \n" <> ToString[NumberForm[max2, 3]]
       , max2 * {1.4, 0.8} + {1, 1}, {0, -1}
      ]}]]
 ]
```

Appendix A.4

Triangle Theory - Toth Isotherm

Notation

```
<< Notation`
```

```
Symbolize[ ϵ* ]
```

```
Symbolize[ ϵb ]
```

```
Symbolize[ ϵp ]
```

```
Symbolize[ v* ]
```

```
Symbolize[ Vs ]
```

```
Symbolize[ Vl ]
```

```
Symbolize[ css ]
```

```
Symbolize[ ca ]
```

```
Symbolize[ cA ]
```

```
Symbolize[ cb ]
```

```
Symbolize[ cB ]
```

```
Symbolize[ nss ]
```

```
Symbolize[ na ]
```

```
Symbolize[ nA ]
```

```
Symbolize[ nb ]
```

```
Symbolize[ nB ]
```

```
Symbolize[ qs ]
```

```
Notation[ n_{a_,b_} ⟺ D[n_{a_}, c_{b_}, NonConstants → {n_1, n_2}] ]
Notation[ n^{ss}_{i_} ⟺ n^{ss}_{i_} ]
Notation[ n^{a}_{i_} ⟺ n^{a}_{i_} ]
Notation[ n^{A}_{i_} ⟺ n^{A}_{i_} ]
Notation[ n^{b}_{i_} ⟺ n^{b}_{i_} ]
Notation[ n^{B}_{i_} ⟺ n^{B}_{i_} ]
Notation[ c^{ss}_{i_} ⟺ c^{ss}_{i_} ]
Notation[ c^{a}_{i_} ⟺ c^{a}_{i_} ]
Notation[ c^{A}_{i_} ⟺ c^{A}_{i_} ]
Notation[ c^{b}_{i_} ⟺ c^{b}_{i_} ]
Notation[ c^{B}_{i_} ⟺ c^{B}_{i_} ]

AddInputAlias["eb" → ϵ_b ]
AddInputAlias["ep" → ϵ_p ]
AddInputAlias["phase" → ϵ_p / (1 - ϵ_p) ]
AddInputAlias["ns" → n^{ss}_i ]
AddInputAlias["cs" → c^{ss}_i ]
```

Step 1: Definition of the Problem

```
f = Table[(c_i + v* n_i) ψ, {i, 1, 2}]
g = Table[((1 - μ ϵ_p) c_i - μ (1 - ϵ_p) n_i), {i, 1, 2}]
isotherm = Simplify@Table[ q_s b_i c_i / (1 + (b_1 c_1 + b_2 c_2)^t)^{1/t}, {i, 1, 2}]
```

```
{ψ (c_1 + v* n_1), ψ (c_2 + v* n_2)}
{(1 - ϵ_p μ) c_1 - (1 - ϵ_p) μ n_1, (1 - ϵ_p μ) c_2 - (1 - ϵ_p) μ n_2}
{q_s b_1 c_1 (1 + (b_1 c_1 + b_2 c_2)^t)^{-1/t}, q_s b_2 c_2 (1 + (b_1 c_1 + b_2 c_2)^t)^{-1/t}}
```

```
jacobian[x_] := D[x, {{c_1, c_2}}, NonConstants → {n_1, n_2}]
```

Step 2: Calculate the Various Eigenvalues and Eigenvectors

Actual Calculations

```
jacobianF = Simplify@jacobian[f]
jacobianG = Simplify@jacobian[g]
jacobianIso = Simplify@jacobian[isotherm]
jacobianH = Simplify[Inverse[jacobianG].jacobianF]

eigensFn = Simplify@Eigensystem[jacobianF]
eigensGn = Simplify@Eigensystem[jacobianG]
eigensIso = Simplify@Eigensystem[jacobianIso]
eigensHn = Simplify@Eigensystem[jacobianH]

nReplacement =
 Thread /@ Thread[Table[n_{i,j}, {i, 1, 2}, {j, 1, 2}] → jacobianIso] // Flatten

eigensF = Simplify[eigensFn /. nReplacement, Assumptions → 1 + b1 c1 + b2 c2 > 0]
eigensG = Simplify[eigensGn /. nReplacement, Assumptions → 1 + b1 c1 + b2 c2 > 0]
eigensH = Simplify[eigensHn /. nReplacement, Assumptions → 1 + b1 c1 + b2 c2 > 0]
```

Interesting Properties

The eigenvectors of the F, G, H, and isotherm matricies are the same.

```
eigensHn[[2]] == eigensFn[[2]] == eigensGn[[2]]
FullSimplify[Thread /@ Thread[eigensH[[2]] == eigensIso[[2]]],
 Assumptions → {b1 > 0, b2 > 0, b2 > b1, c1 > 0, c2 ≥ 0, qs > 0, 1 > t > 0}]
```

```
True
```

```
{{True, True}, {True, True}}
```

Step 3: Analyze the Eigenvectors

The Eigenvectors can be Problematic at the Origin

Calculate Vector Directions at the Origin

no need, when $b_1 < b_2$ the directions are {1,0} and {0,1} just like lang

So as we see, the vectors at the origin point along the axes. Which vector is vertical (and therefore has infinite slope) depends on which component is more strongly adsorbed

```
vectors = eigensIso[[2]]
```

Use Origin Information to Inform Choice Between $\frac{\partial c_1}{\partial c_2}$ or $\frac{\partial c_2}{\partial c_1}$

Assume that $b_1 < b_2$

```
slopeEq1 = FullSimplify[vectors[[1, 2]] / vectors[[1, 1]],
  Assumptions → {b1 > 0, b2 > 0, b2 > b1, c1 > 0, c2 ≥ 0, qs > 0, 1 > t > 0}]
slopeEq2 = FullSimplify[vectors[[2, 1]] / vectors[[2, 2]],
  Assumptions → {b1 > 0, b2 > 0, b2 > b1, c1 > 0, c2 ≥ 0, qs > 0, 1 > t > 0}]
```

Prove that the vectors are nonlinear

```
funcSlopes1 = slopeEq1 /. c2 → c2[c1];
funcSlopes2 = slopeEq2 /. c1 → c1[c2];

secondDerivative1 = Simplify[∂c1 funcSlopes1,
    Assumptions → {b1 > 0, b2 > 0, b2 > b1, c1 ≥ 0, c2 ≥ 0, qs > 0, 1 > t > 0}];
secondDerivative2 = Simplify[∂c2 funcSlopes2,
    Assumptions → {b1 > 0, b2 > 0, b2 > b1, c1 ≥ 0, c2 ≥ 0, qs > 0, 1 > t > 0}];

linetest = {Simplify[secondDerivative1 /. c2'[c1] → funcSlopes1,
    Assumptions → {b1 > 0, b2 > 0, b2 > b1, c1 ≥ 0, c2 ≥ 0, qs > 0, 1 > t > 0}],
  Simplify[secondDerivative2 /. c1'[c2] → funcSlopes2,
    Assumptions → {b1 > 0, b2 > 0, b2 > b1, c1 ≥ 0, c2 ≥ 0, qs > 0, 1 > t > 0}]}
```

Step 4: Calculate the Intersection of the Lines

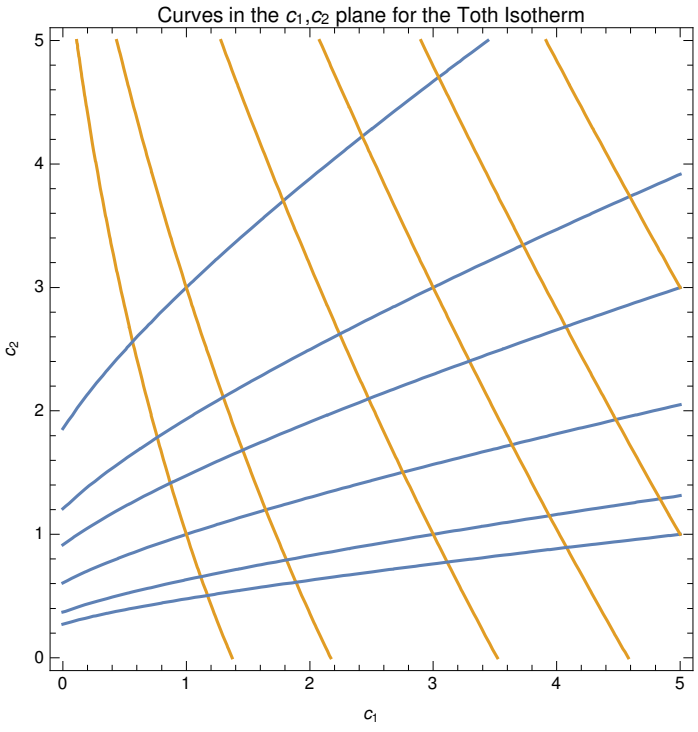
Calculate the curves with NDSolve

```
c2'[c1] == funcSlopes1
c1'[c2] == funcSlopes2

linear[b1_, b2_, c01_, c02_, t0_] :=
 Block[{r = {c2[c1] → c2, c1[c2] → c1, b1 → b1, b2 → b2, t → t0, c1 → c01, c2 → c02}, eq},
  Simplify[linetest //. r]
 ]
```

```
curve1[b1_, b2_, c01_, c02_, t0_, c1start_, c1end_] :=
 Block[{r = {b1 → b1, b2 → b2, t → t0, c1 → c1, c2 → c2}, eq},
  eq = c2'[c1] == funcSlopes1 /. r;
  NDSolve[{eq, c2[c01] == c02}, c2[c1], {c1, c1start, c1end}] /. {c1 → c1, c2 → c2}
 ]
curve2[b1_, b2_, c01_, c02_, t0_, c2start_, c2end_] :=
 Block[{r = {b1 → b1, b2 → b2, t → t0, c1 → c1, c2 → c2}, eq},
  eq = c1'[c2] == funcSlopes2 /. r;
  NDSolve[{eq, c1[c02] == c01}, c1[c2], {c2, c2start, c2end}] /. {c1 → c1, c2 → c2}
 ]
```

View Some Results

```
Table[
  Flatten[{curve1[.1, .2, ca, cb, .5, 0, 5], curve2[.1, .2, ca, cb, .5, -0.001, 5]}],
  {ca, 1, 5, 2}, {cb, 1, 3, 2}];
{c2 == c2[c1], c1[c2] == c1} /. %;
Show[Map[
  ContourPlot[Evaluate[#], {c1, 0, 5}, {c2, 0, 5}, PlotLabel →
     "Curves in the c1,c2 plane for the Toth Isotherm", FrameLabel → {c1, c2}] &
  , %, {2}
 ]]
```



Make the line2 equation piecewise, for performance

```
curve2simpler[b1_, b2_, c01_, c02_, t0_, c1start_, c1end_] :=
 Piecewise[{{curve2[b1, b2, c01, c02, t0, c1start, c1end], c01 ≠ 0 && c02 ≠ 0},
   {curve2[b1, b2, c01, -0.001, t0, c1start, c1end], c01 ≠ 0 && c02 == 0},
   {{c1[c2] → 0}}, True}]
```

Calculate the c_1 and c_2 intercepts of the curves from a point

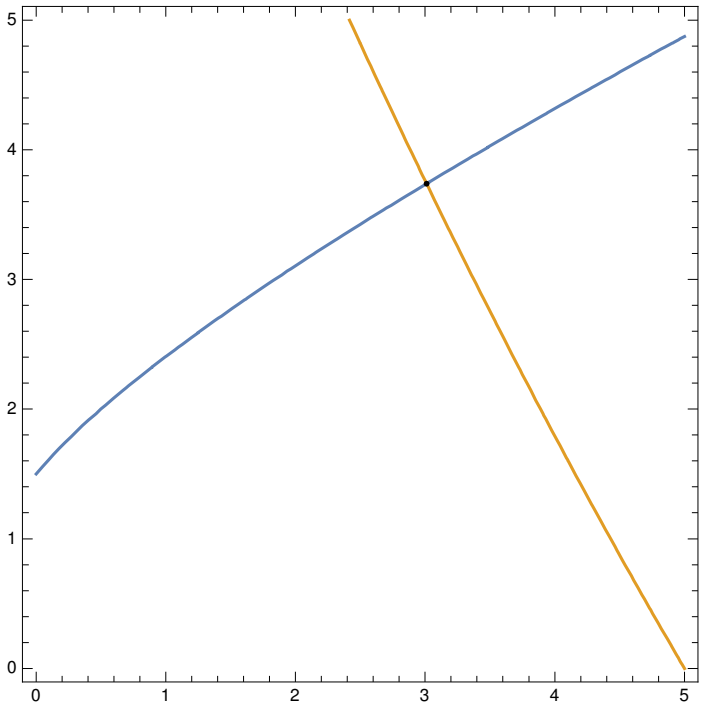
```
curve1intercept[b1_, b2_, c01_, c02_, t0_] :=
 Block[{r = {b1 → b1, b2 → b2, t → t0, c1 → c1, c2 → c2}, eq, intercept, curve},
  eq = c2'[c1] == funcSlopes1 /. r;
  curve = First@NDSolve[{eq, c2[c01] == c02}, c2[c1], {c1, c01, 0},
      Method → {"EventLocator", "Event" → c1, "EventAction" :> (intercept = c2[c1])}
     ] /. {c1 → c1, c2 → c2};
  {{0, intercept}, First@curve}
 ]
curve2intercept[b1_, b2_, c01_, c02_, t0_] :=
 Block[{r = {b1 → b1, b2 → b2, t → t0, c1 → c1, c2 → c2}, eq, intercept, curve},
  eq = c1'[c2] == funcSlopes2 /. r;
  curve = First@NDSolve[{eq, c1[c02] == c01}, c1[c2], {c2, c02, -0.00001},
      Method → {"EventLocator", "Event" → c2, "EventAction" :> (intercept = c1[c2])}
     ] /. {c1 → c1, c2 → c2};
  {{intercept, 0}, First@curve}
 ]
axesIntercepts[b1_, b2_, c01_, c02_, t0_] :=
 Transpose@Through[{curve1intercept, curve2intercept}[b1, b2, c01, c02, t0]]
```

Calculate the intersections of curves from two points on the c_1 c_2 axes

```
findIntersect[b1_, b2_, c01_, c02_, t0_] :=
 Block[{r = {b1 → b1, b2 → b2, t → t0, c1 → c1, c2 → c2},
   eq1, eq2, curve1, curve2, tempfunc, intercept},
  eq1 = c2'[c1] == funcSlopes1 /. r;
  curve1 = NDSolve[{eq1, c2[0] == c02}, c2[c1], {c1, 0, c01}];

  tempfunc = Function[{c1}, Evaluate[c2[c1] /. curve1]];

  eq2 = c1'[c2] == funcSlopes2 /. r;
  curve2 =
   NDSolve[{eq2, c1[-0.00001] == c01}, c1[c2], {c2, -0.00001, (tempfunc[c01])},
    Method → {"EventLocator", "Event" → tempfunc[c1[c2]] - c2,
      "EventAction" :↦ (intercept = {c1[c2], c2})}
   ];

  {intercept, Flatten[{curve1, curve2}]} /. {c1 → c1, c2 → c2}
 ]
```

```
findIntersect[.1, .2, 5, 1.5, .5];
ContourPlot[
 Evaluate[{c2 == c2[c1], c1 == c1[c2]} /. %[[2]] ],
 {c1, 0, 5}, {c2, 0, 5},
 Epilog → {Point[%[[1]]]}
]
```

```
{{3.01139, 3.7387}, {c2[c1] → InterpolatingFunction[ ⊞ Domain: {{0., 5.}}
                                                     Output: scalar ][c1],
  c1[c2] → InterpolatingFunction[ ⊞ Domain: {{−0.00001, 4.87}}
                                  Output: scalar ][c2]}}
```



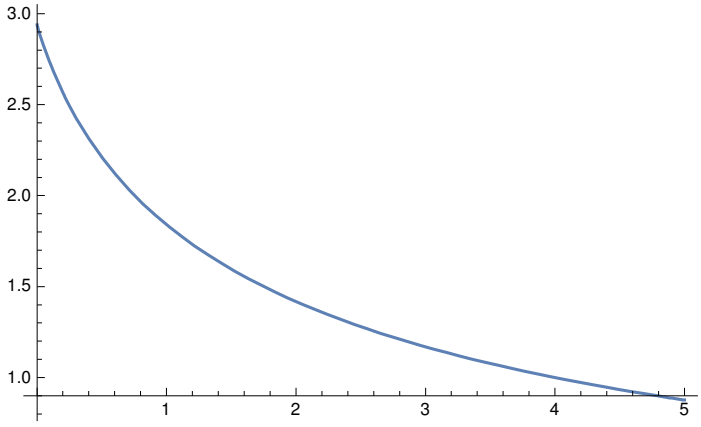
Check that Intercets->intercepts gets you back where you started

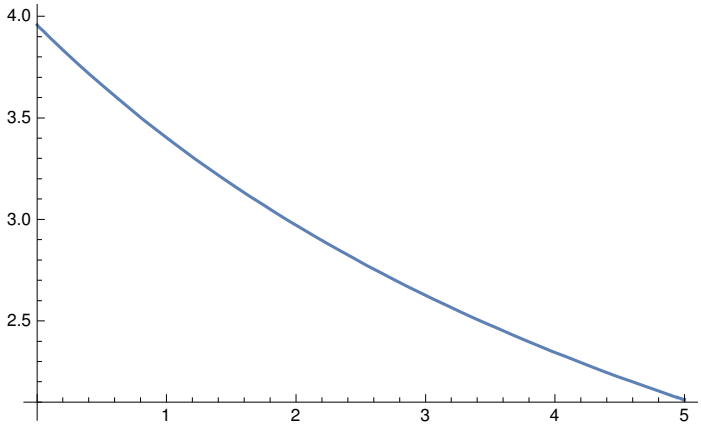
```
findIntersect[.1, .2, 5, 1.5, .5]
axesIntercepts[.1, .2, %[[1, 1]], %[[1, 2]], .5]
```

Step 5: Calculate the Eigenvalues Along the Curves

```
eigenvalues = eigensIso[[1]]
```

```
eigenvalueCurve1[transitionFunc_, b1_, b2_, t0_, qs_] :=
 eigenvalues[[1]] /. {c2 → ( c2[c1] /. transitionFunc), b1 → b1, b2 → b2, t → t0, qs → qs}
eigenvalueCurve2[transitionFunc_, b1_, b2_, t0_, qs_] :=
 eigenvalues[[2]] /. {c1 → ( c1[c2] /. transitionFunc), b1 → b1, b2 → b2, t → t0, qs → qs}

axesIntercepts[.1, .2, 5, 5, .5];
eigenvalueCurve1[%[[2, 1]], .1, .2, .5, 71];
Plot[Evaluate[%], {c1, 0, 5}]
eigenvalueCurve2[%%%[[2, 2]], .1, .2, .5, 71];
Plot[Evaluate[%], {c2, 0, 5}]
```





```
axesIntercepts[.1, .2, 5, 5, .5]

axesIntercepts[.1, .2, 5, 5, .5][[2, 2]];
eigenvalueCurve2[%, .1, .2, .5, 71];
ParametricPlot[Evaluate[{{%, c1[c2] /. %%}, {%, c2}}],
 {c2, 0, 5}, AspectRatio → 1, PlotRange → Full]

eigenvalueCurve1[c2[c1] → 0, .1, .2, .5, 71];
ParametricPlot[Evaluate[{{%, c1}, {%, 0}}],
 {c1, 0, 7.992}, AspectRatio → 1, PlotRange → Full]
```

Shockwaves

$$\theta\text{shocks}[\text{init\_}, \text{fin\_}, \text{b1\_}, \text{b2\_}, \text{t0\_}, \text{qs\_}] := \frac{1}{\text{fin} - \text{init}}$$

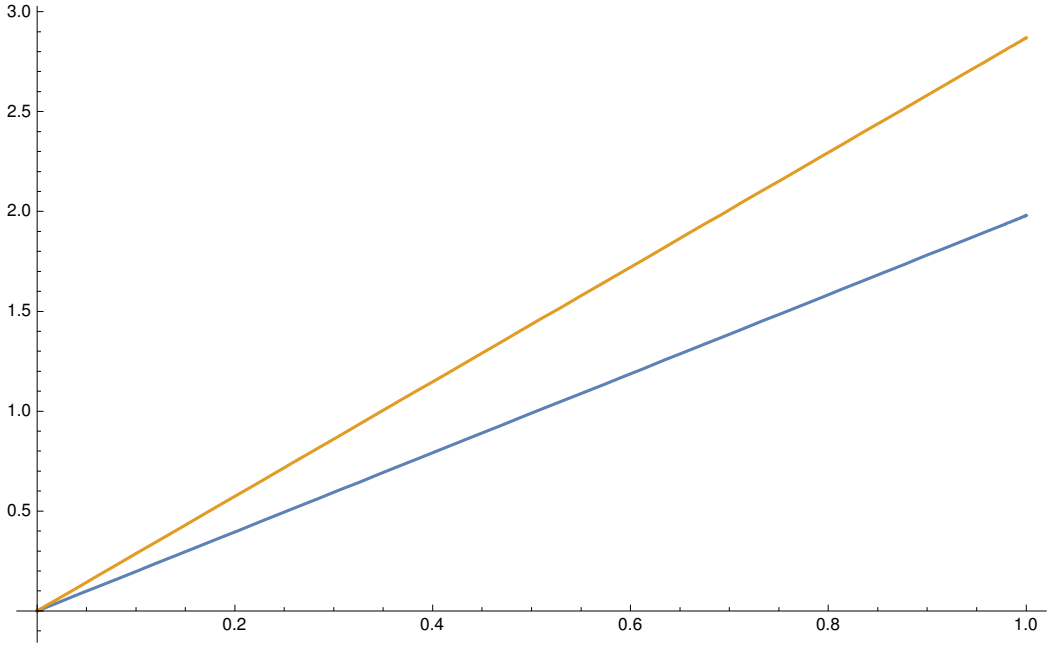
```
((isotherm /. Thread[{c1, c2} → fin]) - (isotherm /. Thread[{c1, c2} → init])) /.
 {b1 → b1, b2 → b2, t → t0, qs → qs}
```

```
axesIntercepts[.1, .2, 5, 5, .5];
{θshocks[{0, 0}, %[[1, 2]], .1, .2, .5, 71][[1]],
 θshocks[{5, 5}, %[[1, 2]], .1, .2, .5, 71][[2]]}
Plot[Evaluate[% * τ], {τ, 0, 1}]
```

Power::infy : Infinite expression $\frac{1}{0}$ encountered. ≫

Infinity::indet : Indeterminate expression $0\,(1+0^t)^{-1/t}\,q_s$ ComplexInfinity b_2 encountered. ≫

```
{1.97919, 2.86898}
```

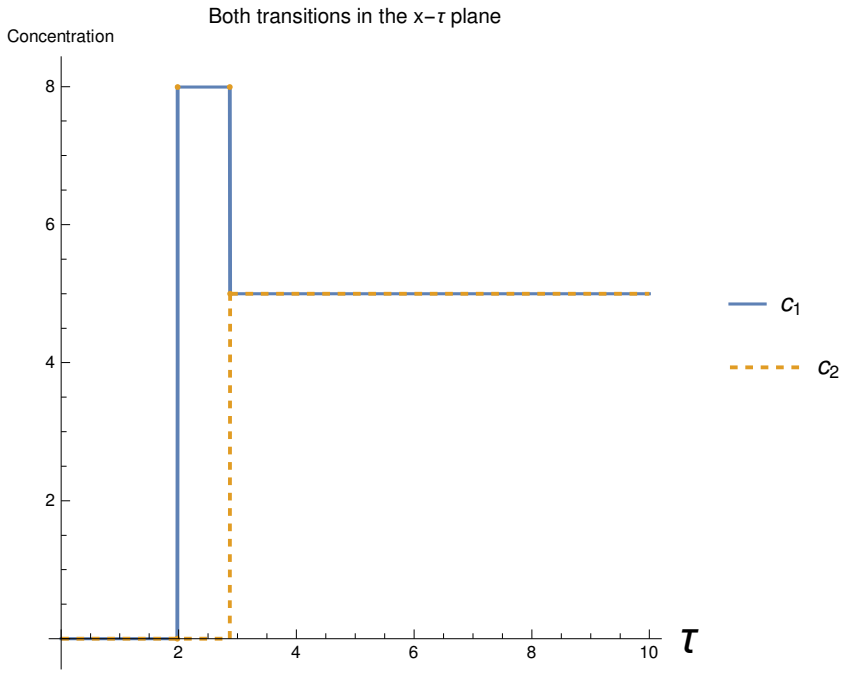


Shockwave Plot

```
Block[{inter = axesIntercepts[.1, .2, 5, 5, .5], shocks},
 shocks =
  {θshocks[{0, 0}, inter[[1, 2]], .1, .2, .5, 71][[1]],
   θshocks[{5, 5}, inter[[1, 2]], .1, .2, .5, 71][[2]]};
 Show[
  Plot[Evaluate[
    ({ 0                  τ < shocks[[1]]
     { inter[[1, 2, 1]]   shocks[[1]] ≤ τ ≤ shocks[[2]]
     { 5                  True
   ], {τ, 0, 10},
   ExclusionsStyle →
    {Directive[Thick, ColorData[97][1]], Directive[Thick, ColorData[97][2]]},
   PlotLegends → {"c₁"}, AspectRatio → 1,
   AxesLabel → {Style["τ", Large], "Concentration"},
   PlotLabel → "Both transitions in the x-τ plane"
  ],
  Plot[Evaluate[
    ({ 0                  τ < shocks[[1]]
     { inter[[1, 2, 2]]   shocks[[1]] ≤ τ ≤ shocks[[2]]
     { 5                  True
   ], {τ, 0, 10},
   ExclusionsStyle → {Directive[Thick, Dashed, ColorData[97][2]]},
   PlotStyle → {Directive[Thick, Dashed, ColorData[97][2]]},
   PlotLegends → {"c₂"}, AspectRatio → 1,
   AxesLabel → {Style["τ", Large], "Concentration"},
   PlotLabel → "Both transitions in the x-τ plane"
  ]
 ]
]
```

Power::infy : Infinite expression $\frac{1}{0}$ encountered. ≫

Infinity::indet : Indeterminate expression $0\,(1+0^t)^{-1/t}\,q_s$ ComplexInfinity b_2 encountered. ≫



Step 6: Combine the Differential Equations With Mass Balances

Describe the various constraints

Zone II

```
zone2state = {c1 → 0, c2 → cII2};
zone2thetas = FullSimplify[eigenvalues /. zone2state,
    Assumptions → {b1 > 0, b2 > 0, b2 > b1, c1 ≥ 0, c2 ≥ 0, qs > 0, 1 > t > 0}];
zone2constraint[b1_, b2_, c2_, t0_, qs_, m2_] := zone2thetas[[1]] ∈ Reals &&
    zone2thetas[[2]] ∈ Reals && zone2thetas[[1]] ≤ m2 ≤ zone2thetas[[2]] /.
  {b1 → b1, b2 → b2, t → t0, cII2 → c2, qs → qs, m2 → m2}
```

Zone III

```
θshocks[init_, fin_, b1_, b2_, t0_, qs_]
zone3theta = FullSimplify[θshocks[{cIII1, 0}, {cβ1, cβ2}, b1, b2, t, qs],
    Assumptions → {b1 > 0, b2 > 0, b2 > b1, c1 ≥ 0, c2 ≥ 0, qs > 0, 1 > t > 0}];
```

```
zone3constraint[b1_, b2_, c3_, cb1_, cb2_, t0_, qs_] :=
 Evaluate[{zone3theta[[1]] ∈ Reals && m3 ≤ zone3theta[[1]]} /.
   {b1 → b1, b2 → b2, t → t0, cIII1 → c3, qs → qs, cb1 → cβ1, cb2 → cβ2}]
```

Feed concentration constraint

The steady state concentrations cannot exceed the concentrations predicted by

$(c\gamma_1 == cF_1,\ c\gamma_2 == cF_2)\ cIII_1 \le c_1$

```
feedConstraint[b1_, b2_, cf1_, cf2_, t0_, c2_] :=
 Simplify[c2 ≤ c2] /. (c2 → axesIntercepts[b1, b2, cf1, cf2, t0][[1, 1, 2]])
```

cγ constraints

```
cγConstraints = {cγ1 ∈ Reals && cγ1 > 0, cγ2 ∈ Reals && cγ2 > 0}
```

$\{c\gamma_1 \in \text{Reals} \&\& c\gamma_1 > 0,\ c\gamma_2 \in \text{Reals} \&\& c\gamma_2 > 0\}$

Calculate the various relationships

Calculate the steady state concentrations from the mass balances

```
balanceEq1 = -(m3 - m2) cF2 == m2 cII2 - (isotherm[[2]] /. {c1 → 0, c2 → cII2})
balanceEq2 = (m3 - m2) cF1 == m3 cIII1 - (isotherm[[1]] /. {c1 → cIII1, c2 → 0})
```

$cF_2\ (m_2 - m_3) == -q_s\ b_2\ cII_2\ (1 + (b_2\ cII_2)^t)^{-1/t} + cII_2\ m_2$

$cF_1\ (-m_2 + m_3) == -q_s\ b_1\ cIII_1\ (1 + (b_1\ cIII_1)^t)^{-1/t} + cIII_1\ m_3$

```
balSolutions[b1_, b2_, cf1_, cf2_, t0_, qs_, m2_, m3_] :=
  Flatten[{balanceEq1, balanceEq2} /.
    {m2 → m2, m3 → m3, b1 → b1, cF1 → cf1, cF2 → cf2, b2 → b2, t → t0, qs → qs}];
```

Combine the two steady state concentration relationships

```
test =.
test[b1_, b2_, cf1_, cf2_, t0_, qs_, m2_?NumericQ, m3_?NumericQ] :=
 And[
  m2 ≤ m3,
  Block[{eq2, c2ss, bal = balSolutions[b1, b2, cf1, cf2, t0, qs, m2, m3]},
   eq2 = FindRoot[bal[[1]] , {cII₂, 1}] // Quiet;
   And[
    Or @@ (cII₂ ∈ Reals /. eq2),
    (Abs[bal[[1, 1]] - bal[[1, 2]]] /. eq2) < 0.001,
    (c2ss = cII₂ /. eq2);
    feedConstraint[b1, b2, cf1, cf2, t0, c2ss],
    zone2constraint[b1, b2, c2ss, t0, qs, m2] // Quiet
   ]
  ]
 ]

sepRegion[b1_, b2_, cf1_, cf2_, t0_, qs_, m2min_, m2max_, m3min_, m3max_] :=
 Block[{plot1, nums, plot2},
  plot1 = RegionPlot[
      test[b1, b2, cf1, cf2, t0, qs, m2, m3]
      , {m2, m2min, m2max}, {m3, m3min, m3max},
      PlotLabel → "Complete Separation Region",
      FrameLabel → {m₂, m₃},
      PlotPoints → 80,
      MaxRecursion → 2
     ] // Quiet;

  nums =
   DeleteDuplicates@Flatten[Cases[Normal@plot1, Polygon[x_] :→ x, Infinity], 1];
  max = First@MaximalBy[nums, (#[[2]] - #[[1]] &)];

  Show[plot1, Graphics[{
      Arrow[{max * {1.1, 1.3}, max}],
      Text["Optimal point at \n" <> ToString[NumberForm[max, 3]]
       , max * {1.1, 1.3}, {0, -1}
      ]}]]
 ]
```

Calculate the zones 1 and 4 constraints

In zone 1, $m_1 > \theta_2[0, 0]$, so lets calculate that limit!

```
FullSimplify[eigenvalues, Assumptions →
  {b2 - b1 (1 + b2 c2) > 0, b2 + b1 (-1 + b2 c1) > 0, b2 > 0, c2 == 0, c1 ≥ 0, b1 > 0, b2 > b1}]
θzero = Limit[%, c1 → 0, Assumptions → {0 < t < 1, b1 < b2}]
```

$$\left\{\frac{1}{2} q_s \left(1 + (b_1 c_1)^t\right)^{-\frac{1+t}{t}} \left(b_1 + b_2 \left(1 + (b_1 c_1)^t\right) - \sqrt{\left(b_1 - b_2 \left(1 + (b_1 c_1)^t\right)\right)^2}\right),\right.$$
$$\left.\frac{1}{2} q_s \left(1 + (b_1 c_1)^t\right)^{-\frac{1+t}{t}} \left(b_1 + b_2 \left(1 + (b_1 c_1)^t\right) + \sqrt{\left(b_1 - b_2 \left(1 + (b_1 c_1)^t\right)\right)^2}\right)\right\}$$

$$\{q_s b_1, q_s b_2\}$$

```
zone1constraint = m1 ≥ θzero[[2]]
```

$$m_1 \geq q_s b_2$$

In zone 4, $m_4 < \theta_1[c_R, 0]$

```
FullSimplify[eigenvalues,
 Assumptions → {b2 > 0, c2 == 0, c1 ≥ 0, b1 > 0, b2 > b1, 1 > t > 0}]
θraff = %[[1]] /. c1 → cR1
FullSimplify[1 / θraff]
zone4MassBal = cR1 → (m3 - m2) cF1 / (m3 - m4)
```

$$cR_1 \to \frac{cF_1 \left(-m_2 + m_3\right)}{m_3 - m_4}$$

```
zone4constraint = FullSimplify[m4 ≤ θraff /. zone4MassBal,
  Assumptions → {b2 > 0, c1 ≥ 0, b1 > 0, b2 > b1, 1 > t > 0}]
region2test[b1_, b2_, cf1_, cf2_, t0_, qs_, m2_, m3_, m1_, m4_] := With[
  {r = {b1 → b1, b2 → b2, cF1 → cf1,
     cF2 → cf2, t → t0, qs → qs, m1 → m1, m2 → m2, m3 → m3, m4 → m4}},
  Simplify@And[
    zone1constraint /. r,
    zone4constraint /. r
   ]
 ]
```

```
separationRegion2[b1_, b2_, cf1_, cf2_,
  t0_, qs_, m2_, m3_, m1min_, m1max_, m4min_, m4max_] :=
 Block[{eq, plot1, nums},
  eq = region2test[b1, b2, cf1, cf2, t0, qs, m2, m3, m1, m4];
  plot1 = RegionPlot[
      eq
      , {m4, m4min, m4max}, {m1, m1min, m1max},
      PlotLabel → "Complete Separation Region",
      FrameLabel → {m4, m1},
      PlotPoints → 80,
      MaxRecursion → 2
     ] // Quiet;
  nums =
   DeleteDuplicates@Flatten[Cases[Normal@plot1, Polygon[x_] :→ x, Infinity], 1];
  max2 = First@MaximalBy[nums, (#[[1]] - #[[2]] &)];

  Show[plot1, Graphics[{
      Arrow[{max2 * {1.3, 0.8}, max2}],
      Text["Optimal point at \n" <> ToString[NumberForm[max2, 3]]
       , max2 * {1.3, 0.8}, {0, 1}
      ]}]]
 ]
```

Appendix A.5

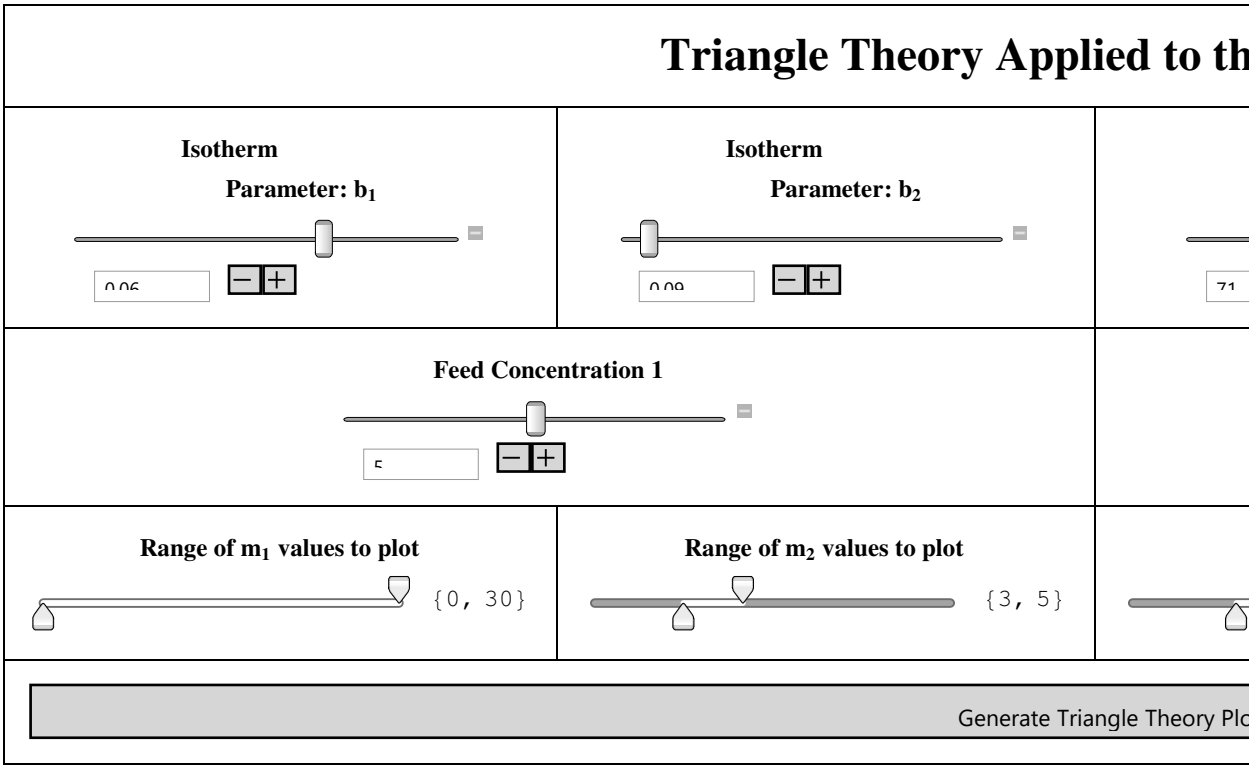
GUI Code

Toth

In[1]:=
```
DynamicModule[
 {b1 = .06, b2 = .09, cf1 = 5, cf2 = 5, update = 0, qs = 71, t0 = 0.77, m2min = 3,
  m3min = 3.5, m2max = 5, m3max = 5.5, m1min = 0, m4min = 0, m1max = 30, m4max = 6},
 Grid[
  {
   {Text[Style["Triangle Theory Applied to the Toth Isotherm", Bold, Large]],
    SpanFromLeft},
   Inner[Labeled[#1, #2, Top] &,
    {
     Manipulator[Dynamic[b1], {0, Dynamic[b2]}, Appearance → "Open",
      AppearanceElements → {"InputField", "StepLeftButton", "StepRightButton"},
      ContinuousAction → True],
     Manipulator[Dynamic[b2], {Dynamic[b1], 1}, Appearance → "Open",
      AppearanceElements → {"InputField", "StepLeftButton", "StepRightButton"},
      ContinuousAction → True],
     Manipulator[Dynamic[qs], {0, 100}, Appearance → "Open",
      AppearanceElements → {"InputField", "StepLeftButton", "StepRightButton"},
      ContinuousAction → True],
     Manipulator[Dynamic[t0], {0, 1}, Appearance → "Open",
      AppearanceElements → {"InputField", "StepLeftButton", "StepRightButton"},
      ContinuousAction → True]
    },
    Text[Style[#, Bold]] & /@ {"Isotherm Parameter: b₁",
      "Isotherm Parameter: b₂", "Saturation Loading: qₛ", "Toth Exponent: t"},
    List
   ],
   Riffle[#, SpanFromLeft, {2, -1, 2}] &@Inner[Labeled[#1, #2, Top] &,
     {
      Manipulator[Dynamic[cf1], {0, 10}, Appearance → "Open",
       AppearanceElements → {"InputField", "StepLeftButton", "StepRightButton"},
       ContinuousAction → True],
      Manipulator[Dynamic[cf2], {0, 10}, Appearance → "Open",
       AppearanceElements → {"InputField", "StepLeftButton", "StepRightButton"},
       ContinuousAction → True]
```

```
      },
      Text[Style[#, Bold]] & /@ {"Feed Concentration 1", "Feed Concentration 2"},
      List
     ],
    Inner[Labeled[#1, #2, Top] &,
     {
      Grid[{{IntervalSlider[{Dynamic[m1min], Dynamic[m1max]}, {0, 30, 1}],
         {Dynamic[m1min], Dynamic[m1max]}}}],
      Grid[{{IntervalSlider[{Dynamic[m2min], Dynamic[m2max]}, {0, 12, 0.1}],
         {Dynamic[m2min], Dynamic[m2max]}}}],
      Grid[{{IntervalSlider[{Dynamic[m3min], Dynamic[m3max]}, {0, 12, 0.1}],
         {Dynamic[m3min], Dynamic[m3max]}}}],
      Grid[{{IntervalSlider[{Dynamic[m4min], Dynamic[m4max]}, {0, 12, 0.1}],
         {Dynamic[m4min], Dynamic[m4max]}}}]
     },
     Text[Style[#, Bold]] & /@
      {"Range of m₁ values to plot", "Range of m₂ values to plot",
       "Range of m₃ values to plot", "Range of m₄ values to plot"},
     List
    ],
    {
     Button["Generate Triangle Theory Plots",
      Print[
       Grid[{{
          sepRegion[b1, b2, cf1, cf2, t0, qs, m2min, m2max, m3min, m3max],
          SpanFromLeft,
          separationRegion2[b1, b2, cf1, cf2,
           t0, qs, max[[1]], max[[2]], m1min, m1max, m4min, m4max],
          SpanFromLeft
         },
         {m₁ == max2[[2]], m₂ == max[[1]], m₃ == max[[2]], m₄ == max2[[1]]}}, Frame → All]
      ], Method → "Queued"
     ]

     , SpanFromLeft}
   }, Frame → All, Spacings → {2, 2}
  ]
 ]
```

Out[1]=



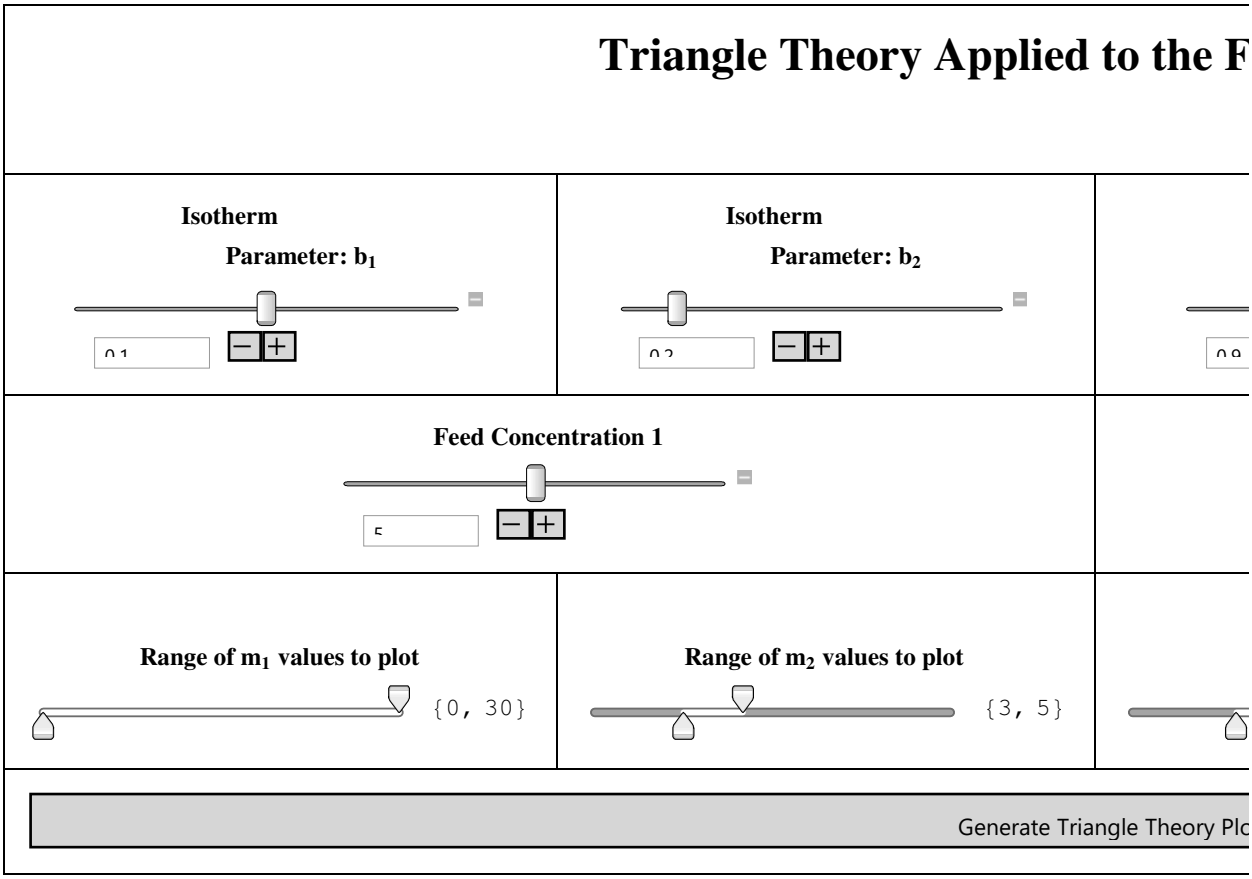
Freundlich

In[2]:=
```
DynamicModule[{h1 = .1, h2 = .2, cf1 = 5, cf2 = 5, update = 0, f0 = 0.9, m2min = 3,
  m3min = 3.5, m2max = 5, m3max = 5.5, m1min = 0, m4min = 0, m1max = 30, m4max = 6},
 Grid[
  {
   {Text[Style["Triangle Theory Applied to the Freundlich Isotherm", Bold, Large]],
    SpanFromLeft},
   Inner[Labeled[#1, #2, Top] &,
    {
     Manipulator[Dynamic[h1], {0, Dynamic[h2]}, Appearance → "Open",
      AppearanceElements → {"InputField", "StepLeftButton", "StepRightButton"},
      ContinuousAction → True],
     Manipulator[Dynamic[h2], {Dynamic[h1], 1}, Appearance → "Open",
      AppearanceElements → {"InputField", "StepLeftButton", "StepRightButton"},
      ContinuousAction → True],
     Manipulator[Dynamic[f0], {0, 1}, Appearance → "Open",
      AppearanceElements → {"InputField", "StepLeftButton", "StepRightButton"},
      ContinuousAction → True]
    },
    Text[Style[#, Bold]] & /@ {"Isotherm Parameter: b1",
```

```
    "Isotherm Parameter: b2", "Freundlich Exponent: t"},
   List
  ],
  Riffle[#, SpanFromLeft, {2, -1, 2}] &@Inner[Labeled[#1, #2, Top] &,
    {
     Manipulator[Dynamic[cf1], {0, 10}, Appearance → "Open",
      AppearanceElements → {"InputField", "StepLeftButton", "StepRightButton"},
      ContinuousAction → True],
     Manipulator[Dynamic[cf2], {0, 10}, Appearance → "Open",
      AppearanceElements → {"InputField", "StepLeftButton", "StepRightButton"},
      ContinuousAction → True]
    },
    Text[Style[#, Bold]] & /@ {"Feed Concentration 1", "Feed Concentration 2"},
    List
   ],
  Inner[Labeled[#1, #2, Top] &,
   {
    Grid[{{IntervalSlider[{Dynamic[m1min], Dynamic[m1max]}, {0, 30, 1}],
       {Dynamic[m1min], Dynamic[m1max]}}}],
    Grid[{{IntervalSlider[{Dynamic[m2min], Dynamic[m2max]}, {0, 12, 0.1}],
       {Dynamic[m2min], Dynamic[m2max]}}}],
    Grid[{{IntervalSlider[{Dynamic[m3min], Dynamic[m3max]}, {0, 12, 0.1}],
       {Dynamic[m3min], Dynamic[m3max]}}}],
    Grid[{{IntervalSlider[{Dynamic[m4min], Dynamic[m4max]}, {0, 12, 0.1}],
       {Dynamic[m4min], Dynamic[m4max]}}}]
   },
   Text[Style[#, Bold]] & /@
    {"Range of m1 values to plot", "Range of m2 values to plot",
     "Range of m3 values to plot", "Range of m4 values to plot"},
   List
  ],
  {
   Button["Generate Triangle Theory Plots",
    Print[
     Grid[{{
        sepRegion[b1, h2, cf1, cf2, t0, qs, m2min, m2max, m3min, m3max],
        SpanFromLeft,
        separationRegion2[b1, h2, cf1, cf2,
         t0, qs, max[[1]], max[[2]], m1min, m1max, m4min, m4max],
        SpanFromLeft
       },
       {m1 == max2[[2]], m2 == max[[1]], m3 == max[[2]], m4 == max2[[1]]}}, Frame → All]
    ], Method → "Queued"
```

```
    ]

    , SpanFromLeft}
  }, Frame → All, Spacings → {2, 2}
 ]
]
```

Out[2]=



Sipps

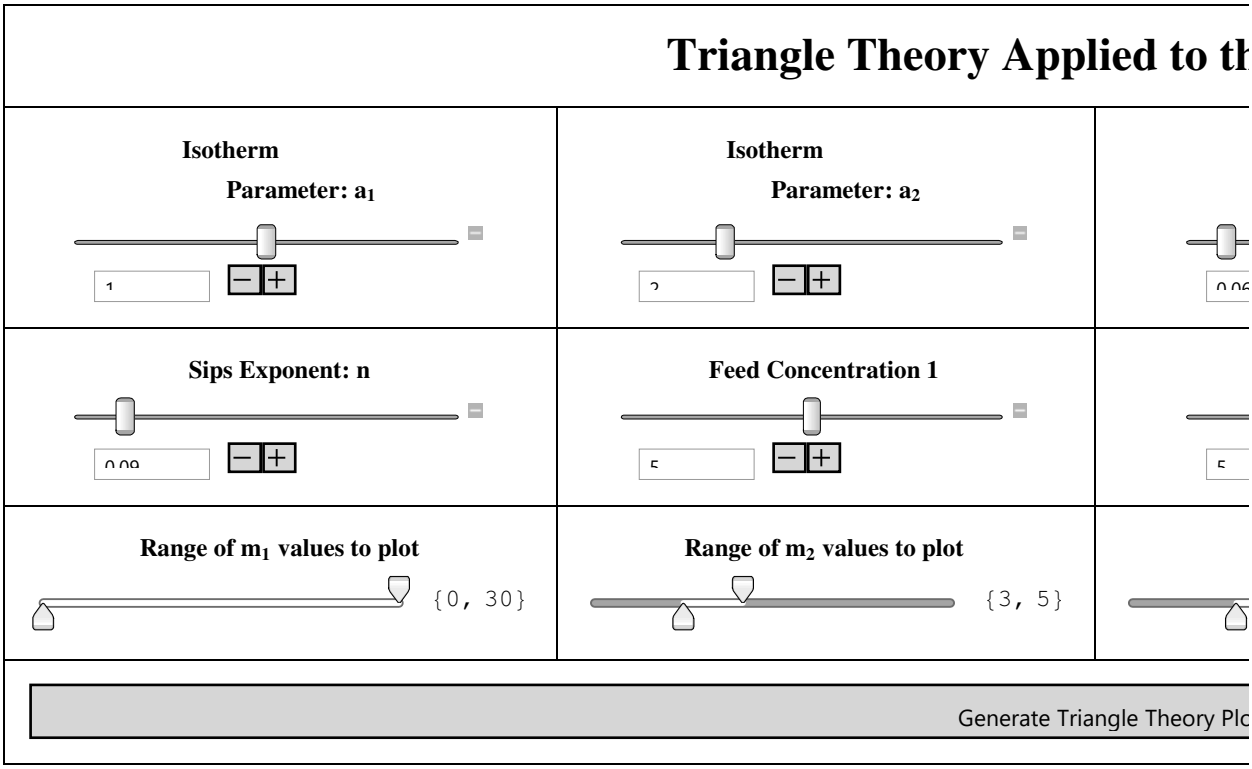
In[3]:=

```
DynamicModule[
 {a1 = 1, a2 = 2, b1 = .06, b2 = .09, cf1 = 5, cf2 = 5, update = 0, t0 = 0.9, m2min = 3,
  m3min = 3.5, m2max = 5, m3max = 5.5, m1min = 0, m4min = 0, m1max = 30, m4max = 6},
 Grid[
  {
   {Text[Style["Triangle Theory Applied to the Sips Isotherm", Bold, Large]],
    SpanFromLeft},
   Inner[Labeled[#1, #2, Top] &,
    {
     Manipulator[Dynamic[a1], {0, Dynamic[a2]}, Appearance → "Open",
```

```
    AppearanceElements → {"InputField", "StepLeftButton", "StepRightButton"},
    ContinuousAction → True],
   Manipulator[Dynamic[a2], {Dynamic[a1], 5}, Appearance → "Open",
    AppearanceElements → {"InputField", "StepLeftButton", "StepRightButton"},
    ContinuousAction → True],
   Manipulator[Dynamic[b1], {0, 1}, Appearance → "Open",
    AppearanceElements → {"InputField", "StepLeftButton", "StepRightButton"},
    ContinuousAction → True],
   Manipulator[Dynamic[b2], {0, 1}, Appearance → "Open",
    AppearanceElements → {"InputField", "StepLeftButton", "StepRightButton"},
    ContinuousAction → True]
  },
  Text[Style[#, Bold]] & /@ {"Isotherm Parameter: a1", "Isotherm Parameter: a2",
    "Isotherm Parameter: b1", "Isotherm Parameter: b2"},
  List
 ],
 Inner[Labeled[#1, #2, Top] &,
  {
   Manipulator[Dynamic[b2], {0, 1}, Appearance → "Open",
    AppearanceElements → {"InputField", "StepLeftButton", "StepRightButton"},
    ContinuousAction → True],
   Manipulator[Dynamic[cf1], {0, 10}, Appearance → "Open",
    AppearanceElements → {"InputField", "StepLeftButton", "StepRightButton"},
    ContinuousAction → True],
   Manipulator[Dynamic[cf2], {0, 10}, Appearance → "Open",
    AppearanceElements → {"InputField", "StepLeftButton", "StepRightButton"},
    ContinuousAction → True]
  },
  Text[Style[#, Bold]] & /@
   {"Sips Exponent: n", "Feed Concentration 1", "Feed Concentration 2"},
  List
 ],
 Inner[Labeled[#1, #2, Top] &,
  {
   Grid[{{IntervalSlider[{Dynamic[m1min], Dynamic[m1max]}, {0, 30, 1}],
      {Dynamic[m1min], Dynamic[m1max]}}}],
   Grid[{{IntervalSlider[{Dynamic[m2min], Dynamic[m2max]}, {0, 12, 0.1}],
      {Dynamic[m2min], Dynamic[m2max]}}}],
   Grid[{{IntervalSlider[{Dynamic[m3min], Dynamic[m3max]}, {0, 12, 0.1}],
      {Dynamic[m3min], Dynamic[m3max]}}}],
   Grid[{{IntervalSlider[{Dynamic[m4min], Dynamic[m4max]}, {0, 12, 0.1}],
      {Dynamic[m4min], Dynamic[m4max]}}}]
  },
```

```
    Text[Style[#, Bold]] & /@
     {"Range of m1 values to plot", "Range of m2 values to plot",
      "Range of m3 values to plot", "Range of m4 values to plot"},
    List
   ],
   {
    Button["Generate Triangle Theory Plots",
     Print[
      Grid[{{
          sepRegion[h1, b2, cf1, cf2, t0, qs, m2min, m2max, m3min, m3max],
          SpanFromLeft,
          separationRegion2[h1, b2, cf1, cf2,
           t0, qs, max[[1]], max[[2]], m1min, m1max, m4min, m4max],
          SpanFromLeft
         },
         {m1 == max2[[2]], m2 == max[[1]], m3 == max[[2]], m4 == max2[[1]]}}, Frame → All]
     ], Method → "Queued"
    ]

    , SpanFromLeft}
  }, Frame → All, Spacings → {2, 2}
 ]
]
```

Out[3]=



Appendix B.1

Standing Wave Design - Freundlich

Nonlinear ideal algorithm as described in “Simulated Moving Bed Chromatography for Chiral Separation”

Notation

In[127]:=
```
Needs["Notation`"]
Notation[ ip[[x_, y_]] ⇔ ip_{x_,y_} ]
Symbolize[ c_A ]
Symbolize[ c_B ]
Symbolize[ γ_+ ]
Symbolize[ γ_- ]
Symbolize[ A_+ ]
Symbolize[ A_- ]
Symbolize[ q_A ]
Symbolize[ q_B ]
Symbolize[ a_1 ]
Symbolize[ a_2 ]
Symbolize[ b_1 ]
Symbolize[ b_2 ]
Symbolize[ δ1_2 ]
Symbolize[ δ2_1 ]
Symbolize[ δ3_2 ]
Symbolize[ δ4_1 ]
```

Notes

This notebook requires the corresponding triangle theory notebook (i.e. the Freundlich notebook) be run first. This notebook makes use of the functions derived there.

Terminology Differences:
In this notebook, the "Dilution Concentration" is used to refer to the cβ state, i.e. the liquid state entering zone 3, after the feed.
The "Peak" concentrations refer to the steady state concentrations in zones 2 and 3, i.e. cII_2 and $cIII_1$

Preliminary Calculations

Calculate various system parameters to use them in later calculations.

In[145]:=
```
lumpedCoefficients = 15 * ϵp * diff / (particleRadius)^2;
phaseRatio = (1 - ϵb) / ϵb;
crossSection = N[Pi * (columnDiameter / 2)^2];
raffinateFlow1 = feedFlowrate * feedConcentration[[1]] * yieldTarget; // Quiet
```

Iterative Steps

Steps 1-8 define a the series of equations necessary to perform the Standing Wave Design. All the equations are fully symbolic at first, but are later simplified when the system parameters and design targets are specified (see the System Parameters and Design Targets section)

Step 1: Calculate $C_{peak,1}^{(0)}$ and $C_{peak,2}^{(0)}$ from $C_{s,1}^{(0)}$ and $C_{s,2}^{(0)}$

In this step, we use the intercept function to calculate the peak concentrations

```
step1[{cs1_, cs2_}] :=
 With[{intercepts = axesIntercepts[b1, b2, cs1, cs2, t0][[1]]},
  {intercepts[[2, 1]], intercepts[[1, 2]]}
 ]
```

Step 2: Calculate δ_2^{I} δ_1^{II} δ_2^{III} and δ_1^{IV}

δ values are related to the zones' maximum possible capacity of each component

In[150]:=
```
allAssumptions = {b1 > 0, b2 > 0, t > 1, b2 > b1}
```

Out[150]=
```
{b1 > 0, b2 > 0, t > 1, b2 > b1}
```

In[157]:=
```
δequations = ϵp + (1 - ϵp) *
    Block[{ja = jacobianIso, vec = eigensIso[[2]], eq1, eq2, eq3, eq4},
     eq1 = Simplify[ja[[1]].(vec[[1]] / vec[[1, 1]])];
     eq2 = Simplify[ja[[2]].vec[[2]]];
     eq3 = θshocks[{cs1, cs2}, {0, 0}, b1, b2, t][[2]];
     eq4 = θshocks[{cp1, 0}, {0, 0}, b1, b2, t][[1]];
     Simplify[
      {
       Limit[eq2 /. {c1 → 0}, c2 → 0, Assumptions → allAssumptions],
       eq1 /. {c1 → 0, c2 → cp2},
       eq3,
       eq4
      }
      , Assumptions → allAssumptions]
    ] // Quiet
```

Out[157]=
$$\left\{\epsilon_p,\ \epsilon_p + (1-\epsilon_p)\, b_1^{\frac{1}{t}} \left(cp2\, b_2^{\frac{1}{t}}\right)^{-1+t},\ b_2^{\frac{1}{t}} \left(b_1^{\frac{1}{t}}\, cs1 + b_2^{\frac{1}{t}}\, cs2\right)^{-1+t} (1-\epsilon_p) + \epsilon_p,\ b_1\, cp1^{-1+t}\, (1-\epsilon_p) + \epsilon_p\right\}$$

In[158]:=
```
δFunction[cp1_, cp2_, cs1_, cs2_, b1_, b2_, t0_, ϵp_] :=
 Evaluate[δequations /. {b1 → b1, b2 → b2, t → t0, ϵp → ϵp}]
```

In[159]:=
```
δFunction[cp1, cp2, cs1, cs2, b1, b2, t, ϵp]
```

Out[159]=
$$\left\{\epsilon_p,\ \epsilon_p + (1-\epsilon_p)\, b_1^{\frac{1}{t}} \left(b_2^{\frac{1}{t}}\, cp_2\right)^{-1+t},\ \epsilon_p + b_2^{\frac{1}{t}}\, (1-\epsilon_p) \left(b_1^{\frac{1}{t}}\, cs_1 + b_2^{\frac{1}{t}}\, cs_2\right)^{-1+t},\ \epsilon_p + b_1\, (1-\epsilon_p)\, cp_1^{-1+t}\right\}$$

In[160]:=
```
step2[{cs1_, cs2_}, {cp1_, cp2_}] := Block[{δ1₂, δ2₁, δ3₂, δ4₁},
  {δ1₂, δ2₁, δ3₂, δ4₁} = δFunction[cp1, cp2, cs1, cs2, b1, b2, t, ϵp];
  Evaluate[{δ1₂, δ2₁, δ3₂, δ4₁}]
 ]
```

Step 3: Calculate v, u_0^{I} u_0^{II} u_0^{III} and u_0^{IV}

The definitions of the interstitial velocities, coupled with a mass balance based on the feed flow rate lets us calculate the velocity in each zone (5 equations with 5 unknowns)

In[33]:=
```
step3[δ1_, δ2_, δ3_, δ4_] :=
 Block[{δ = {δ1, δ2, δ3, δ4}, v, interstitialVelocity},
  v = -feedFlowrate / (crossSection phaseRatio ϵb (δ[[2]] - δ[[3]]));
  interstitialVelocity = v * (1 + phaseRatio * #) & /@ δ;
  {v} ~Join~ interstitialVelocity
 ]

step3[{cs1_, cs2_}] := step3 @@ step2[{cs1, cs2}]
```

Step 4: Calculate $C_{peak,2}^{(1)}$ from $C_{Feed,1}$ and $C_{Feed,2}$

This value is constant.

In[35]:=
```
step4[] := step1[feedConcentration][[2]]
```

Step 5: Estimate $C_{peak,1}^{(1)}$ from a mass balance

raffinateFlow1 is a constant calculated in the initial calculations section, it is simply:
`feedFlowrate * feedConcentration[[1]] * yieldTarget`

```
step5[v_, u1_, u2_, u3_, u4_] := With[{raf = u3 - u4}, raffinateFlow1/(eb * crossSection * raf)]
```

In[37]:=
```
step5[{cs1_, cs2_}] := step5 @@ step3[{cs1, cs2}]
```

Step 6: Check the tolerances on $C_{peak,1}^{(1)}$ and $C_{peak,2}^{(1)}$

This is where the iteration is defined. If the tolerances are in bounds, the final cp value is returned. Otherwise, step 8 is invoked (which implicitly invokes step 7) to create a new estimate of cs1 and cs2. This new estimate is passed into step 6, which implicitly invokes steps 1-5 to calculate the new cp values.

In[38]:=
```
step6[cpOld_, cpNew_] := With[{tol = Abs[cpNew - cpOld]},
  { cpNew              And @@ Thread[tol < convergenceTolerance]
  { step6@step8[cpNew] True
 ]
step6[{cs1_, cs2_}] := step6[step1[{cs1, cs2}], {step4[], step5[{cs1, cs2}]} ]
```

Step 7: Calculate γ_+ and γ_- from $C_{peak,1}^{(1)}$ and $C_{peak,2}^{(1)}$

In[40]:=
```
step7[cp1_, cp2_] := With[{γ+ = (a2 - a1 - a1 b2 cp2)/(a2 b1 cp2), γ- = (a1 b2 cp1)/(a1 - a2 - a2 b1 cp1)},
  {γ+, γ-}
 ]
```

Step 8: Calculate $C_{s,1}^{(1)}$ and $C_{s,2}^{(1)}$ from γ_+ and γ_-

In[41]:=
```
Block[{cs1, cs2, cs1New, cs2New, denominator},
 denominator = a1 a2 b1 b2 (γ+ + γ-) + a1^2 b2^2 + a2^2 b1^2 γ+ γ-;
 cs1New = Simplify[a2 b1 γ+ γ- (a1 - a2) / denominator];
 cs2New = Simplify[a1 b2 (a2 - a1) / denominator];
 newEstimateEquations = {cs1New, cs2New} /. {γ+ → γ1, γ- → γ2}
]
```

Out[41]=
$$\left\{\frac{(a_1 - a_2)\, a_2\, b_1\, \gamma 1\, \gamma 2}{(a_1\, b_2 + a_2\, b_1\, \gamma 1)\, (a_1\, b_2 + a_2\, b_1\, \gamma 2)}, -\frac{a_1\, (a_1 - a_2)\, b_2}{(a_1\, b_2 + a_2\, b_1\, \gamma 1)\, (a_1\, b_2 + a_2\, b_1\, \gamma 2)}\right\}$$

In[42]:=
```
step8[γ1_, γ2_] := Evaluate[newEstimateEquations];
```

In[43]:=
```
step8[{cp1_, cp2_}] := step8 @@ step7[cp1, cp2]
```

Appendix B.2

Standing Wave Design - Sips

Nonlinear ideal algorithm as described in “Simulated Moving Bed Chromatography for Chiral Separation”

Notation

```
Needs["Notation`"]
Notation[ ip[[x_, y_]] ⇔ ip_{x_,y_} ]
Symbolize[ c_A ]
Symbolize[ c_B ]
Symbolize[ γ_+ ]
Symbolize[ γ_- ]
Symbolize[ A_+ ]
Symbolize[ A_- ]
Symbolize[ q_A ]
Symbolize[ q_B ]
Symbolize[ a_1 ]
Symbolize[ a_2 ]
Symbolize[ b_1 ]
Symbolize[ b_2 ]
Symbolize[ δ1_2 ]
Symbolize[ δ2_1 ]
Symbolize[ δ3_2 ]
Symbolize[ δ4_1 ]
```

Notes

This notebook requires the corresponding triangle theory notebook (i.e. the Sips notebook) be run first. This notebook makes use of the functions derived there.

Terminology Differences:
In this notebook, the "Dilution Concentration" is used to refer to the cβ state, i.e. the liquid state entering zone 3, after the feed.
The "Peak" concentrations refer to the steady state concentrations in zones 2 and 3, i.e. cII_2 and $cIII_1$

Preliminary Calculations

Calculate various system parameters to use them in later calculations.

```
lumpedCoefficients = 15 * ϵp * diff / (particleRadius)^2;
phaseRatio = (1 - ϵb) / ϵb;
crossSection = N[Pi * (columnDiameter / 2)^2];
raffinateFlow1 = feedFlowrate * feedConcentration[[1]] * yieldTarget; // Quiet
```

Iterative Steps

Steps 1-8 define a the series of equations necessary to perform the Standing Wave Design. All the equations are fully symbolic at first, but are later simplified when the system parameters and design targets are specified (see the System Parameters and Design Targets section)

Step 1: Calculate $C_{peak,1}^{(0)}$ and $C_{peak,2}^{(0)}$ from $C_{s,1}^{(0)}$ and $C_{s,2}^{(0)}$

In this step, we use the intercept function to calculate the peak concentrations

In[127]:=
```
step1[{cs1_, cs2_}] :=
 With[{intercepts = axesIntercepts[a1, a2, b1, b2, cs1, cs2, t0][[1]]},
  {intercepts[[2, 1]], intercepts[[1, 2]]}
 ]
```

Step 2: Calculate δ_2^{I} δ_1^{II} δ_2^{III} and δ_1^{IV}

δ values are related to the zones' maximum possible capacity of each component

In[159]:=
```
allAssumptions = {a1 > 0, a2 > 0, b1 > 0, b2 > 0, a2 > a1,
   1 > t > 0, c1 ≥ 0, c2 ≥ 0, a2 (1 + b1 c1^t) + a1 (-1 + (-1 + t) b1 c1^t) ≥ 0}
```

Out[159]=
```
{a1 > 0, a2 > 0, b1 > 0, b2 > 0, a2 > a1, 1 > t > 0,
 c1 ≥ 0, c2 ≥ 0, a2 (1 + b1 c1^t) + a1 (-1 + (-1 + t) b1 c1^t) ≥ 0}
```

In[160]:=
```
δequations = ϵp + (1 - ϵp) *
    Block[{ja = jacobianIso, vec = eigensIso[[2]], eq1, eq2, eq3, eq4},
     eq1 = Simplify[ja[[1]].(vec[[1]] / vec[[1, 1]])];
     eq2 = Simplify[ja[[2]].vec[[2]]];
     eq3 = θshocks[{cs1, cs2}, {0, 0}, a1, a2, b1, b2, t][[2]];
     eq4 = θshocks[{cp1, 0}, {0, 0}, a1, a2, b1, b2, t][[1]];
     Simplify[#, Assumptions → allAssumptions] & /@
      {
       Limit[eq2, c2 → 0, Assumptions → allAssumptions] /. c1 → 0,
       Limit[eq1, c1 → 0, Assumptions → allAssumptions] /. {c2 → cp2},
       eq3,
       eq4
      }

    ];
```

Power::infy : Infinite expression $\frac{1}{0}$ encountered. >>

In[162]:=
```
δFunction[cp1_, cp2_, cs1_, cs2_, a1_, a2_, b1_, b2_, t0_, ϵp_] :=
 Evaluate[δequations /. {a1 → a1, a2 → a2, b1 → b1, b2 → b2, t → t0, qs → qs, ϵp → ϵp}]
```

In[163]:=
```
δFunction[cp1, cp2, cs1, cs2, a1, a2, b1, b2, t, ϵp]
```

Out[163]=
$$\left\{\epsilon_p + (1-\epsilon_p)\, a_2,\ \epsilon_p + \frac{(1-\epsilon_p)\, a_1}{1 + b_2\, cp_2^t},\ \epsilon_p + \frac{(1-\epsilon_p)\, a_2}{1 + b_1\, cs_1^t + b_2\, cs_2^t},\ \epsilon_p + \frac{(1-\epsilon_p)\, a_1}{1 + b_1\, cp_1^t}\right\}$$

```
step2[{cs1_, cs2_}, {cp1_, cp2_}] := Block[{δ12, δ21, δ32, δ41},
  {δ12, δ21, δ32, δ41} = δFunction[cp1, cp2, cs1, cs2, a1, a2, b1, b2, t, ϵp];
  Evaluate[{δ12, δ21, δ32, δ41}]
 ]
```

Step 3: Calculate v, u_0^{I} u_0^{II} u_0^{III} and u_0^{IV}

The definitions of the interstitial velocities, coupled with a mass balance based on the feed flow rate lets us calculate the velocity in each zone (5 equations with 5 unknowns)

In[33]:=
```
step3[δ1_, δ2_, δ3_, δ4_] :=
 Block[{δ = {δ1, δ2, δ3, δ4}, v, interstitialVelocity},
  v = -feedFlowrate / (crossSection phaseRatio ϵb (δ[[2]] - δ[[3]]));
  interstitialVelocity = v * (1 + phaseRatio * #) & /@ δ;
  {v} ~Join~ interstitialVelocity
 ]

step3[{cs1_, cs2_}] := step3 @@ step2[{cs1, cs2}]
```

Step 4: Calculate $C_{peak,2}^{(1)}$ from $C_{Feed,1}$ and $C_{Feed,2}$

This value is constant.

```
step4[] := step1[feedConcentration][[2]]
```

Step 5: Estimate $C_{peak,1}^{(1)}$ from a mass balance

raffinateFlow1 is a constant calculated in the initial calculations section, it is simply: `feedFlowrate * feedConcentration[[1]] * yieldTarget`

```
step5[v_, u1_, u2_, u3_, u4_] := With[{raf = u3 - u4}, raffinateFlow1/(eb * crossSection * raf)]
```

In[37]:=
```
step5[{cs1_, cs2_}] := step5 @@ step3[{cs1, cs2}]
```

Step 6: Check the tolerances on $C_{peak,1}^{(1)}$ and $C_{peak,2}^{(1)}$

In[38]:=
```
step6[cpOld_, cpNew_] := With[{tol = Abs[cpNew - cpOld]},
  {cpNew, And @@ Thread[tol < convergenceTolerance]
   step6@step8[cpNew], True}
 ]
step6[{cs1_, cs2_}] := step6[step1[{cs1, cs2}], {step4[], step5[{cs1, cs2}]}]
```

Step 7: Calculate γ_+ and γ_- from $C_{peak,1}^{(1)}$ and $C_{peak,2}^{(1)}$

In[40]:=
```
step7[cp1_, cp2_] := With[{γ+ = (a2 - a1 - a1 b2 cp2)/(a2 b1 cp2), γ- = (a1 b2 cp1)/(a1 - a2 - a2 b1 cp1)},
  {γ+, γ-}
 ]
```

Step 8: Calculate $C_{s,1}^{(1)}$ and $C_{s,2}^{(1)}$ from γ_+ and γ_-

In[41]:=
```
Block[{cs1, cs2, cs1New, cs2New, denominator},
 denominator = a1 a2 b1 b2 (γ+ + γ-) + a1^2 b2^2 + a2^2 b1^2 γ+ γ-;
 cs1New = Simplify[(a2 b1 γ+ γ- (a1 - a2))/denominator];
 cs2New = Simplify[(a1 b2 (a2 - a1))/denominator];
 newEstimateEquations = {cs1New, cs2New} /. {γ+ → γ1, γ- → γ2}
]
```

Out[41]=
$$\left\{\frac{(a_1 - a_2)\, a_2\, b_1\, \gamma 1\, \gamma 2}{(a_1\, b_2 + a_2\, b_1\, \gamma 1)\, (a_1\, b_2 + a_2\, b_1\, \gamma 2)}, -\frac{a_1\, (a_1 - a_2)\, b_2}{(a_1\, b_2 + a_2\, b_1\, \gamma 1)\, (a_1\, b_2 + a_2\, b_1\, \gamma 2)}\right\}$$

In[42]:=
```
step8[γ1_, γ2_] := Evaluate[newEstimateEquations];
```

In[43]:=
```
step8[{cp1_, cp2_}] := step8 @@ step7[cp1, cp2]
```

Appendix B.3

Standing Wave Design - NonLinear

Nonlinear ideal algorithm as described in “Simulated Moving Bed Chromatography for Chiral Separation”

Notation

```
Needs["Notation`"]
Notation[ ip[[x_, y_]] ⇔ ip_{x_,y_} ]
Symbolize[ c_A ]
Symbolize[ c_B ]
Symbolize[ γ_+ ]
Symbolize[ γ_- ]
Symbolize[ A_+ ]
Symbolize[ A_- ]
Symbolize[ q_A ]
Symbolize[ q_B ]
Symbolize[ a_1 ]
Symbolize[ a_2 ]
Symbolize[ b_1 ]
Symbolize[ b_2 ]
Symbolize[ δ1_2 ]
Symbolize[ δ2_1 ]
Symbolize[ δ3_2 ]
Symbolize[ δ4_1 ]
```

Notes

This notebook requires the corresponding triangle theory notebook (i.e. the Toth notebook) be run first. This notebook makes use of the functions derived there.

Terminology Differences:
In this notebook, the “Dilution Concentration” is used to refer to the cβ state, i.e. the liquid state entering zone 3, after the feed.
The “Peak” concentrations refer to the steady state concentrations in zones 2 and 3, i.e. cII_2 and $cIII_1$

Preliminary Calculations

Calculate various system parameters to use them in later calculations.

```
lumpedCoefficients = 15 * ϵp * diff / (particleRadius) ^ 2;
phaseRatio = (1 - ϵb) / ϵb;
crossSection = N[Pi * (columnDiameter / 2) ^ 2];
raffinateFlow1 = feedFlowrate * feedConcentration[[1]] * yieldTarget; // Quiet
```

Iterative Steps

Steps 1-8 define a the series of equations necessary to perform the Standing Wave Design. All the equations are fully symbolic at first, but are later simplified when the system parameters and design targets are specified (see the System Parameters and Design Targets section)

Step 1: Calculate $C_{peak,1}^{(0)}$ and $C_{peak,2}^{(0)}$ from $C_{s,1}^{(0)}$ and $C_{s,2}^{(0)}$

In this step, we use the intercept function to calculate the peak concentrations

```
step1[{cs1_, cs2_}] :=
 With[{intercepts = axesIntercepts[a1, a2, b1, b2, cs1, cs2, t0][[1]]},
  {intercepts[[2, 1]], intercepts[[1, 2]]}
 ]
```

Step 2: Calculate δ_2^{I} δ_1^{II} δ_2^{III} and δ_1^{IV}

δ values are related to the zones’ maximum possible capacity of each component

```
allAssumptions = {b1 > 0, b2 > 0, b2 > b1, c1 > 0, c2 ≥ 0, qs > 0, 1 > t > 0};
```

```
δequations = ϵp + (1 - ϵp) *
    Block[{ja = jacobianIso, vec = eigensIso[[2]], eq1, eq2, eq3, eq4},
     eq1 = Simplify[ja[[1]].(vec[[1]] / vec[[1, 1]])];
     eq2 = Simplify[ja[[2]].vec[[2]]];
     eq3 = θshocks[{cs1, cs2}, {0, 0}, b1, b2, t, qs][[2]];
     eq4 = θshocks[{cp1, 0}, {0, 0}, b1, b2, t, qs][[1]];
     Simplify[
      {
       Limit[eq2 /. {c1 → 0}, c2 → 0, Assumptions → allAssumptions],
       eq1 /. {c1 → 0, c2 → cp2},
       eq3,
       eq4
      }
      , Assumptions → allAssumptions]
    ];
```

Power::infy : Infinite expression $\frac{1}{0}$ encountered. ≫

Infinity::indet : Indeterminate expression $0\,(1+0^{t})^{-1/t}\,q_s$ ComplexInfinity b_2 encountered. ≫

```
δFunction[cp1_, cp2_, cs1_, cs2_, b1_, b2_, t0_, qs_, ϵp_] :=
 Evaluate[δequations /. {b1 → b1, b2 → b2, t → t0, qs → qs, ϵp → ϵp}]
```

```
δFunction[cp1, cp2, cs1, cs2, b1, b2, t, qs, ϵp]
```

$$\left\{\epsilon_p + q_s\,(1-\epsilon_p)\,b_2,\ \epsilon_p + q_s\,(1-\epsilon_p)\,b_1\,(1+(b_2\,cp_2)^t)^{-1/t},\right.$$
$$\left.\epsilon_p + q_s\,(1-\epsilon_p)\,b_2\,(1+(b_1\,cs_1+b_2\,cs_2)^t)^{-1/t},\ \epsilon_p + q_s\,(1-\epsilon_p)\,b_1\,(1+(b_1\,cp_1)^t)^{-1/t}\right\}$$

```
step2[{cs1_, cs2_}, {cp1_, cp2_}] := Block[{δ12, δ21, δ32, δ41},
  {δ12, δ21, δ32, δ41} = δFunction[cp1, cp2, cs1, cs2, a1, a2, b1, b2, t, ϵp];
  Evaluate[{δ12, δ21, δ32, δ41}]
 ]
```

Step 3: Calculate v, u_0^{I} u_0^{II} u_0^{III} and u_0^{IV}

The definitions of the interstitial velocities, coupled with a mass balance based on the feed flow rate lets us calculate the velocity in each zone (5 equations with 5 unknowns)

```
step3[δ1_, δ2_, δ3_, δ4_] :=
 Block[{δ = {δ1, δ2, δ3, δ4}, v, interstitialVelocity},
  v = -feedFlowrate / (crossSection phaseRatio ϵb (δ[[2]] - δ[[3]]));
  interstitialVelocity = v * (1 + phaseRatio * #) & /@ δ;
  {v} ~Join~ interstitialVelocity
 ]
```

```
step3[{cs1_, cs2_}] := step3 @@ step2[{cs1, cs2}]
```

Step 4: Calculate $C_{peak,2}^{(1)}$ from $C_{Feed,1}$ and $C_{Feed,2}$

This value is constant.

```
step4[] := step1[feedConcentration][[2]]
```

Step 5: Estimate $C_{peak,1}^{(1)}$ from a mass balance

raffinateFlow1 is a constant calculated in the initial calculations section, it is simply:
`feedFlowrate * feedConcentration[[1]] * yieldTarget`

```
step5[v_, u1_, u2_, u3_, u4_] := With[{raf = u3 - u4}, raffinateFlow1/(eb * crossSection * raf)]

step5[{cs1_, cs2_}] := step5 @@ step3[{cs1, cs2}]
```

Step 6: Check the tolerances on $C_{peak,1}^{(1)}$ and $C_{peak,2}^{(1)}$

```
step6[cpOld_, cpNew_] := With[{tol = Abs[cpNew - cpOld]},
  { cpNew                And @@ Thread[tol < convergenceTolerance]
  { step6@step8[cpNew]   True
 ]
step6[{cs1_, cs2_}] := step6[step1[{cs1, cs2}], {step4[], step5[{cs1, cs2}]} ]
```

Step 7: Calculate γ_+ and γ_- from $C_{peak,1}^{(1)}$ and $C_{peak,2}^{(1)}$

```
step7[cp1_, cp2_] := With[{γ+ = (a2 - a1 - a1 b2 cp2)/(a2 b1 cp2), γ- = (a1 b2 cp1)/(a1 - a2 - a2 b1 cp1)},
  {γ+, γ-}
 ]
```

Step 8: Calculate $C_{s,1}^{(1)}$ and $C_{s,2}^{(1)}$ from γ_+ and γ_-

```
Block[{cs1, cs2, cs1New, cs2New, denominator},
 denominator = a1 a2 b1 b2 (γ+ + γ-) + a1^2 b2^2 + a2^2 b1^2 γ+ γ-;
 cs1New = Simplify[(a2 b1 γ+ γ- (a1 - a2))/denominator];
 cs2New = Simplify[(a1 b2 (a2 - a1))/denominator];
 newEstimateEquations = {cs1New, cs2New} /. {γ+ → γ1, γ- → γ2}
]
```

$$\left\{\frac{(a_1 - a_2)\, a_2\, b_1\, \gamma 1\, \gamma 2}{(a_1\, b_2 + a_2\, b_1\, \gamma 1)\, (a_1\, b_2 + a_2\, b_1\, \gamma 2)}, -\frac{a_1\, (a_1 - a_2)\, b_2}{(a_1\, b_2 + a_2\, b_1\, \gamma 1)\, (a_1\, b_2 + a_2\, b_1\, \gamma 2)}\right\}$$

```
step8[γ1_, γ2_] := Evaluate[newEstimateEquations];

step8[{cp1_, cp2_}] := step8 @@ step7[cp1, cp2]
```

Appendix B.4

Standing Wave Design - Ideal Langmuir

Notation

```
Needs["Notation`"]
```

```
Symbolize[ u_0^i ]
Symbolize[ u_0^ii ]
Symbolize[ u_0^iii ]
Symbolize[ u_0^iv ]
Notation[ ip[[x_, y_]] ⟺ ip_{x_,y_} ]
Symbolize[ c_A ]
Symbolize[ c_B ]
Symbolize[ γ_+ ]
Symbolize[ γ_- ]
Symbolize[ A_+ ]
Symbolize[ A_- ]
Symbolize[ q_A ]
Symbolize[ q_B ]
Symbolize[ a_1 ]
Symbolize[ a_2 ]
Symbolize[ b_1 ]
Symbolize[ b_2 ]
Symbolize[ δ1_2 ]
Symbolize[ δ2_1 ]
Symbolize[ δ3_2 ]
Symbolize[ δ4_1 ]
```

Preliminary Calculations

Calculate various system parameters to use them in later calculations.

```
lumpedCoefficients = 15 * ϵp * diff / (particleRadius) ^ 2;
phaseRatio = (1 - ϵb) / ϵb;
crossSection = N[Pi * (columnDiameter / 2) ^ 2];
raffinateFlow1 = feedFlowrate * feedConcentration[[1]] * yieldTarget; // Quiet
```

Isotherm Definition

```
loadingA[ca_, cb_] := a1 ca / (1 + b1 ca + b2 cb);
loadingB[ca_, cb_] := a2 cb / (1 + b1 ca + b2 cb);
```

Ideal SWD steps

Steps 1-8 define a the series of equations necessary to perform the Standing Wave Design. All the equations are fully symbolic at first, but are later simplified when the system parameters and design targets are specified (see the System Parameters and Design Targets section)

Step 1: Calculate $C_{peak,1}^{(0)}$ and $C_{peak,2}^{(0)}$ from $C_{s,1}^{(0)}$ and $C_{s,2}^{(0)}$

```
Block[{cA, cB, γ, qA = loadingA[cA, cB], qB = loadingB[cA, cB], γPolynomialSolutions},

 γPolynomial = γ^2 ∂cA qB + (∂cB qB - ∂cA qA) γ - ∂cB qA == 0;

 γPolynomialSolutions = Simplify@Solve[γPolynomial, γ];
 {A+, A-} = (a1 - a2) / (a2 b1 + a1 b2 / γ) /. γPolynomialSolutions;
 step1eq =
  Simplify[{A-, -A+ / γ} /. First[γPolynomialSolutions]] /. {cA → cs1, cB → cs2}
]
```

$$\left\{\frac{(a_1-a_2)\Big/\left(a_2 b_1+\frac{2 a_1 a_2 b_1 b_2 cs2}{a_2+a_2 b_1 cs1-a_1 (1+b_2 cs2)-(1+b_1 cs1+b_2 cs2)\sqrt{\left((a_2+a_2 b_1 cs1)^2+(a_1+a_1 b_2 cs2)^2+2 a_1 a_2 (-1-b_2 cs2+b_1 cs1 (-1+b_2 cs2))\right)\Big/(1+b_1 cs1+b_2 cs2)^2}}\right)}{},\right.$$
$$\left.-\frac{(2 (a_1-a_2)\, cs2)\Big/\left(a_2+a_2 b_1 cs1+a_1 (-1+b_2 cs2)+(1+b_1 cs1+b_2 cs2)\sqrt{\left((a_2+a_2 b_1 cs1)^2+(a_1+a_1 b_2 cs2)^2+2 a_1 a_2 (-1-b_2 cs2+b_1 cs1 (-1+b_2 cs2))\right)\Big/(1+b_1 cs1+b_2 cs2)^2}\right)}\right\}$$

```
step1[{cs1_, cs2_}] := Evaluate[step1eq]
```

Step 2: Calculate δ_2^{I} δ_1^{II} δ_2^{III} and δ_1^{IV}

```
step2[{cs1_, cs2_}, {cp1_, cp2_}] := Block[{δ1₂, δ2₁, δ3₂, δ4₁},
  δ1₂ = ϵp + (1 - ϵp) * a₂;
  δ2₁ = ϵp + (1 - ϵp) a₁ / (1 + b₂ cp2);
  δ3₂ = ϵp + (1 - ϵp) a₂ / (1 + b₁ cs1 + b₂ cs2);
  δ4₁ = ϵp + (1 - ϵp) a₁ / (1 + b₁ cp1);
  Evaluate[{δ1₂, δ2₁, δ3₂, δ4₁}]
 ]

step2[{cs1_, cs2_}] := step2[{cs1, cs2}, step1[{cs1, cs2}]]
```

Step 3: Calculate v, u_0^{I} u_0^{II} u_0^{III} and u_0^{IV}

```
step3[δ1_, δ2_, δ3_, δ4_] :=
 Block[{δ = {δ1, δ2, δ3, δ4}, v, interstitialVelocity},
  v = - feedFlowrate / (crossSection phaseRatio ϵb (δ[[2]] - δ[[3]]));
  interstitialVelocity = v * (1 + phaseRatio * #) & /@ δ;
  {v} ~Join~ interstitialVelocity
 ]

step3[{cs1_, cs2_}] := step3 @@ step2[{cs1, cs2}]
```

Step 4: Calculate $C_{peak,2}^{(1)}$ from $C_{Feed,1}$ and $C_{Feed,2}$

```
step4[] := step1[feedConcentration][[2]]
```

Step 5: Estimate $C_{peak,1}^{(1)}$ from a mass balance

raffinateFlow1 is a constant calculated in the initial calculations section, it is simply:
`feedFlowrate * feedConcentration[[1]] * yieldTarget`

```
step5[v_, u1_, u2_, u3_, u4_] := With[{raf = u3 - u4}, raffinateFlow1 / (ϵb * crossSection * raf)]


step5[{cs1_, cs2_}] := step5 @@ step3[{cs1, cs2}]
```

Step 6: Check the tolerances on $C_{peak,1}^{(1)}$ and $C_{peak,2}^{(1)}$

This is where the iteration is defined. If the tolerances are in bounds, the final cp value is returned. Otherwise, step 8 is invoked (which implicitly invokes step 7) to create a new estimate of cs1 and cs2.

This new estimate is passed into step 6, which implicitly invokes steps 1-5 to calculate the new cp values.

```
step6[cpOld_, cpNew_] := With[{tol = Abs[cpNew - cpOld]},
  { cpNew                And @@ Thread[tol < convergenceTolerance]
  { step6@step8[cpNew]   True
 ]
step6[{cs1_, cs2_}] := step6[step1[{cs1, cs2}], {step4[], step5[{cs1, cs2}]} ]
```

Step 7: Calculate γ_+ and γ_- from $C_{peak,1}^{(1)}$ and $C_{peak,2}^{(1)}$

```
step7[cp1_, cp2_] := With[{γ+ = (a2 - a1 - a1 b2 cp2)/(a2 b1 cp2), γ- = (a1 b2 cp1)/(a1 - a2 - a2 b1 cp1)},
  {γ+, γ-}
 ]
```

Step 8: Calculate $C_{s,1}^{(1)}$ and $C_{s,2}^{(1)}$ from γ_+ and γ_-

```
Block[{cs1, cs2, cs1New, cs2New, denominator},
 denominator = a1 a2 b1 b2 (γ+ + γ-) + a1^2 b2^2 + a2^2 b1^2 γ+ γ-;
 cs1New = Simplify[(a2 b1 γ+ γ- (a1 - a2))/denominator];
 cs2New = Simplify[(a1 b2 (a2 - a1))/denominator];
 newEstimateEquations = {cs1New, cs2New} /. {γ+ → γ1, γ- → γ2}
]
```

$$\left\{\frac{(a_1 - a_2)\, a_2\, b_1\, \gamma 1\, \gamma 2}{(a_1\, b_2 + a_2\, b_1\, \gamma 1)\, (a_1\, b_2 + a_2\, b_1\, \gamma 2)},\ -\frac{a_1\, (a_1 - a_2)\, b_2}{(a_1\, b_2 + a_2\, b_1\, \gamma 1)\, (a_1\, b_2 + a_2\, b_1\, \gamma 2)}\right\}$$

```
step8[γ1_, γ2_] := Evaluate[newEstimateEquations];
step8[{cp1_, cp2_}] := step8 @@ step7[cp1, cp2]
```

System Parameters

These parameters must be known or estimated beforehand. To change a value, simply type in a new number, then press shift+enter

```
particleRadius = Quantity[.027, "mm"];
ϵp = 0.55;
ϵb = 0.38;
ϵt = ϵb + ϵp (1 - ϵb);
diffusivity1 = Quantity[0.007, "cm^2/s"];
diffusivity2 = Quantity[0.007, "cm^2/s"];
a1 = .448;
a2 = 1.44;
b1 = Quantity[.0102, "L/g"];
b2 = Quantity[.0354, "L/g"];
zoneLength = Quantity[10, "cm"];
columnDiameter = Quantity[1, "cm"];
dispersion1 = Quantity[0.1, "cm^2/min"];
dispersion2 = Quantity[0.1, "cm^2/min"];

filmMTC = Quantity[2.25, "cm/min"];

particleRadius = QuantityMagnitude[UnitConvert[particleRadius, "m"]];
diffusivity1   = QuantityMagnitude[UnitConvert[diffusivity1, "m^2/s"]];
diffusivity2   = QuantityMagnitude[UnitConvert[diffusivity2, "m^2/s"]];
b1             = QuantityMagnitude[UnitConvert[b1, "m^3/kg"]];
b2             = QuantityMagnitude[UnitConvert[b2, "m^3/kg"]];
zoneLength     = QuantityMagnitude[UnitConvert[zoneLength, "m"]];
columnDiameter = QuantityMagnitude[UnitConvert[columnDiameter, "m"]];
dispersion1    = QuantityMagnitude[UnitConvert[dispersion1, "m^2/s"]];
dispersion2    = QuantityMagnitude[UnitConvert[dispersion2, "m^2/s"]];
filmMTC        = QuantityMagnitude[UnitConvert[filmMTC, "m/s"]];

diff           = {diffusivity1, diffusivity2};
ip1            = {a1, a2};
ip2            = {b1, b2};
disp           = {dispersion1, dispersion2};
```

Unit Conversions

Unfortunately, leaving units in the equations drastically reduces the performance of the entire algorithm. Therefore, the dimensional units are converted to SI and then stripped of their units.

Design Targets

These parameters must be specified, it seems that the final design is optimized by changing these values

```
yieldTarget = .99;
feedConcentration1 = Quantity[5, "g/L"];
feedConcentration2 = Quantity[5, "g/L"];

feedConcentration1 = QuantityMagnitude[UnitConvert[feedConcentration1, "kg/m^3"]];
feedConcentration2 = QuantityMagnitude[UnitConvert[feedConcentration2, "kg/m^3"]];

feedFlowrate = Quantity[.5, "mL/min"];
feedFlowrate       = QuantityMagnitude[UnitConvert[feedFlowrate, "m^3/s"]];

feedConcentration = {feedConcentration1, feedConcentration2};

convergenceTolerance = 0.00001;
```

Results

```
idealDesign[{cs1_, cs2_}] :=
 With[{converged = step6[{cs1, cs2}]}, step3@step8[converged]]

calculateMvals[velocities_] := Block[{v = velocities[[1]],
    zonev = velocities[[2 ;;]], tsw, vBed = crossSection * zoneLength, mvals},
  tsw = zoneLength / v;
  mvals = (zonev * crossSection * tsw - εt vBed) / (vBed (1 - εt));
  velocities ~Join~ mvals ~Join~ {tsw}
 ]

makePretty[result_] := Grid[{{"Standing Wave Design Results", SpanFromLeft},
    {"v", "u_0^1", "u_0^II", "u_0^III", "u_0^IV", "m_1", "m_2", "m_3", "m_4", "t_sw"}, result},
  Frame → All, ItemStyle → {Automatic,
    {Directive[Bold, FontSize → 14], Directive[Bold, FontSize → 14]}},
  Background → {None, {LightBlue, LightBlue}}, Spacings → {1, 1}]

makePretty@calculateMvals@idealDesign[feedConcentration]
```

Standing Wave Design Results									
v	u_0^1	u_0^{II}	u_0^{III}	u_0^{IV}	m_1	m_2	m_3	m_4	t_{sw}
0.000 4870 7	0.001 4391 1	0.001 0590 8	0.001 3383	0.001 0771 4	8.005 85	5.2093	7.264 01	5.342 16	205.3 09

Appendix B.5

Standing Wave Design - Non-Ideal Langmuir

Notation

```
Needs["Notation`"]
```

```
Symbolize[ u_0^i ]
Symbolize[ u_0^ii ]
Symbolize[ u_0^iii ]
Symbolize[ u_0^iv ]
Notation[ ip[[x_, y_]] ⟺ ip_{x_,y_} ]
Symbolize[ c_A ]
Symbolize[ c_B ]
Symbolize[ γ_+ ]
Symbolize[ γ_- ]
Symbolize[ A_+ ]
Symbolize[ A_- ]
Symbolize[ q_A ]
Symbolize[ q_B ]
Symbolize[ a_1 ]
Symbolize[ a_2 ]
Symbolize[ b_1 ]
Symbolize[ b_2 ]
Symbolize[ δ1_2 ]
Symbolize[ δ2_1 ]
Symbolize[ δ3_2 ]
Symbolize[ δ4_1 ]
```

Preliminary Calculations

Calculate various system parameters to use them in later calculations.

```
lumpedCoefficients = 15 * ϵp * diff / (particleRadius) ^ 2;
phaseRatio = (1 - ϵb) / ϵb;
crossSection = N[Pi * (columnDiameter / 2) ^ 2];
raffinateFlow1 = feedFlowrate * feedConcentration[[1]] * yieldTarget; // Quiet
```

Isotherm Definition

```
loadingA[ca_, cb_] := a1 ca / (1 + b1 ca + b2 cb);
loadingB[ca_, cb_] := a2 cb / (1 + b1 ca + b2 cb);
```

Equations from the ideal design that are useful

```
Block[{cA, cB, γ, qA = loadingA[cA, cB], qB = loadingB[cA, cB], γPolynomialSolutions},

 γPolynomial = γ^2 ∂cA qB + (∂cB qB - ∂cA qA) γ - ∂cB qA == 0;

 γPolynomialSolutions = Simplify@Solve[γPolynomial, γ];
 {A+, A-} = (a1 - a2) / (a2 b1 + a1 b2 / γ) /. γPolynomialSolutions;
 step1eq =
  Simplify[{A-, -A+ / γ} /. First[γPolynomialSolutions]] /. {cA → cs1, cB → cs2}
]
idealStep1[{cs1_, cs2_}] := Evaluate[step1eq]
```

$$\Big\{(a_1 - a_2) \Big/ \Big(a_2 b_1 + (2\, a_1\, a_2\, b_1\, b_2\, cs2) \Big/ \Big(a_2 + a_2 b_1\, cs1 - a_1\,(1 + b_2\, cs2) - (1 + b_1\, cs1 + b_2\, cs2)\sqrt{\Big((a_2 + a_2 b_1\, cs1)^2 + (a_1 + a_1 b_2\, cs2)^2 + 2\, a_1\, a_2\,(-1 - b_2\, cs2 + b_1\, cs1\,(-1 + b_2\, cs2))\Big) \Big/ (1 + b_1\, cs1 + b_2\, cs2)^2}\Big)\Big),$$
$$-\Big((2\,(a_1 - a_2)\, cs2) \Big/ \Big(a_2 + a_2 b_1\, cs1 + a_1\,(-1 + b_2\, cs2) + (1 + b_1\, cs1 + b_2\, cs2)\sqrt{\Big((a_2 + a_2 b_1\, cs1)^2 + (a_1 + a_1 b_2\, cs2)^2 + 2\, a_1\, a_2\,(-1 - b_2\, cs2 + b_1\, cs1\,(-1 + b_2\, cs2))\Big) \Big/ (1 + b_1\, cs1 + b_2\, cs2)^2}\Big)\Big)\Big\}$$

Non-Ideal SWD Steps

```
idealDesign[{cs1_, cs2_}] := With[{converged = step6[{cs1, cs2}]},
  step3@step8[converged] ~Join~ step8[converged]]
```

Step 0: Initialization, define the methods for calculating E_b, k_f, and K_f

```
niStep0[] := {Eb, kf, Kf}
niStep0[] := {dispersion1, filmMTC, 1 / (particleRadius^2 / (15 ϵp diffusivity1) + particleRadius / (3 filmMTC))}
```

Step 1: Calculate the β values

```
niStep1[v_, u1_, u2_, u3_, u4_, cs1_, cs2_] :=
 Block[{β1, β2, β3, β4, ce2, c12, ce1, cr1, cr2, c41, css1, css2,
   raf = u3 - u4, ext = u1 - u2, feed = u3 - u2},
  cr1 = (feed * feedConcentration1 * yieldTarget)/raf;
  ce1 = (feed * feedConcentration1 * (1 - yieldTarget))/ext;
  css1 = (u3 * cs1 - feed * feedConcentration1)/u2;
  c41 = (u1 * ce1)/u4;
  cr2 = (feed * feedConcentration2 * (1 - yieldTarget))/raf;
  ce2 = (feed * feedConcentration2 * yieldTarget)/ext;
  css2 = (u3 * cs2 - feed * feedConcentration2)/u2;
  c12 = (u4 * cr2)/u1;

  β1 = Log[ce2/c12];
  β2 = Log[css1/ce1];
  β3 = Log[cs2/cr2];
  β4 = Log[cr1/c41];
  {β1, β2, β3, β4}
 ]

niStep1[{cs1_, cs2_}] := niStep1 @@ (idealDesign[{cs1, cs2}])
  (* If this step is called with a concentration only,
  it gets velocity estimates from the ideal design*)
```

Step 2: Calculate the zone flow rates

```
niStep2[β1_, β2_, β3_, β4_, d1_, d2_, d3_, d4_, E_, k_, K_] :=
 Block[{v, u1, u2, u3, u4},
  v =
   (K zoneLength (-d2 + d3 + √((d2 - d3)^2 - (4 (d2^2 β2 + d3^2 β3) (feedFlowrate * zoneLength +
                    E * crossSection * (β2 + β3) * ϵb)) / (K * zoneLength^2 *
                phaseRatio * crossSection * ϵb))))) / (2 (d2^2 β2 + d3^2 β3));

  u1 = (1 + phaseRatio * d1) v + β1/zoneLength (E + phaseRatio * v^2 * d1^2/K);
  u2 = (1 + phaseRatio * d2) v + β2/zoneLength (E + phaseRatio * v^2 * d2^2/K);
  u3 = (1 + phaseRatio * d3) v - β3/zoneLength (E + phaseRatio * v^2 * d3^2/K);
  u4 = (1 + phaseRatio * d4) v - β4/zoneLength (E + phaseRatio * v^2 * d4^2/K);

  {v, u1, u2, u3, u4}
 ]
```

Step 3: Check the zone flow rates for convergence

```
niStep3[vOld_, vNew_, cs_] := With[{tol = Abs[vNew - vOld]},
   {vNew, niStep3[vNew, cs]}  And @@ Thread[tol < convergenceTolerance]
 ]
```

With an estimate of the velocities and cs, we've got a normal iteration

```
niStep3[vel_, cs_] :=
 With[{δest = niStep9[cs], βest = niStep1 @@ vel ~Join~ cs, mtc = niStep0[]},
  niStep3[vel, niStep2 @@ (βest ~Join~ δest ~Join~ mtc), cs]
 ]
```

Without an estimate of the velocities, we get the values from the ideal design. That estimate is then used as the first estimate in the normal iteration

```
niStep3[cs_] := With[{design = idealDesign[cs]},
  niStep3[design[[1 ;; 5]], design[[-2 ;;]]]
 ]
```

Step 4: Calculate the estimates of the peak concentration. See steps 4 and 5 of the ideal system.

```
niStep4[v_, u1_, u2_, u3_, u4_, cs1_, cs2_] :=
 With[{raf = u3 - u4}, {raffinateFlow1 / (εb * crossSection * raf), idealStep1[feedConcentration][[2]]}]
```

Step 5: Check the peak concentrations for convergence

```
niStep5[cpOld_, cpNew_, vel_] := With[{tol = Abs[cpNew - cpOld]},
  If[ {vel, cpNew}          And @@ Thread[tol < 0.001]
  [ niStep5[cpNew, vel]  True
 ]
```

With a velocity estimate. When we have an estimate of cp and velocity, we can calculate a new velocity estimate using step 3. The new estimate is used to calculate a new cp estimate using step 4.

```
niStep5[cp_, vel_] := Block[{newVel, newCS = niStep7[cp]},
  newVel = niStep3[vel, newCS];
  niStep5[cp, niStep4 @@ newVel ~Join~ newCS, newVel]
 ]
```

Without a velocity estimate. The velocity is calculated by passing the concentration only to step 3, which causes step 3 to invoke the ideal design. This results in a cp estimate which is immediately used in an iteration. This means that the ideal cp is not compared against the non-ideal cp, a potential inefficiency

```
niStep5[{cs1_, cs2_}] := Block[{vel = niStep3[{cs1, cs2}], cpNew },
  cpNew = niStep4 @@ (vel ~Join~ {cs1, cs2});
  niStep5[cpNew, vel]
 ]
```

Step 6: Calculate γ_+ and γ_- from $C_{peak,1}^{(1)}$ and $C_{peak,2}^{(1)}$

```
niStep6[cp1_, cp2_] := With[{γ+ = (a2 - a1 - a1 b2 cp2) / (a2 b1 cp2), γ- = (a1 b2 cp1) / (a1 - a2 - a2 b1 cp1)},
  {γ+, γ-}
 ]
```

Step 7: Calculate $C_{s,1}^{(1)}$ and $C_{s,2}^{(1)}$ from γ_+ and γ_-

```
Block[{cs1, cs2, cs1New, cs2New, denominator},
 denominator = a1 a2 b1 b2 (γ+ + γ-) + a1^2 b2^2 + a2^2 b1^2 γ+ γ-;
 cs1New = Simplify[ a2 b1 γ+ γ- (a1 - a2) / denominator ];
 cs2New = Simplify[ a1 b2 (a2 - a1) / denominator ];
 newEstimateEquations = {cs1New, cs2New} /. {γ+ → γ1, γ- → γ2}
]
```

$$\left\{\frac{(a_1 - a_2)\, a_2\, b_1\, \gamma 1\, \gamma 2}{(a_1\, b_2 + a_2\, b_1\, \gamma 1)\, (a_1\, b_2 + a_2\, b_1\, \gamma 2)}, -\frac{a_1\, (a_1 - a_2)\, b_2}{(a_1\, b_2 + a_2\, b_1\, \gamma 1)\, (a_1\, b_2 + a_2\, b_1\, \gamma 2)}\right\}$$

```
niStep7[γ1_, γ2_] := Evaluate[newEstimateEquations];
niStep7[{cp1_, cp2_}] := niStep7 @@ (niStep6 @@ {cp1, cp2})
```

Step 8: Calculate $C_{peak,1}^{(0)}$ and $C_{peak,2}^{(0)}$ from $C_{s,1}^{(0)}$ and $C_{s,2}^{(0)}$

```
niStep8[cs1_, cs2_] := idealStep1[{cs1, cs2}]
niStep8[{cp1_, cp2_}] := niStep8 @@ niStep7[{cp1, cp2}]
```

Step 9: Calculate δ_2^{I} δ_1^{II} δ_2^{III} and δ_1^{IV}

```
niStep9[{cs1_, cs2_}, {cp1_, cp2_}] := Block[{δ12, δ21, δ32, δ41},
  δ12 = ep + (1 - ep) * a2;
  δ21 = ep + (1 - ep) a1 / (1 + b2 cp2);
  δ32 = ep + (1 - ep) a2 / (1 + b1 cs1 + b2 cs2);
  δ41 = ep + (1 - ep) a1 / (1 + b1 cp1);
  Evaluate[{δ12, δ21, δ32, δ41}]
 ]


niStep9[{cs1_, cs2_}] := niStep9[{cs1, cs2}, niStep8 @@ {cs1, cs2}]
```

System Parameters

These parameters must be known or estimated beforehand. To change a value, simply type in a new number, then press shift+enter

```
particleRadius = Quantity[.027, "mm"];
ϵp = 0.55;
ϵb = 0.2;
ϵt = ϵb + ϵp (1 - ϵb);
diffusivity1 = Quantity[0.0007, "cm^2/s"];
diffusivity2 = Quantity[0.0007, "cm^2/s"];
a1 = .448;
a2 = 1.44;
b1 = Quantity[.0102, "L/g"];
b2 = Quantity[.0354, "L/g"];
zoneLength = Quantity[10, "cm"];
columnDiameter = Quantity[1, "cm"];
dispersion1 = Quantity[0.1, "cm^2/min"];
dispersion2 = Quantity[0.1, "cm^2/min"];

filmMTC = Quantity[2.25, "cm/min"];

particleRadius = QuantityMagnitude[UnitConvert[particleRadius, "m"]];
diffusivity1   = QuantityMagnitude[UnitConvert[diffusivity1, "m^2/s"]];
diffusivity2   = QuantityMagnitude[UnitConvert[diffusivity2, "m^2/s"]];
b1             = QuantityMagnitude[UnitConvert[b1, "m^3/kg"]];
b2             = QuantityMagnitude[UnitConvert[b2, "m^3/kg"]];
zoneLength     = QuantityMagnitude[UnitConvert[zoneLength, "m"]];
columnDiameter = QuantityMagnitude[UnitConvert[columnDiameter, "m"]];
dispersion1    = QuantityMagnitude[UnitConvert[dispersion1, "m^2/s"]];
dispersion2    = QuantityMagnitude[UnitConvert[dispersion2, "m^2/s"]];
filmMTC        = QuantityMagnitude[UnitConvert[filmMTC, "m/s"]];

diff           = {diffusivity1, diffusivity2};
ip1            = {a1, a2};
ip2            = {b1, b2};
disp           = {dispersion1, dispersion2};
```

Unit Conversions

Unfortunately, leaving units in the equations drastically reduces the performance of the entire algorithm. Therefore, the dimensional units are converted to SI and then stripped of their units.

Design Targets

These parameters must be specified, it seems that the final design is optimized by changing these values

```
yieldTarget = .99;
feedConcentration1 = Quantity[5, "g/L"];
feedConcentration2 = Quantity[5, "g/L"];

feedConcentration1 = QuantityMagnitude[UnitConvert[feedConcentration1, "kg/m^3"]];
feedConcentration2 = QuantityMagnitude[UnitConvert[feedConcentration2, "kg/m^3"]];

feedFlowrate = Quantity[.5, "mL/min"];
feedFlowrate       = QuantityMagnitude[UnitConvert[feedFlowrate, "m^3/s"]];

feedConcentration = {feedConcentration1, feedConcentration2};

convergenceTolerance = 0.00001;
```

Results

```
idealDesign = .
idealDesign[{cs1_, cs2_}] := With[{converged = step6[{cs1, cs2}]},
  (step3@step8[converged] ~Join~ step8[converged])]

niStep5[feedConcentration]
```

```
{{0.105848, 0.67422, 0.446102, 0.446633, 0.389319}, {0.0458193, 4.62984}}
```