DISCONTINUITIES IN MATHEMATICAL MODELLING: ORIGIN, DETECTION AND RESOLUTION

Tareg M. Alsoudani

A thesis submitted for the degree of Doctorate of Philosophy of
University College London

Department of Chemical Engineering
University College London
London WC1E 7JE

March 2016

I, Tareg M. Alsoudani confirm that the work presented in this thesis is my own. Where information has been derived from other sources, I confirm that this has been indicated in the thesis.

Abstract

When modelling a chemical process, a modeller is usually required to handle a wide variations in time and/or length scales of its underlying differential equations by eliminating either the faster or slower dynamics. When compelled to deal with both and simultaneously simplify model structure, he/she is sometimes forced to make decisions that render the resulting model discontinuous.

Discontinuities between adjacent regions, described by different equation sets, cause difficulties for ODE solvers. Two types exist for handling discontinuities in ODEs. Type I handles a discontinuity from the ODE solver side without paying any attention to the ODE model. This resolution to discontinuities suffer from underestimating the proper location of the discontinuity and thus results in solution errors. Type II discontinuity handlers resolve discontinuities at the model level by altering model structure or introducing bridging functions. This type of discontinuity handling has not been thoroughly explored in literature.

I present a new hybrid (Type I and Type II) algorithm that eliminates integrator discontinuities through two steps. First, it determines the optimum switch point between two functions spanning adjacent or overlapping domains. The optimum switch point is determined by searching for a “jump point” that minimizes a discontinuity between adjacent/overlapping functions. Two resolution approaches exist. Approach I covers the entire overlap domain with an interpolating polynomial. Approach II relies on a moving vector to track a function trajectory during simulation run. Then, the discontinuity is resolved using an interpolating polynomial that joins the two discontinuous functions within a fraction of the overlap domain.

The developed algorithm is successfully tested in models of a steady state chemical reactor exhibiting a bivariate discontinuity and a dynamic Pressure Swing Adsorption Unit exhibiting a univariate discontinuity in boundary conditions. Simulation results demonstrated a substantial increase in models' accuracy with a reduction in simulation runtime.

Dedication

To my father who I still feel his positive presence after he passed away 27 years ago,

To my mother who taught me how to crave my way through difficulties,

To my wife Amani for the valuable support, infinite love and cheerful encouragement she provided throughout my study time, and

To my children Ziyad, Ludan, Siba and Joud whom I haven't had much time to spend with while studying for this degree.

Acknowledgement

Very special thanks to my advisor, Professor I.D.L. Bogle, for his continuous support, guidance and above all patience during my study at UCL. His attitude towards explaining points that I missed without pointing them, pushing me when feeling exhausted, and being patient while his ideas are still to be digested in my mind is unforgettable. I learnt so much by interacting with him through my study at UCL.

Contents

List of Figures

List of Tables

Chapter 1:

Introduction

Chemical Engineering is one of the most versatile disciplines in science. Its stamp is sensed in almost every aspect of our life from the fuel that drives our cars to the cement that builds our homes and to pesticides that remove harmful bacteria and insects from agricultural products, etc. The current chemical engineering practice covers wider areas than it used to be in the old days when the discipline was just shaping. Chemical engineers are now contributing to areas such as design of integrated circuits and production of composite materials.

In most of these disciplines, experimenting with a product to improve its quality or reduce production costs comes at a cost. Sometimes the cost is so high that plant managers will prohibit engineers from making any changes to an existing process unless or until it is bullet-proofed that these changes will cause no harm to the plant production. Even when plants are green built, companies resort to old practices that are proven to work over the uncertainty that accompanies new innovations. Of course, such practices, although acceptable and sometimes appreciated, hinder development. To overcome difficulties associated with adopting newly developed practices, engineers resort to either the use of pilot plants that mimic current operating practices or to the use of mathematical models that simulate the behaviour of the system.

Pilot plants, when properly built, constitute a very effective method to experiment with a small model of an existing process, optimize or completely alter it to make the same product or redesign it to produce a better product. However, in general, pilot plants are expensive to build and operate. Their cost is sometimes prohibitive to justify their construction.

The other route to prove the feasibility of a new idea would be to construct a mathematical model that resembles the process to be tested whether an operating one or just being newly built. This route is normally less expensive than building pilot plants. It is also not uncommon that successful simulation results justify the construction of a pilot plant.

In order for a mathematical model to be useful, it needs to serve a purpose [Cameron et al, 2005]. Serving the purpose requires a balance between the level of model detail and its accuracy. Detailed models would always be preferable if it were not to the fact that they take longer time to build and more time to test and troubleshoot. Thus, a compromise is usually struck between model accuracy and its level of detail. This compromise is achieved through the use of simplified models that only address the main contributing phenomena to a process while either ignoring or simplifying models of non-core phenomena. An Inclusion/exclusion of a certain phenomena into/from a mathematical model is both scientific and judgemental.

When modelling dynamics of slow processes, faster dynamics that occur below a specific time scale are ignored. Similarly, when modelling faster dynamics, slow dynamics occurring beyond a certain time scale are ignored. For example, when modelling ecological systems, scientists seldom care about the fast changes that are happening

within a tissue of a living species. Similarly, when modelling the cellular activity of a human body, human life span is seldom included in such models.

Many similar examples exist in chemical engineering. For example, when modelling flow distribution networks between several plants, the modeller usually ignores modelling of smaller plant constituting equipment such as pumps and valves because they exits at a lower detail level. Also, modellers who model industrial reactors are usually not concerned with including equations that model molecular level dynamics and vice versa.

There are several reasons behind excluding or approximating models resembling non-core phenomena:

1. The time and effort used to build such models might not be justifiable considering the added accuracy. New developments in multi-scale modelling might reduce the time required to build such models. However, this approach to modelling is still at its infancy.

2. Computational power required to run such models might not be justifiable. Development of faster computers might resolve the required computational power for today's produced models. However, with advances in computational power, scientists are usually tempted to move from simplified models to more rigorous ones, increasing the demand for more computational power.

Until the above mentioned problems are resolved, scientists will almost always be forced to simplify models by excluding non-core phenomena. However, the line that is drawn between core and non-core phenomena is itself a blurred one. Simulation results deviate from accuracy when important phenomena are ignored or not properly modelled in the

name of simplicity.

Nature can be thought of as a sequence of numerous continuous events. Studying nature as a whole is virtually an impossible task. That's why scientists prefer encapsulating selected pieces of a system before studying them in a controlled environment. To extend the usability of such experiments, scientists fit the results obtained from such experiments to equations that are preferably derived from basic principles. When it is not possible to formulate an analytical equation to describe a certain phenomena, scientists resort to fitting results to empirical or semi-empirical formulas. Regardless of the origin of the fit, the resulting equation only resembles the generated output within a specified accuracy. More experiments at different controlled conditions lead to generating different formulas with differing accuracies.

These scientific practices lead to differing formulas to calculate a certain property at different conditions. Such situations leave the modeller no choice but to use two differing formulas to model the behaviour of a particular phenomena that extends between the boundaries provided by two differing formulas. The model switches from calculating the property using one formula to the other through the use of conditional statements. If a condition is met, the model uses one formula. Otherwise, it uses the other one.

This direct use of conditional statements raises what is referred to in mathematics as a “jump discontinuity”. Such discontinuities raise difficulties when solving mathematical models. Handling of a discontinuity is a solver dependent problem. Some solvers reinitialize model equations while others generate bridging interpolating functions. This means that for the same discontinuous model, different solvers will probably produce different model outputs. The lack of generality when addressing such a problem raises a

question about the accuracy of one provided solution over the other.

A mathematical “removable discontinuity” is generated when the formulas at both sides of the conditional statement do not cover the full range of their respective independent variables. Because of the current modelling practices, none of the available modelling languages or their respective solvers is able to detect such discontinuities. In this work, I illustrate how a better modelling practice reveals such discontinuities. I will also provide means to resolve them.

Sometimes, reinitialization is inevitable, mainly because of restrictions imposed by current modelling practices. However, how much information is lost because of reinitialization? and whether there is a better solution that avoids reinitialization? are two questions that remain unanswered.

The first objective of this work is to prove the inaccuracy of some of the practices adopted with simplified models. In particular, the focus is devoted to the way a simplified model behaves when crossing two adjacent domains possessing different modelling equations. The second objective is to provide a better solution to this problem that requires minimum intervention from the modeller.

I will provide a brief introductory to modelling in Chapter 2. I will start by defining what is meant by a model and provide a brief history of modelling. Then, I will discuss model development and highlight how assumptions emerge during the modelling process. A brief introductory to error analysis will also be provided in this chapter. Due to their importance in simulation of mathematical models in general and to this work in particular, a small section is devoted to integrating stiff mathematical models and integrator variable step sizing.

In Chapter 3, I survey the available approaches to handling discontinuities in mathematical models. I classify the approaches into two types and survey current practices of each type separately.

Chapter 4 focuses on the encountered discontinuities in the models that are constructed to prove the novel concepts outlined in Chapter 5. A particular emphasis are drawn to the way I modelled the Pressure Swing Adsorption (PSA) column. The PSA column model is built with an objective in mind to build a model that includes all possible steps occurring in today's operating PSA columns. Then, an MINLP optimizer would be built on the top of the model to determine the optimum operating conditions of a PSA unit based on a particular feed and an objective function. Integer parameters in the optimization include the minimum required number of PSA columns in addition to elimination or inclusion of some steps. The maximum number of pressure equalization steps is also included as integer optimisation parameter.

In Chapter 5, I discuss a generic methodology to handle discontinuities in mathematical models. I start by introducing the concept and illustrate how it applies to one-dimensional systems. Then, the concept is expanded to cover multi-dimensional systems.

Lastly, Chapter 6 will demonstrate how the ideas developed in Chapter 5 apply to the complex models constructed in Chapter 4.

CHAPTER 2:

An Overview of Modelling with Emphasis on Mathematical Models

---

The purpose of this chapter is to provide the reader a brief background on modelling. Through the discussion, I shed light at the definition of a model in its general and restricted forms. Then, I follow it by a brief introductory to the history of modelling. The core ideas behind the development of mathematical models are explored in the third section. I also discuss why model assumptions originate and their implications on the numerical integration of the model. I also briefly discuss the sources of numerical errors associated when integrating an ODE system and how variable integration step-size contribute to increased solution accuracy. Last, I briefly introduce stiff systems and how they're specially handled through the use of implicit integration methods.

2.1. Definition of a Model

According to [Schichl, 2004], the word modelling originated from the Latin word *modellus*. Modellus refers to the regular way humans cope with reality. [Ritchey, 2012] states that various kinds of models are used in all aspects of science in ways that make it difficult to agglomerate the definition of modelling into a clear and concise statement. However, he points out that two distinguishing criteria stand behind each model. First, a model should have more than one dimension or as he puts it: *mental construct*. Each dimension should support ranges of values or states. Second, a relationship must exist between model dimensions or their respective ranges.

In earlier papers, an extra restrictive criterion on the definition was imposed, namely: connections between dimensions should be identified on the basis of connections between their respective value ranges. However, this additional criterion rules out some of the classic models such as influence diagrams and analytical hierarchy models. Such models do not possess direct relationships between the values of their dimensions. In fact, the dimensions of some of these models are not defined as they are treated as black-boxes. [Ritchey, 2012] also claims that by the above definition, he relaxed his earlier criterion for defining a model from those presented in his earlier papers ([de Waal and Ritchey, 2007], [Ritchey, 2011]).

[Frigg and Hartman, 2012] point out that models refer to a variety of *things*. They named physical and fictional objects, descriptions and equations as examples. Below is a brief description of each of these *things:*

1. Fictional Objects: Models are also constructed to represent fictional objects and hence named fictional models. Examples of this class include the Bohr model of

an atom, a frictionless moving object, ideal collision of billiard balls, etc. Such models only exist in the mind of the scientist and they don't need to be physically realizable. In this class, the model becomes the object.

2. Physical objects: Models are constructed to model physical objects. The class of models that falls into this category covers all physical entities. In such instances, the model serves as a representation of something else. Wooden models of vehicles, planes, ships are examples of models that fall into this category. However, interestingly, [Frigg and Hartman, 2012] points out that some living creatures can be and are looked at as models to other creatures. Such an analogy is very evident in life sciences where animals such as rats and monkeys are used as models to understand human reactions to certain internal/external influences. Science refers to such models as material models.

3. Descriptions: Some scientists think of models as a stylized description of the objects under study [Achinstein, 1968][Black, 1962]. Each model is assumed to be uniquely identified with a description. However, this unique identification raises a contradiction. If the description is simplified, would it still be representative of the same model? Would it represent a different model? If a model can be uniquely identified with a description A, then any other identifying description of the same model would have to be connected to a different model. Thus, models cannot be equated with descriptions as the relationship is not one-to-one based.

4. Equations: Some scientists indulge the idea that models are equations. This view of models also suffers from the same drawback of treating models as descriptions.

[Frigg and Hartman, 2012] also point out that models can be constructed as a combination

of these elementary constructs. Thus, they define models as representations of objects. Represented objects can be either real (representing phenomena) or mere theories. A model can even extend further to include a combination of a partial representation of reality and a posed theory [Morgan, 2001].

[Schichl, 2004] provides the following definition: "*A model is a simplified version of something that is real*". [Hangos and Cameron, 2001] follow a similar path when they define a model as an imitation of reality. The term *real* implies that the fictional models, discussed earlier, cannot be considered as models in these definitions. This restricted form of the definition undermines the role of fictional models into the development of science.

Looking at the above introductory concepts, one can easily deduce that a clear and concise definition that encapsulates all types of models is difficult to construct. [Ritchey, 2012] clearly articulates this problem:

> "*What is, and what is not, to be considered a scientific model is a matter of convention, as long as we make our*[1] *rules clear and we apply them consistently*"

2.2. Brief History of Modelling

Ancient cavemen, and cavewomen, paintings are evident examples of humans early use of abstractions to represent objects. Paintings are considered as models crudely representing reality whether that reality is an event, a sequence of events (story) or a mere representation of a number. However, according to [Schichl, 2004], breakthroughs in modelling were introduced by cultures of the Ancient Near East and Ancient Greece. [Schichl, 2004] claims that mathematical models can be traced back to ancient

1 As per the author, it has been mistakenly scripted as "are" instead of "our" in the original paper.

civilizations in Babylon, Egypt, and India. These cultures used mathematical models to better organize and advance their life. In particular, algorithmic mathematics was used to solve problems related to irrigation, tax payments, construction, etc.. [Schichl, 2004]

Deductive reasoning emerged with developments in philosophy and its interaction with mathematics. This development gave rise to the seeds of mathematical theory in the Hellenic era. Thales of Miletus (624-546 BC), a pre-Socratic Greek philosopher, started the use of geometry to analyse reality. This introduction of geometry into analysis of reality facilitated the development of pure mathematics as a science that is independent from its applications [Kallrath, 2004].

Succeeding Thales, Pythagoras of Samos (570-495 BC) is known as the first pure mathematician basing his work on Thales. In the 300 years to follow, philosophers such as Aristotle (384-322 BC) and Eudoxos (408-355 BC) added more contributions to the science of mathematics. Climax was reached in 300 BC by Euclid of Alexandria (Mid 4th century- Mid 3rd century BC). He scripted a collection of books that contained most of the available mathematical knowledge available at his time. *The Elements* was the title of the collection. *The Elements* contained the first concise axiomatic description of geometry. It also included a treatment on number theory among other subjects. *The Elements* remained as a classic text for teaching mathematics for hundreds of years to follow. Around 250 BC, the theories in The Elements were used by Eratosthenes of Cyrene (276-194 BC) to calculate distances between Earth and Sun and between Earth and Moon. Eratosthenes of Cyrene is claimed to be the first applied mathematician [Kallrath, 2004].

By 150 AD, a mathematical model describing the solar system with circles and epicircles to predict the movement of the sun and the surrounding planets was developed by Ptolemy (100-170 AD). The accuracy of the model ensured its application for years that

followed. In 1619, Johannes Kepler (1571-1630 AD) devised a better and simpler model to predict planetary motion. Newton and Einstein enhanced Kepler's motion model. The final model is still in use until today.

Development in models and model building methods was not only evident in Europe. Similar modelling methods were independently developed in countries such as China, India and Islamic countries such as Persia. Abu Abd-Allah Ibn Musa Al-Khwarizmi (780-850 AD) was famous of his work on algorithms. In fact, the word algorithm was derived from his last name. He authored a collection of books on what was known at the time as Indian numbers (known now as Arabic numerals). His book titled "Al-kitab Al-Mukhtasar fi Hisab Al-gabr wa Al-muqabala" is rich in mathematical models and problem solving algorithms. The term *Algebra* was derived from the title of this book [Kallrath, 2004].

After the decline of Greek civilization, Leonardo da Pisa Fibonacci (1170-1250 AD) is considered the first great western mathematician. Fibonacci realized the advantage of Indian numbers over their Roman counterparts. He used the algebraic methods recorded in Al-Khwarizmi books to succeed as a merchant. In 1202, he authored his book *Liber Abaci*. The book marked the introduction of the zero as a number to Europe [Kallrath, 2004].

To advance the use of visual models, artists started novel development in the principles of geometry. Giotto (1267-1336 AD) and Filippo (1377-1446 AD) were among the first to introduce the concept of perspective into visual models in Anatomy [Kallrath, 2004].

Although Diophant (201-285 or 215-299 AD) and Al-Khwarizmi made great contributions to algebra, it wasn't until Vieta (1540-1603 AD) that variables were systematically introduced into mathematics. Nevertheless, it took 300 more years to fully articulate and understand the role of variables in the formulation of mathematical theory.

Cantor (1845-1918 AD) and Russel (1823-1913 AD) were among the first scientists that contributed to variable formulations. Deriving laws of physics that described the principles of nature was the major driving force behind developments in modelling and mathematical theory. The introduction of models into economics came at a later stage [Kallrath, 2004].

Process Systems Engineering (PSE) has evolved as a science after the industrial revolution to advance problem solving techniques using models derived from many of the physical sciences and engineering disciplines. The motive behind this development is the growing trend to reduce complex physical behaviour to simple mathematical forms for easier process design. This motive has continued and increased after the Second World War. The development of faster computers, high level programming languages and advances in mathematical modelling have all lead to considerable progress in the area of Process Systems Engineering. [Hangos and Cameron, 2001]

2.3. Model Development

The great interest in model building and model use is because it is a means to gaining insight into the behaviour of systems, probing them, controlling them, and optimizing them. The process of model building and use is divided into four steps [Hangos and Cameron, 2001]:

1. Transforming a real problem into a mathematical model.
2. Solving the mathematical model.
3. Interpreting model output.
4. Using results in real world.

Models are built to serve a variety of functions. Examples include:

1. *Explaining Phenomena*: Most developed theories in physics fall into this category. Examples include : Newton's mechanics, thermodynamics, Einstein's theory of relativity, quantum mechanics, the Standard Model of particle physics, etc. Models in engineering follow a similar trend. Examples include models of distillation columns, fluid behaviour around objects, circuit analysis, channel hydraulics, etc.

2. *Making Predictions*: Models that are built to explain certain phenomenon can be further used to predict the behaviour of a system under certain conditions. The most obvious example falling into this category is weather forecast models.

3. *Decision Making*: quite a number of models are built to aid in decision making process.

The process of designing models begins with a goal in mind. The modelling goal specifies the intended use of the model. The modelling goal has a major impact on the level of detail and on the mathematical form of the model to be built. Models can be developed to test dynamic or steady state aspects of a system, to help in design problems and to address process control issues [Hangos and Cameron, 2001]. According to [Cameron et al, 2005], modelling objectives in current modelling practice are forgotten, implicitly considered, or remembered in a blurred manner at a later stage in the building cycle of the model. The lack of an explicit goal statement significantly affects the focus, task efficiency, model coherency and eventually might lead to termination of model cycle, especially in model conceptualization. Lack of explicit goals often results in a model that is not suitable for the stated purpose, consumes an enormous amount of time to develop and is either too simplistic or exhaustively complex for the required application. In their paper, [Cameron

et al, 2005] defined a modelling goal triplet

<<*Model*>> to/for <<*Application*>> of/from/for <<*System*>>

The short notation for this triplet is <<M-A-S>>. They also pointed out that decomposition of a modelling objective is not an easy task. Their means-end analysis for process modelling is focused on introducing a framework that permits the modeller to develop model structures that possess the required functionality to achieve the declared goals. Model functionality includes a model representation of the basic character of the system as well as the required functionality for the application area [Cameron et al, 2005].

The effort of setting up a detailed mathematical model for a chemical process is high due to the large variety of chemical process units and physical phenomena in addition to increasing requirements on the sophistication of models. To overcome this modelling bottleneck, considerable effort needs to be exerted into systemising process models, formalizing their representation, and developing knowledge-based software tools. [Bogusch et al, 2001] used conferences, industrial project meetings and a field study to collect requirements on modelling environments from a practitioner's point of view. Their findings are summarized as follows:

1. Support should be provided for development and storage of groups of models that serve a particular process need.
2. Interaction between modeller and modelling package needs to be transformed from equation level to knowledge level.
3. Support should be provided to maintain process models up-to-date with process changes.
4. Modelling packages should be capable of storing and retrieving modelling knowledge to be used to guide the model development process.

5. A library of predefined model building blocks of fine granularity should be supplied.
6. Automation of some stages of the modelling process. For examples, Knowledge propagation, documentation and report generation can be automated.
7. Models are more than equations. Model assumptions and limitations, degrees of freedom, model initialization, etc, should also be included in a model representation.

In their paper, [Bogusch et al, 2001] describe how ModKit has evolved as a process modelling environment to meet these needs.

Mathematical models can be classified in pairs, as (Mechanistic or Empirical), (Stochastic or Deterministic), (Lumped or Distributed), (Linear or Nonlinear) or (Continuous, Discrete or Hybrid) [Hangos and Cameron, 2001]. The approach to modelling a particular problem can also be classified. [Marquardt, 1996] classified modelling approaches in current commercial process simulators into two groups: block-oriented (or modular) and equation-oriented. In the Block-oriented approach, the main focus is to model at the flowsheet level. Process entities are described through block diagrams that are built from standard library of building blocks. The blocks and their connectivities model the behaviour of a process unit or part of it. Blocks are connected via signals representing stream flow of information, material and energy. Standard formats are used to construct each stream. Advantages of this modelling approach include ease of accessibility and use. Despite its great advantages, this approach is considered inadequate for supporting solutions of more complex problems. The reason is the lack of precoded models for various unit operations with adequate level of detail. Examples include multiphase reactors, membrane processes, polymer reactors and most units involving particulates. As

a result, expensive and time consuming model development for a particular unit is often needed during project work [Marquardt, 1996].

Equation-oriented modelling tools implement unit models and incorporate them in a model library through declarative modelling languages. Aspen+ equation-oriented modeller and Aspen dynamics (formerly SPEED UP) are examples of this approach. Languages developed in these modelling tools extensively support model implementation. However, users do not have the freedom of developing models from basic engineering concepts. In addition, support is lacking for appropriate design and documentation of the model library. Thus, the concept of validated model re-usability, by a group of engineers, for these types of models is impossible. Reinventing models becomes imperative. Models that are initially well developed deteriorate over time. [Marquardt, 1996].

The realised disadvantages in both approaches has excited researchers to look for better modelling approaches. The main aim is to ease model development and maintenance through developing model formulation capabilities, enhancing model reuse and adaptation as well as facilitating model's maintenance and documentation.

Recent developments have led to modelling languages that are more declarative (explicit and symbolic) and multilevel model based. These developments can generally be classified into four groups:

1. *Process Modelling Languages*: Although the fundamental concepts of these languages are similar to those of the generic modelling languages, they are designed from the start to address the specific issues of a selected application domain in the definition of the language. Examples include MODEL.LA or VEDA. In these languages, chemical engineering specific elements are included in

the language definition. It should be noted that VEDA is a basic language definition platform [Marquardt, 1996]. The language definition syntax is based on the use of Objects derived from Object Oriented programming concepts originating from computer science. Although originally developed to model chemical engineering phenomena, the structure of the language is generic enough to accommodate models from any scientific discipline. Currently, implementations are carried out using different software platforms. An example is the frame-based knowledge representation language FRAMETALK [Rathke, 1993] as well as the expert system shell G2 [Gensym, 1995] and the process-centred design environment PROART/CE [Dömges et al , 1996].

2. *Modelling Expert Systems*: the objective behind these modelling environments is to produce a sufficient process model from a formal description of the modelling problem that is initially introduced by a user with a minimum or no further interaction. Similar to all expert systems, the system should contain a knowledge base that is built on some formalism of a hybrid knowledge representation, an interface for knowledge acquisition, a description facility in addition to a discrete reasoning system that allows automatic model generation from the provided specification. The early attempts to model such a system constituted MODEX. In addition, MODASS exhibits some aspects of this general idea. After prototypes of both modelling languages were implemented and enhanced, they were suspended. PROFIT encompasses recent advances in expert systems. In PROFIT, the user supplies details of structural specification in addition to the phenomenological characteristics that comprehensively define the considered process abstraction. An inference engine, that is rule-based, automatically determines a set of balance

equations based on the supplied facts [Marquardt, 1996].

3. *Interactive (Knowledge-Based) Modelling Environments*: knowledge-based design environments or construction kits are built to enforce the traditional concept of elementary building blocks that result in a robust model. Various possible configurations can result from those elementary building blocks because of their generic structure that provides few restrictions on combined blocks. Instead of directly solving the problem, the system provides a variety of solution paths that a modeller can select from. Consequently, problem specifications are constructed side-by-side with the solution. There isn't, so far, a practically built system that complies with this idea. Nevertheless, Some of its concepts are found in MODASS or in knowledge-based user interface of DIVA [Bär and Zeitz, 1990].

4. *General modelling languages*: Examples of this group include DYMOLA, OMOLA , ASCEND or gPROMS. These languages can be looked at as the second generation of equation oriented simulation languages that can be traced back to the 1960s specification of CSSL [Augustin et al, 1967]. Their design supports hierarchical decomposition of complex models. This hierarchical decomposition facilitates model reuse and maintenance. All of these languages utilise concepts originated from Semantic Data Modelling [King and Hull, 1987] and Object Oriented Programming [Stephik and Bobro, 1986]. They exhibit structured representations of encapsulated submodels that are organized in terms of inheritance and aggregation hierarchies. The use of these languages is not restricted to chemical engineering applications. This is because the definition of the language is reduced to a relatively small number of generic elements [Marquardt, 1996].

The development of any software package that supports an engineering task requires a model conceptualization of the problem domain. This abstraction level should eventually reveal a reasonable process modelling methodology that well suits computer implementation. This methodology should include:

1. Models' decomposition and identification of elementary modelling objects that can be combined to form a coherent model of virtually any chemical process.
2. Generic modelling algorithms that support building models from the ground up, maintenance and modifications of existing models to serve the requirements of a new context [Marquardt, 1996].

2.4. Assumptions in Mathematical Model Building

Mathematical models constitute a class of models that are built based on mathematical equations to study the behaviour of an existing system under different scenarios or to study the effect of pushing the system close to or beyond its known boundaries.

In general, equations in a mathematical model are divided into conservation laws and constitutive equations [Hangos and Cameron, 2001]. Conservation laws are equations that restrict and align the behaviour of the model with the system it is presenting. When modelling, the differential variables belonging to this class of equations are called state variables as they determine the state of the system at any particular time or spatial instant. Integration routines usually integrate these variables from particular initial to final conditions, between predetermined boundary conditions, or a combination of initial and boundary conditions. Differential variables are assumed continuous in nature. However, discontinuities may occur in differential equations. Such discontinuities usually result from model formulation and its underlying assumptions. Let us illustrate this with an example.

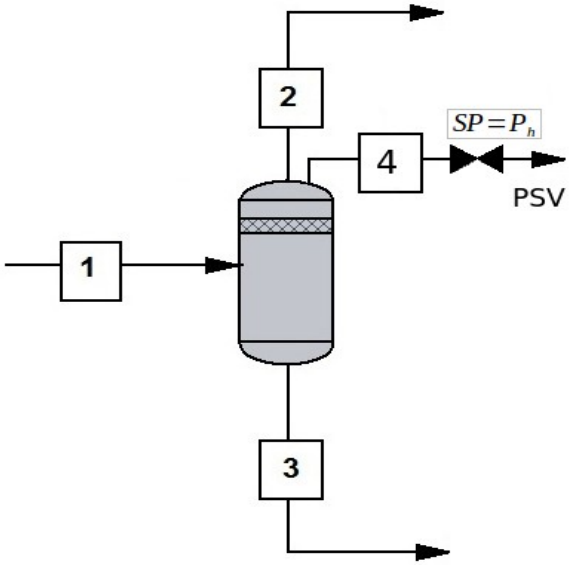


Figure 2.1 : A flash drum with a pressure safety valve.

Example 2.1 An over-pressurized column.

Let us draw a mass balance envelope around a simple flash drum that contains a pressure safety relief valve as illustrated in Figure . In normal process operating conditions, the pressure relief valve is closed since the pressure is lower than the set value of the relief valve P_h. *In such conditions, the overall dynamic mass balance around the flash drum can be written as:*

$$\frac{dm}{dt} = \dot{m}_1 - \dot{m}_2 - \dot{m}_3 \tag{2.1}$$

Once drum pressure reaches the pressure set by the PSV (P_h*), the mass balance will immediately shift to the form:*

$$\frac{dm}{dt} = \dot{m}_1 - \dot{m}_2 - \dot{m}_3 - \dot{m}_4 \tag{2.2}$$

This sudden change in the mass balance equation results in an explicit model discontinuity.

Conventional integration routines properly tackle this type of discontinuity mainly

because the discontinuity appears in the state variable. Such routines use an interpolating polynomial to bridge the gap between the two sides of the discontinuity. Some of modern integration routines (e.g. [gPROMS, 2012]) prefer re-initialization of variables over bridging with an interpolating polynomial. However, in such cases, bridging with an interpolating polynomial should arguably provide a more accurate solution than mere initialization. The increase in accuracy is attributed to the fact that an interpolating polynomial would implicitly assume that there is a spatial or temporal transition between the two adjacent sides of the discontinuity. Smooth transition more resembles reality regardless of the difference between the relative rates of change exhibited by system behaviour and the interpolating polynomial representing the transition over the discontinuity.

On the other hand, reinitialization assumes an instantaneous transition between the sides of the discontinuity. This instantaneous transition overlooks the smoothness of the system transition. In doing so, model behaviour information during transition is not captured. In addition, the use of an interpolating polynomial is computationally less exhaustive as I will prove in section 6.1. Reinitialization is computationally exhaustive because the integrating routine does not only reinitialize the discontinuous variable or equation. Rather, it reinitializes the entire system of equations. Thus, computational effort is directly proportional to model size when reinitializing. Such computational deficiency mandates the use of more powerful computing platforms as the size of the model increases.

Let us turn our attention to constitutive equations. These equations are formulated and added to conservation laws (equations) in order to determine values of particular constants/variables appearing in the differentiable equations. The reason behind the need

for such equations lies in the fact that when conservation balances are written, few of their underlying terms require either definition or calculation [Hangos and Cameron, 2001]. Constitutive equations are, unlike balance equations, particular to the system under study. They define the characteristics of a particular system and to some extent differentiate it from other systems [Aris, 1999]. Examples of model variables that can be calculated using constitutive equations include the density of a two-phase fluid in a crystallizer, thermal conductivity of a substance, the overall heat transfer coefficient of a particular system, stresses within a rock, etc. These properties are usually functions of the state of the system (temperature, pressure, flow and composition) in addition to other system specifications.

In some cases, the constitutive variable may reduce to a simple constant such as the resistance in a simplified electrical circuit. However, in other cases, equations may extend beyond that. The complexity of calculating a constitutive variable in a conservation equation is usually a direct function of the accuracy required for the value of that variable. Thus, in general, more accurate values require more complex equations.

To overcome the need to implement high accuracy calculations over the entire range of the property to be estimated, scientists and engineers resort to formulating relatively simplified equations that calculate the value of a constitutive variable to a certain degree of accuracy. Such equations are based on theoretical grounds, experimental data or a combination of both. Regardless of the origin of the calculation method, it is almost always associated with a domain at which it can be applied with some confidence, a minimum acceptable accuracy and few simplifying assumptions.

Extrapolating the use of the calculation method beyond its applicability domain results in loss of either confidence or accuracy of the reported values, if not resulting in both. To

overcome this barrier, researchers opt to define an equation or a set of equations that satisfy minimum acceptable accuracy for each of the domains a simulation model might run into. This approach works well within the applicability domain of the equation. However, it introduces another problem when simulation moves from the applicability domain of one equation (or correlation) to that of another. The problem is illustrated in Example 2.2.

Example 2.2 Viscosities of liquid and vapour benzene:

The viscosities of saturated pure vapour and liquid benzene against the temperature are plotted in Figure 2.2. Saturated liquid viscosity is plotted on the left y-axis while saturated vapour viscosity is plotted on the right axis. The saturated liquid viscosity at any given temperature is roughly about thirty times that of the saturated vapour. A modeller can account for the value of the viscosity at any given phase through an expression such as:

```
if Phase = Vapour
    Viscosity = Vapour Viscosity
else if Phase = Liquid
    Viscosity = Liquid Viscosity
endif
```

A simulation model involving a transition between the two phases will most probably run into a discontinuity at the phase transition point because of the large differences between the viscosity values of the two phases.

Since the origin of constitutive equations differ from one applicability domain to the other, it becomes natural to realise that these equations will most probably violate continuity at the intersecting points of their applicability domains

although they are calculating the value of the same property. Such a discontinuity introduces a problem when a simulation integration routine moves from one domain to an adjacent one exhibiting different equations to calculate the same variable.

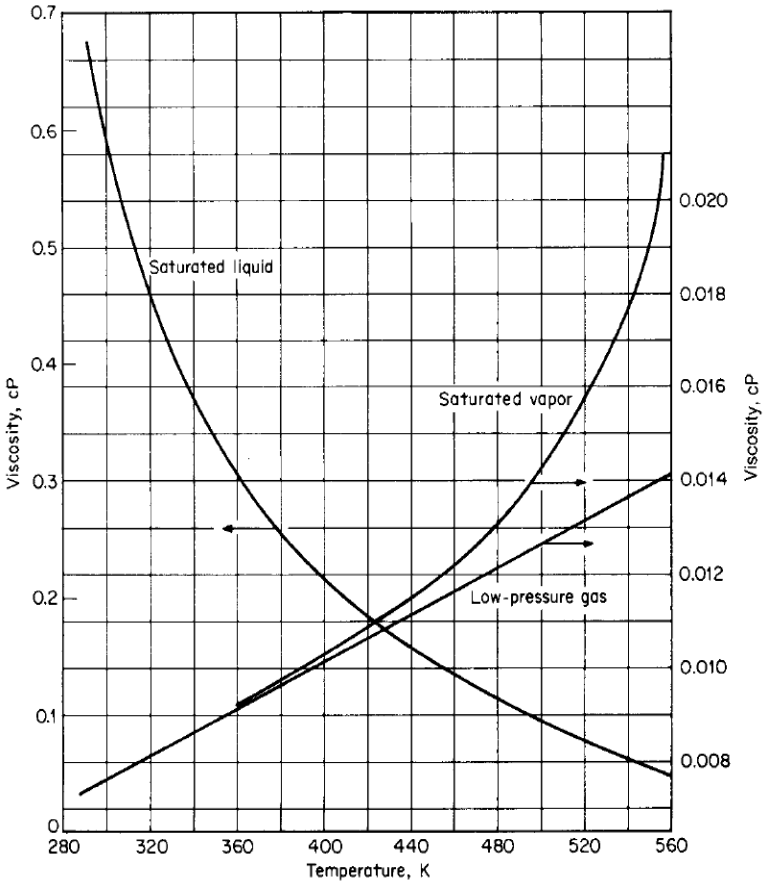


Figure 2.2 : Vapour and liquid benzene viscosities as functions of temperatures. [Reid et al, 1987]

As discussed earlier, conventional integration routines use an interpolating polynomial to resolve the discontinuity. However, conventional integration routines cannot detect the exact location of the discontinuity. They rather detect the discontinuity in the state variable resulting from a discontinuous constitutive equation. Since discontinuity is detected at the state variable level, the bridging interpolating polynomial is constructed at the state variable level. Thus, the resulting interpolating polynomial is not representative

of system behaviour any more. Such a resolution leads to:

1. a diversion of the simulation from its original trajectory. This diversion creates an error and reduces confidence in simulation results post discontinuity. The error accumulates with every passage through a constitutive-equation discontinuity. What worsens the situation is that the error is not calculated as it is passed undetected. At best, the modeller is merely notified of the existence of a discontinuity and its respective resolution.

2. a situation known in literature as a sticky discontinuity. A sticky discontinuity happens when the change in the simulation trajectory, introduced by the interpolating polynomial, lands the model at a pre-discontinuity point leading to a regeneration of the same polynomial and a re-landing at the same pre-discontinuity conditions. The situation continues until the integrating routine surrenders after a certain preconfigured number of iterations.

Modern solvers such as [gPROMS, 2012] reintialize the entire model equation when such a discontinuity is encountered. Reinitialization in this situation is better than the use of an interpolating polynomial since it, at least, preserves the structure of the model and avoids sticky discontinuities. However, the aforementioned reinitialization problems still exist and a proper solution remains to be found.

A third form of a discontinuity appears in a model when a sudden change exists, not in model equations but in their respective boundary and/or initial conditions. Examples of such discontinuity include a sudden open/closure of a motor-operated valve, the start-up or shut-off of a pump or a sudden reroute of flow network. The discussion is best explained through an example.

Example 2.3 Pressurizing and de-pressurizing a vessel.

In this example, I will model a simple gaseous pressurization of a vessel through one end and its immediate depressurization through the other end. The interest is focused on concentration and velocity profiles throughout the vessel over space and time. Thus, I will discretize the axial dimension of the vessel. Uniformity will be assumed in radial direction. To further simplify the problem, I will assume isothermal conditions and negligible pressure gradient. The differential component concentration of the system can be written as:

$$\frac{dc_i}{dt} = D_L \frac{d^2 c_i}{d z^2} - \frac{d(c_i u)}{dz} \tag{2.3}$$

Also, since no reaction or adsorption is occurring inside the vessel, the total concentration becomes a function of pressure only. Assuming an ideal gas behaviour:

$$C_t = f(P) = \frac{P}{RT} \tag{2.4}$$

Thus, velocity becomes a function of total concentration and its time derivative:

$$\frac{dv}{dz} = \frac{1}{C_t} \frac{dC_t}{dt} \tag{2.5}$$

To complete the problem specification, I need a function representing the change in vessel pressure with respect to time ($P = f(t)$). *An exponential form is presented in equation* (2.6)

$$P = P_{low} - (P_{high} - P_{low})\left[1 - e^{-M_p t}\right] \tag{2.6}$$

Since the component concentration balance is presented through a PDE

that is first order in time and a second order in space, I need to specify the initial conditions as well as the boundary conditions. For this example, The focus is devoted to boundary conditions of the PDE. Thus, initial conditions (feed component concentrations) can be arbitrary selected.

When pressurizing the vessel, the feed is introduced at one end (z=0) while the other is closed (z=L). The boundary conditions for the feed introduction end and the closed end during pressurization step are respectively outlined below:

$$-D_L \frac{\partial c_i}{\partial z}\Big|_{z=0} = u\Big|_{z=0}\left(c_i^f - c_i\Big|_{z=0}\right) \tag{2.7}$$

$$-D_L \frac{\partial c_i}{\partial z}\Big|_{z=L} = 0 \tag{2.8}$$

For the depressurization step, the respective boundary conditions are as follows:

$$-D_L \frac{\partial c_i}{\partial z}\Big|_{z=0} = 0 \tag{2.9}$$

$$-D_L \frac{\partial c_i}{\partial z}\Big|_{z=L} = 0 \tag{2.10}$$

Note how the boundary condition changes form from equation 2.7 to equation 2.9. Such a change creates a discontinuity in the mathematical formulation of the problem.

Almost all modelling literature treats discontinuities in boundary conditions similarly. Simply stated, no known integration routine can smoothly integrate over changing boundary conditions. Thus, almost all modelling languages allow modellers the flexibility to split a discontinuity in boundary condition into two separately treated problems. The

integration routine integrates over the first set of mathematical equations, stops integration, reinitializes model equations and continues integrating into the next set of equations.

As I stated earlier, although reinitialization overcomes the discontinuity, it comes at the cost of introducing an error into subsequent integration steps. In addition, it is computationally exhaustive as all system equations need reinitialization and not only the discontinuous set.

It appears from the above discussion that there is still a room to improve the accuracy and computational efficiency when integrating discontinuous functions whether the discontinuity occurs in the state variable, the constitutive equation or the boundary condition.

2.5. Numerically Integrating Mathematical Models and the Inherent Errors

In order to solve any mathematical model, it needs to be reduced to a set of ordinary differential equations (ODEs) before linking it to an integration routine (sometimes an integration routine is referred to as a solver). If a model contains higher order differential equations such as Partial Differential Equations (PDEs), the equations are reduced to a set of ODEs using readily available techniques in literature before passing the final system to the integration routine.

A typical relationship between the model and the solver, as implemented in most conventioal solvers, is represented int Figure 2.3. As illustrated in the figure, the main driver routine (Block *A*) is responsible for providing the initial conditions and the overall integration interval. This routine is almost always written by the model developer. Once information is passed to the ODE integrator (Block *B*), the integration routine initializes

integration and starts integrating between initial and final points defined by the main driver routine in a sequence of integration steps.

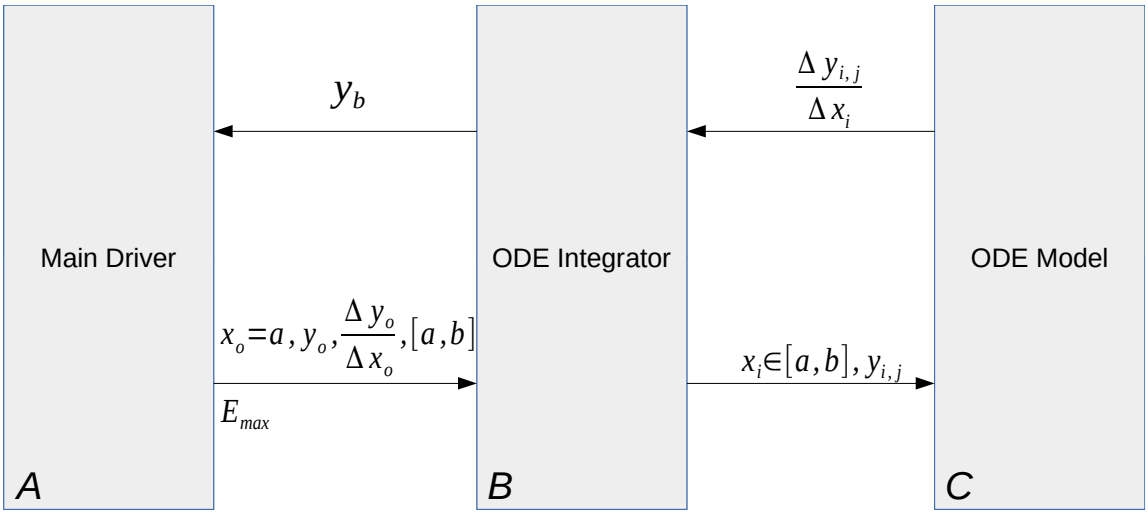


Figure 2.3 : A diagram illustrating the flow of information between entities of a conventional integration routine, its associated main driver and the model routine.

For each integration step *i*, the integration routine passes the current integration position (x_i), the values of the $y_{i,j}$ vector evaluated at x_i and the integration step size $h=\Delta x$ to the ODE model routine (Block *C*). The ODE model routine evaluates the $\Delta y_{i,j}/\Delta x_i$ and passes results to the integration routine. Once the integration routine receives a new set of $\Delta y_{i,j}/\Delta x_i$, it checks solution accuracy by one of the following methods:

1. Recalculating derivatives using x_i and $y_{i,j}$ vectors that correspond to a smaller h (normally half of the original one) while maintaining the integration algorithm.

$$\left| \frac{\Delta y_{i,j}}{\Delta x_i}\bigg|_{x_{x_i}+\Delta x_i} - \frac{\Delta y_{i,j}}{\Delta x_i}\bigg|_{x_{x_i}+0.5\Delta x_i} \right| < \epsilon \qquad \forall\, j \tag{2.11}$$

2. Computing the error using two different integration algorithms with the first (A) being more computationally efficient while the second (B) being more accurate. Both algorithms will integrate through a fixed integration step size $h=\Delta x$.

$$\left| \frac{\Delta y_{i,j}}{\Delta x_i} \right|_{x_{x_i}+\Delta x_i}^{A} - \frac{\Delta y_{i,j}}{\Delta x_i} \left|_{x_{x_i}+\Delta x_i}^{B} \right| < \epsilon \qquad \forall i \tag{2.12}$$

Regardless of the error calculation method, the integration step is accepted if the error is less than a specified tolerance ϵ and h is increased for the subsequent integration step. Otherwise, h is reduced and integration is repeated over the newly calculated h.

The difference between the two integral values, calculated using either of equations 2.11 or 2.12, constitutes the local error, or at least an approximate numerical representation of it. The inaccuracy that results from using equation 2.11 arises from the fact that integrating using any value $\bar{h}$, representing the magnitude of the halved step, that is less than h carries its own errors. An exact representation of the local error is only achievable when $\bar{h}$ approaches an absolute 0. Of course, the calculation then becomes computationally prohibitive. So, a compromise is usually struck between acceptable accuracy and computational efficiency.

The inaccuracy associated with using equation 2.12 as error evaluation criterion stems from the fact that the more accurate algorithm is not the exact solution to the integral. Thus, it also carries its own error within its computation. We are simply stating that a numerical solution is as good as the computing algorithm and, with an infinite computational power and/or highly accurate numerical solution algorithm, the numerical solution might reach the exact one.

Errors resulting from the use of a particular numerical algorithm can be reduced by deploying better numerical algorithms, increasing efficiency of certain existing ones or tightening solution error-tolerance criterion. The first two solutions are handled by the modelling language developer while the last one is handled by the modeller.

In addition to errors resulting from the use of a particular numerical algorithm, there is

another source of numerical error that is associated with machine precision. It is sometimes referred to as the *round-off* error. Each computing machine stores numbers to a finite precision. If the calculated number requires a precision that is more than what the machine can store, an error is introduced that is equivalent to the difference between the true numeric value and that stored by the machine.

[Cheney and Kincaid, 1999] state that round-off errors are negligible when integrating few steps. However, error magnitudes start playing an important role when integrating over hundreds to thousands of steps. The IEEE 754 double precision format, illustrated in Figure 2.4, stores a float number using 15-17 decimal figures (depending on the sign). This number representation significantly reduces errors associated with rounding-off.

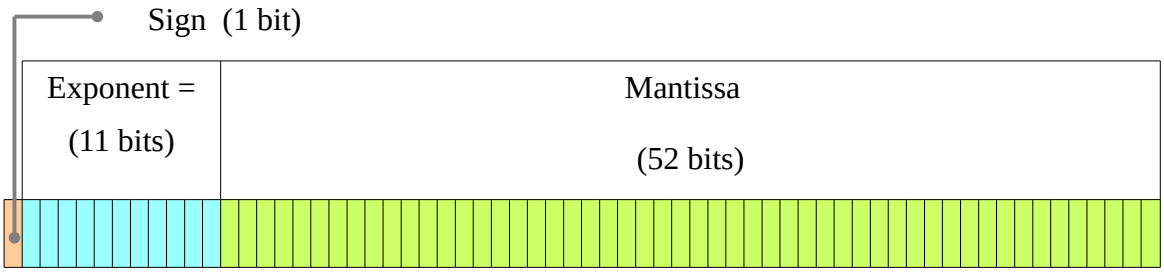


Figure 2.4 : The number of machine bits reserved for a double-precision variable as outlined by IEEE 754 standard

Another solution that overcomes machine precision limitations is rescaling of ODE variables. Sometimes, it is also called *normalization*. Basically, the ODE variables are transformed from their original domains to normalized ones. For example, let us assume that an integral of an ODE $y'(x) = f(x,y)$ is required to be carried over $x \in [a,b]$ with an initial condition $y(a) = g$ and a known y domain of $y \in [c,d]$. All variables and their respective domains can be normalized to fall within a range of [0,1]. For the independent variable x, the transformation will take the form $\bar{x} = (x-a)/(b-a)$ resulting in $\bar{x} \in [0,1]$. Similar transformation over the dependent variable y using $\bar{y} = (y-c)/(d-c)$, results in

$\bar{y} \in [0,1]$ and an initial condition of $\bar{y}(0) = (g-c)/(d-c)$.

The error that results from a one-time execution of the numerical algorithm is referred to as the *local error.* This error is the sum of the last two pre-mentioned errors over a single execution interval of a particular numerical algorithm. When integrating polynomial ODEs, the local error resulting from a single integration step can be easily calculated using Taylor's series expansion of the form:

$$E_L = \frac{f'(x_i, y_i)}{2!}h^2 + \frac{f''(x_i, y_i)}{3!}h^3 + \frac{f^{(4)}(x_i, y_i)}{4!}h^4 + ... \frac{f^{(n)}(x_i, y_i)}{n!}h^n \qquad (2.13)$$

The integer *n* in equation (2.13) corresponds to the order of the polynomial to be integrated since any derivatives beyond the *n*th derivative will be zeros as per polynomial definition. The calculation of the local error is more accurate when exact derivatives of (2.13) are available and computable. When these derivatives are not available, their numerical counterparts can replace them with a compromise on accuracy.

When a particular numerical algorithm is repeatedly executed to solve a particular numerical problem (as in ODE integration), the sum of the local errors introduced by a particular execution step in addition to errors introduced by previous executions is called the *Cumulative* or *Global error.* When the exact solution is available, for comparative purposes, the global errors is calculated as the difference between the exact solution and its numerical counterpart..

2.6. Stiffness and Stiff Mathematical Models

In this discussion, a particular interest is devoted to stiffness of ODE systems because discontinuities in ODEs originate at the boundaries of stiffness where conventional numerical integration methods do not apply. Conventional methods to resolving discontinuities in ODE systems are discussed in Chapter 3: .

A stiff system of equations is a system that inherently involves mixed, slow and fast, dynamics. The bigger is the difference between the fast and slow dynamics of a system, the stiffer is the system [Chapra and Cancale, 2002].

In numerical mathematics, stiffness is described as a phenomenon rather than a property of the system. This is mainly because there is no concise definition to stiffness. In addition to the description outlined earlier, here are few more definitions:

- An ODE system is considered stiff if the size of the integration step is defined by a stability criterion and not by solution accuracy.
- An ODE system is is considered still if explicit integration methods fail to integrate it or take longer time to integrate.
- A linear ODE system is stiff if all its associated eigenvalues posses negative real part, and the stiffness ratio (the ratio of the magnitudes of the real parts of the largest to smallest eigenvalues) is large.
- In general, An ODE system is considered stiff if the magnitudes of eigenvalues of its Jacobian matrix greatly differ.

In the vast majority of systems, the rapid changing dynamics are only evident in a fraction of the integration interval. Afterwards, the system behaviour is dictated by the slower dynamics [Chapra and Cancale, 2002]. For example, consider the ODE system:

$$\begin{aligned} \frac{dy_1}{dt} &= 1000 * (1 - y_1) \\ \frac{dy_2}{dt} &= 1 - y_2 \end{aligned} \tag{2.14}$$

with initial conditions $y_1(0) = y_2(0) = 0$. The analytical solution takes the form:

$$y_1(t)=1-e^{-1000t}$$
$$y_2(t)=1-e^{-t} \tag{2.15}$$

The behaviour of the system is plotted in Figure 2.5. Note how fast the response of y_1(t) compared to y_2(t).

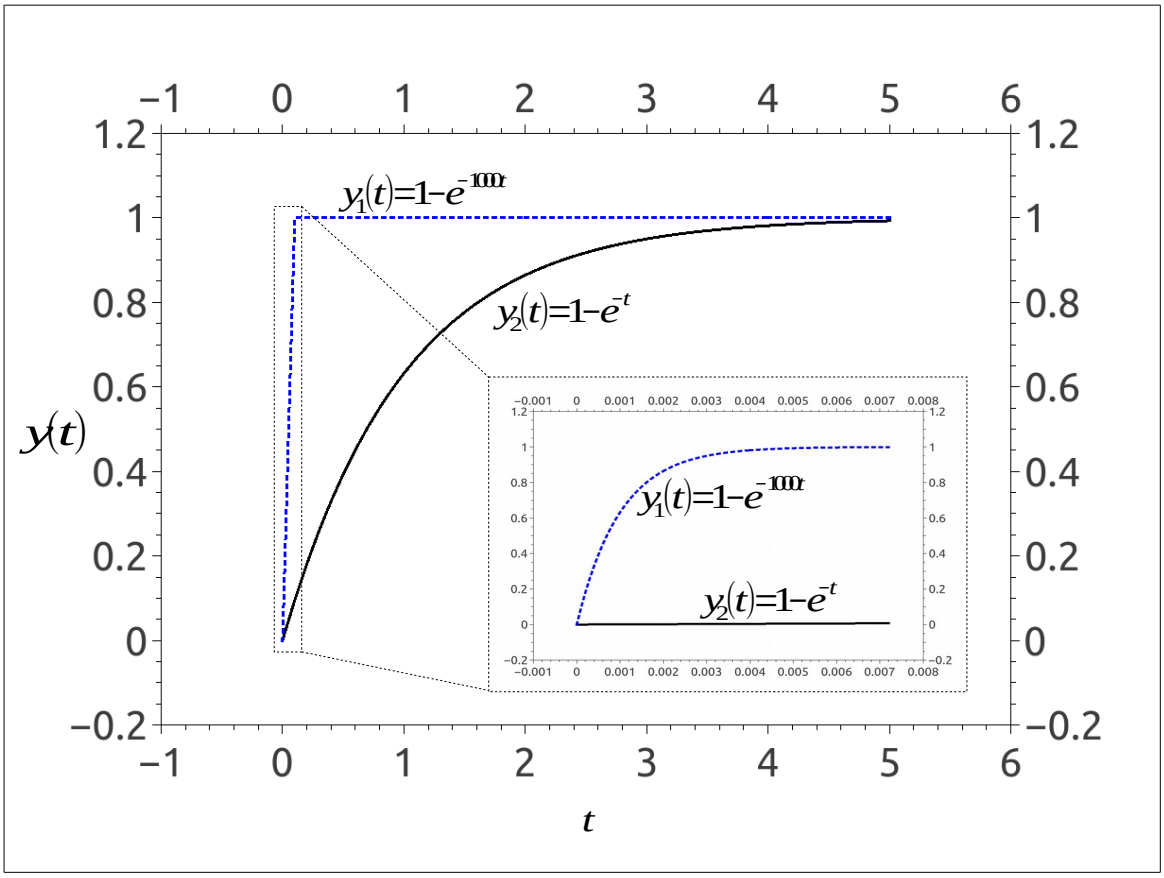


Figure 2.5: The behaviour of the stiff system defined by equation (2.15).

If a small integration step *h* is used, the dynamics of the fast response ODE will be captured. However, despite the fact that the fast response ends after a fraction of the integration interval, any variable step-size routine that is not equipped to handle stiff systems (mainly explicit integration routines) will fail to increase the step-size afterwards [Chapra and Cancade, 2002]. Note the difference in time constants defining the system in equation 2.14 (0.001 and 1). If the time constant of the fastest response equation in an ODE system is denoted as $\tau_{fastest}$ and that of the slowest response equation is denoted as $\tau_{slowest}$, stiffness ratio R_s is defined as:

$$R_S = \frac{\tau_{slowest}}{\tau_{fastest}} \tag{2.16}$$

2.7. Concluding remarks

In this introduction to modelling, I defined modelling and provided a brief historical background. The importance of defining a modelling goal is also navigated. The concepts of equation-oriented and block-oriented modelling were introduced. I also provided a summary of available modelling languages and their categorization. The difference between conservation laws and constitutive equation has been highlighted. I also provided an insight on how discontinuities appear in formulation of mathematical equations. I discussed the building blocks required to integrate any given model and introduced variable step-size as a mean to to efficiently integrate ODEs without a significant loss of accuracy or overload of the computing machine. Lastly, I briefly introduced Stiff ODE systems with methods to integrate them.

When the response time of the fastest ODE in the system approaches 0, R_S in equation 2.16 approaches infinity. Literature refers to this type of problem as a discontinuity problem. Discontinuities in mathematical ODEs require special handling techniques that are presented in Chapter 3. The chapter presents conventional approaches to resolve a discontinuity in an ODE system. Chapter 4 introduces the models that are constructed to prove the novel concepts in Chapter 5. In Chapter 5, I present a novel approach to handle discontinuities. This novel approach better bounds the discontinuity, minimizes the error around it and reduces computational power. Chapter 6 presents some of the applications to the novel approach presented in Chapter 5.

CHAPTER 3:

Discontinuities and Their Conventional Resolutions

In this chapter, I define the mathematical discontinuity, shed light on the previous work dedicated to handling discontinuities in modelling languages. The previous work on handling discontinuities is classified into two types. This chapter reviews previous literature on both types.

A process can be thought of as a complex system that is described by, mostly, continuous mathematical functions (algebraic or differential). Solution of these differential equations, usually through integration, brings insights into the behaviour of the process under study. However, as discussed earlier, the continuity of these mathematical functions is sometimes broken by internal or external influences. Breakage of a continuity occurs because of the tendency of scientists to treat each process condition with differing constitutive equations and/or boundary conditions. Once simulation shifts from one condition to another, the underlying equations change; usually with no reservation of mathematical continuity. A rapid phase change or flow reversal are examples of an internally generated discontinuity in a ODE/DEA system whereas switching a pump on or off can be considered as an external influence that raises a mathematical discontinuity in the modelled system.

A mathematically continuous function at a point *c* is one that satisfies three conditions [Swokowski, 1991]:

f(c) is defined (3.1a)

$\lim_{x \to c} f(x)$ exists (3.1b)

$\lim_{x \to c} f(x) = f(c)$ (3.1c)

Satisfying condition (3.1c) implies that(3.1a) and (3.1b) are automatically satisfied. Discontinuities in mathematical functions arise when one or more of the above conditions are not satisfied. Mathematics classify discontinuity into removable, jump and infinite.

Figures 3.1 illustrate the various forms of discontinuities encountered in mathematics. Figure 3.1a and 3.1b illustrate two types of *removable discontinuities*. For Figure 3.1a, the value of the function at point *c* is not defined. Thus, condition 3.1a is not satisfied and the function is deemed discontinuous at *c*. Figure 3.1b illustrates a different type of

removable discontinuity. Although the function is defined at point c (condition 3.1a), condition 3.1c is not satisfied as $\lim_{x \to c} f(x) \neq f(c)$. The discontinuity in Figure 3.1c is generally referred to as *jump discontinuity.* Note that although *f(c)* is defined at one side of the function, condition 3.1c is still not satisfied as $\lim_{x \to c-} f(x) \neq f(c)$. The last form of discontinuity is called *infinite discontinuity* and is illustrated by the example in Figure 3.1d. In such cases, condition 3.1a and 3.1b are always not satisfied. Note that at this stage of the discussion we are only addressing the continuity of a function but not the continuity of its respective derivatives.

A discontinuity in a mathematical model arises because of a change in a system state leading to a change in mathematical equations representing the system. In some cases, the discontinuity presents itself explicitly in the form of a conditional statement to describe a transition from one state of the system to another. For example, a modeller would transit from a laminar to turbulent flow regime through a conditional statement that sets the boundaries for each regime. Because each regime is described by a different function (correlation), the conditional statement used to transit simulation between two adjacent regimes would probably cause a jump discontinuity.

Other discontinuities might not be modelled in an explicit conditional statement form. However, the structure of the model causes a state change that consequently alters the underlying mathematical equations and eventually leads to a model discontinuity. Examples of this form include model boundary conditions related to disc ruptures, pump start/stop, sudden opening/closure of valves, etc. Such discontinuities can be triggered by a time, space or state-variable event. Such discontinuities can still be reformulated as conditional statements and hence facilitate the derivation of a unified solution for this

class of problems resulting from discontinuities in conditional statements.

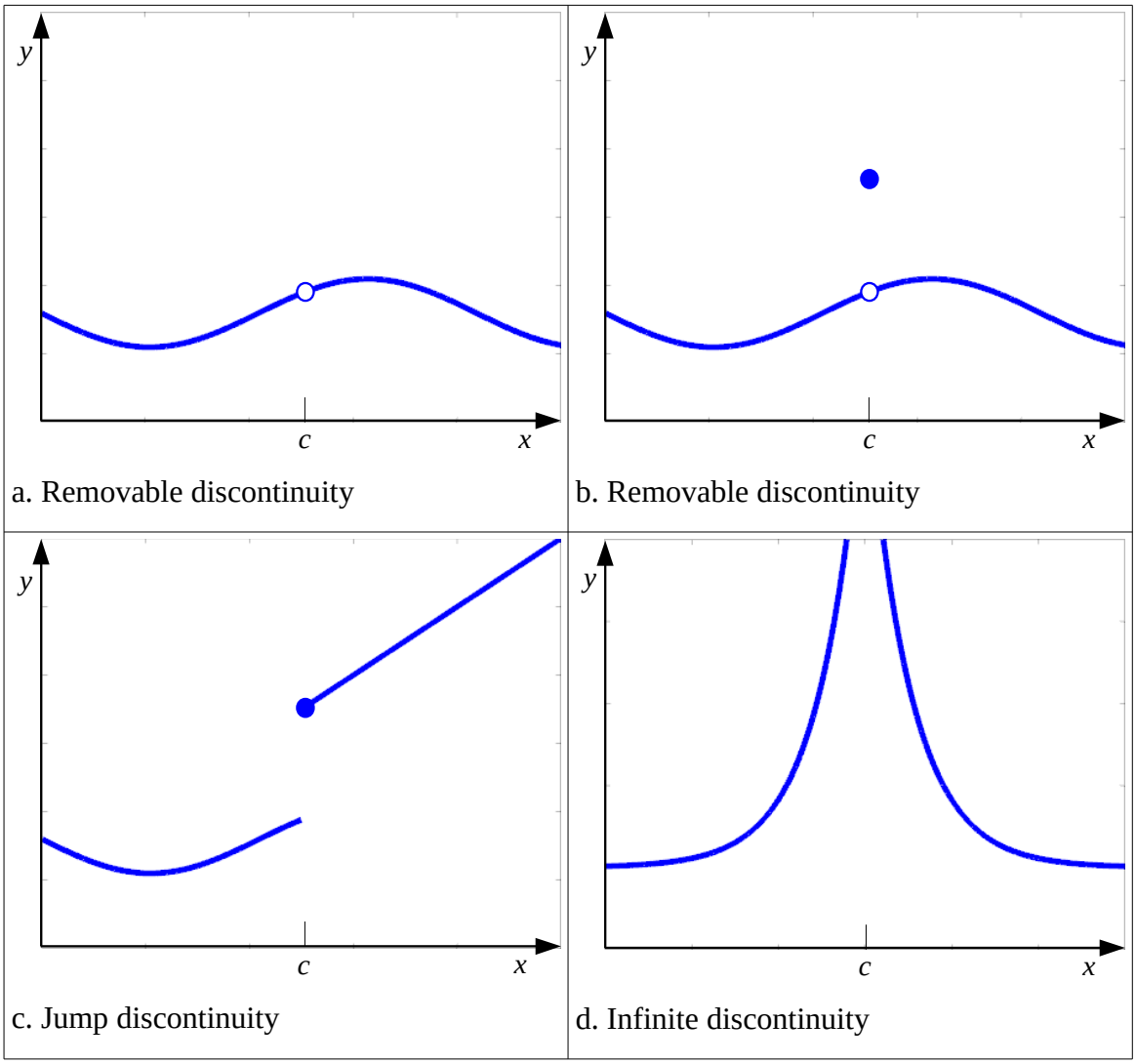


Figure 3.1 : Types of mathematical discontinuities [Swokowski, 1991].

Ideally, conditional statements should not be used to describe continuous dimensions as continuous dimensions are described by continuous functions. Thus, if functions representing continuous models exist with an equivalent accuracy to those with discretized models, continuous functions should be preferred over discretized ones. The method of negative saturations for modelling two-phase compositional flow [Abadpour and Panfilov, 2009] presents an interesting example that resolves a discontinuity in model equations through reformulating the problem definition to eliminate the discontinuity.

However, in some cases, the modeller would want to simplify the modelling task because

of computational cost, inapplicability to the problem at hand, insignificance of rigorously modelling some parts of the model, etc. In other instances, information about specific parts of the model are not readily available. As [Cameron et al, 2005] stated, a model is built to fit a purpose. Thus, if the purpose does not call for a rigorous model, a simplified model is constructed. In such cases, the modeller probably resorts to assumptions that lead to discretizing some of model's continuous dimensions through the use of conditional statements. Discretization contradicts the nature of the assumed continuity of the original rigorous continuous function and presents itself as a jump discontinuity that mandates a resolution during a simulation run.

Even when rigorously tested functions/correlations are available in literature, they are usually bound by the conditions set for their validation experiments. Such bounds leave the modeller no choice but to combine more than one function to cover a certain applicability domain for the intended simulation. Any combination of heterogeneous functions leads to a model discontinuity.

Once a discontinuity in a simulation run is detected, it should be properly handled by the ODE/DAE solver. Handling discontinuity through ODE/DAE solvers is performed through two steps: discontinuity detection and discontinuity resolution; although some solvers combine the two steps [Mao and Petzold, 2002]. The literature refers to the problem of locating a discontinuity as discontinuity detection [Javey, 1988]. Process simulators usually couple their integrators with the modelling language. This coupling eases detection of jump discontinuities.

Regardless of the form or source of discontinuity, it needs to be resolved either before starting to integrate the ODE/DAE system (if possible) or whenever it is encountered during the evolution of integration process. Methods for the resolution of discontinuities

arising during integration of differential equations can be divided into two types:

1. Type I tries to handle discontinuities using methods that are usually integrated with the solver (integrator) of the ODE/DAE system. Those methods are usually generic, irrespective of the system to be modelled and handle discontinuities at the time they are encountered during integration (or simulation). Most literature on discontinuity detection and resolution covers this class (eg. [Ellison, 1981], [Mao and Petzold, 2002], [Javey, 1988] and [Park and Barton, 1996]).
2. Type II handles discontinuities using knowledge about the process to be modelled. It remodels the ODE/DAE system in a way that eliminates discontinuities. Literature is very sparse in this area (e.g. [Borst, 2008], [Brackbill et al, 1992] [Helenbrook et al, 1999] and [Carver, 1978]).

[Borst, 2008] refers to the two types as discretization and regularization, respectively (Figure 3.1). He also points out that internal model discontinuities are better handled using type II methods irrespective of the solver integration routine. Surprisingly, both types use some form of an interpolation to convert a discontinuous region into a continuous one when dealing with internally generated discontinuities. Externally generated discontinuities are usually handled by reinitialization of the model equations and their respective new initial and boundary conditions. In the following discussion, I will briefly touch on recent literature covering each of the categories.

3.1. Type I - Integrator Based Discontinuity Resolution

[Cellier, 1979] demonstrated that the most efficient approach to locating a state event is through discontinuity locking. In discontinuity locking, the system of ODE/DAE is locked until the end of the integration step regardless of the existence of a state event

during the step. After completion of the integration step that involves a state event, the exact location of the state event is detected. Several event location algorithms that use discontinuity locking mechanism are reported and for a comprehensive review of state event detection algorithms, the reader may refer to [Park and Barton, 1996].

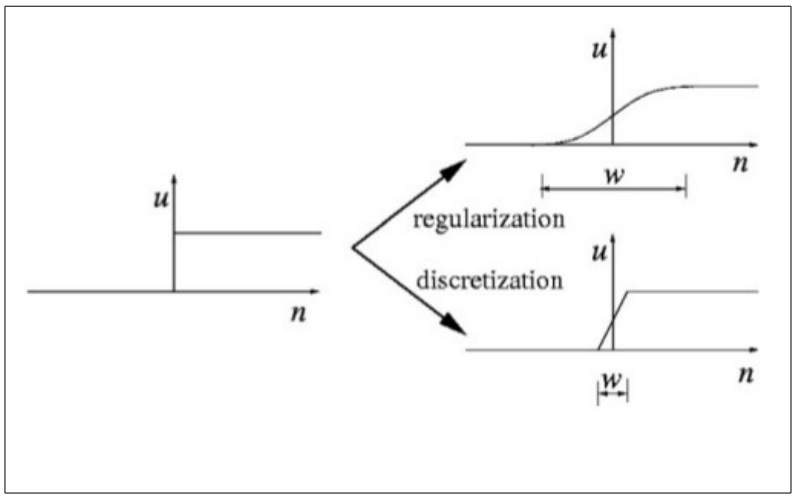


Figure 3.1: Transformation of a discontinuity into either a regularization or discretization problem. [Borst, 2008]

[Mao and Petzold, 2002] have introduced an event detection algorithm that is based on regulating the integration step size based on discontinuity functions that are appended to the DAE system. Recently, [Archibald et al, 2008] introduced a state event detection algorithm that is based on polynomial annihilation techniques. Their method relies on the difference of the Taylor series expansions behaviour between continuous and non-continuous intervals of the tested function. The authors also indicate that their method is applicable to one-dimensional problems only.

Once a discontinuity is detected, it needs to be resolved before the integrator passes it. [Javey, 1988] reports three methods for resolving discontinuities. In all methods, the integrator checks the sign change of a discontinuity-function after each integration step as an indication of having located a discontinuity:

1. Once the discontinuity is located, the integrator switches modelling equations to

those after the discontinuity and starts at the end of the current step. This procedure is inaccurate as it accumulates error each time a discontinuity is encountered. [Mao and Petzold, 2002] warn about mere stepping over discontinuities without carefully handling them with some rigour.

2. Once the discontinuity is located, the integrator halves the step and repeats the last integration step in a hope to resolve the discontinuity. Resolution is generally achieved if the function is continuous but the integrator fails to resolve the discontinuity due to the use of a large integration step. Thus, repeating the integration step with smaller step sizes, where the discontinuity is detected should eventually reveal the continuity of the function. This solution, although better than the first one, is still considered inefficient because the integrator needs to iterate at the discontinuity until an acceptable error tolerance is achieved. If the acceptable error tolerance is not achieved after repeated step-halving (usually because of an instantaneous discontinuity) , the integrator aborts integration. The method is then unable to resolve the discontinuity [Carver, 1978].

3. Once the discontinuity is located, the integrator reinitializes the differential and algebraic variables using post discontinuity conditions after interpolating all differential and algebraic variables at the discontinuity using a discontinuity function (an interpolating polynomial). It should be noted that this method implies mathematical continuity of differential equations through the discontinuity domain regardless of the validity of the resulting solution, as demonstrated by [Cellier, 1979]. This method is the most commonly adopted in recent integration routines used for process simulation.

The mismatch between the results obtained using the interpolating polynomial and

those obtained when reinitializing the ODE/DAE system after crossing a discontinuity sometimes creates what is known as a sticky discontinuity. Sticky discontinuities occur because sometimes after reinitializing the ODE/DAE system, the state of the differential variables returns to the value it had before triggering the discontinuity resolution resulting in an infinite loop: locating the discontinuity, interpolating to conditions after the discontinuity, reinitializing ODE/DAE after the discontinuity, re-evaluating discontinuity trigger and falling back to the same discontinuity, interpolating to conditions after discontinuity, etc.

Two problems arise from Type I discontinuity resolution:

1. Reinitialization effort is directly proportional to the number of DAE/ODE equations. Even if a discontinuity is encountered in one equation of the system, the integrator still needs to reinitialize the entire system. This procedure is computationally exhaustive. What we need is an approach that detects and eliminates localized discontinuities leaving the rest of the system's continuous functions intact.
2. Some integration routines use interpolating polynomials to bridge discontinuous domains. The use of integrator-based interpolating polynomials can produce inaccurate results at or after the discontinuous region. [Park and Barton, 1996] demonstrate that sticky discontinuities arise because the interpolating polynomial used by the integrator to overcome a ODE/DAE discontinuity may land the ODE system at a point before the discontinuity. This is mainly due to the difference in behaviours between the ODE/DAE system and the interpolating polynomial that is used to approximate its behaviour at the discontinuity although both the ODE/DAE system and the interpolating polynomial share the same initial

conditions at the location immediately preceding the discontinuity.

We may easily deduce that even if the interpolating polynomial has managed to cross the discontinuity, it will probably land at a location post the discontinuity that is different from that corresponding to the destination of the ODE/DAE system. So, even when discontinuities are resolved using integrator-based interpolating polynomials, the solution post a discontinuity loses accuracy. The error accumulates with every resolved discontinuity.

3.2. Type II – System Dependent Discontinuity Resolution

In this section, we shed light on resolution of discontinuities using bridging functions that are derived from laws surrounding the physical system or their approximation. The first published attempt was by [Carver, 1978]. He appended the discontinuous functions to the ODE system after a slight transformation. Then, he applied [Gear, 1970] algorithm to detect discontinuities. Carver's attempt was the only encountered attempt to generalize a solution using Type II although the problem was still left discretized (i.e. no regularization functions used). [Brackbill et al, 1992] resolved a discontinuity resulting from the contact of two fluids at an interface point by a smooth interpolation between discontinuities using the following function:

$$P(x)=\begin{cases} C_1(FLUID\,1) \\ C_2(FLUID\,2) \\ 0.5*(C_1+C_2)(INTERFACE) \end{cases} \tag{3.2}$$

[Helenbrook et al, 1999] criticized Brackbill's approach as introducing an error that is linearly proportional to the formed grid. Instead they recommended replacing discontinuities with moving boundaries that retain the interface region between the two fluids. [Borst, 2008] emphasized that the use of regulating functions derived from the physics of the problem (Type II) will better eliminate discontinuities than the sole use of

Type I discretization techniques . He attributes the enhancement to the increase in length (or time) scale over that resulting from the use of discretization techniques as illustrated in Figure 3.1. He illustrated the concept by modelling fractures of solid material at their break points.

3.3. Concluding Remarks

In this chapter, I discussed how conventional numerical integration routines (solvers) handle discontinuities. I also highlighted the drawbacks of handling discontinuities using conventional integrator-based approaches.

Conventional approaches to handling discontinuities are classified into Integrator-Based (Type I) and System-Dependent (Type II). Type II focuses on model behaviour during integration rather than model equations. It addresses the resolution through devising better regularizing functions. Literature favours Type II discontinuity resolution approach over Type I. However, apart from the attempt by [Carver, 1978], literature reports no generic methodology for Type II resolutions.

In Chapter 4, I will introduce the discontinuities in the constructed models that are used to prove the applicability of the novel approach introduced in this work. I will also highlight the sources of the embedded discontinuities within these models.

In Chapter 5, I provide a generic approach to Type II problems that is problem independent. Once included within a simulation package, this approach eliminates the need for the solver to reinitialize state variables whenever a discontinuity is located. In addition, since the approach tackles discontinuities at their appropriate level, interpolating polynomials resulting from this approach more resemble the accurate simulation path than those generated by an integration routine that resolve discontinuities at state variable level only. The resolution is generic enough to be adopted in:

1. implicitly defined discontinuities arising from discontinuous constitutive equations.
2. implicitly defined discontinuities arising from discontinuities in state variables.
3. explicitly defined discontinuities that are formulated as boundary conditions.

An implicit discontinuity is a discontinuity arising from model differential or constitutive equations. On the other hand, an explicit discontinuity is a discontinuity raised through a sudden change in model boundary conditions.

Chapter 4:

Discontinuities in Constructed Models

In this chapter, I will present discontinuities arising in the modelling of a chemical reactor and a Pressure Swing Adsorption (PSA) unit. The reactor model posses an implicit two-dimensional discontinuity in the calculation of its heat transfer coefficient when transitioning between Laminar and Turbulent flow regimes.

The constructed PSA model exhibits multiple one-dimensional discontinuities in its boundary conditions when the PSA column shifts between each of its cyclic steps. To simulate various PSA column configurations, additional intermediate steps are modelled along with the basic cyclic steps reported by [Skarstrom, 1960]. The additional steps include co-current depressurization and multiple pressure equalization steps.

The PSA model is structured to allow its use as an optimisation model for PSA units. I will devote some pages to outline the modelling scheme I followed to include various PSA column steps in one model in order to construct a PSA model that will prove useful for synthesis and optimisation of PSA units.

To demonstrate the ideas on discontinuity handling presented in this thesis, I need to prove that the concept is applicable to both implicitly and explicitly defined discontinuities. Thus, I need to construct models exhibiting implicit and/or explicit discontinuities. In the next two sections, I will walk through model construction, illustrate the philosophy behind constructing each model and highlight encountered discontinuities in the process of model building.

4.1. Discontinuities in the Reactor Model

A simplified model of the isomerization reactor patented by [Minkkinen et al, 1993] is constructed. The reactor is basically used to isomerize part of the normal alkanes introduced by the process feed to elevate the feeds octane number. Details of reactor modelling and validation are discussed in Appendix B. In this section, my primary focus is to present discontinuities occurring in the constructed model.

Discontinuities in the reactor model arise when transitioning from laminar to turbulent flow regimes and vice versa. Modelling any constitutive equation that posses a separate function to represent Laminar flow regime and another one to represent turbulent flow regime will result in a discontinuity when simulation shifts from one flow regime to another. Unless the values of the two functions are close enough for the integrator routine to pass its error tolerance test, a discontinuity is inevitable.

To simplify the problem and only focus on a single discontinuity, I reduced the values of the other variables calculated through constitutive equations to constants evaluated at feed conditions. The only exception is the fluid heat transfer coefficient. To calculate fluid heat transfer coefficient for Laminar flow, I used the simplified constant heat-flux equation of $Nu_d = 4.364$. I assumed that *Reynolds* number ranges from 0 to 2,310. For turbulent flow,

I used the Gnielinski correlation [Keith, 2000]:

$$Nu_d = \frac{(f/2)(Re_d - 1000)\,Pr}{1 + 12.7(f/2)^{1/2}(Pr^{2/3} - 1)}\left[1 + \left(\frac{d}{L}\right)^{2/3}\right] \tag{4.1}$$

where: $f = \left[1.58 \ln(Re_d) - 3.28\right]^{-2}$

$2300 < Re_d < 10^6$

$0.6 < Pr_d < 2000$

$0 < d/L < 1$

Thus, *Nusselt* number for the range covering both laminar and turbulent regimes becomes:

$$Nu_d = \begin{cases} 4.34 & 1 < Re_d < 2{,}310 \\ \dfrac{(f/2)(Re_d - 1000)\,Pr}{1 + 12.7(f/2)^{1/2}(Pr^{2/3} - 1)}\left[1 + \left(\dfrac{d}{L}\right)^{2/3}\right] & 2300 < Re_d < 10^6,\ 0.6 < Pr_d < 2000 \end{cases} \tag{4.2}$$

A plot of Nu_d versus *Re* and *Pr* for Laminar and Turbulent flow regimes is illustrated in Figure 4.1.

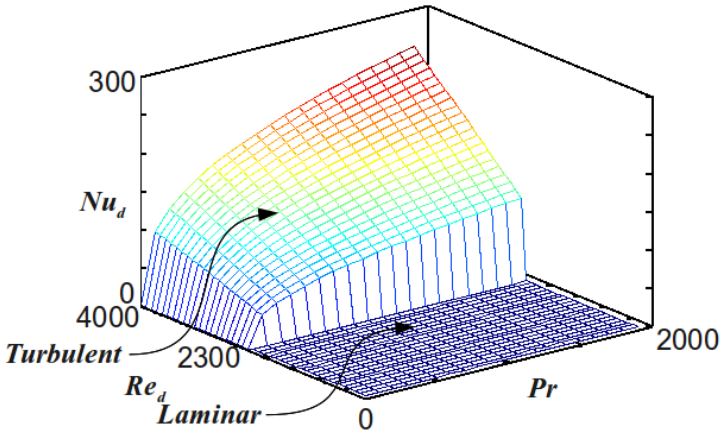


Figure 4.1 :A plot of *Nusselt* number versus *Prandtl* and *Reynolds* numbers illustrating a discontinuity in the transition between Laminar and Turbulent flow regimes at Re = 2300.

A typical pseudo code of equation 4.2 is presented in 4.3:

If (Re < 2300)

$Nu_d = 4.364$ $Nu_d = 4.364$

Else

$$Nu_d = \frac{(f/2)(Re_d - 1000)Pr}{1 + 12.7(f/2)^{1/2}(Pr^{2/3} - 1)}\left[1 + \left(\frac{d}{L}\right)^{2/3}\right]$$

EndIf (4.3)

A typical mistake, that modellers usually fall into, is not accounting for the proper boundaries of both branches of the conditional statement. A better conditional statement encapsulating the bounds of 4.2 would be in a form similar to 4.4:

If (Re > 1) and (Re < 2310)

$Nu_d = 4.364$

ElseIf (Re > 2300) and (Re < 10^6)

if (Pr > 0.6) and (Pr < 2000)

$$Nu_d = \frac{(f/2)(Re_d - 1000)Pr}{1 + 12.7(f/2)^{1/2}(Pr^{2/3} - 1)}\left[1 + \left(\frac{d}{L}\right)^{2/3}\right]$$

Else

flag a warning and continue or flag an error and quit

EndIf

EndIf (4.4)

Note how expression 4.4 well encapsulates the composite Nu_d function within its proper bounds. However, such encapsulation creates a problem during simulation run. What if *Re* started or passed through at a value that is less than 1? What if *Re* is above 2310 but *Pr* is less than 0.6 or greater than 2000?

Also, from the structure of the conditional statement, the language compiler or interpreter would not shift to the second branch of the conditional statement until the the first logical statement evaluates to false although an overlap exists between the domains of the two sub-functions representing both sides of the conditional statement ($\mathrm{Re} \in [2300, 2310]$). Is it better to leave the conditional statement intact or alter it to a better one? If a better one

exists, on what basis should we alter the expression?

Lastly, in modern modelling languages, any transition between two consecutive branches of a conditional statement is treated as a discontinuity that mandates reinitialization of all state variables and their underlying constitutive equations. But, do we need to reinitialize all model equations when the discontinuity is occurring only in a subset of the model equations? In the context of this work, I will provide a generalized framework to better treat models involving discontinuities. In the discussion, I will be providing answers to all of these questions.

4.2. PSA Model Construction and Discontinuities

Pressure Swing Adsorption is one of the very competitive separation techniques to distillation. When the right adsorbent is identified, purities can reach values beyond those of conventional distillation columns. PSA is also useful in separating equiboiling point mixtures that are otherwise deemed difficult or expensive to separate using distillation columns.

This introductory will begin by a process description of PSA. Within the description, I will highlight differences between each of the cyclic steps and the boundary conditions surrounding each of the steps.

4.2.1. PSA Process Description and Differential Equations

The first PSA patents were published between 1930 and 1933. However, early published work on PSA processes was overlooked by recent authors in favour of the works published separately by [Skarstrom, 1960] (filed in 1958 and accepted in 1960) and [Guerin and Domine, 1957] (filed in December 1957).

The [Skarstrom, 1960] PSA cycle consisted of four main steps: pressurization, adsorption,

counter current depressurization (blowdown) and desorption. It used an inert material to desorb. On the other hand, [Guerin and Domine, 1957] used vacuum to desorb material off adsorbents.

After the introductory of the basic steps, few other steps were added that contributed to either an increased purity or a reduced energy utilization. Examples of the later added steps include co-current de-pressurization, pressure equalization and strong-adsorptive purge steps. Also, Rapid PSA eliminated the adsorption step from the basic cycle.

In this section, I will detail the efforts I made to to create a generalized PSA model encompassing most of the available steps. The intent is to use the mode as a synthesis optimization tool that determines the best combination of steps that serve a particular feed with specified objectives (purity and/or recovery). In the following paragraphs, I will describe each of the modelled steps, outline the underlying differential equations, their respective boundary conditions and the available optimisation variables.

In this discussion, the term *adsorbent* refers to the solid pellets which adsorb certain components from the gas phase. Sometimes, it is referred to as *molecular sieve*. The term *inert* is used to refer to the material that is weakly adsorbed from the gas phase by adsorbent. The term *adsorbate* refers to the material that is strongly adsorbed into the adsorbent.

During adsorption step, as the mixture to be separated passes through the adsorbent bed, adsorbent pellets preferentially adsorb some of the mixture components over others based on either separation kinetics or equilibrium constants of mixture constituents. As time passes, more adsorbates accumulate in the adsorbents. At a certain point, adsorbents reach a saturation limit beyond which no adsorption occurs. Once the entire bed, or a portion of it, reaches a certain saturation level, the bed needs to be purged to remove the adsorbed

material. Following the analogy of liquid-liquid extraction, the stream containing the weakly adsorbed components (inerts) is sometimes referred to as the *Raffinate* and that containing the strongly adsorbed is referred to as the *Extract*.

Adsorption is usually favoured by high pressure and low temperature and desorption is hence favoured by low pressure and high temperature. Thus, PSA beds continuously cycle over periods of high and low pressures and temperatures. Most of the Raffinate is collected at high pressure cyclic steps and most of the Extract is collected at low pressure ones (Figure 4.2). Between these two cyclic steps, a PSA vessel, naturally, needs to pressurize and depressurize. Figure 4.3 illustrates a typical PSA cycle pressure profile starting with Pressurization step and moving through Adsorption and De-pressurization steps before concluding with a Desorption step.

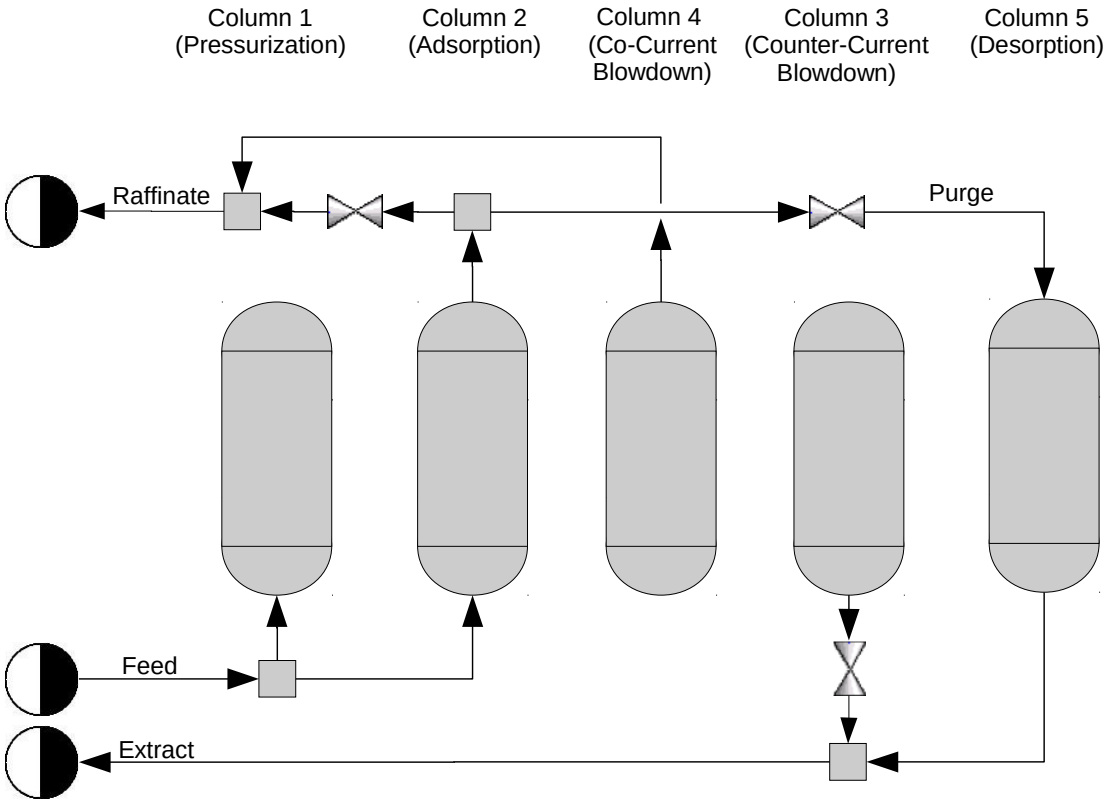


Figure 4.2: A PSA process flow diagram illustrating the connections between feed and product streams for columns undergoing pressurization, adsorption, blowdown (co- & counter- current) and desorption steps respectively.

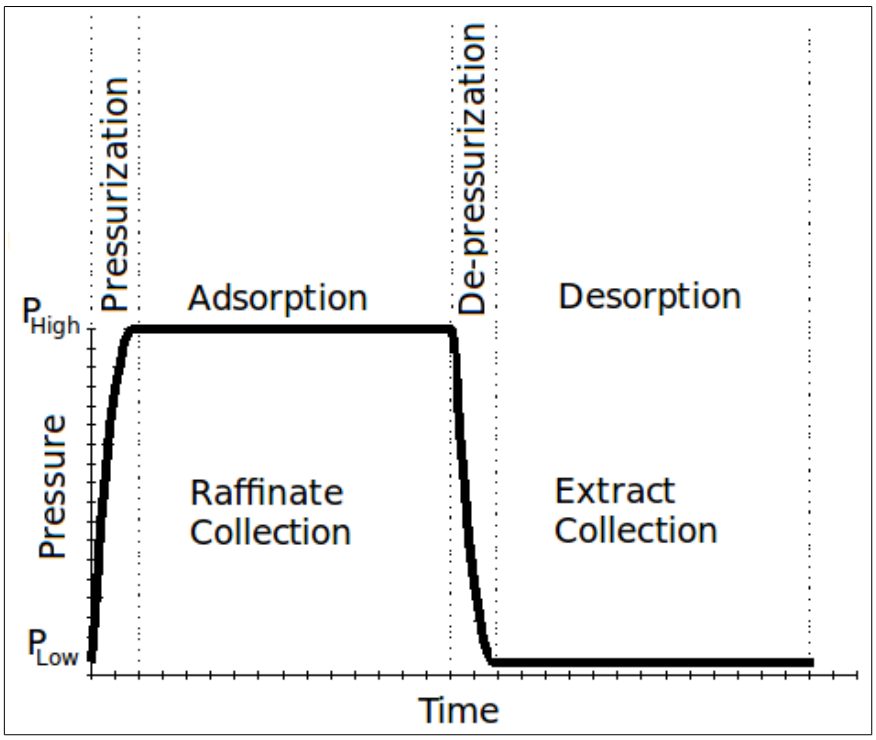


Figure 4.3 : Pressure profile versus time for a single [Skarström, 1960] PSA Cycle.

When constructing the PSA model, I started with the four-step process described by [Skarström, 1960]; namely: Pressurization, Adsorption (feed introduction), Counter-current Blowdown (Depressurization) and Desorption. Figure 4.4 illustrates the interconnections of streams between columns undergoing various steps.

After constructing the basic [Skarström, 1960] cycle, I introduced the co-current blowdown ([Cassidy and Holmes, 1984][Keller II, 1983][Avery and Lee, 1962]) and pressure equalization steps ([March et al][Berlin, 1966][Wagner, 1969]).

Because of computational difficulty of modelling the full set of PSA units, I initially opted for simulating one PSA unit and scaling the resulting output to neighbouring non-PSA units as suggested by [Nilchan and Pantelides, 1998]. However, this modelling scheme proved to be inaccurate when modelling pressure equalization steps as I will discuss later. This limitation mandated the modelling of multiple PSA columns.

I used an axial dispersion model to model the PSA column. To discretize the spatial

dimension, I used the finite difference method. Thus, fluid phase component mass balance is written as:

$$-D_L\frac{\partial^2 c_i}{\partial z^2}+\frac{\partial(uc_i)}{\partial z}+\frac{\partial c_i}{\partial t}+\rho_s\frac{(1-\varepsilon)}{\varepsilon}\frac{\partial q_i}{\partial t}=0 \tag{4.5}$$

The overall mass balance is written as:

$$C_t\frac{\partial u}{\partial z}+\frac{\partial C_t}{\partial t}+\rho_s\frac{(1-\varepsilon)}{\varepsilon}\sum_j\frac{\partial q_j}{\partial t}=0 \tag{4.6}$$

The mass transfer rate follows a linear driving force (LDF) expression:

$$\rho_s\frac{\partial q_i}{\partial t}=a_p k_{gl}(c_i-<c_i>) \tag{4.7}$$

The adsorption equilibrium isotherm follows that introduced by [Nitta et al, 1984]:

$$<c_i>RT=\frac{1}{K_{i,ads}}\frac{\theta_i}{\left(1-\sum_j\theta_j\right)^{n_i}} \tag{4.8}$$

Fluid phase energy balance is written as:

$$\varepsilon K_L\frac{\partial^2 T_g}{\partial z^2}=\varepsilon C_{\rho g}C_t\frac{\partial(uT_g)}{\partial z}+\varepsilon C_{\rho g}C_t\frac{\partial T_g}{\partial t}+(1-\varepsilon)a_p h_p(T_g-T_s)+a_i h_{wi}(T_g-T_w) \tag{4.9}$$

Energy balance around adsorbent is written with the assumption that adjacent adsorbent pellets do not exchange heat and that heat is only exchanged with the surrounding fluid. This assumption reduces heat balance around adsorbent pellets from a PDE to an ODE:

$$\rho_s C_{ps}\frac{\partial T_s}{\partial t}=a_p h_p(T_g-T_s)+\sum_j(-\Delta H_{j,ads})\rho_s\frac{\partial q_j}{\partial t} \tag{4.10}$$

Energy balance around the column shell is formulated as:

$$k_w\frac{\partial^2 T_w}{\partial z^2}=\rho_w C_{\rho w}\frac{\partial T_w}{\partial t}+h_{wi}a_{wi}(T_w-T_g)+h_{we}a_{we}(T_w-T_\infty) \tag{4.11}$$

The pressure drop inside the column is assumed to follow Ergun's equation:

$$\frac{\partial P}{\partial z}=\frac{150\,u\,\varepsilon}{d_p^2}\frac{(1-\varepsilon)^2}{\varepsilon^3}+\frac{1.75\rho_g u^2}{d_p}\frac{(1-\epsilon)}{\varepsilon^3} \tag{4.12}$$

Although Ergun correlation is originally derived to estimate pressure drop across the entire length of the column, it has also been widely used to estimate infinitesimal pressure drop across two points along the axial dimension of the bed (e.g. [Yang et al, 1998] and [Buzanowski and Yang, 1989]). In this work, I adopt the latter use. [Crittenden et al, 1994] showed that pressure drop predictions from Ergun equation do not accurately represent experimental data. Nevertheless, they should suffice for the material to be demonstrated in this thesis.

The boundary conditions for the energy balance around the wall are the same regardless of the cyclic step the column is undergoing:

$$\frac{\partial T_w}{\partial z}\Big|_{z=0}=\frac{\partial T_w}{\partial z}\Big|_{z=L}=0 \tag{4.13}$$

Boundary conditions for other differential equations are cyclic step dependent. I will detail them after a brief description of their respective steps.

Pressurization step is regarded as the first step in a PSA cycle. The purpose of the pressurization step is to elevate the pressure from a predetermined low to high value. The feed to this step can be introduced from a battery-limit fresh feed, from a bleed (recycle) stream from the raffinate or a combination of both. Pressurizing with a recycled stream from the raffinate has the advantage of enhancing raffinate purity.

It should be noted that it is always better to have a higher [high : low] pressure ratio as it enhances separation. However, higher pressure ratios are accompanied with higher compression power costs. No effluent stream is collected from this step (Figure 4.2). Once the pressure reaches its high value, this step ends and the closed end is opened for

raffinate collection signalling the start of the *Adsorption* step.

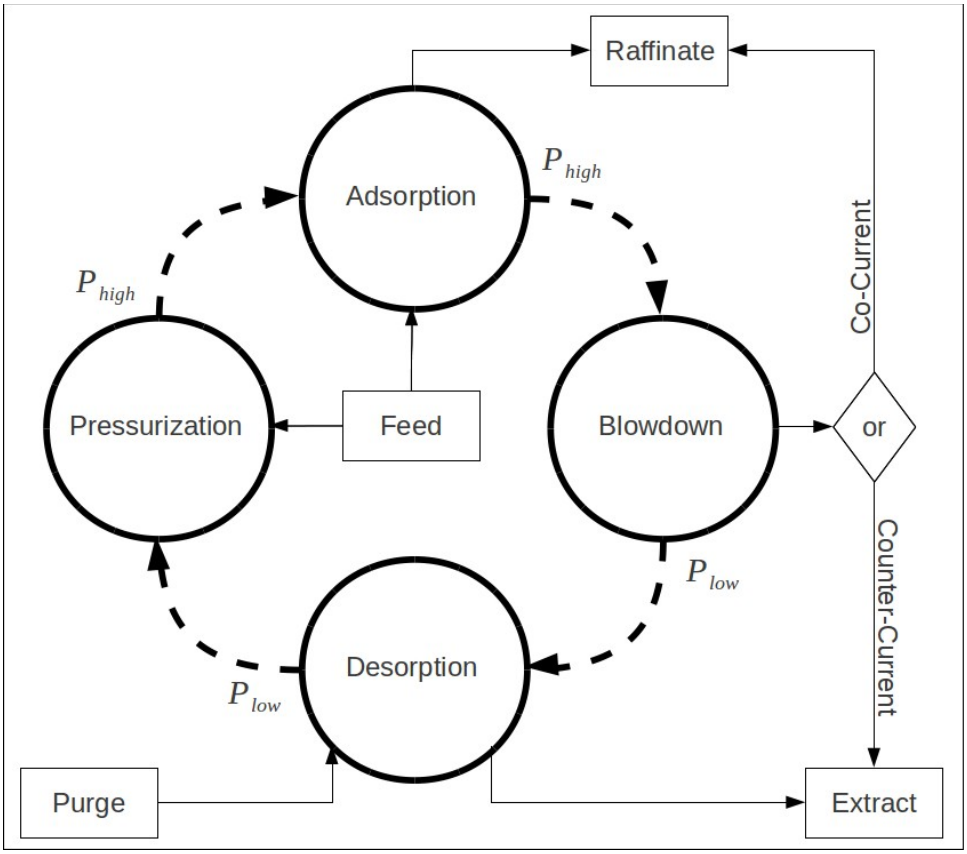


Figure 4.4: A diagram illustrating the basic [Skarstrom, 1960] cycles a PSA column undergoes.

The diagram also indicates the steps where feed is introduced and those where Raffinate and Extracts are collected in addition to the effluent of the cocurrent-blowdown step.

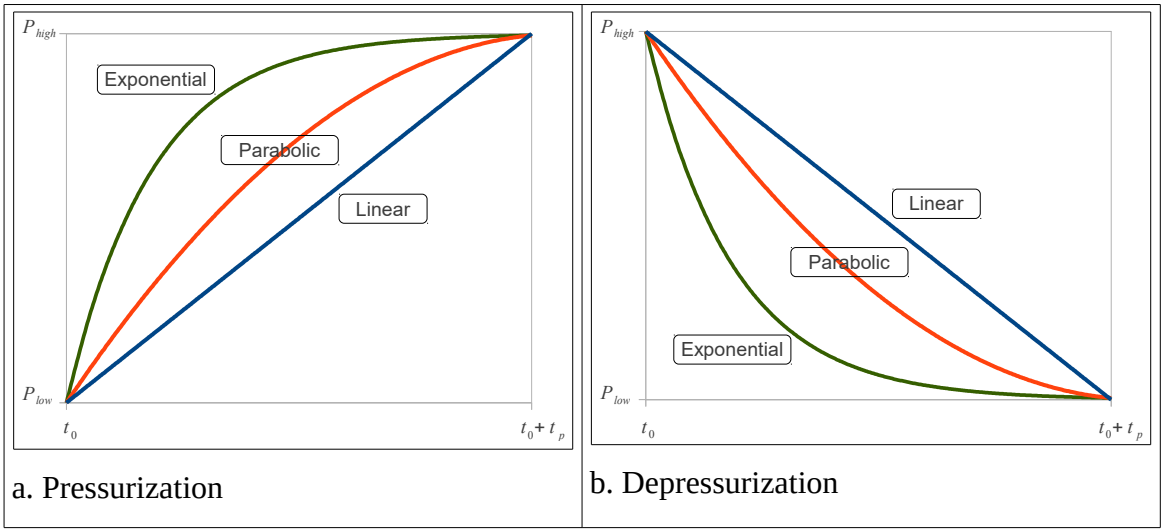
Literature reports the use of three functions to simulate pressurizing and depressurizing a vessel; namely: linear, parabolic and exponential. Figure 4.5 illustrates the shape of the curves for respective functions during pressurization and depressurization steps.

The linear pressurization profile is the simplest to model although it does not represent the reality of a fast pressurization rate when the driving force is high (Pressure difference between feed and vessel) and a low pressurization rate when the driving force reduces. Linear pressurization equation is presented in 4.14 and linear depressurization equation is presented in 4.15.

$$P = P_{low} + \left[\frac{P_{high} - P_{low}}{t_p}\right] t \tag{4.14}$$

$$P = P_{high} + \left[\frac{P_{low} - P_{high}}{t_p}\right] t \tag{4.15}$$

Two functions that demonstrate a better behaviour are the exponential and the parabolic functions. The exponential function provides a steeper departure pressure at the start of the pressurization/depressurization step with a pressure profile that is close to flat line towards the end of the step. On the other hand, the parabolic function provides a relatively even distribution of pressure profile.



a. Pressurization b. Depressurization

Figure 4.5: Comparison between linear, parabolic and exponential pressure profiles for pressurization and depressurization steps.

Equations 4.16 and 4.17 represent parabolic pressure profiles for pressurization and depressurization steps, respectively. Similarly, equations 4.18 and 4.19 represent exponential pressure profiles for pressurization and depressurization steps, respectively.

$$P = P_{high} - \left(P_{high} - P_{low}\right)\left[\frac{t}{t_p} - 1\right]^2 \tag{4.16}$$

$$P = P_{low} - \left(P_{low} - P_{high}\right)\left[\frac{t}{t_p} - 1\right]^2 \tag{4.17}$$

$$P = P_{low} - \left(P_{low} - P_{high}\right)\left[1 - e^{-M_p t}\right] \tag{4.18}$$

$$P = P_{high} - \left(P_{high} - P_{low}\right)\left[1 - e^{-M_{dp} t}\right] \tag{4.19}$$

Nevertheless, all the above modelling equations suffer a fundamental drawback. They all exhibit an instantaneous change in the feed velocity when the feed is initially introduced to the pressurization step. A better novel treatment of this drawback is presented in Appendix A where a combination of parabolic and exponential pressure profile equations is used to provide a realistic inlet velocity evolution from the start to the end of the pressurization step.

Another typical optimization variable for this step is the pressurization rate (M_p) when using exponential pressurization profiles or the pressurization velocity when using parabolic pressurization profiles. Typical boundary conditions for this step are as follow:

$$-D_L \frac{\partial c_{A_i}}{\partial z}\Big|_{z=0} = u\Big|_{z=0}\left(c_{A_{if}} - c_{A_i}\Big|_{z=0}\right) \tag{4.20}$$

$$-D_L \frac{\partial c_{A_i}}{\partial z}\Big|_{z=L} = 0 \tag{4.21}$$

$$-K_L \frac{\partial T_g}{\partial z}\Big|_{z=0} = \varepsilon C_{pg} C_t u\Big|_{z=0}\left(T_{gf} - T_g\Big|_{z=0}\right) \tag{4.22}$$

$$-K_L \frac{\partial T_g}{\partial z}\Big|_{z=L} = 0 \tag{4.23}$$

$$u\Big|_{z=L} = 0 \tag{4.24}$$

Pressurization-Equalization step can be considered as a partial pressurization step from the perspective of the vessel to be pressurized. The difference between a pressurization step and pressurization-equalization step lies in the feed. In the pressurization step, the feed is usually coming from a continuous stream with a fixed pressure, flow and composition such as fresh feed from unit battery-limits or a recycled raffinate. However,

in pressurization-equalization, the vessel that is at the end of an adsorption step is connected to pressurize a vessel that has just been purged; resulting in pressure changes for both vessels during pressure equalization. The main reason behind pressure equalization steps is the conservation of mechanical energy, that would otherwise be drawn from a compressor, by equalizing pressures of these two connected vessels.

Boundary conditions of a pressurization-equalization step can be regarded as similar to those of a pressurization step. However, [Delgado and Rodrigues, 2008] have shown that these boundary conditions do not conserve mass and energy between interconnected vessels; especially for long equalization times. They analysed two sets of boundary conditions from literature. They also proposed a third set of boundary conditions and concluded, from simulation runs, that the third set better conserves mass and energy between interconnected beds. Nevertheless, and for the purposes of this study, I will stick to those boundary conditions that are similar to pressurization step for reasons outlined in the next few paragraphs.

In modelling multiple vessels, I followed the suggestions by [Nilchan and Pantelides, 1998]. They suggested that modelling one PSA vessel is sufficient to predict bed profiles of the entire PSA cycle. Indeed, modelling one vessel and scaling the output to multiple vessels substantially reduces simulation computational power and consequently time. However, to incorporate [Delgado and Rodrigues, 2008] suggestions regarding equalization step boundary conditions, at least two vessels need to be simulated: one undergoing pressurization-equalization and the other undergoing blowdown-equalization. An additional vessel is needed per each additional equalization step. To compromise, I opted for the use of an intermediate vessel to store a well-mixed product of the bed undergoing blowdown-equalization. The amount stored in the intermediate vessel will be

discharged to a running PSA bed when the bed reaches the next pressurization-equalization step. The intermediate vessel acts as a well-mixed tank. Thus, time and spatial profiles are not stored. Only the integral of the amount released from the bed and its average concentration over the elapsed time are stored for later use. I am still using the exact boundary conditions of regular pressurization and blowdown steps for the beds undergoing pressurization-equalization and blowdown-equalization, respectively. The idea of introducing and intermediate storage vessel is not new. It was implemented in the original patent that introduced equalization steps to the community [Marsh et al, 1964] before eliminating the intermediate vessel in the patents filed by [Berlin, 1966] and [Wagner, 1969].

The question would then be, why should we still treat this step as a separate one instead of treating it as a pressurization step? It is mainly to conserve mass balance. As would be expected, the mass of an equalization step is conserved between the interconnected high and low PSA vessels. No raffinates or extracts are collected during equalization steps. In addition, this segregation allows independent future developments of separate boundary conditions for depressurization, depressurization-equalization, pressurization and pressurization equalization steps inside the model.

The final pressure of an equalization step lies somewhere between the pressures of the two interconnected vessels. Arithmetic ($P_{eq}=0.5*(P_{high}+P_{low})$) and geometric ($P_{eq}=\sqrt{(P_{high}P_{low})}$) means are used in literature to calculate the final settling (equalization) pressure. Examples of works that use these formulas include [Chiang, 1996] and [Banerjee et al, 1990]. [Warmuzinski, 2002] showed that arithmetic mean corresponds to the frozen solid approximation. However, due to the nature of this step, both averages do not reflect the actual final settling pressure. [Warmuzinski and Tanczyk,

2003] calculated the equalization pressure for a binary adsorbed components using this equation (assuming component A is the strongly adsorbed):

$$P_{eq} = \sqrt[C+1]{P_{high}^{C} P_{low}} \tag{4.25}$$

Where:

$$C = \frac{1}{\alpha\, y_A^f + 1}\ ,\ \alpha = \frac{\alpha_B}{\alpha_A}\ ,\ \alpha_i = \frac{\epsilon_t}{\epsilon_b} + \frac{1-\epsilon_b}{\epsilon_b} K_{i,ads}\ ,\ i = \text{A, B}$$

However, their analysis is based on linear isotherms. Since we are fitting our adsorption isotherm curves to a non-linear model [Nitta et al, 1984], more testing is required to verify the validity of this formula. [Chahbani and Tondeur, 2010] have proved that, for an accurate prediction of equalization pressure, segregation of the equalization step into pressurization-equalization and blowdown-equalization steps ceases to be valid as I noted earlier. This demonstrates the invalidity of the assumption that modelling a single PSA bed suffices to predicting the performance of an entire PSA unit, proposed by [Nilchan and Pantelides, 1998], when it comes to equalization steps. As can be seen from Figure 4.6 , there is a noticeable mass imbalance between the two interconnected vessels when assuming that each vessel preserves independent boundary conditions, as reported by [Delgado and Rodrigues, 2008]. However, I opted to accept this difference and reinitialize content of the virtual tank after the end of each pressurization-equalization step. The constructed model is designed to allow the calculation of equalization pressure using arithmetic, geometric or [Warmuzinski and Tanczyk, 2003] equation based on user selection.

Since this work is aimed as a proof of a concept more than a rigorous design and/or operation, I think [Nilchan and Pantelides, 1998] assumption is sufficient for the purpose. However, for the PSA optimization work discussed in section 4.2.2, pressure equalization

is modelled using a number of PSA units.

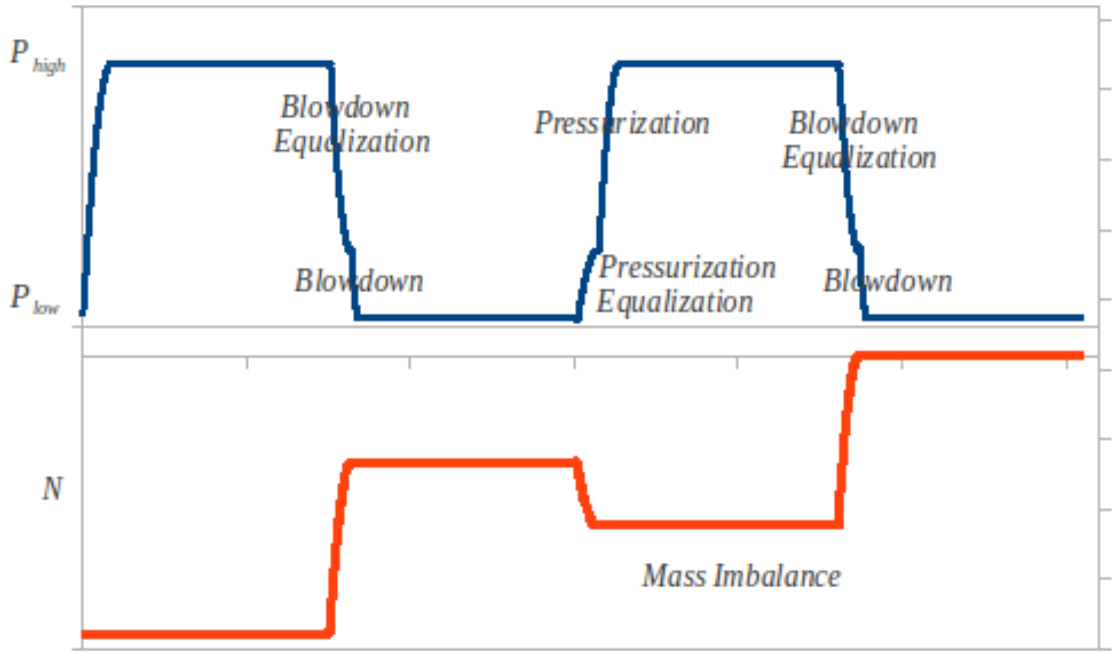


Figure 4.6: Trends illustrating the imbalance in mass when assuming that pressure equalization steps act as two separate steps; namely: pressurization-equalization and blowdown-equalization.

The mass of the virtual tank is trended at the lower section of the figure. Trends were produced using geometric average pressure.

The typical optimization parameter for equalization steps is the number of equalization steps to be performed with a column undergoing pressurization and a set of columns that need to be de-pressurized. The absence of equalization steps result in a considerable loss of mechanical energy that needs to be compensated by power-driven compressors; leading to energy inefficient process. On the other hand, after a certain number of equalization steps, the driving force (pressure difference) between the interconnected vessels reaches a very low value that renders further equalizations infeasible. Boundary conditions for this step are the same as those for pressurization steps (eq. 4.20-4.24).

Adsorption step (sometimes referred to as feed introduction step) is the high pressure step since pressure remains at its high value for the entire period of the step. This is also the step at which raffinate is collected (Figure 4.2). When PSA units were introduced, this

step used to be run until the bed was saturated with adsobates before switching to counter-current blowdown (depressurization) step. However, after introduction of the co-current blowdown step, beds are prematurely switched to co-current blowdown to allow additional recovery of raffinate. A typical optimization parameter for this step is the step duration (t_a). Boundary conditions for adsorption step are written as:

$$-D_L \frac{\partial c_{A_i}}{\partial z}\Big|_{z=0} = u\Big|_{z=0}\left(c_{A_{if}} - c_{A_i}\Big|_{z=0}\right) \tag{4.26}$$

$$-D_L \frac{\partial c_{A_i}}{\partial z}\Big|_{z=L} = 0 \tag{4.27}$$

$$-K_L \frac{\partial T_g}{\partial z}\Big|_{z=0} = \varepsilon C_{pg} C_t u\Big|_{z=0}\left(T_{gf} - T_g\Big|_{z=0}\right) \tag{4.28}$$

$$-K_L \frac{\partial T_g}{\partial z}\Big|_{z=L} = 0 \tag{4.29}$$

$$u\Big|_{z=0} = u_f \tag{4.30}$$

The discussion related to pressurization-equalization step is also applicable to depressurization-*equalization* step. The purpose of the depressurization-equalization step is to reduce the pressure from its high value, to an intermediate value, by pressurizing a vessel at a lower pressure. This step allows for conservation of mechanical energy required to pressurize low-pressure vessels. No products are collected during this step. The Boundary conditions for depressurization-equalization step are:

$$-D_L \frac{\partial c_{A_i}}{\partial z}\Big|_{z=0} = 0 \tag{4.31}$$

$$-D_L \frac{\partial c_{A_i}}{\partial z}\Big|_{z=L} = 0 \tag{4.32}$$

$$-K_L \frac{\partial T_g}{\partial z}\Big|_{z=0} = 0 \tag{4.33}$$

$$-D_L \frac{\partial c_{A_i}}{\partial z}\Big|_{z=0} = 0 \tag{4.31}$$

$$-K_L \frac{\partial T_g}{\partial z}\Big|_{z=L} = 0 \tag{4.34}$$

$$u\big|_{z=L} = 0 \tag{4.35}$$

De-pressurization (blowdown) is originally the step that is used to reduce bed pressure from its high value to the low one. However, after introduction of equalization steps, this step became either an intermediate step between equalization steps (e.g. [Cassidy and Holmes, 1984]) or a final step after a series of equalization steps to bring bed pressure to the value of the purge stream in the desorption step. The main difference between this step and an equalization step is that the bed in this step is connected to a low pressure end (in contrast to a variable pressure vessel in equalization step). The direction of the flow of this step determines the collecting end. Co-current blowdown effluent is usually collected as a raffinate while counter-current blowdown effluent is usually collected as an extract as illustrated in 4.2. In both cases, one end of the vessel is closed. The advantage of co-current blowdown, before saturating the bed, is that it increases the concentration of the strongly adsorbed components in the gas phase by discharging the weakly adsorbed components that were trapped in the adsorbent to the raffinate product. The resulting increased concentration of strongly adsorbed components enhances extract purity when collected later at the counter-current blowdown step. Thus, this step simultaneously enhances raffinate and extract recoveries and purities.

The depressurization rate (M_{dp}) or depressurization time (t_{dp})is a typical optimization variable. Another optimization variable is the fractional time utilized for co-current pressurization versus that of the counter-current pressurization in relation to the total time devoted for depressurization (t_{dp}). Boundary conditions for the blowdown step are exactly

the same as those of the Blowdown-equalization step.

Desorption step is the last step in a cycle. The purpose of this step is to clean the saturated adsorbent from the adsorbate that was mainly adsorbed during adsoprion step. Since desorption is favoured by low pressure, this step is entirely run at low pressure. In addition, part of the raffinate is used as a purge gas. In fact, raffinate recovery and purity are influenced by the amount of the purge used. So, for an operating unit, more purge results in a purer raffinate at the expense of its recovery and vice versa. Extract is collected as an effluent from this step. Desorption step (t_d) duration is a typical optimization variable. Typical desorption step boundary conditions are:

$$\frac{\partial c_{A_i}}{\partial z}\Big|_{z=0} = 0 \tag{4.36}$$

$$-D_L \frac{\partial c_{A_i}}{\partial z}\Big|_{z=L} = u|_{z=L}\left(c_{A_{ip}} - c_{A_i}|_{z=L}\right) \tag{4.37}$$

$$-K_L \frac{\partial T_g}{\partial z}\Big|_{z=0} = 0 \tag{4.38}$$

$$-K_L \frac{\partial T_g}{\partial z}\Big|_{z=L} = \varepsilon C_{pg} C_t u|_{z=L}\left(T_{gp} - T_g|_{z=L}\right) \tag{4.39}$$

$$u|_{z=L} = u_p \tag{4.40}$$

To accurately represent the unit, two separate tanks are added to store both products' (raffinate and extract) quantities and qualities. Also, to avoid the infinite accumulation of mass, as the simulation, progresses, tanks' respective inventories are reduced, or simply reinitialized, to a specified inventory once the inventory exceeds the specified limit. In addition to mimicking real PSA units, this provision prevents the tanks from turning into concentration sinks; specially after the passage of a large number of cycles.

All beds initially contain no adsorbates in both fluid and solid phases. Also, initial bed

temperature is assumed to be equal to the fresh feed temperature. Thus, initial conditions become:

$$c_i(z,t=0)=0 \tag{4.41}$$

$$q_i(z,t=0)=0 \tag{4.42}$$

$$T(z,t=0)=T_f \tag{4.43}$$

where C_{Ai} refers to the concentration of each adsorbate component.

4.2.2. Formulation of the PSA synthesis problem

As I indicated earlier, the PSA model was developed generically enough to be applied to the synthesis of any PSA process provided that constitutive equations related to the composition of the feed to be processed and those related to the adsorbent are available. In this section, I will outline the formulation of the optimization problem as a disjunctive programming problem [Grossmann and Ruiz, 2011].

since this is a synthesis optimization problem, the objective function can be written as:

$$max \quad P=(Y_R F_R \$_R + Y_E F_E \$_E - P_C \$_C - N_{SD} \$_{SD}) C_L - (N_C \$_{N_C} + N_{Aux} \$_{Aux}) \tag{4.44}$$

where:

Y_R : composition of valuable components in Raffinate stream

F_R : Raffinate stream flow

$\$_R$: Raffinate stream Price

Y_E : composition of valuable components in Extract stream

F_E : Extract stream flow

$\$_E$: Extract stream price

P_C : consumed power (mainly compression)

$\$_C$: Price of consumed power

N_{SD} : Number of shut downs per cycle length

$\$_{SD}$: Cost of production loss per shut down

N_C : Number of PSA columns (optimisation variable)

$\$_{N_c}$: Capital cost of a single PSA column

N_{Aux} : Number of auxiliary equipment (mainly compressors)

$\$_{Aux}$: Capital cost of a single compressor

C_L : Life cycle

The first right hand side term corresponds to the operating cost while the second term corresponds to the capital cost. For simplicity, all auxiliary equipment (piping, valves, compressors, etc) are combined into a compressor term. This is usually a valid assumption since the capital cost of the compression supersedes the cost of other equipment.

Compression power P_C is represented as combination of the compression power saved with pressure equalization steps and that consumed during elevation of extract pressure to feed pressure before using it to co-current purge at high pressure:

$$P_{CN} = P_{Press} + P_{SA} - P_{EQ} \tag{4.45}$$

where:

P_{Press} : Total compression power required to pressurize a vessel.

P_{SA} : Power required to elevate the extract pressure from its low value to that of the strong adsorptive purge pressure.

P_{EQ} : Compression power required if equalization steps are used.

Both terms in equation will be discussed later in this section.

When there is a premium on the quality of either Raffinate or Extract flows, the premium can be included as a variable cost function:

$$\$_R = f(Y_R) \tag{4.46}$$

$$\$_E = f(Y_E) \tag{4.47}$$

The overall material balance is a constraint:

$$F_F = F_R + F_E \tag{4.48}$$

where:

F_F : fresh feed stream flow

For the pressurization step, the only optimization variable is the pressurization rate (M_p) or the pressurization time (t_p). [Shirley and Lemcoff, 1996] demonstrated that the performance of an Air-nitrogen PSA separation unit approaches a maximum as the pressurization rate increases before dropping afterwards. I expect other PSA units to follow similar behaviour. Thus, pressurization rate is added as an optimisation variable.

For the adsorption (feed introduction) step, the only optimization variable is the duration of the step (t_a). Low durations result in high purity raffinate and maintain bed temperatures at relatively steady values, preventing high temperature swings between adsorption and desorption steps. However, a low step duration might underutilize the PSA bed, resulting in frequent shifts between cycle steps. These frequent shifts lead to short valve life cycles. Cost of valve replacements is usually not that high. However, the cost of production loss due to unplanned shut downs is high enough. PSA units are usually used as intermediate units to aid in production. Thus, the cost of a unit shut down is usually not directly associated with the cost of separated products from the PSA unit but is directly associated with the cost of the final products produced from the plant.

Longer adsorption step durations result in an increase in bed temperature. This increase in

bed temperature lowers adsorption capacity (adsorption capacity increases with the decrease in temperature). Thus, a longer adsorption step duration is also not favourable. an optimum adsorption time for a specified process that balances between process failure and separation efficiency should exist.

The term $N_{SD}\$_{SD}$ captures the cost of production loss due to probable shut downs resulting from a valve failure. Assuming a valve can function for a specified number of open/close sequences (S_{MAX}), dividing the number of total open/close sequences (S) over S_{MAX} calculates the number of probable shut downs. To include the term as part of the operating cost, it needs to be divided by the life cycle (C_L). Thus,

$$N_{SD} = \frac{S}{S_{MAX} C_L} \tag{4.49}$$

Co-current Purging with strongly adsorptive (Extract) product was introduced in the patent by [Tamura, 1974]. The basic idea is to purge the amount of feed that is left inside a PSA column with a portion of the Extract stream after elevating Extract stream pressure to that of the feed as illustrated in Figure 4.7a. The effluent of this step is combined with the effluent of the adsorption step and thus is considered as part of the Raffinate. The introduction of this step (in addition to co-current depressurization) enabled the production of high purity extract in addition to high purity raffinate [Yang, 1987]. However, the downside of this step is that it involves pressure elevation for the amount of extract that will be used as a purge stream. Remember that extract is mostly (with the exception of counter-current de-pressurization) a low-pressure product. Thus, elevation to a higher pressure incurs power costs. The consumed compression power during purge pressure elevation is captured within the objective function in the variable (P_P).

[Yang, 1987] suggested that an optimization opportunity may exist if purging with strong adsorptive is performed between two co-current de-pressurization intervals as illustrated

in Figure 4.7b. The amount of power saved by elevating the Extract pressure to a value that is lower than that of the feed might justify the suggestion. However, no attempt has been made to verify the feasibility of this suggestion. One of the objectives of this PSA model development work is to prove such feasibility. An optimization variable (x_{cc}) will be introduced. An x_{cc}=0 indicates that the high pressure purge will occur immediately after the adsorption step as illustrated in Figure 4.7a. This leads to purging with a pressure that is equivalent to that of the feed. An x_{cc}=1 indicates that the purge step will occur after the co-current pressurization step as illustrated in Figure 4.7c. This would result in the strong-adsorptive purge taking place at the lowest possible pressure at which raffinate is collected. An x_{cc} value between 0 and 1 would indicate a strong-adsorptive purge that occurs sandwiched between two co-current de-pressurization steps as illustrated in Figure 4.7b. The value of x_{cc} will dictate the amount of de-pressurization time after which the strong-adsorptive purge step would occur.

Once a strong-adsorptive purge step is introduced, the duration of this step ($t_{sa,p}$) becomes an optimisation variable. A short duration will result in lower recovery of inerts (weakly adsorptive). A long duration will result in an escape of the strong adsorptive components into the raffinate leading to lower raffinate purity. A good estimate for the upper bound of the duration would be the length of the adsorption step (t_a). Using pressurization step time duration (t_p) as an upper bound might not be sufficient to discharge all inerts from the column if the column is too long.

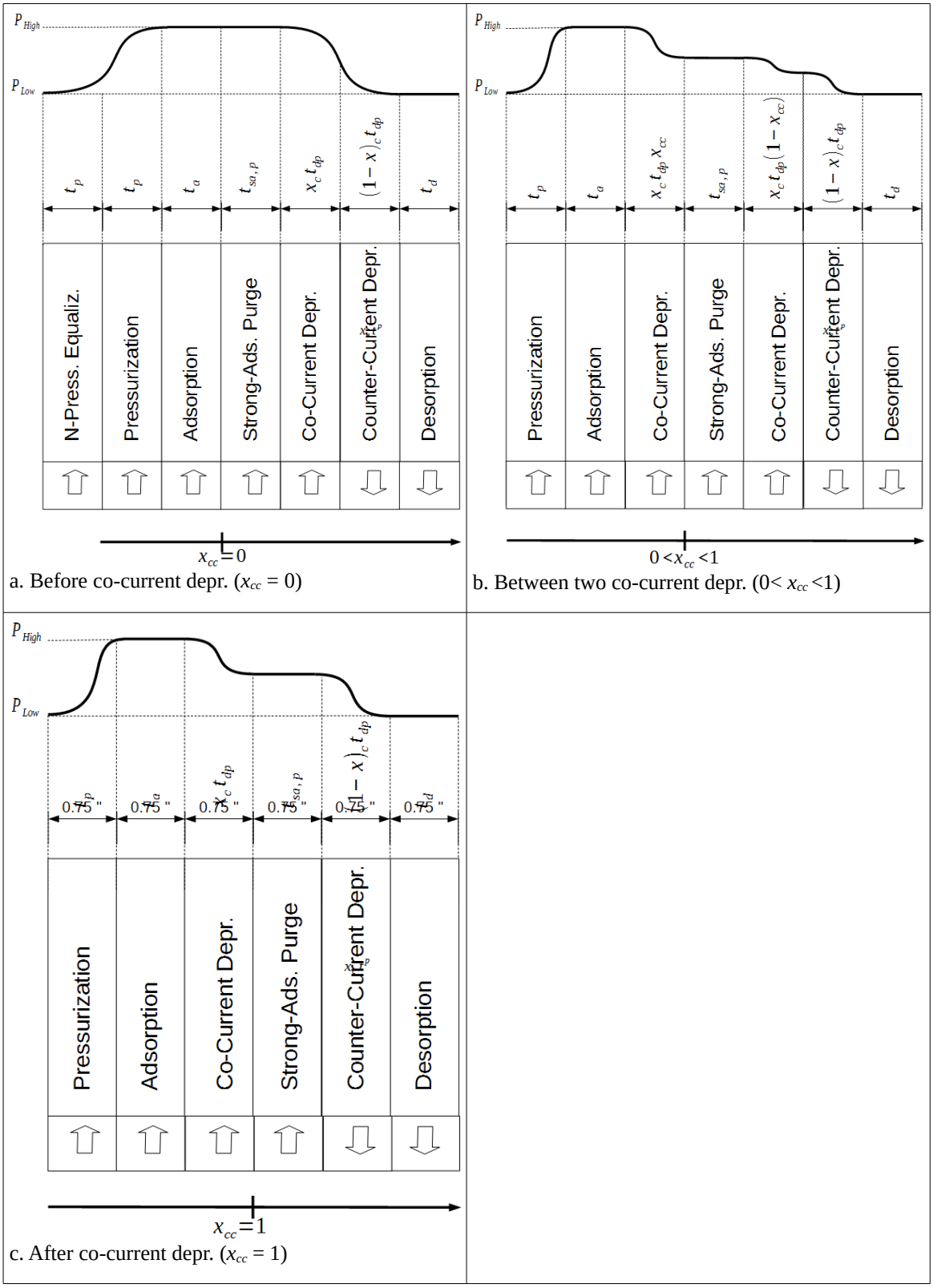


Figure 4.7: Location of the strong-adsorptive purge step relative to the co-current depressurization step as suggested, but not verified, by [Yang, 1987]. Arrows indicate the flow direction for each of the steps.

For Depressurization (blowdown) step, the first optimisation variable is the depressurization rate (M_{dp}) or the depressurization time (t_{dp}). The second optimization variable is the fraction of the depressurization time that is devoted to co-current depressurization (x_c). The remaining depressurization period (1-x_c), after subtracting the time required for pressure equalization, is devoted to counter-current depressurization.

For Pressure Equalization step, the optimization variable would be the number of feasible equalization steps (N_E). Since each equalization occurs between two columns, the minimum number of columns required for a PSA process that involves equalization steps is 3. The third PSA column is required to maintain continuity of production. Also, for the same reason, the maximum number of equalizations should not exceed the number of available PSA columns.

After a number of successive equalization steps, the pressure difference between the column to be pressurized and the pressurizing column becomes small enough to hinder subsequent equalizations. Thus, an optimum number of equalization steps exists.

For desorption step, the optimisation step is desorption step duration (t_d). Short t_d values result in under desorption of strongly adsorptive from adsorbent pellets. Long t_d values lead to lower raffinate recovery.

Another variable that affects the performance of the desorption step is the location of the effluent stream at the desorption step. In their patent, [Guerin and Domine, 1957] purged their extract from the middle of the PSA column (not from either of the column ends). Purging from the middle of the column cuts the residence time of the material inside the vessel by almost a half. The location of desorption step effluent stream (x_d) also constitutes an optimisation variable with the optimum leaning probably towards the feed end. An $x_d=0$ indicates an extract that is collected from the feed end. An x_d=1 indicates an

extract that is collected from the product end (z=L). The importance of the location of the desorption step effluent has not been studied in any earlier work. The second objective of this work is to determine the optimum location of the effluent stream during desorption step.

The last optimisation variable of the Desorption step is the [purge : feed] ratio. In his patent, [Skarstrom, 1960] indicated that for the desorption step to be effective, the volumes of the feed and purge streams, at their respective pressures, should at least be the same. This suggestion proved to be useful in future PSA implementations. It also sets the minimum purge volume (or volumetric flow rate). It can be formulated as a minimum constraint. Assuming ideal gas behaviour, the constraint can be formulated as:

$$\left[V_P = \frac{n_P R T_P}{P_P}\right] \geq \left[V_F = \frac{n_F R T_F}{P_F}\right] \tag{4.50}$$

Dividing V_P by V_F in Equation 4.51 , the ratio becomes:

$$\frac{V_P}{V_F} = \frac{n_P T_P}{P_P}\frac{P_F}{n_F T_F} \geq 1 \tag{4.51}$$

To complete problem formulation, I need to specify a minimum raffinate purity and/or recovery or a minimum extract purity and/or recovery. I also need to specify the maximum number of columns required to achieve such specifications. The problem can be further extended to optimize columns sizing (i.e. length and diameter). Thus, the optimization problem can be summarized as:

$$max \quad P = \left(Y_R F_R \$_R + Y_E F_E \$_E - P_C \$_C - N_{SD} \$_{SD}\right) C_L - \left(N_C \$_{N_c} + N_{Aux} \$_{Aux}\right)$$

s.t. :

1. Pressurization rate: $M_{(p,min)} < M_p < M_{(p,max)}$
2. Pressurization Feed (fresh or recycled raffinate) [Boolean]:
   $\left[P_F = 0\right] \vee \left[P_F = 1\right]$

3. Adsorption step duration: $0 < t_a < t_{a,max}$
4. Strong Adsorptive Purge:
$$\begin{bmatrix} 0 < x_{cc} \le 1 \\ 0 < t_{sa,p} \le t_{a,max} \end{bmatrix} \vee \begin{bmatrix} x_{cc} = 0 \\ t_{sa,p} = 0_{a,max} \end{bmatrix}$$
5. Depressurization rate: $M_{(dp,min)} < M_{dp} < M_{(dp,max)}$
6. Fraction co-current de-pressurization from the total de-pressuirzation time: $0 \le x_c \le 1$
7. Desorption step duration: $0 < t_d \le t_{d,max}$
8. Desorption step effluent stream location: $0 \le x_d \le 1$
9. Column length: $L_{min} \le L \le L_{max}$
10. Column diameter: $d_{c,min} \le d_c \le d_{c,max}$
11. Number of PSA columns: $1 \le N_c \le N_{c,max}$
12. Number of Presssure-Equalization steps: $0 < N_E \le N_C - 1$
13. Minimum raffinate purity: $Y_{R,min} < Y_R \le 1$
14. Minimum extract purity: $Y_{E,min} < Y_E \le 1$

The only equality constraint is the total material balance:

1. Material Balance: $F_D = F_R + F_E$

Note that the N_E and N_C are pure integer variables and not an either-or boolean variables. I am intending to connect these variables to an MIP optimizer using real variables. The optimisation routine should search an integer space of the these two variables and not the real one. This constitutes a mapping problem. How should the optimiser behave when it requests the value of the cost function at N_C =1.5? To overcome this difficulty, I am planning to introduce an intermediate layer between the optimizer and the constructed model specifically tailored to these two variables as illustrated in Figure 4.8.

The purpose of the intermediate layer is to rescale the integer variables in order to present them as real variable to the optimizer. Thus, using the transformation $x_C = N_c / N_{C,Max}$, the N_C

variable is transformed to the variable x_C that is bounded by real bounds: $x_C \in [0,1]$. Applying similar scaling transforms the N_E variable: $x_E = N_E / N_{c,max}$. The variables N_C *and* N_E will be visible to the constructed model while their respective transformations x_C and x_E will be visible to the optimizer. The reader should note that since the lower bound of N_E is dependent on N_C ($N_E \in [0, N_C]$), a variable-bound optimisation routine should be selected for optimization.

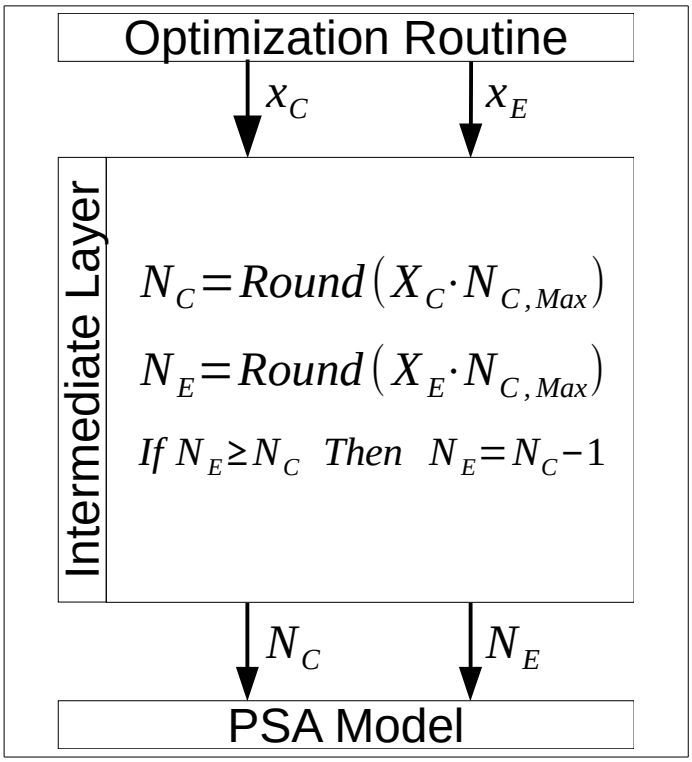


Figure 4.8: Optimising integer variables as continuous ones through the introduction of an intermediate layer.

The third objective behind this optimisation exercise is to search for a possible existence of any new PSA operating region that was not revealed in any of the earlier PSA works by freely varying all optimization variables within their specified limits.

In summary, this section outlined the developmental work performed in modelling the PSA unit that was used to prove the concepts developed in this thesis. The next section outlines the discontinuities occurring as a result of shifts between boundary conditions of

the intermediate steps that form a PSA cycle.

4.2.3. Encountered Discontinuities in the PSA Model

Noting the variations in boundary conditions between each of the steps a PSA column undergoes, one can easily deduce that each change in boundary conditions requires a model reinitialization. A typical set of [Skarström, 1960] cycle boundary conditions is outlined in Figure 4.9.

Although the same set of differential equations is used throughout a PSA model, each of the pre-mentioned steps carry its own boundary and initial conditions. To shift from one to another set of boundary conditions, integration of the previous step is stopped and model equations are reinitialized to the new set of boundary conditions before resuming integration. Because transitions between boundary conditions occur within the time line, modellers don't usually think of the altering sequence of boundary conditions as a composite function. Nevertheless, it is a composite one. Taking component mass balance as an example, one can view the alteration of boundary conditions, at a specified vessel end, as a strip of time or state events. For example, the conditional statements in 4.52a and 4.52b (and their respective mathematical representations in 4.53 and 4.54) illustrate how boundary conditions change as a function of cyclic time (t_{cycle}) at each of the respective ends of the column. Boundary conditions switch between Nuemann and Robin throughout the cycle.

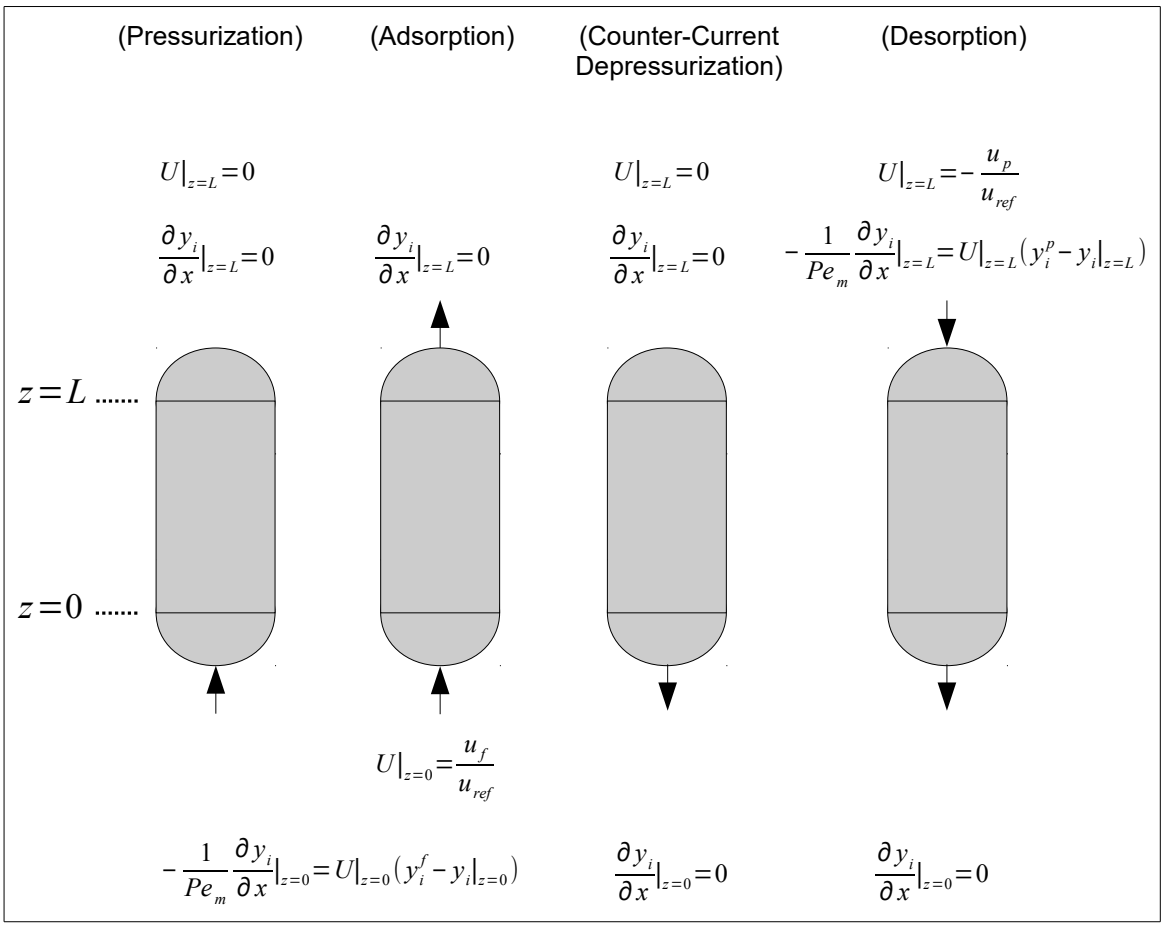


Figure 4.9: Velocity and component balance boundary conditions for each of [Skarström, 1960] PSA cyclic steps.

Similarly, the axial velocity vector initial condition value and location change as the active column step changes. The respective mathematical formulation of this composite function is illustrated in 4.55.

The PSA model was constructed using [gPROMS, 2012] modelling language. For the purposes of this work, I converted all state based transitions into time based transitions. For example, pressurization step concludes when the pressure of the PSA vessel reaches that of the feed. Since Pressure variation is modelled as function of time as outlined earlier, it becomes an easy task to calculate the time required for the pressure to move from a lower value to a higher one and thus replacing the state transition between pressurization and adsorption steps into a time transition. Thus, transitions between

boundary conditions are written as functions of time only as illustrated in equations 4.53, 4.54 and 4.55. The reason behind taking this course is that it facilitates the construction of a composite discretized boundary conditions function within gPROMS as illustrated in 4.52. It also prevents gPROMS from reinitializing variables when a state transition is encountered in a regularized boundary conditions model. These concepts will be discussed in the next chapter.

If ($t_{cycle} > 0$) and ($t_{cycle} \le t_{Pressurization}$)	*If* ($t_{cycle} > 0$) and ($t_{cycle} \le t_{Pressurization}$)
$-D_L \frac{\partial c_{A_i}}{\partial z}\Big\|_{z=0} = u\Big\|_{z=0}\left(c_{A_{if}} - c_{A_i}\Big\|_{z=0}\right)$	$-D_L \frac{\partial c_{A_i}}{\partial z}\Big\|_{z=1} = 0$
ElseIf ($t_{cycle} > t_{Pressurization}$) and ($t_{cycle} \le t_{Adsorption}$)	*ElseIf* ($t_{cycle} > t_{Pressurization}$) and ($t_{cycle} \le t_{Adsorption}$)
$-D_L \frac{\partial c_{A_i}}{\partial z}\Big\|_{z=0} = u\Big\|_{z=0}\left(c_{A_{if}} - c_{A_i}\Big\|_{z=0}\right)$	$-D_L \frac{\partial c_{A_i}}{\partial z}\Big\|_{z=1} = 0$
ElseIf ($t_{cycle} > t_{Adsorption}$) and ($t_{cycle} \le t_{De\text{-}pressurization}$)	*ElseIf* ($t_{cycle} > t_{Adsorption}$) and ($t_{cycle} \le t_{De\text{-}pressurization}$)
$-D_L \frac{\partial c_{A_i}}{\partial z}\Big\|_{z=0} = 0$	$-D_L \frac{\partial c_{A_i}}{\partial z}\Big\|_{z=1} = 0$
ElseIf ($t_{cycle} > t_{De\text{-}pressurization}$) and ($t_{cycle} \le t_{Desorption}$)	*ElseIf* ($t_{cycle} > t_{De\text{-}pressurization}$) and ($t_{cycle} \le t_{Desorption}$)
$-D_L \frac{\partial c_{A_i}}{\partial z}\Big\|_{z=0} = 0$	$-D_L \frac{\partial c_{A_i}}{\partial z}\Big\|_{z=1} = u\Big\|_{z=1}\left(c_{A_{if}} - c_{A_i}\Big\|_{z=1}\right)$
EndIf	*EndIf*
(4.52a). $BC_{z=0}(t_{cycle})$	(4.52b). $BC_{z=L}(t_{cycle})$

I should emphasise that conversion of state events into time events does not limit the applicability of the concepts that will be discussed in the next chapter. It just facilitates proving the concept when the modeller is not at liberty to alter the code of the simulation package.

$$\frac{\partial C_i}{\partial z}\Big|_{z=0} = f(t_{Cycle}) = \begin{cases} -(u|_{z=0}/D_L)(C_i^f - C_i|_{z=0}) & 0 \le t_{Cycle} \le T_{Pressurization} \\ -(u|_{z=0}/D_L)(C_i^f - C_i|_{z=0}) & T_{Pressurization} < t_{Cycle} \le T_{Adsorption} \\ 0 & T_{Adsorption} < t_{Cycle} \le T_{Depressurization} \\ 0 & T_{Depressurization} < t_{Cycle} \le Time_{DesorptionStep} \end{cases} \quad (4.53)$$

$$\frac{\partial C_i}{\partial z}\Big|_{x=L} = f(t_{Cycle}) = \begin{cases} 0 & 0 \le t_{Cycle} \le T_{Pressurization} \\ 0 & T_{Pressurization} < t_{Cycle} \le T_{Adsorption} \\ 0 & T_{Adsorption} < t_{Cycle} \le T_{Depressurization} \\ -(u|_{z=L}/D_L)(C_i^p - C_i|_{z=L}) & T_{Depressurization} < t_{Cycle} \le T_{Desorption} \end{cases} \quad (4.54)$$

$$u|_{z=0 \text{ or } z=L} = f(t_{Cycle}) = \begin{cases} u|_{Z=L} = 0 & 0 \le t_{Cycle} \le T_{Pressurization} \\ u|_{Z=0} = u_f & T_{Pressurization} < t_{Cycle} \le T_{Adsorption} \\ u|_{z=L} = 0 & T_{Adsorption} < t_{Cycle} \le T_{Depressurization} \\ u|_{z=L} = -u_p & T_{Depressurization} < t_{Cycle} \le Time_{Desorption\,Step} \end{cases} \quad (4.55)$$

Note that gPROMS immediately reinitializes state variables when transiting between each two consecutive cyclic steps as illustrated by the conditional statement in 4.56. Also, note how the entire simulation run time is converted into a sequence of repetitive steps in 4.56.

The question to be posed at this stage is whether immediate/instantaneous transition between boundary conditions, constitute a good modelling practice? Can we avoid reinitialization and yet achieve the same results? Can we avoid reinitialization and yet achieve better simulation results? I will answer these questions in the next chapter.

```
Cycle = 0
Repeat
        Cycle Time = 0
        If  ( Cycle Time >= 0 ) and ( Cycle Time <= Pressurization Time )
            Step = Pressurization; reinitialize model equations based on Pressurization BCs;
            run simulation for cyclic step time-span;
        ElseIf  ( Cycle Time > Pressurization Time ) and ( Cycle Time < Adsorption Time )
            Step = Adsorption; reinitialize model equations based on Adsorption BCs;
            run simulation for cyclic step time-span;
        ElseIf  ( Cycle Time >= Adsorption Time ) and ( Cycle Time <= Depressurization Time )
            Step = Depressurization; reinitialize model equations based on Depressurization BCs;
            run simulation for cyclic step time-span;
        ElseIf ( Cycle Time > Depressurization Time ) and ( Cycle Time < Desorption Time )
            Step = Desorption; reinitialize model equations based on Desorption BCs;
            run simulation for cyclic step time-span;
        Endif
        Cycle = Cycle + 1
Until (Cycle = Max Cycles) or ( | Y_Cycle − Y_Cycle-1 | <= Tolerance)
```

(4.56)

4.3. Concluding Remarks

In this chapter, I highlighted the discontinuities encountered in the developed reactor and PSA columns. For the reactor model, the discontinuity occurs as the reactor moves from laminar to turbulent flow region because of the increase in feed flow. The discontinuity affects the wall heat transfer coefficient. It is a two-dimensional discontinuity as Nusselt number is dependent on both Reynolds and Prandtl numbers.

In the PSA model, I demonstrated how the shift in boundary conditions from PSA cyclic sub-step to the other (e.g. pressurization to adsorption and adsorption to depressurization, etc) results in a one-dimensional discontinuity.

I also took the opportunity to present the ongoing work on the formulation and construction of the generic optimization of the PSA synthesis problem. I also introduced a novel method to properly model transient inlet velocity profile during the pressurization and depressurization of columns without introducing discontinuities (Appendix A).

CHAPTER 5:

Regularizing Discrete Functions

This chapter discusses regularizing discrete functions that were introduced in Chapter 3. I begin by introducing some of terminologies that will be adopted throughout the discussion. Then, I will discuss the resolution starting with univariate functions and, later, extending it to bivariate and multivariate functions. For all resolutions, I will discuss novel approaches to detection and resolution of discontinuities.

5.1. One-dimensional Functions

Let us assume that we have a composite function f that is defined by two separate sub-functions $f_1(x)$ and $f_2(x)$ that span two adjacent domains $[a', b]$ and $[a, b']$, respectively:

$$f(x)=\begin{cases} f_1(x), x\in[a', b] \\ f_2(x), x\in[a, b'] \end{cases} \tag{5.1}$$

For demonstration purposes, we will assume that $a > a'$. The ideal situation for the modeller is to have a continuous composite function across the entire simulation domain regardless of the sub-domains defining the respective sub-functions. To achieve this situation, the switch between f_1 and f_2 has to occur at a changeover (switch) location g satisfying the following condition (Figure 5.1a):

$$f_1(g)=f_2(g) \tag{5.2}$$

However, switch point g is seldom searched for, or even considered, when modelling. Instead the modeller usually opts for the selection of a point g' based usually on a widely adopted convention. A Reynolds number (Re) of 2300 is an example of a conventionally used break point between Laminal and Turbulent flows. If Re is below 2300, the flow is assumed Laminar. Otherwise, it is Turbulent. Such an arbitrary selection often raises a discontinuity between sub-domains at any arbitrary switch point g' as illustrated in Figure 5.1a. In such a case, the objective is to eliminate a discontinuity between two intersecting functions spanning overlapping domains. In this case, sub-functions intersect and functions' domains overlap. Thus, their exists a point g that satisfies equation 5.2.

When equation 5.2 is not satisfied, the sub-functions are said to be non-intersecting as illustrated in Figures 5.1b and 5.1c. For non-intersecting functions, there is usually a location g, along the dimension of the independent variable, that minimizes the distance between the two functions and hence allows for a smoother jump. Jumping between the

two functions at any point other than *g* would result in an extra effort by the integration routine to resolve the discontinuity. Thus, in such cases, the objective of this work is to minimize jump effort between two overlapping but non-intersecting sub-functions spanning overlapping domains as illustrated in Figure 5.1c. It should be noted that Figure 5.1b is a special case of Figure 5.1c where the intersection domain reduces to a single point. In such cases respective sub-functions' domains overlap ($a \leq b$). However, unlike the case in Figure 5.1a, sub-functions do not intersect.

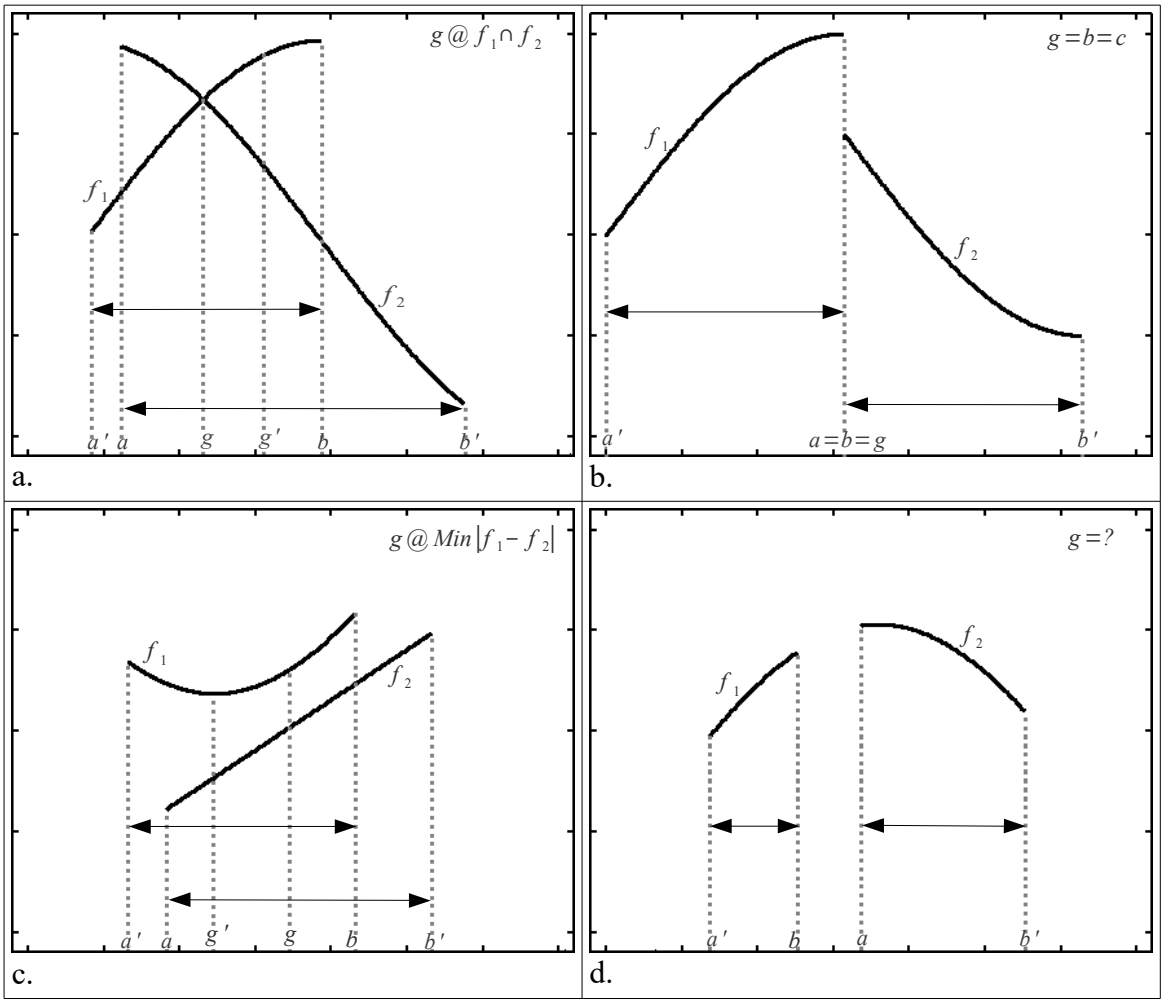


Figure 5.1: Forms of domain switch points between two functions and types of discontinuities between two adjacent domains.

The first objective of this work is to find the best switch point *g* for any given set of two overlapping sub-functions, whether intersecting or non-intersecting. The second objective is to eliminate discontinuities in non-intersecting sub-functions by devising an

interpolating polynomial at the location of the discontinuity between the two functions. To achieve both objectives, the method is decomposed into discontinuity detection and discontinuity resolution sub-problems.

5.1.1. *One-dimensional* Discontinuity Detection

First, we must sort the ranges for the respective sub-functions using their starting points in an ascending or descending order and then compare the location of the domain end of one function, (e.g. b for f_1), with the domain start of its successor, (e.g. a for f_2). If the end and start domain limits of two respective successive sub-functions are equal (i.e. $a=b$), the discontinuity is said to be non-overlapping. The point g is immediately identified for non-overlapping domains as $g=a=b$ *as* illustrated in Figure 5.1b. Sorting and comparison will also immediately detect if sub-functions f_1 and f_2 do not satisfy the continuity assumed for the main function f spanning [a', b'] as illustrated in Figure 5.1d. A resolution to removable discontinuities, such as that illustrated in Figure 5.1d, is presented in section 5.2.3.

Having identified an overlapping domain, to find g for overlapping discontinuous sub-functions, we will transform the problem into an optimization problem. As an example, the overlap domain for Figure 5.1a and Figure 5.1c is [a,b]. We define an error function as:

$$e(x)=\left|f_1(x)-f_2(x)\right| \tag{5.3}$$

Our objective is to find a point g that minimizes $e(x)$ over the domain [a,b]. It can be argued that the use of the absolute function will alter the convexity of the objective function as illustrated in Figures 5.2a and 5.2b. However, it should be noted that, in this problem, the objective is to search for $e(x)=0$, not the minimum $e(x)$. The problem is a

root finding problem, not an optimization one. Thus, using the absolute value function helps formulating a better solution in this case. If the value of *e(x)* is always above zero, the optimisation algorithm will report the optimum *x* that corresponds to the minimum *e(x)*. Even if the function contains multiple zeros (Figure 5.2c), locating one of the zeros is sufficient for the search algorithm to succeed. Nevertheless, since the absolute value function is not differentiable at sign-change locations, it will introduce problems when used by optimization routines. A better differential function that achieves the same objective is the square function (illustrated in Figure 5.2d):

$$e(x)=[f_1(x)-f_2(x)]^2 \tag{5.4}$$

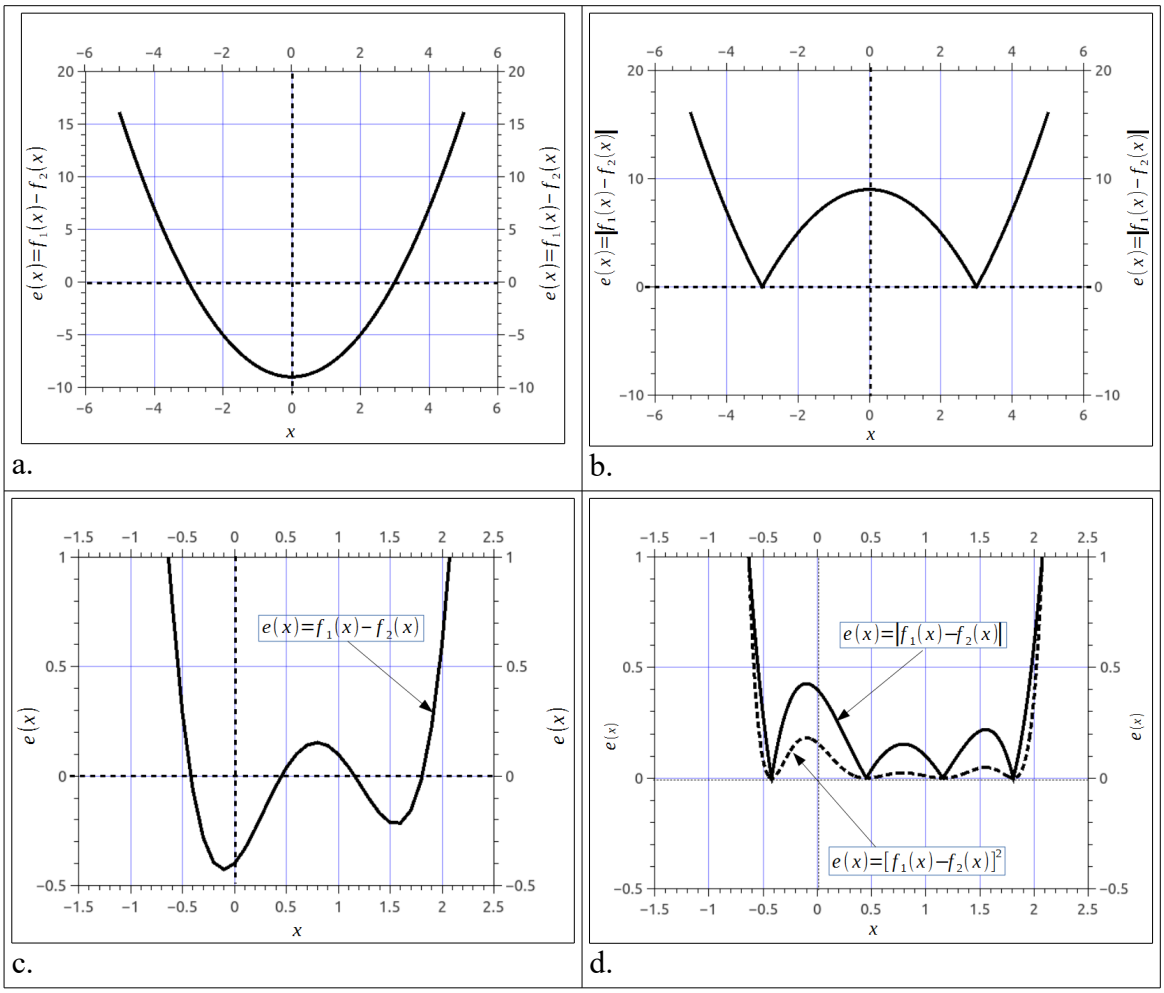


Figure 5.2: Behaviours of various error (difference) functions *e(x)*.

The advantage of formulating the problem as an optimization problem instead of a root-finding one is that the optimum will always return a value whether roots are available or not. When roots are available, the minimum resembles that of Figure 5.1a (i.e. $e(g) = 0$). When roots are not available, the minimum resembles one of the cases illustrated in Figures 5.1b and 5.1c. Since this is a fairly simple optimization problem, it can be solved using any of the commercially available optimisation routines.

Once g is detected, it can be immediately inserted into the conditional statement of the composite function; replacing any arbitrary selected g' by the modeller. For example, if the detection algorithm resulted in locating a minimum jump effort point g between two discontinuous functions f_1 and f_2, g can easily be inserted into the final conditional statement as illustrated in (5.5):

If (x<g) then	(Domain I)		*If (x<=g) then*	(Domain I)	
$f = f_1(x)$		*or*	$f = f_1(x)$		(5.5)
Else if (x>=g) then	(Domain II)		*Else if (x>g) then*	(Domain II)	
$f = f_2(x)$			$f = f_2(x)$		

For cases where sub-functions intersect and overlap (Figure 5.1a), a discontinuity detection algorithm is sufficient to grant at least smooth continuity between the discontinuous functions but not their respective first and second derivatives. For cases where functions touch or overlap but do not intersect (Figures 5.1b and 5.1c, respectively), discontinuity detection algorithm might be sufficient if the simulation integrator routine is able to jump between the functions without the need for reinitializing the state variables. As indicated by [Borst, 2008], resolution of discontinuity using Type I discontinuity handlers might not always be appropriate because of the exhaustive need to reinitialize state variables and the fact that, in some cases, re-initialization might alter the solution path. Thus, I propose a discontinuity resolution algorithm to avoid falling into

state-variable re-initialization.

5.1.2. One-dimensional Discontinuity Resolution

Discontinuity resolution takes the form of bridging the two discontinuous domains through an interpolating polynomial, f_3. Linear interpolation requires at least two points. However, we will attempt to link functions using a smooth interpolating polynomial preferably to the 3rd degree. Linking functions with a third degree interpolating polynomial ensures continuity up to the second derivative of the interpolating function. To construct any smooth polynomial, we need at least three points. One would think that three points are sufficient to construct the polynomial around the discontinuity point. However, as we will demonstrate later, at least four points are required in order to minimize first and second derivatives' discontinuities at the junction points between the interpolating polynomial f_3 and the corresponding discontinuous sub-functions f_1 and f_2.

To simplify computations, I will evenly separate the points by an interval h from each other. Their exact locations will be relative to the location of the discontinuity location (g) in the independent variable dimension. The location of the mesh control points, relative to g, takes one of three forms depending on its location within the overlap domain [a,b]:

- If a minimum $g \in (a, b)$ exists, mesh control points will be respectively located at distances $g-1.5h$, $g-0.5h$, $g+0.5h$ and $g+1.5h$ as illustrated in Figure 5.3a. This selection of points' locations ensures even distribution of the interpolating points on both sides of the point g.
- If the minimum $g \notin (a, b)$, then g must reside at one end of the domain. If g is located at the start of the overlap domain ($g=c$), mesh control points will be respectively located at g, $g+h$, $g+2h$ and $g+3h$ as illustrated in Figure 5.3b.

- If g is located at the end of the overlap domain ($g=b$), mesh control points will be respectively located at g, $g-h$, $g-2h$ and $g-3h$ as illustrated in Figure 5.3c.

To perform a smooth transition, we need at least one point to lie on each of the functions' curves at the respective sides of the discontinuity location. Let us call these points *point 1* and *point 2*. Taking Figure5.3a as an example for the case where $g \in (a, b)$, the respective locations of points 1 and 2 will be ($g-1.5h$, $f_2(g-1.5h)$) and ($g+1.5h$, $f_1(g+1.5h)$), respectively. Of course, one can argue that we could also position the points at ($g-1.5h$, $f_1(g-1.5h)$) and ($g+1.5h$, $f_2(g+1.5h)$). However, we should bear in mind that the sorting algorithm, explained earlier, decides on the order of the functions based on their span over the independent variable dimension.

For the case where g is located at the start of the overlap domain ($g=a$), the respective locations of points 1 and 2 will be (g, $f_2(g)$) and ($g+3h$, $f_1(g+3h)$). For the case where g is located at the end of the overlap domain ($g=b$), points 1 and 2 will be located at ($g-3h$, $f_1(g-3h)$) and (g, $f_2(g)$), respectively. Respective examples of both cases are illustrated in Figures 5.3b and 5.3c. Of course, the sorting algorithm argument still holds.

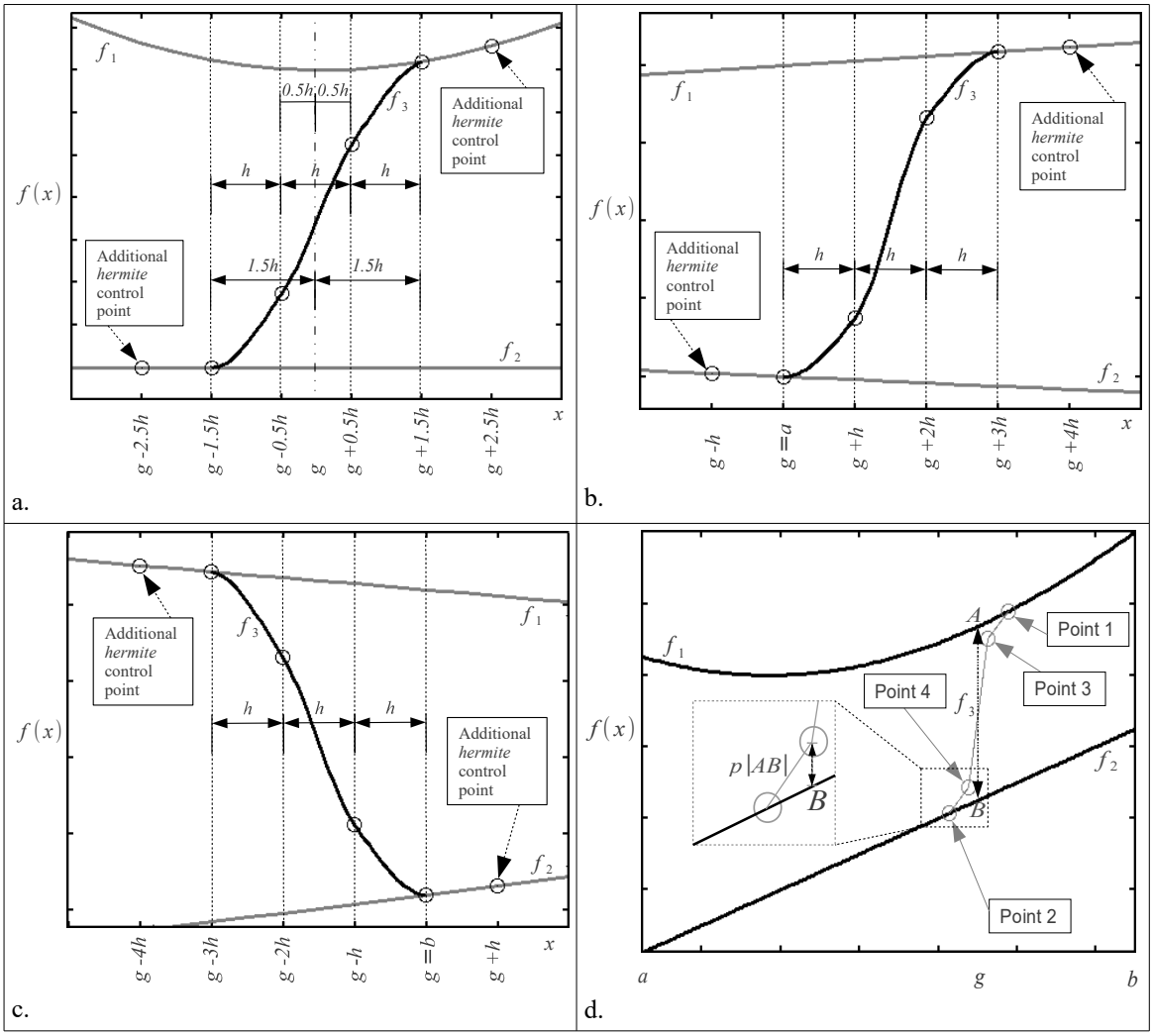


Figure 5.3: Location of mesh control points relative to the minimum jump-effort point *g*.

For the last two points (*points* 3 and 4), of the four point set, we utilized the length of the line segment |AB|, defined by equation 5.6 and illustrated in Figure 5.3d, to shift f_3 function values at these points from the respective discontinuous functions values. Since the location of *g*, on the independent variable dimension, corresponds to the point that exhibits minimum distance between the two functions f_1 and f_2 within the overlap domain, the length of the line segment $|AB|$ corresponds to that minimum distance.

As an example, let us take the case where $g \in (a, b)$. The *y*-axis values of the points located at distances *-0.5h* and *+0.5h* from the point *g* will be calculated as the values of the functions at these respective points after adding or subtracting a fraction *p* of $|AB|$.

For lower valued functions (e.g. f_2), Point 3 would have the coordinates $(g-0.5\mathrm{h}, f_2(g-0.5\mathrm{h})+p|AB|)$. For higher valued functions (e.g. f_1), Point 4 would have the coordinates $(g+0.5\mathrm{h}, f_1(g+0.5\mathrm{h})-p|AB|)$.

$$|AB|=|f_1(g)-f_2(g)| \tag{5.6}$$

[Fritsch and Carlson, 1980] detail the necessary and sufficient conditions to ensure monotonicity of the interpolating polynomial control points. Basically, they prove that in order to ensure a monotonically increasing or decreasing function, slopes of control points should have the same sign or a value of zero. To emphasise the same concept, the value of p should satisfy the condition in 5.7:

$$0 \le p \le 0.5 \tag{5.7}$$

Naturally, providing a separate p value for each of the functions f_1 and f_2 would add to the degrees of freedom as long as they satisfy the condition in (5.7). These two p values can act as tuning parameters to smooth the transition between f_3 and the discontinuous sub-functions f_1 and f_2. In addition, the original formulation of *hermite* interpolating polynomials (to be discussed later) uses a tension parameter (t) that extends between 0 and 1. We could use either t or p to perfect the resulting interpolation curve. However, we intend to keep both parameters in order to smooth the transition between the interpolating polynomial and the discontinuous sub-functions.

To demonstrate the effect of the interpolation algorithm on the conditional statement, let us consider the case in (5.5) and assume $g \in (a, b)$. After generating the four-point interpolating polynomial, the logical statements in (5.5) will be transformed into (5.8).

We should also note that, because of the uniqueness of the solution for one-dimensional functions, the devised procedure can be run off-line prior to the start of the simulation

run. Indeed, we recommend embedding the algorithm into the modelling language compiler to automate generation of polynomials and their respective additional conditional expressions.

If (x<g-1.5h)	(Domain I)	(5.8)
f = f_1(x)		
ElseIf (|x-g|≤1.5h)	*(Interpolating polynomial Domain)*	
f = f_3(x)		
ElseIf (x>g+1.5h)	(Domain II)	
f = f_2(x)		
EndIf		

The algorithm can be extended to account for complex conditional statements such as (5.9) by solving *w(x)* for *x*. It can also be extended to account for complex logical expressions involving logical operators $\wedge$ or $\vee$.

If (w(x) ≤ 0)	(Domain I)	(5.9)
f = f_1(x)		
ElseIf (w(x)>0)	(Domain II)	
f = f_2(x)		
EndIf		

Lastly, an additional side benefit resulting from the use of the line segment |*AB*| to locate the intermediate points at *g-0.5h* and *g+0.5h* is that the locations of these points automatically coincide with the locations of the respective sub-functions f_1 and f_2 if f_1 and f_2 posses a common intersection point since |*AB*|=0 in this case regardless of the value of *p*. This benefit indicates that detection and resolution algorithms can be integrated seamlessly without the need to treat intersecting sub-functions separately. Figure 5.4 illustrates the resulting interpolating polynomial linking two intersecting sub-functions.

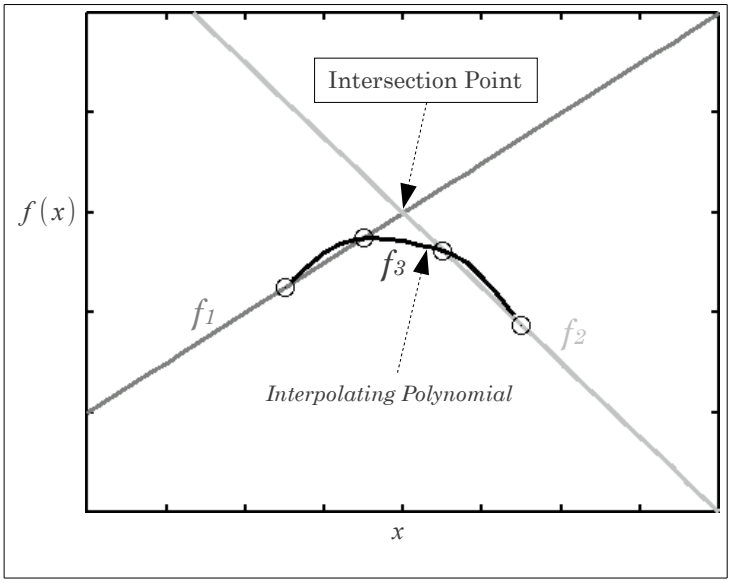


Figure 5.4: A four-point *hermite* interpolating polynomial between two intersecting unidimensional functions using *tension (t)*=0.

5.1.3. Perfecting the Connection and the Bounding Box Problem

The smoothing of the transition between the interpolating polynomial and the discontinuous functions can be transformed into an optimization problem that minimizes first or second derivative differences between the interpolating polynomial and the discontinuous functions at Point 1 and Point 2. The optimization problem can be formulated as:

$min:[f'^{-}_{P1}-f'^{+}_{P1}]^2+[f'^{-}_{P2}-f'^{+}_{P2}]^2$ $s.t.=\begin{cases} 0\le p_i<0.5 \\ 0\le t\le 1 \end{cases}$	$min:[f''^{-}_{P1}-f''^{+}_{P1}]^2+[f''^{-}_{P2}-f''^{+}_{P2}]^2$ $s.t.=\begin{cases} 0\le p_i<0.5 \\ 0\le t\le 1 \end{cases}$
a. first order derivative optimization	b. second order derivative optimization

(5.10)

If the derivatives of the discontinuous sub-functions, appearing in the cost function, are readily available, they can be directly evaluated through the available expressions. Otherwise, any derivative estimation numerical technique (e.g. secant method) can be used to evaluate the required derivatives.

Once the position of the points is determined, we need to connect them with a continuous

interpolating function that is preferably 2nd order smooth to aid in calculation of Jacobian and Hessian matrices when required by the numerical ODE/DAE solver. Two interpolation methods satisfy our criterion: cubic splines and cubic *hermite* interpolating polynomials (Appendix C). However, we selected *hermite* interpolating polynomials for the following reasons:

1. For the same set of interpolating points, cubic spline interpolating polynomials exhibit more overshoot than their cubic *hermite* counterparts [Fritsch and Carlson, 1980].

2. Cubic *hermite* interpolating polynomials have one more degree of freedom to better control the shape of the interpolating polynomial ([Kochanek and Bartels, 1984] and [Bartels et al, 1987]). This degree of freedom is granted by the extra tension parameter (t). As the name implies, t is roughly a measure of how stretched or lose is the connecting polynomial between the mesh control points. Assuming that mesh control points are connected through a thread, a t=0 indicates a loose thread while a t=1 indicates a tightly wrapped thread. I encourage using *hermite* interpolating polynomials for the extra degree of freedom they provide. The discussion from this point onward assumes the utilization of *hermite* interpolating polynomials. *Hermite* interpolating polynomials are discussed in Appendix C.

Nevertheless, the reader should note that *hermite* interpolating polynomials require two additional mesh control points over cubic splines as illustrated in Figures 5.3a, 5.3b and 5.3c. Interpolation will still occur between the four control points discussed earlier. The two additional control points only aid in forming the shape of the curve.

Let us now turn our attention to an issue that will further constrain the value of the *p* parameter. At lower values of p_i and/or tension parameter (t), the bounds of the interpolating polynomial tends to cross the maximum function boundaries set by the control points as illustrated in Figure 5.5. This situation might not create an issue for most discontinuous functions. However, certain types of discontinuous functions mandate proper bounding of interpolating polynomial to the upper and lower limits set by the control points. For example, if x denotes valve opening and $f(x)$ represents flow, then it would not be expected for the flow to arrive at its maximum value until valve opening reaches 100% (x=1). An interpolating polynomial that is not properly bounded will result in the undesirable situation leading to either a maximum flow before reaching 100% valve opening or worse leading to a negative flow before the valve is fully closed. This problem is known as the *bounding-box* problem in computer-graphics literature [Filip et al, 1986].

To resolve the problem, we need to bound the maximum and minimum values of the interpolating polynomial to the values set by control points 1 and 2 so that:

$$f_1(x_{P_1}) \le f_3(x) \le f_2(x_{P_2}) \qquad for \qquad x_{P_1} \le x \le x_{P_2} \tag{5.11}$$

The solution to the problem comes straight forwardly from calculus. To do so, the optimization routine needs to identify the maximum and minimum values of $f_3(x)$, compare them to those of control points 1 and 2, and finally, reject or accept the pair of (p_i, t_i) values based on adherence to condition 5.11.

5.1.4. Are four control points enough?

The discussion, so far, has assumed that we need at least four points to properly interpolate. However, we need a good justification to favour four points over three or five. This can be demonstrated by considering the plots of the *hermite* interpolating polynomial

for three, four and five interpolating points shown in Figures 5.5a, 5.5b and 5.5c.

When using a three-point interpolating polynomial, two of the points lie on the respective discontinuous functions. The *x* coordinate of the third point corresponds to the minimum jump effort location (*g*). The only degree of freedom available to tune the curvature, excluding the *hermite* tension parameter, is through the manipulation of the function value at the minimum jump effort point *g*. I varied p/|AB| values from 0 to 0.5 relative to f_1 and f_2 in the upper and lower sections of the figure, respectively. As illustrated in Figure 5.5a, the drawback of a three-point interpolating polynomial is that it always favours better closure towards one of the discontinuous functions over the other.

For the case of four control points, I omitted the *g* point and relied only on two points separated by a distant *h* from each of the sides of the minimum jump effort location *g*. The interpolating function values, at the junctions with f_1 and f_2 are fixed at the values of their respective functions f_1 and f_2. I used equal values of *p* to distance interpolating function values at points 4 and 5. Thus, one degree of freedom is remaining (again excluding *hermite* tension) to smooth the transition between the interpolating polynomial f_3 and the functions f_1 and f_2, namely *p*. The common intersection point between all generated curves is purely curvature related and has no relation to the *g* point discussed earlier.

For the case of five control points, I made use of the minimum jump effort location (*g*) to add the fifth point. The value of the interpolating polynomial f_3, at this point, is calculated and fixed at the mean of the two discontinuous functions f_1 and f_2 (i.e. $f_3(g) = 0.5[f_1(g) + f_2(g)]$). The values of the control points at the junctions with f_1 and f_2 are assigned the respective values of the functions. The values of these two points are also fixed. I also used constant values of *p* to distance the points located at *g*-*h* and *g*+*h* from their

respective functions f_1 and f_2. The resulting interpolating values of $p/|AB|$ ranging from 0 to 0.5 are plotted in Figure 5.5c.

The resulting curves for four-point interpolating polynomials (Figure 5.5b) provide similar degrees of curvature to those obtained using five-point interpolating polynomial (Figure 5.5c). Thus, we may comfortably conclude that a four-point interpolating polynomial is sufficient to provide good closure between the interpolating polynomial and the discontinuous functions.

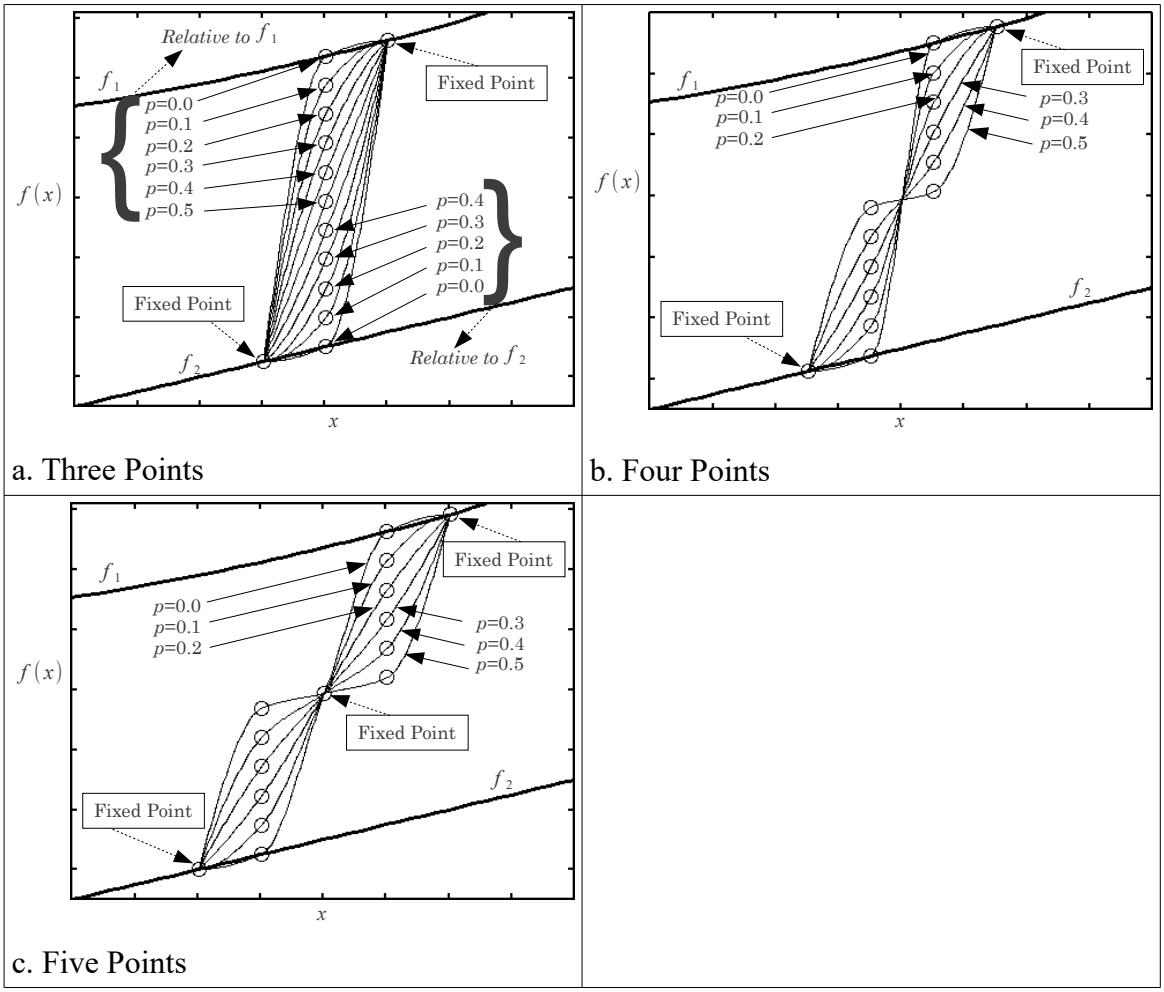


Figure 5.5: Comparison between 3, 4 and 5 control points using a *hermite* interpolating polynomial with various p values.

5.1.5. Regularizing boundary and initial conditions

Discontinuities in boundary conditions usually take the form presented in Figure 5.1b (i.e.

g=a=b). Because the overlap domain is so small, any regularization will force f_3 to lie outside the overlap region. Moreover, since the switch between value reported by each side of the conditional statements (f_1 to f_3) or (f_2 to f_3) can be state variable or time dependent, we cannot evenly distribute f_3 span between f_1 and f_2. Even distribution could violate state variable dependency. Thus, the solution would be to insert an additional time interval to accommodate f_3 between f_1 *and* f_2. This makes sense since the set of boundary conditions at the overlap region does not coincide with any of the sets of boundary conditions belonging to the either of the discontinuous sub-functions.

Regularizing the form in Figure 5.1b can take one of the forms in Figures 5.3a-c. Using the forms presented in Figure 5.3a and 5.3c would require calculation of more control points at locations before f_3 (points at the left side of the *g* point when replacing the *x*-axis with a time axis). The use of the form presented in Figure 5.3b reduces the number of points located to the left of the *g* point to only one point, namely the additional control point required by the *hermite* interpolating polynomial.

I should mention that accurate estimation of the value of the state variable at this point is not very important. This is due to the fact that the additional *hermite* control points are used to adjust the shape of the resulting curve bounded by the four points discussed earlier. The algorithm would work with any arbitrary value of the state variable at that point. However, accurate determination provides a better initial interpolation curve. After optimizing the shape of the curve through (5.10), the final curve would have better closure at both ends of the interpolation region than a curve optimized with an arbitrary selection of the additional *hermite* control point.

To accurately calculate the value of f_1 at this point, the integrator needs to pass through the control point and record a snapshot of the boundary condition values at that point. For

time events, the event can be marked in the integrator time-line. For state events, the integrator needs to switch to the branch of the conditional statement containing the regularization function before realizing the existence of a shift in boundary conditions. Then, it needs to return back an interval h in time to record the snapshot. In both time and state event cases, such approaches add an extra unnecessary burden on the integrator. To mask the problem from the integrator, I allowed the integration routine to freely control integration step-size while taking snapshots of the time steps taken by the integrator. Once the regular expression shifts to the regularizing function, the location of that *hermite* control point is calculated through approximating past integration steps with an interpolating polynomial.

The concept is illustrated in Figure 5.6. The past points (diamond-shaped) along with the g-point (intersecting f_2 with the interpolating polynomial) are used to estimate the value of $f_2(g\text{-}h)$. In the figure, four interpolating points are used to generate a third degree interpolating polynomial that past-interpolates to find $f_2(g\text{-}h)$. Of course, we could have used a *hermite* interpolating polynomial to perform the same task. However, there is no added benefit in using *hermite* polynomials for past interpolation as no tight control over the estimation of function value is required. In this case, a *hermite* interpolating polynomial would unnecessarily increase computational power.

Although computationally exhaustive, I think this approach provides a better estimation of the past value of the state variable. To avoid such computations, we can assume the value of the state variable at the left *hermite* control point to be equal to that at the g point. This assumption is used to calculate the additional *hermite* control point located to the right side of the g point.

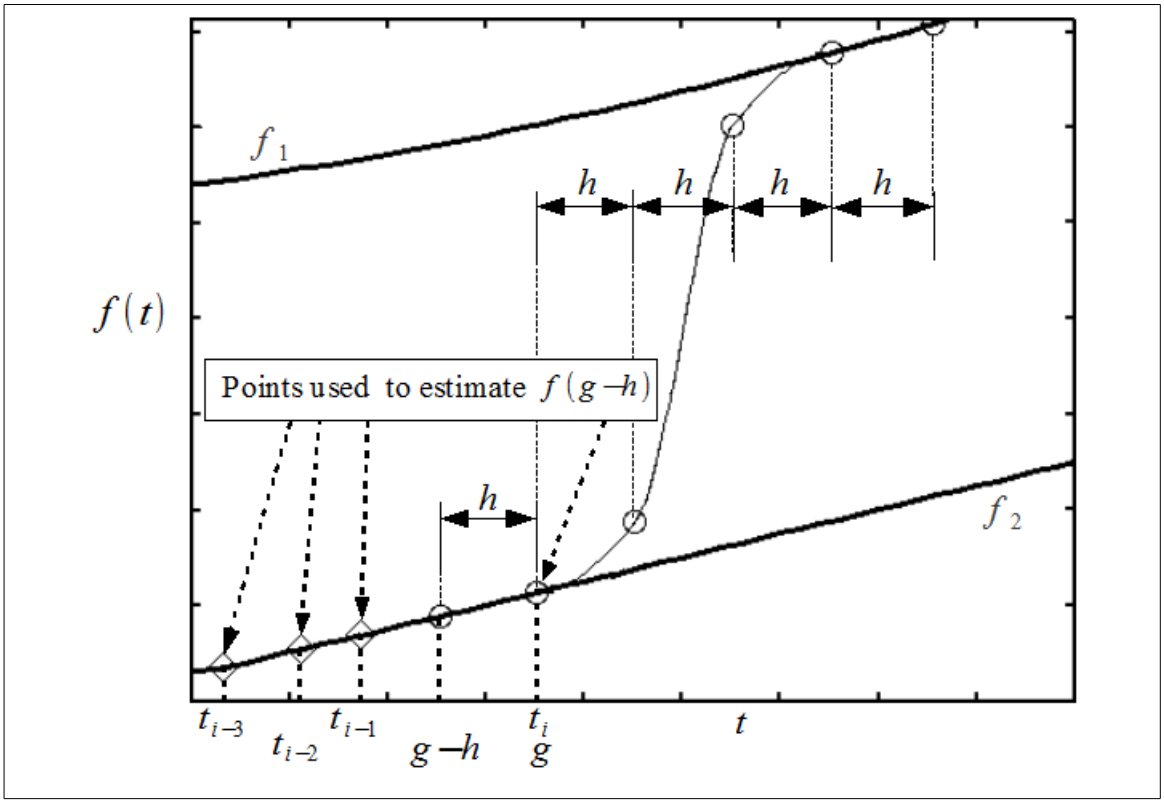


Figure 5.6: Past interpolation points at t_{i-1}, t_{i-2} and t_{i-3} in addition to the g point at t_i are used to estimate the value of f_2 at $g-h$.

5.1.6. Regularizing conflicting boundary conditions

The main reason behind conventional initialization of variables at a discontinuity is the large change in one or more of the state-variables. The change is usually larger than the accepted value of the tolerance set by integration routine. The large change is sometimes a direct result of a conflicting boundary conditions between the two discontinuous functions. An example of such conflict is the sudden changes in flow or flux directions between the one set of boundary conditions and its neighbouring one.

Conflicting boundary conditions arise when the boundary conditions before reinitialization of variables conflict with those after reinitialization. A discontinuity in a boundary condition resulting from flow reversal can be regarded as a conflicting boundary condition. The flow before the discontinuity occurs in one direction. After the discontinuity, the flow direction reverses.

The problem that arises with regularizing conflicting boundary conditions is that the developed algorithm cannot directly move from one boundary condition to the other without stumbling in the middle and eventually failing. Taking the example of flow reversal at the discontinuous region, we can realize that one set of boundary conditions is mandating the flow to move in one direction while the other set is asking it to move in a counter-current direction to the first set.

Reinitialization of variables resolves the conflict by simply ignoring past boundary conditions and focusing only on the present boundary conditions. However, such a resolution introduces an error into the model as it assumes that flow reversal happened exactly at the start of the discontinuity. Reinitialization assumes the existence of no intermediate transition region.

The solution to such regularization problems lies in breaking the discontinuous region into two regularized regions that share a common interchange point. This common interchange point is hopefully physically realizable. For example, before a flow reverses its direction, it needs to move from a positive or negative flow to a point were the fluid is stagnant. This stagnation point is a good transition point between the two sets of boundary conditions as the point belongs to both sets of boundary conditions.

The concept is best understood with an example. In section 4.2, I detailed the general layout of a discretized PSA model. Components boundary conditions of the model are illustrated in Figure 4.9. One-interval regularization between the two steps is illustrated in Figure 5.7. Note how the direction of spatial flux for the component mass balance changes from Desorption step to Pressurization step. In the Desorption step, velocity and component fluxes move in a direction that is counter-current to that of the Pressurization step. Trying to directly bridge the discontinuity at the two boundaries (z=0 or x=L) using

one regularization interval results in a regularizing function having a negative flux at one end while exhibiting a positive one at the other end. This situation leads to a solver instability and eventually results in the solver failing to integrate. Indeed, the solver should not integrate such a scenario as it is not physically realizable.

Looking deep into the process, it can easily be realized that the boundary discontinuity is summing two process actions. At the end of Desorption step, the purge valve starts closing. After the purge valve is completely closed, the feed valve is opened and feed is introduced at high pressure signifying the start of the Pressurization step. So, effectively, the discontinuity is compacting two process actions in an instantaneous time point.

Two regularization intervals are required to resolve this problem. The first regularization interval closes the purge valve, effectively moving the flow and its respective component mass fluxes from their negative direction to an intermediate stagnant point where there is no flow in any of the directions. The flow and component fluxes then start moving into the positive direction with the opening of the feed valve. The two-interval regularization concept is illustrated in Figure 5.7. Also, two-interval regularization between Desorption and Pressurization steps is illustrated in Figure 5.9.

As I outlined, the two-interval regularization solves the problem. However, it comes at an expense. It is not an easy task for an algorithm to decide whether a discontinuity requires one- or two-interval regularization. Until a future algorithm is devised to tackle such a limitation, it becomes the task of the modeller to point the number of regularizations required per a discontinuity to the modelling language. In addition, for a two-interval regularization, the modeller needs to define the intermediate point that is shared by both regularization intervals and define its corresponding boundary conditions.

5.1.7. Differential models embedding other models

Complex models usually combine boundary conditions, initial conditions and constitutive equations. The model in such cases is built from different layers. However, as outlined in Chapter 3: , the integrating routine focuses only on the layer that it immediately integrates through. This is the layer at which model state variables are integrated with respect to an independent variable such as time. Other model layers are normally overlooked by conventional integrators.

For example, in the PSA model outlined in section 4.2.1, velocity distribution is a function of the spatial dimension and not the temporal one. The distribution is modelled as an initial value problem in space only although a small time contributing factor is evident from the component adsorption term. Thus, to a conventional integrating routine, velocity distribution does not exist and hence will not be regularized unless pointed out through any mean by the modeller to the integrating routine implementing the regularization algorithm. Moreover, the fact that the location of the initial conditions for velocity (whether at x=0 or at x=1) is a process step dependent (refer to Figure 4.9) adds to the complexity of the situation.

It is an easy task for a modelling language/algorithm to identify the state variables in a model. This easiness facilitates the insertion of appropriate state-variable regularization algorithms. However, this is not the case with embedded models since these models are transparent to the modelling language. Some of the embedded models might require regularization. Others might not. Thus, when regularizing models, modelling languages should provide the modeller the option to select which of the embedded models to regularize along with model state variables and which to ignore.

Unless the integration routine is clever enough (normally not) to realize the existence of

embedded constitutive equations within the model that require regularization, it becomes a difficult task for it to regularize these embedded equations. Currently, very little information is exchanged between the model and the integrating routine (refer to Figure 2.3). I think this problem marks a good direction for continuing research on this subject.

5.2. Two-Dimensional Functions

So far, we have discussed tackling the problem for one dimensional functions. What if *z* is a function of two variables (e.g. $z = f(x,y)$), where *z* poses one or more discontinuities along each of the dimensions. The discontinuous function may take a form like:

$$f(x,y)=\begin{cases} f_1(x,y), x\in[a'_x, b_x], y\in[a'_y, b_y] \\ f_2(x,y), x\in[a_x, b'_x], y\in[a_y, b'_y] \end{cases} \tag{5.12}$$

Assuming $a'_x < a_x \le b_x < b'_x$ and $a'_y < a_y \le b_y < b'_y$ (Figure 5.10a), if g'_x and g'_y are arbitrary selected as discontinuity boundaries along the *x* and y dimensions, respectively, a possible pseudo code of (5.12) could be written as either of the forms in (5.13).

```
If ( a'x < x < g'x )
        If ( ay < y < b'y)
                f(x,y) = f1(x,y)
        ElseIf ( ax < x ) and ( a'y < y <ay )
                f(x,y) = f2(x,y)
        EndIf
ElseIf ( g'x < x < b'x )
        If (a'y < y < by)
                f(x,y) = f2(x,y)
        ElseIf [(x < bx) and (by < y < b'y)]
                f(x,y) = f1(x,y)
        EndIf
EndIf
```

(5.13a).

```
If (a'y < y < g'y )
        If ( a'x < x < ax ) and ( y > ay )
                f(x,y) = f1(x,y)
        ElseIf [( ax < x < b'x )]
                f(x,y) = f2(x,y)
        EndIf
ElseIf ( g'y < y < b'y )
        If ( a'x < x < bx )
                f(x,y) = f1(x,y)
        ElseIf [(bx < x < b'x) and ( y < by)]
                f(x,y) = f1(x,y)
        EndIf
EndIf
```

(5.13b).

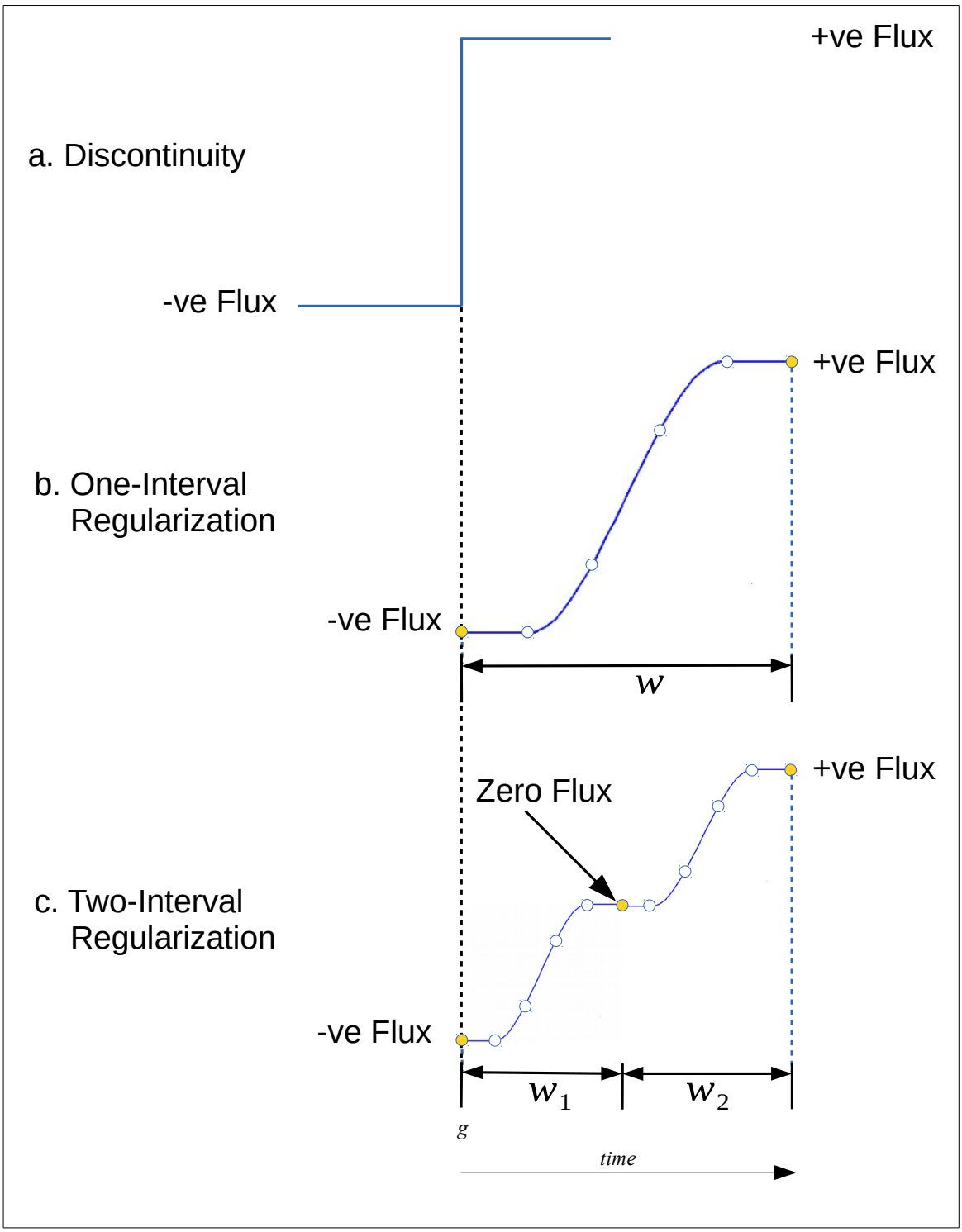


Figure 5.7: One- and two-interval regularizations of a conflicting boundary discontinuity.

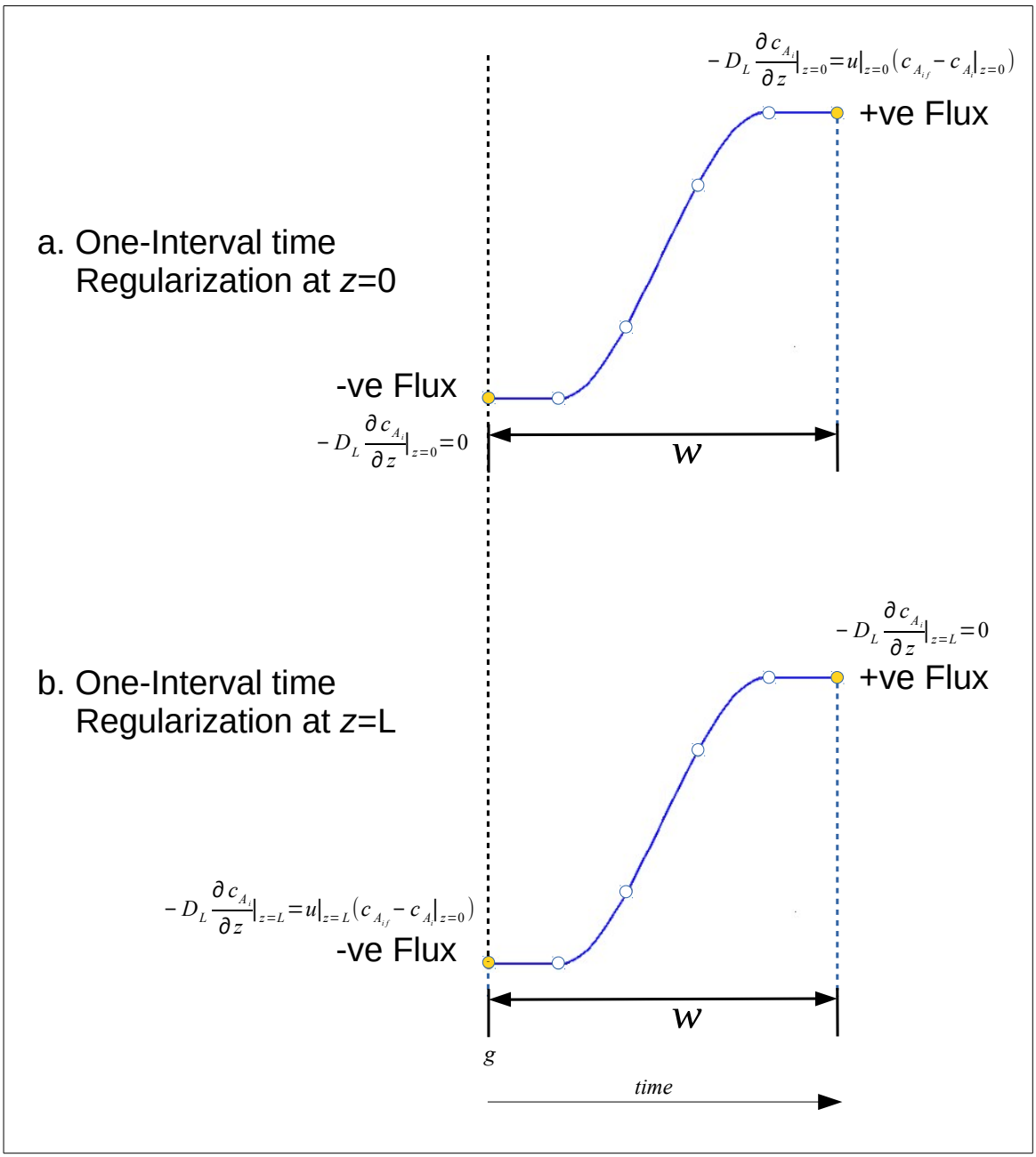


Figure 5.8: One-interval regularization of the conflicting boundary discontinuity between Desorption and Pressurization steps in a PSA unit.

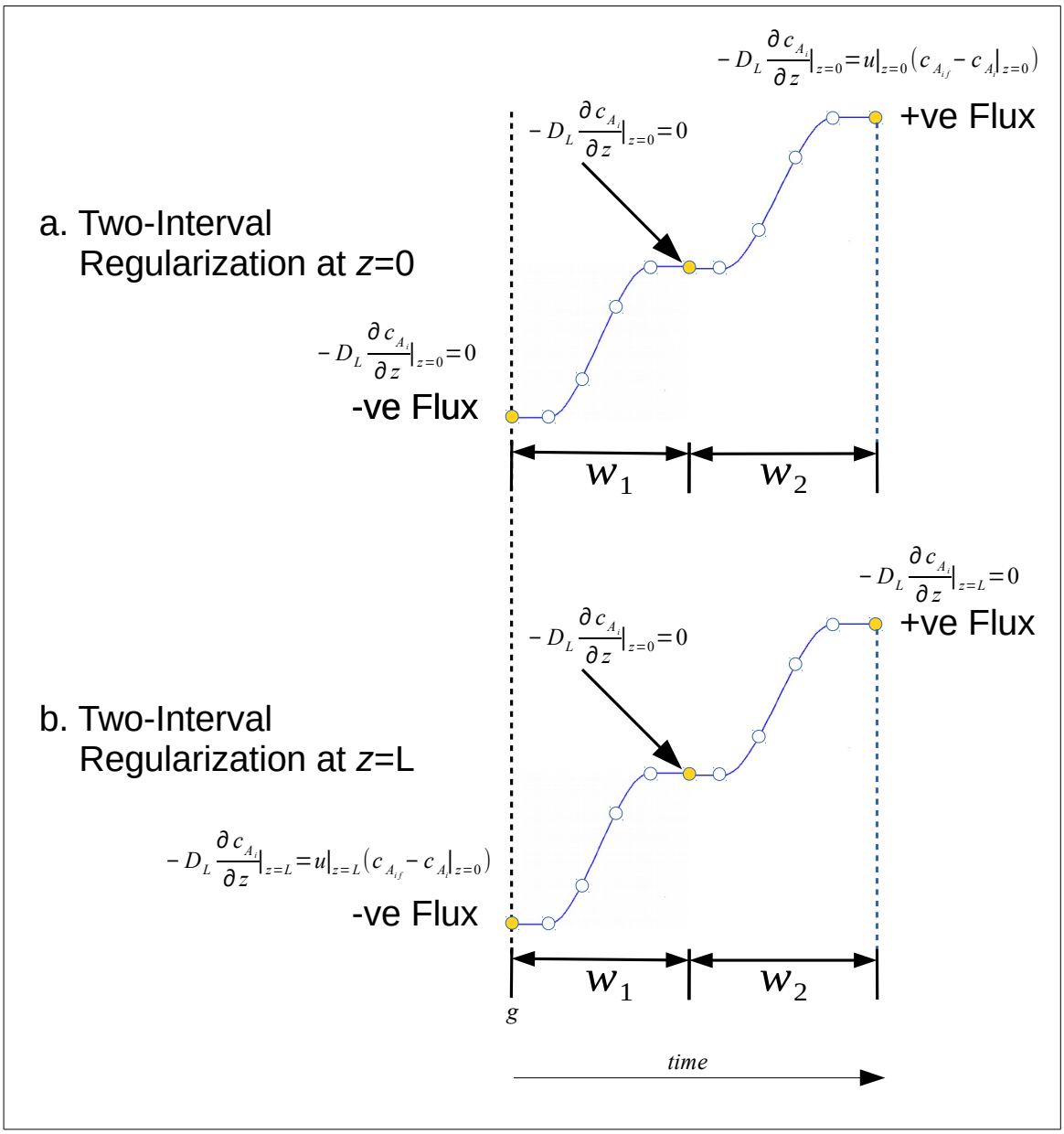
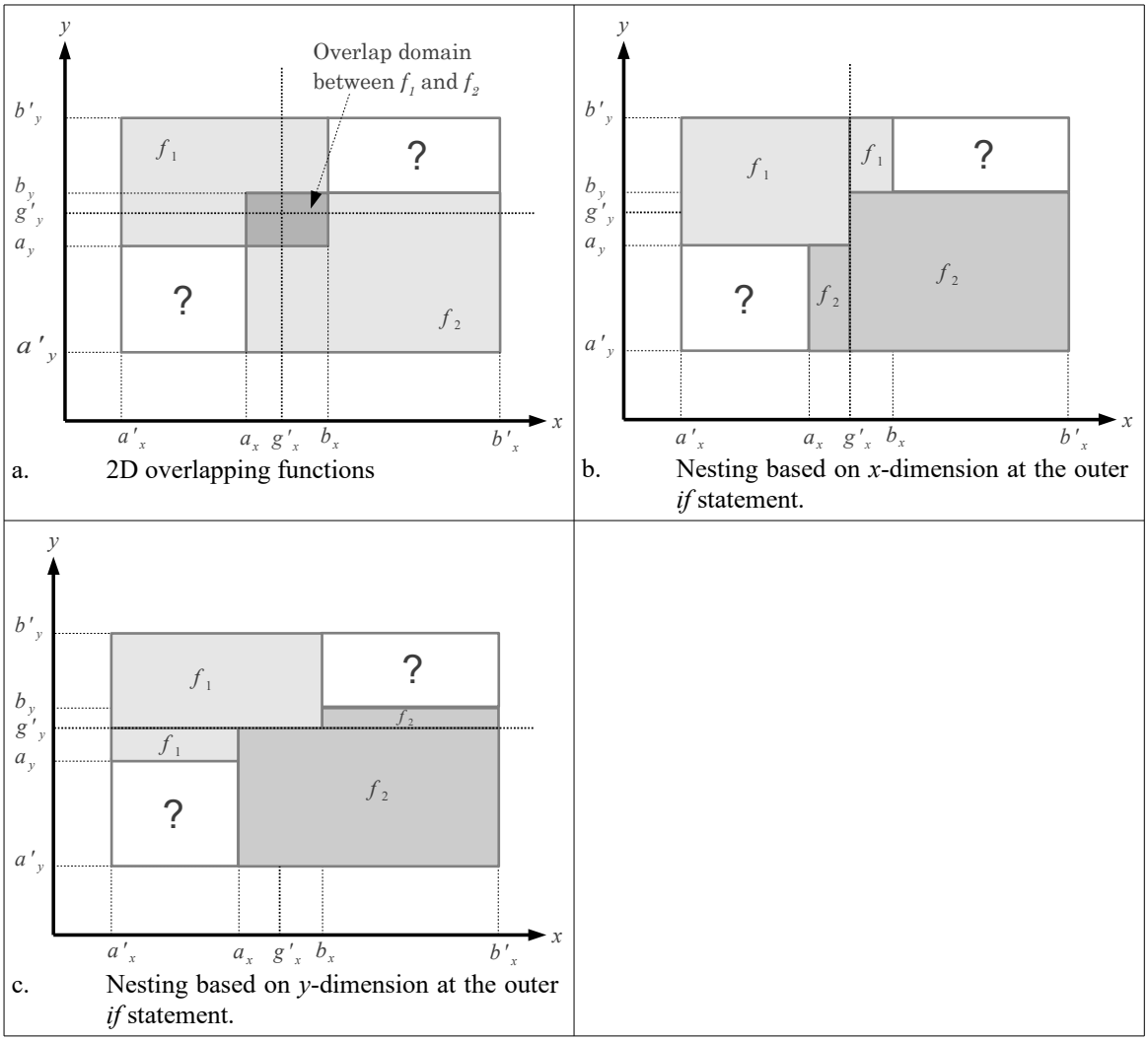


Figure 5.9: Two-interval regularization of the conflicting boundary discontinuity between

Desorption and Pressurization steps in a PSA unit.When dealing with two dimensional relations, discontinuities present themselves as planes as illustrated in Figure 5.10a. We can deduce some conclusions from projecting the domains of f_1 and f_2 into the *x-y* plane. The discontinuity planes formed by using form (5.13a) are illustrated in Figure 5.10b. Similarly, The discontinuity planes formed by using form (5.13b) are illustrated in Figure 5.10c. Notice that the difference in nesting of conditional statements only affects the resulting output within the overlap domain that is illustrated in Figure 5.10a.

The solution strategy remains the same as for one dimension: the problem is still decomposed into discontinuity detection and discontinuity resolution sub-problems.



a. 2D overlapping functions

b. Nesting based on *x*-dimension at the outer *if* statement.

c. Nesting based on *y*-dimension at the outer *if* statement.

Figure 5.10: An example illustrating applicability domains of two-dimensional overlapping functions f_1 and f_2 and the effect of conditional nesting on boundaries segregation.

5.2.1. Two-Dimensional Discontinuity Detection

Before elaborating on the approach to handle discontinuity detection and resolution in 2D, let us look at how functions overlap in two dimensional space. Figure 5.10a illustrates the case where there are overlaps between the two functions in both domains. In such cases the detection algorithm will detect an optimum switch point for each of the domains respective overlap intervals. When functions are adjacent to each other in one

dimension and overlap in the other, the overlap domain in 5.10a reduces to a line. In such cases, the detection algorithm will only have one degree of freedom: that is to find the optimum switch point for the dimension where overlap exists. When functions are adjacent to each other in both domains, the overlap domain reduces to a point in the projected 2D space. The detection algorithm has zero degrees of freedom in this case and the resulting discontinuity locations will correspond to the intersection point between the two functions.

It should be noted that, in 2D problems, detection of optimum switch points does not guarantee passage of the simulation trajectory through these points. It only helps in formulating the conditional statement around the minimum jump effort point to aid in minimizing discontinuity while switching. This conclusion stimulates us to questioning the credibility of the obtained conventional simulation results when the simulation trajectory does not pass through an overlapping domain (shown as question marks in Figure 5.10). When not passing through an overlap domain, conditional expressions will extrapolate the use of discontinuous functions regardless of extrapolation applicability. This statement holds for all conditional statements involving the use of functions bounded by specified intervals. Since conventional modelling packages do not provide an apparent fix to this problem, it becomes the responsibility of the modeller to either ensure that the selected functions cover the intended unknown simulation path, or to insert as many functions as possible (with differing domains) to cover a wider area to, hopefully, minimize extrapolation. Thus, I think it is essential to include the applicability domains of each logical branching expression as part of the model input file. Then, the simulation package would check whether the solution falls within the specified applicability domains and flags an alert (or stops simulation execution) when the simulation trajectory deviates

from the applicable domains of the branched conditional statements.

The detection of an optimum jump points for 2D functions can be formulated as an extension of the 1D problem. For two discontinuous functions overlapping at $[a_x,b_x]$ and $[a_y, b_y]$ in x and y dimensions, respectively; the optimum switch point $g(x,y)$ is found through solving the optimization problem:

$$\begin{aligned} &min.\, e(x,y) = \left| f_1(x,y) - f_2(x,y) \right| \\ &s.t. = \begin{cases} x \in [a_x, b_x] \\ y \in [a_y, b_y] \end{cases} \end{aligned} \tag{5.14}$$

As I indicated in the 1D case, once the g_x and g_y locations are determined, their values can be directly substituted into the constructed conditional statement to minimize jump effort between the two adjacent discontinuous functions. The model can, then, be solved using any of the available integration packages. Nevertheless, since detection of optimum switch points does not always guarantee elimination of reinitialization of the ODE/PDE model at the switch point or accuracy of integrator-based interpolated solution afterwards, the need arises for a discontinuity resolution algorithm.

5.2.2. Two-Dimensional Discontinuity Resolution

Once overlap boundaries between the discontinuous functions are determined through the detection algorithm, we need to interpolate between the discontinuous functions in order to eliminate discontinuity. I propose two approaches and highlight their pros and cons.

The simplest approach (approach I) is to cover the entire overlap domain with an interpolating polynomial. Boundaries of the interpolating polynomial will correspond to those of the continuous function at the boundary location as illustrated in Figure 5.11a. The fact that the values of the interpolating polynomial at its boundaries matches that of

the neighbouring functions facilitates smooth transition in all directions.

However, this approach comes at a cost. For a fixed number of control points per dimension, interpolation mesh size is overlap-domain size dependent. This means that mesh resolution will decrease as the size of the overlap domain increases and vice versa. Of course, increasing the number of control points for large overlap domains will resolve this problem but at a heavy computational cost. Thus, I recommend adopting this approach for a relatively small overlap domain size. A typical *if* structure using this approach (based on Figure 5.11a) is illustrated in (5.15).

Note that the conditional statement well encapsulates the bounding domains of the discontinuous functions. Thus, the last *Else* statement is needed to indicate to the user that simulation trajectory is deviating from the specified functions' boundaries.

An alternative approach (approach II) would be to track a two dimensional trajectory vector $\vec{v}_n$ as simulation progresses and generate the grid points of the interpolating polynomial once the conditional statement shifts to the branch containing the interpolating polynomial as illustrated in Figure 5.4b. The $\vec{v}_n$ vector tracks the coordinates of the independent variables of the composite function as simulation progresses. Full derivation of the underlining equations is presented in Appendix D.

$$
\begin{array}{l}
\textit{If} \quad \left[\{(a'_x \le x < a_x) \wedge (a_y < y < b'_y)\} \vee \{(a_x \le x \le b_x) \wedge (b_y < y \le b'_y)\}\right] \\
\qquad f(x,y) = f_1(x,y) \\
\textit{ElseIf} \left[\{(b_x < x \le b'_x) \wedge (a'_y \le y \le b_y)\} \vee \{(a_x \le x \le b_x) \wedge (a'_y \le y < a_y)\}\right] \\
\qquad f(x,y) = f_2(x,y) \\
\textit{ElseIf} \ [(a_x \le x \le b_x) \wedge (a_y \le y \le b_y)] \\
\qquad f(x,y) = \textit{interpolate} \\
\textit{Else} \\
\qquad \text{Print “Illegal extrapolation”} \\
\textit{EndIf}
\end{array}
\tag{5.15}
$$

In this approach, the mesh is constructed at the intersection point between $\vec{v}_n$ and the

overlap domain. The aim of the constructed mesh is to facilitate transition from the currently active discontinuous function to the function towards which $\vec{v}_n$ is heading. Once transition to the destination discontinuous function is complete the rest of the overlap domain is considered as a seamless part of the destination discontinuous function. This approach allows generation of a high resolution variable grid size that is independent of the size of the overlap domain and with a fixed number of control points.

The approach works well with one exceptional situation. This situation arises when $\vec{v}_i$ changes direction, within the overlap domain, and returns back to the discontinuous function where it originally came from as illustrated in Figure 5.11b. Since the overlap domain, with the exception of the interpolation region, has been replaced with the values of the destination discontinuous function a discontinuity would probably occur at the boundaries of the overlap domain with the function where the vector has originally come from. Such a situation is solvable through formulating an additional exit interpolating polynomial with the original function as illustrated in Figure 5.11c. Note that even the entry region (cross-hatched) is treated as a possible interpolating region to move back to f_1 from the overlap region. The fine-hatched region resembles the entire area at which interpolation might occur. However, the generated mesh will only cover the portion at which $\vec{v}_i$ is heading as illustrated in Figure 5.11d. Note that this problem would never occur if approach I is used because $\vec{v}_i$ will always fall in the region of the interpolating polynomial once it is inside the overlap region as illustrated in Figure 5.11a. Two advantages arise from using approach II:

1. It allows for variable size mesh, i.e. h_x and h_y can be arbitrary selected as long as the resulting mesh does not cross the overlap domain.

2. Only four points are needed (six when using *hermite* interpolating polynomials) per interpolation dimension regardless of the size of the overlap domain.

However, more checks are needed in this approach over approach I. A typical 2D conditional structure pseudo code is illustrated in (5.16).

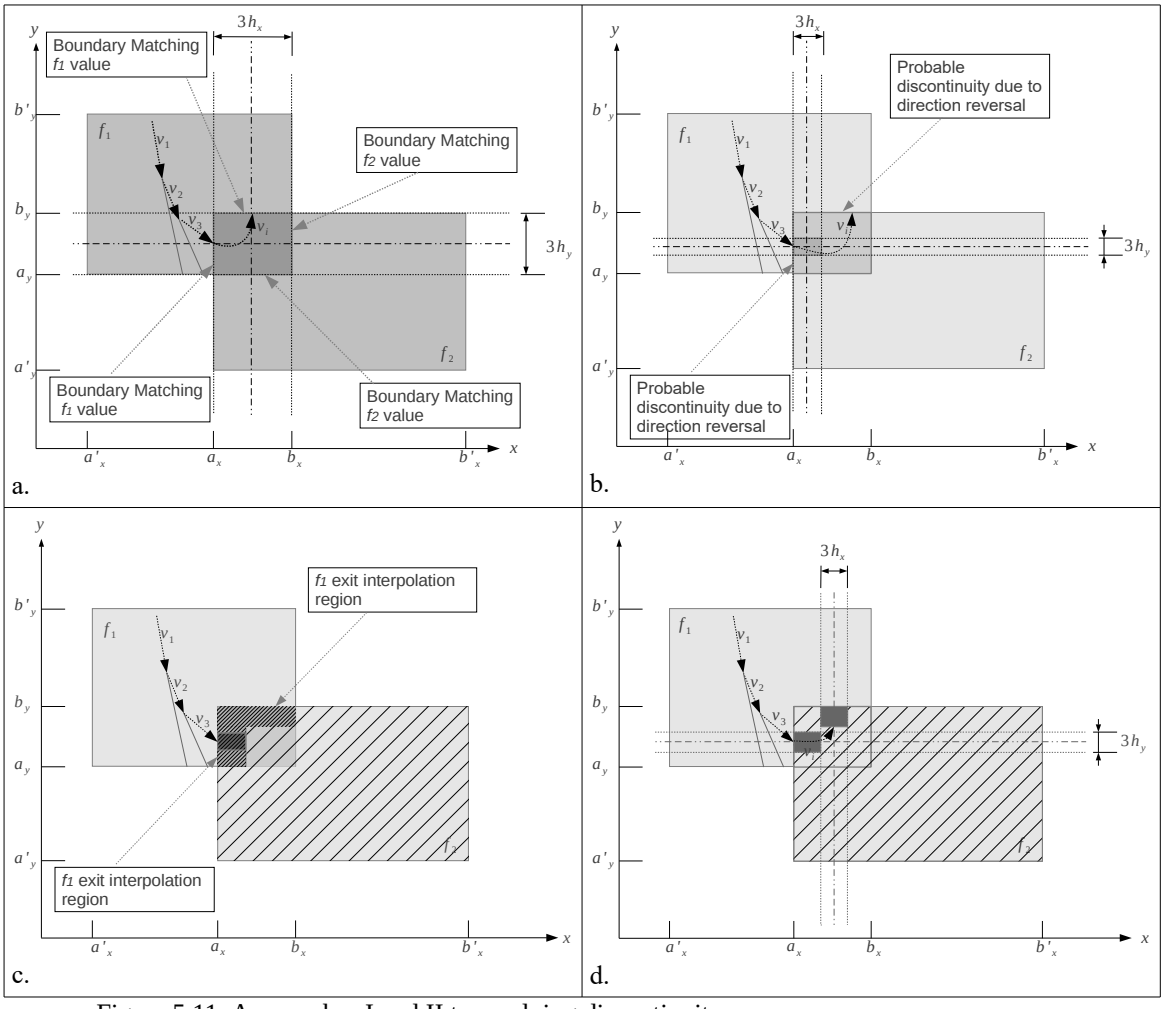


Figure 5.11: Approaches I and II to resolving discontinuity.

5.2.3. *How legal is "illegal" extrapolation?*

As we discussed earlier, extrapolation occurs when trying to join the two discontinuous sub-functions by a polynomial that lies outside their designated domains. This is illustrated in Figures 5.1d and 5.10 (domains marked by question marks) for 1D and 2D functions, respectively. There are two reasons (cases) behind alerting the modeller about illegal extrapolation:

1. The extrapolation domain might be defined by a function exhibiting a behaviour that is different from the behaviours of the sub-functions to be extrapolated. In such cases, extrapolation will result in erroneous simulation output.
2. Either or both of the functions to be linked might not be mathematically defined in the extrapolation region (e.g. division by zero). In such cases control points 3 and 4 cannot be calculated due to unavailability of function values at the location of these points.

The modeller will obtain a less than accurate result in the first case. However, if the modeller is confident about the consistency of the behaviour between the extrapolation region and the functions to be extrapolated, he or she can simply alter domain boundaries of the functions to append the extrapolated region to one of them, divide it between the two functions or, even better, append it to both functions and rely on the detection optimizer to locate the best transition point *g*.

As for the second case, the integrator will simply stop integrating because the values of the functions at points 3 and 4 are dependent on the respective values of functions 1 and 2. However, the dependency can be broken by eliminating function evaluations at these two points. We should recall that function evaluations at points 3 and 4 are needed to calculate the amount of dip based on *p* parameter. If some curvature smoothness at the junction points between the interpolating polynomial and the discontinuous functions can be sacrificed in the quest for continuity, then the integrator can extrapolate between the values of the two discontinuous sub-functions using their respective boundaries that are adjacent to the extrapolation domain.

```
If (active_point_coordinate∉overlap_domain)
        entry_completed = false; first_overlap_entry = true; first_exit_attempt = true
        If (active_point_coordinate∈f1range)
                f= f1(x,y)
                Active_function = f1
        ElseIf (active_point_coordinate∈f2range)
                f= f2(x,y)
                Active_function = f2
        Else
                Print "Illegal Extrapolation"
        EndIf
ElseIf (active_point_coordinate∈overlap_domain)
        detect_entry_intersection_plain;
        If first_overlap_entry = true
                construct_entry_mesh;
                first_overlap_entry = false
        EndIf
        If (active_point_coordinate∈entry_interpolation)∧(not entry_completed)
                f = entry_interpolate
        Else
                entry_completed = true
                If (active_point_coordinate∈exit_interpolation)
                        If first_exit_attempt = true
                                construct_exit_mesh
                                first_exit_attempt = false
                        EndIf
                        f = exit_interpolate
                Else
                        f = fDestination_function(x,y)
                EndIf
        EndIf
EndIf
```

(5.16)

As we might expect, the second solution will work for cases 1 and 2. However, it will not eliminate errors associated with the first extrapolation case. So, it still becomes the modeller's responsibility to tackle the first case by inserting an appropriate function to define the region that might otherwise be erroneously extrapolated.

5.2.4. Mesh Generation

In order to interpolate, a mesh needs to be generated. For one-dimensional problems, the

mesh reduces to a one-dimensional set of points. The 2D+ problems require an elaboration on mesh generation methods.

Mesh generation is an approach dependent exercise. Generating the mesh using approach I is a fairly easy task since the mesh will cover the entire overlap region. The values of the boundary points surrounding the overlap region will always correspond to the neighbouring continuous sub-functions adjacent to the overlap domain as illustrated in Figure 5.11a.

For approach II, mesh generation is more complex. The extra complication arises from the tracking of $\vec{v}_i$. I will discuss four methods to construct the mesh around the intersection of the $\vec{v}_n$ with the discontinuity plane. I will briefly explain each method and provide my reasoning for selecting one of them. For simplicity, I will demonstrate examples using a discontinuity plane orthogonal to *x*-axis. However, the concept applies to discontinuities orthogonal to either of *x*- or *y*-axis.

The first method constructs a squared mesh around the discontinuity point as illustrated in Figure 5.12a. Values of $h_x^{'}$ and $h_y^{'}$ are measured with respect to their respective *x*- and *y*-axes. The size of the mesh is fixed. The distribution of the mesh control points along the sides of $\vec{v}_n$ is dependent on the slope of $\vec{v}_n$. Thus, $\vec{v}_n$ might lean towards some of the control points over others.

The second method is similar to the first one with the exception that the size of the mesh is expandable in the direction that is perpendicular to the discontinuity plane. The advantage of this method is that it allows a better distribution of the control points along each side of the $\vec{v}_n$ vector as illustrated in Figure 5.12b. As can be deduced from the figure, vector $\vec{v}_n$ is still almost always leaning towards one set of the mesh control points

over the other.

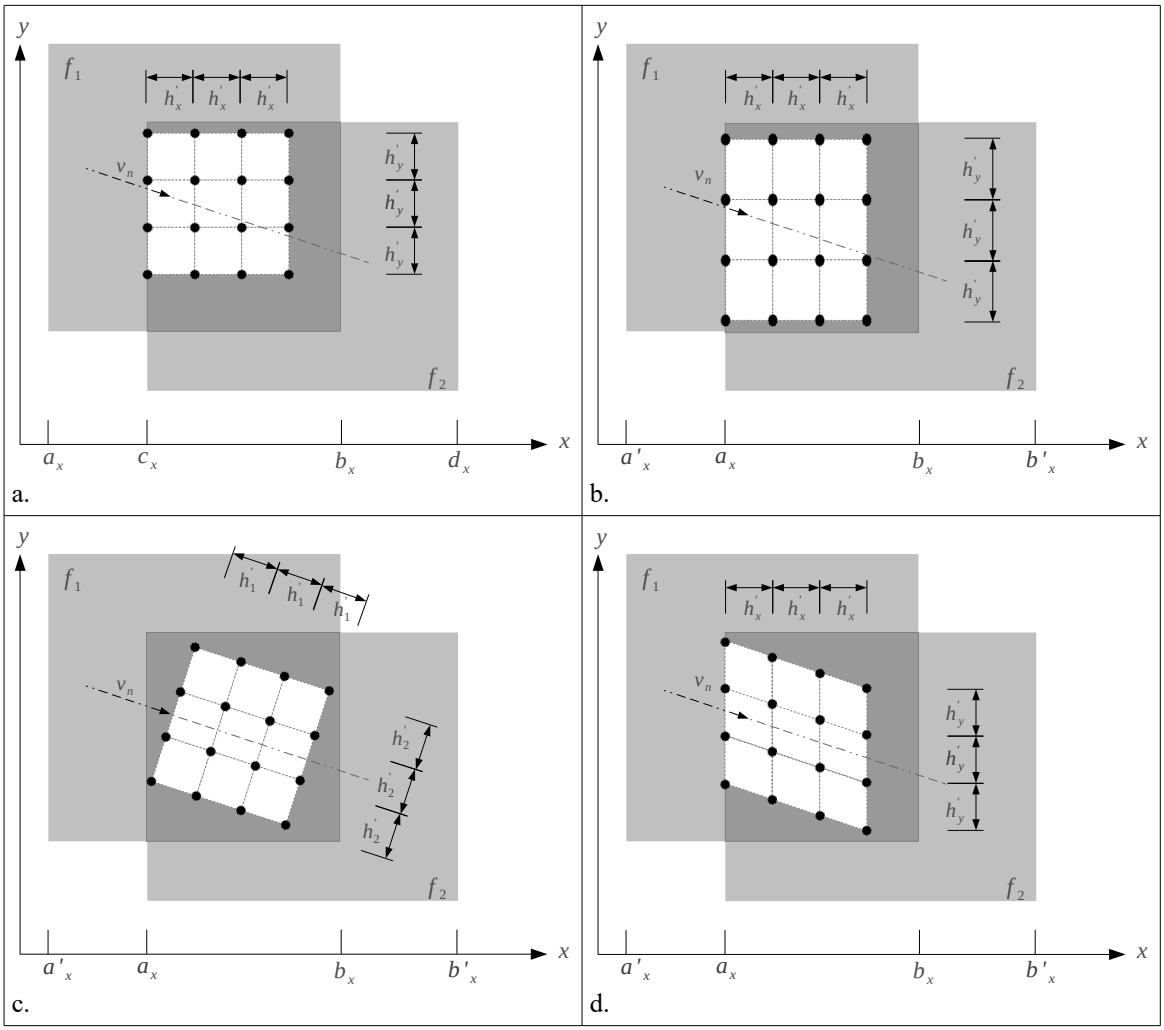


Figure 5.12: Four ways to construct a mesh around a vector-plane intersection point.

The third method aligns the grid with the direction of $\vec{v}_n$. This method better distributes grid points along the sides of $\vec{v}_n$, compared to the former two methods as illustrated in Figure 5.12c. Note that h'_1 and h'_2 are respectively measured parallel and orthogonal to $\vec{v}_n$ but not relative to *x*- and *y*-axes. Since the grid is aligned to $\vec{v}_n$ while the conditional statement is based on a discontinuity that is orthogonal to either *x*- or *y*-axis, logical statements around interpolation region become functions of the direction of $\vec{v}_n$. Since the generated mesh is not aligned with overlap domain, it becomes a difficult task to superimpose the mesh on the conditional statement.

The fourth method relies on fixing an h' along each of the dimensions while shifting the location of the line segments that are parallel to the discontinuous domain to align the grid with $\vec{v}_n$. The fourth method resolves the drawbacks of the previous three methods. Thus, I opted for implementing this method in grid construction for Approach II. The extension of this approach to the construction of 3D meshes is detailed in Appendix C.

5.3. N-Dimensional Functions

5.3.1. N-Dimensional Discontinuity Detection

To generalize, for two *n*-dimensional discontinuous functions, discontinuity detection detects the overlap region between the two discontinuous sub-functions. It also detects the optimum switch point between the two discontinuous functions. The position of the two sub-functions, relative to the overlap region and the location of the optimum switch point, assists in formulating the conditional statement. If sub-functions do not overlap in any of the dimensions, the algorithm flags an error and simulation execution stops.

5.3.2. N-Dimensional Discontinuity Resolution

Discontinuity resolution takes the form of an interpolating polynomial that connects the two discontinuous sub-functions. For one-dimensional discontinuous functions, the interpolating polynomial is best formulated around the minimum jump effort point.

For discontinuous functions of dimensions greater than one, the solution can follow one of two approaches:

1. The first approach relies on constructing an interpolating polynomial that covers the entire overlap domain. This path is suitable for relatively small overlap

regions. For large overlap domains, the interpolating polynomial mesh resolution can be enhanced by increasing the number of control points at a heavy computational cost.

2. The second approach constructs one mesh and possibly a second one. The first mesh is constructed at the entry to the overlap domain. It facilitates smooth transition between the active discontinuous sub-function at the entry point of the overlap domain and the destination one. Once transition occurs, the rest of the overlap domain is treated as if it were part of the destination sub-function. In situations where the simulation vector reverts back to the sub-function where it originally came from within the overlap domain, an exit mesh is constructed to resolve discontinuity at exit location. This path has the advantage of varying the mesh size based on user specification while maintaining a fixed number of control points.

Figures 5.13a and 5.13b illustrate generated meshes for an overlap-domain between two 3D discontinuous functions using approaches I and II to discontinuity resolution, respectively.

The total required number of mesh points is an exponential function of the dimensions of the composite function and can be calculated as:

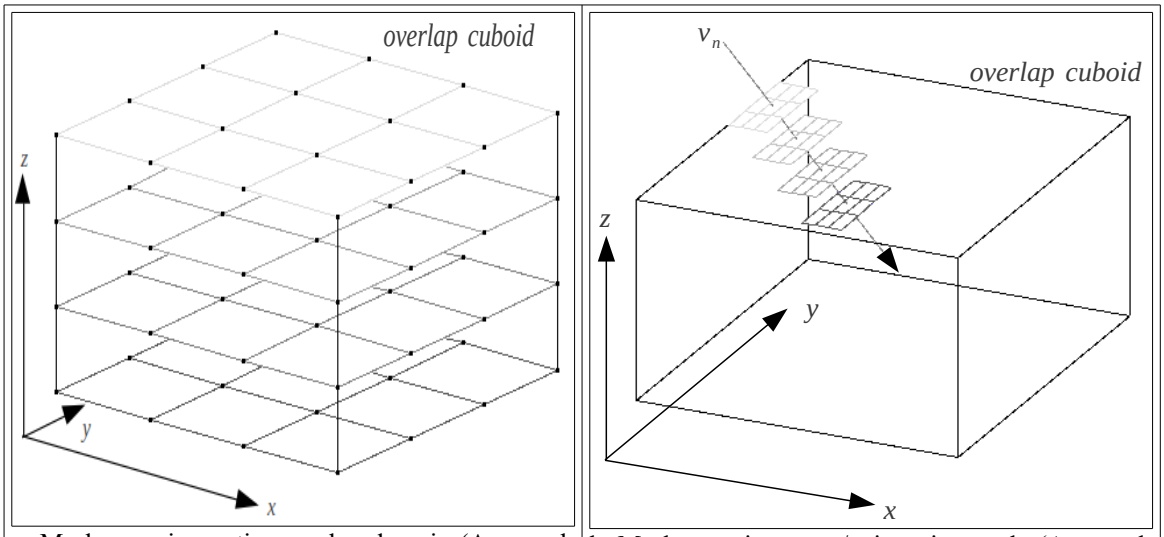
$$Number\ of\ mesh\ points = m^n \tag{5.17}$$

where m: number of control points per dimension

n: number of dimensions

To ensure smooth transition between the two discontinuous sub-functions, at least four control points are required per a dimension. In the case of *hermite* interpolating polynomials, six control points are required per a dimension to assist in curvature closure as outlined in Appendix C. Figure 5.14 illustrates the

relationship between the number of control points required and the dimensions of the composite function. Although computational power and capacity are machine dependent, we can deduce from the plot the existence of a threshold beyond which computational power and machine space (memory or hard disk) becomes prohibitive. For example, for a tenth dimension discontinuous function, a cubic spline would require a mesh composed of 1,048,576 points. That is a megabyte of memory/disk space per discontinuity. The problem becomes worse when using *hermite* interpolating polynomials. For a tenth dimension discontinuous function, the *hermite* interpolating polynomial requires 60,466,176 mesh points. This is about 58 megabytes of memory/disk space per discontinuity encountered.



a. Mesh covering entire overlap domain (Approach I).

b. Mesh covering entry/exit regions only (Approach II).

Figure 5.13: Representation of the two types of generated meshes in a 3D cuboid overlap domain.

A person might think that we could use sparse matrix algebra to conserve memory. However, this is not possible since we only have four or six points per dimension, all of which contribute to the shape formation of the interpolation curve, resulting in a very dense matrix. Yet, some solutions can help reducing the implications of this problem or eliminating it. For example, the number of dimensions can be reduced if any dimension

exhibiting constant values throughout the interpolation region is omitted from the interpolation mesh. Also, since usually hard disk space is more abundant than memory, the entire mesh can be saved in a computer hard drive using binary files to accelerate simulation program access to these mesh-point files. Lastly, instead of generating the mesh once at the first entry to the interpolation region and saving it, the simulation routine can opt to generate the mesh at each interpolation run and immediately dispose it after the composite function value is computed to free memory/hard disk space. The latter resolution saves a tremendous amount of disk space by dynamically allocating mesh space to compute function values and freeing the space once the function value is computed. However, additional CPU time is required to construct the exact same mesh at every function evaluation within the interpolation region.

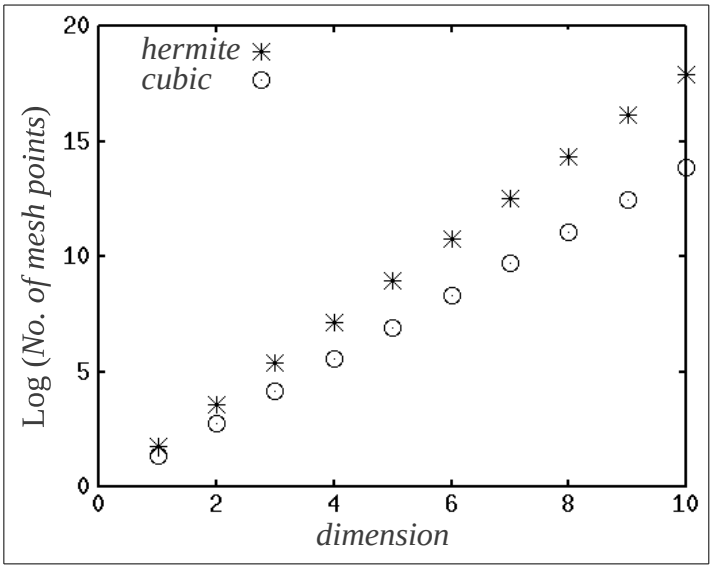


Figure 5.14: A semi-log plot of number of mesh points required versus discontinuous function dimension.

Of course, a combination of one or more of the above resolutions will result in a more efficient and/or robust algorithm. For example, the simulation routine can be programmed to:

1. Generate interpolation mesh only once in memory when memory space is

abundant.

2. Once memory occupied space reaches a specified maximum, the simulation routine switches to storing a one-time generated mesh in the machine hard drive.

3. If hard drive space is limited or has reached a critical level, the routine shifts to dynamically creating and destroying meshes at each function evaluation inside the interpolation region.

To further enhance efficiency, the routine can be programmed to optimize memory utilization by loading lower dimension functions' meshes into memory while saving higher dimension ones to hard disk. The prior knowledge of the dimension of each composite function will assist the simulation routine in calculating the maximum amount of occupied hard disk/memory space beyond which dynamic allocation and destruction of interpolation meshes (bullet 3) should be used instead of a single-time generated mesh (bullets 1 or 2).

Such a resolution is hardware dependent. Thus, below certain machine hardware specifications and based on computed mesh size for each interpolating polynomial in a simulation model, the simulation routine can flag an error message prior to starting simulation run indicating the inability to run the model on a specified machine. However, I think modern hardware capabilities extend far beyond such minimum specifications.

Last, it is good to shed some light on whether this work eliminates the need for implicit solvers and their respective variable integration step size. The answer is no. Taking Figure 5.3 as an example, we notice that slope changes are very evident between each of the sub-functions and their respective interpolating polynomial. An explicit integration routine with a fixed integration step size can easily overlook these slope changes, even in a

regularized composite function, resulting in sever simulation errors. Of course, minimizing integration step length might resolve the issue but at the cost of increased simulation run-length. The use of variable integration step-size in implicit solvers ensures the adjustment of the step-size as required. Larger integration steps are used when integration error is within bounds. Whenever integration error exceeds the bounds, integration step is halved and error is recalculated. The implicit integration routine adjusts integration step size when moving between discontinuous sub-functions and their respective interpolating polynomial. Thus, the use of implicit integration routines is still favoured even after model regularization.

5.4. The Algorithm

The algorithm implementation is programming language dependent as it involves either a modification of conditional statements or a complete rewrite of the discrete composite function to regularize it. In compiler-based modelling languages such as [gPROMS, 2012], it is recommended to embed the code within the language compiler. However, this solution might not be feasible for general purpose modelling languages such as MATLAB or GNU Octave or even general purpose imperative languages such as C++, FORTRAN or Pascal. In such cases, the programmer can write his/her custom code to iterate through discretized composite functions and transform them to their regularized counterparts. Generic packages to perform such tasks can also be developed by the scientific community and added to the language as a language library module.

Regardless of the implementation platform, the modeller needs a mean to enter the domain of each dimension of a sub-function that is part of a composite discontinuous function. The detection algorithm sorts the discontinuous sub-functions of a composite function based on the applicability of the respective domain for each of the dimensions.

Figure 6.3 illustrates a simplified flowchart of the algorithm. A simplified step-by-step procedure that should be executed by the modelling language follows:

STEP-01: Start simulation run

STEP-02: Check for the availability of any functions containing conditional statements or standalone conditional statements involving continuous variables (i.e. of real or float types) inside original model code.

STEP-03: Search for an optimum switch point that minimizes the difference in values between any two sub-functions within their overlap domain.

STEP-04: Adjust the standalone conditional statement or the one within the composite function to account for the new switch point.

STEP-05: If resolution is enabled by the modeller, reconstruct a regularized conditional statement from the discretized one (recommended).

STEP-06: Repeat STEP2 and STEP3 until all conditional statements within modeller's code are handled.

STEP-07: Start the integration and Initialize variables.

STEP-08: The integration routine advances integration step if final integration limit is not reached.

STEP-09: Update $\vec{v_i}$ for each composite regularized function.

STEP-10: If composite regularized function parameters are not within the interpolation region, the value of function *f* is calculated using the provided discontinuous sub-function that lies within the active domain. If parameters are within the overlap domain, check if this is the first entry to the overlap region in order to generate the interpolation grid. If the grid is already generated, use interpolating polynomial f_3 to calculate *f*.

STEP-11: Repeat steps 8-11 until simulation completes.

In the present work, the search for discontinuous functions within a simulation code is not implemented. As I explained earlier, this task is programming-language dependent. However, the search for an optimum switch point within the conditional statements, that are used as examples in this work, is implemented and tested using [gPROMS, 2012] Foreign Object Interface (FOI). It has also been independently tested using GNU Octave [GSL, 2011].

Similarly, regularizing functions have been tested using both [gPROMS, 2012] and GNU Octave [GSL, 2011] programming languages. Again, the automatic formulation of the composite regularizing function is language compiler specific. It is also outside the scope of this work and thus not implemented.

For the online part, the vector tracking algorithm has also been implemented in [gPROMS, 2012] FOI. Binary (record) files are used to record vectors' paths of Prandtl and Reynolds numbers during simulation run of the reactor model. For the PSA model, the same routines are used to track velocity, inlet and outlet concentration profiles.

For Approach II to discontinuity resolution, a complete C++ routine is written to handle the regularization of the discontinuity. The possibility of the vector reversing direction within the interpolation region is also programmed.

A special C++ routine is also written to estimate the location of the left-most control point when regularizing boundary conditions. As discussed earlier, the purpose of the routine is to interpolate using available pre-discontinuity history points in order to calculate the value of the independent variable immediately preceding the regularization region.

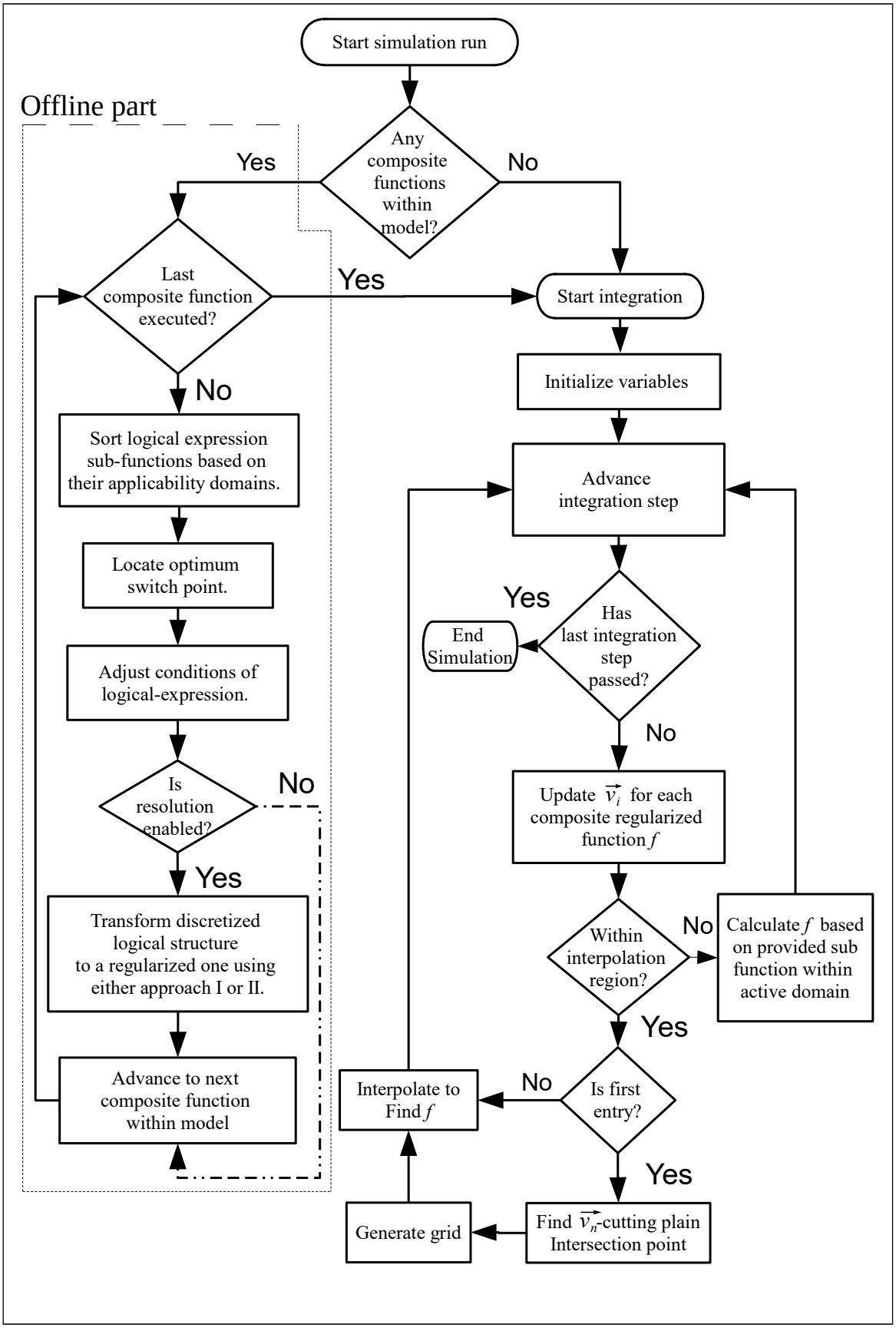


Figure 5.15: A simplified flowchart illustrating flow of the presented algorithm. Solid lines represent the more preferred path while the dashed line represents the less preferred one.

The bounded dotted area represents offline part while the rest represents the online part.

A separate C++ routine is written to handle the generation of the mesh control-points for Approach II to discontinuity resolution. The routine is linked to [gPROMS, 2012] and tested using the reactor model that is described in Appendix B. The mesh generation algorithm is also simultaneously tested using GNU Octave [GSL, 2011].

I implemented the algorithms in a C++ code. Then, I linked the compiled code to a [gPROMS, 2012] models described in Chapter 4 and Appendix B through gPROMS Foreign Object Interface (FOI). A simplified one-dimensional *hermite* interpolation code is presented by [Bourke, 2011]. [Breeuwsma, 2011] presented a general C++ and Java codes for multidimensional interpolation that can be used in conjunction with any one-dimensional interpolation method. I combined the codes of [Bourke, 2011] and [Breeuwsma, 2011] to formulate the C++ multidimensional *hermite* interpolation routines that are used in this work.

5.5. Summary and Concluding Remarks

In this chapter, I introduced a novel approach for detecting and resolving discontinuities originating from the use of conditional statements within a modelling code. The approach is based on targeting the discontinuity at its origin and hence eliminates the need for interpolating polynomials that do not truly represent the discontinuity.

I outlined how the one-dimensional detection and resolution approach can be applied to regularize constitutive equations. I also discussed how the approach can be extended to handle discontinuities resulting from shifts in boundary conditions during simulation run. I demonstrated the uniqueness of the resolution for one-dimensional discontinuous functions. Thus, the one-dimensional detection and resolution approach can be applied offline before starting the model integration.

The one-dimensional detection approach is extendible to multi-dimensional composite functions. For multi-dimensional resolution, I devised two discontinuity resolution approaches. Approach I relies on covering the entire overlap domain with an interpolating polynomial. This approach is more applicable to small overlap domains since the mesh resolution reduces as the size of the overlap domain increases.

Approach II to discontinuity resolution relies on tracking a vector of the independent variables of a composite function. The vector is used to construct the multi-dimensional interpolating polynomial once the conditional statement shifts to the overlap domain. A procedure is also devised to best generate a mesh of control points for the interpolating polynomial based on the direction of a tracked vector.

The last section of this chapter outlined the sequence of the steps for the algorithm and how they should be implemented either within the compiler of the language or as an independent code. The next chapter demonstrates the implementation of the algorithm, discussed in this chapter, to two models of chemical processes.

CHAPTER 6:

Applications to Some Complex Models

In this chapter, I will demonstrate the discussed concepts using two examples, one for one-dimensional functions exhibiting dynamic boundary conditions and the other for a two-dimensional function embedded within a model's constitutive equation.

6.1. Regularizing a Discontinuity in Heat Transfer Coefficient Calculation

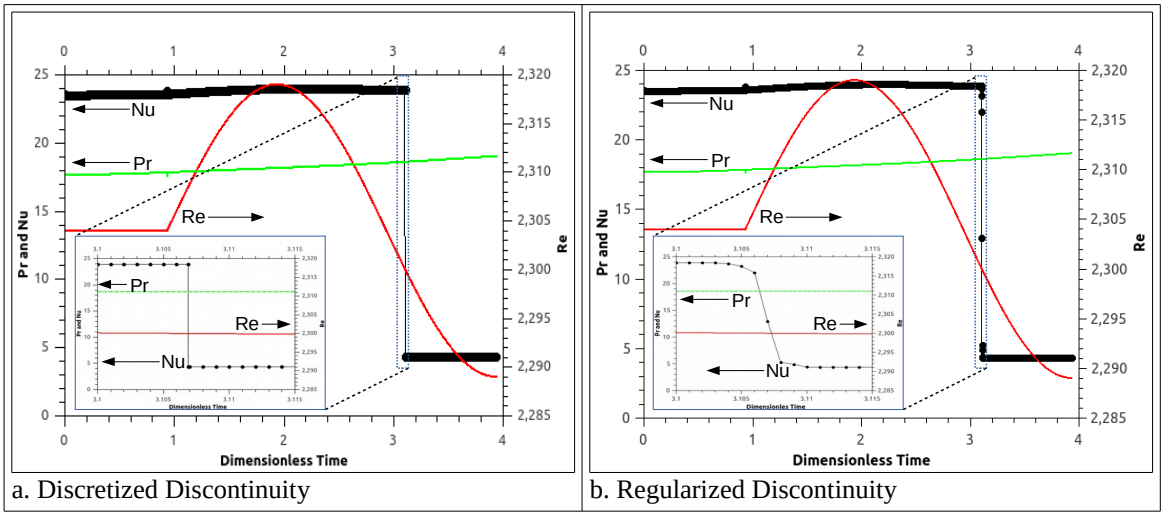
I tested the effect of transition from Laminar to Turbulent flow regimes on the wall heat transfer coefficient described by equation 4.1 and plotted in Figure 4.3.

When using the approach presented in this work, I expected to observe a decline in the time required to perform a simulation run compared with conventional simulation reinitialization procedures. Since the developed reactor model discretizes axial space to convert PDEs to ODEs, I intend to use the number of discretization points as a variable to test our theory.

The code is expected to best perform at large numbers of discretization points. The performance should approach that of conventional simulation techniques as the number of discretization points is reduced. This is due to the fact that the number of equations requiring initialization is directly proportional to the number of discretization points.

To establish a baseline for the analysis and to eliminate the bias introduced by every simulation run on the analysis, I recorded machine time taken to complete a constant velocity simulation that does not pass through any discontinuities for a set of axial discretization nodes that span from 10 to 500 as outlined in Table 1. To eliminate any variance in reported data (due to interfering machine background tasks) I repeated each run three times and reported the average outcome of the three runs on the table. I should also mention that the reported base case is based on conventional simulation runs. A consistent additional one second is noticed when using FOI to report base case results. The additional one second is probably attributed to initiation and termination of the link between [gPROMS, 2012] and the FOI. I should also mention that results on Table 6.1 are generated using a single lumped heat transfer coefficient that is based on feed conditions

and an average axial reactor temperature. Also, the simulation runs were performed on a machine equipped with an Intel i5 processor using 4GB RAM and running a Linux operating system.



a. Discretized Discontinuity b. Regularized Discontinuity

Figure 6.1: (a) Discretized and (b) regularized Nusselt functions plotted against time. The quasi independent variables, Reynolds and Prandtl numbers, are also plotted for illustration purposes.

It should be noted that points in the *Nu* curve do not represent control points but simulation reporting intervals.

After establishing the base case, I applied a sinusoidal input to the feed velocity that crosses Reynolds boundary of 2,300 between the two correlations ten times. Plots of *Nu*, *Pr* and *Re* against time when passing through the first discontinuity are illustrated in Figure 6.1a for the discretized model and in Figure 6.1b for the regularized one. For the regularized model, Figure 6.2 represents a 3D view of the regularized interpolating polynomial that is constructed based on $\vec{v}_n$ direction.

The simulation run-length is plotted against the number of axial discretization nodes for the reference case, the discretized and the regularized models in Figure 6.3a. The difference between the discretized model and the base case run lengths is plotted in Figure 6.3b gainst the number of discretization nodes. The difference between the regularized model and the base case run lengths is also plotted in the same figure. With the exception of the reported time using ten discretization nodes, the rest of the points

closely resemble straight lines. Excluding the point corresponding to ten discretization nodes (explained later) and applying regression analysis between the number of discretization nodes and the absolute simulation run length for the conventional case and this work yields the tabulated results in Table 6.2. The slopes resulting from the regression analysis represent the run length time per discretization node. Dividing the slope resulting from this work (0.12263) by the slope resulting from conventional runs (0.15869) provides the fractional run length time elapsing from this work per elapsed run length of conventional runs (0.7728). The results show that using the approach provided in this work results in about 23% saving in run length time over conventional discontinuity handling techniques at least for 2D discontinuous functions. Of course, the same conclusion would have been achieved had we directly regressed run length time for conventional discontinuity handlers against the results obtained in this work bypassing the inclusion of discretization nodes in regression analysis.

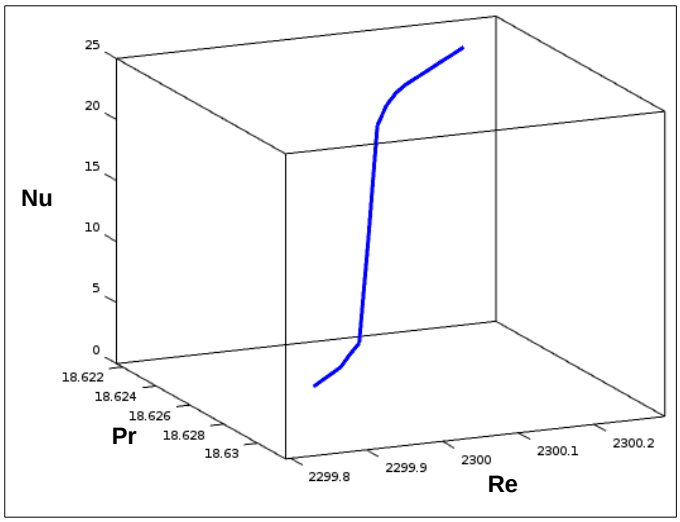


Figure 6.2: A zoomed view of Re-Pr trajectory vector as it approaches the discontinuity and smoothly slides over it.

As it appears from the figures and supported by the computational results, there is a consistent drop in the reported simulation time when using the new approach for two dimensional discontinuous functions. Also, the new approach becomes more attractive as

the number of the state variables, to be initialized, increases.

As the number of state variables decreases, both approaches to resolving discontinuity report closer simulation times. However, since initialization itself introduces errors in the solution, the new approach still holds the advantage of not reinitializing any state variables.

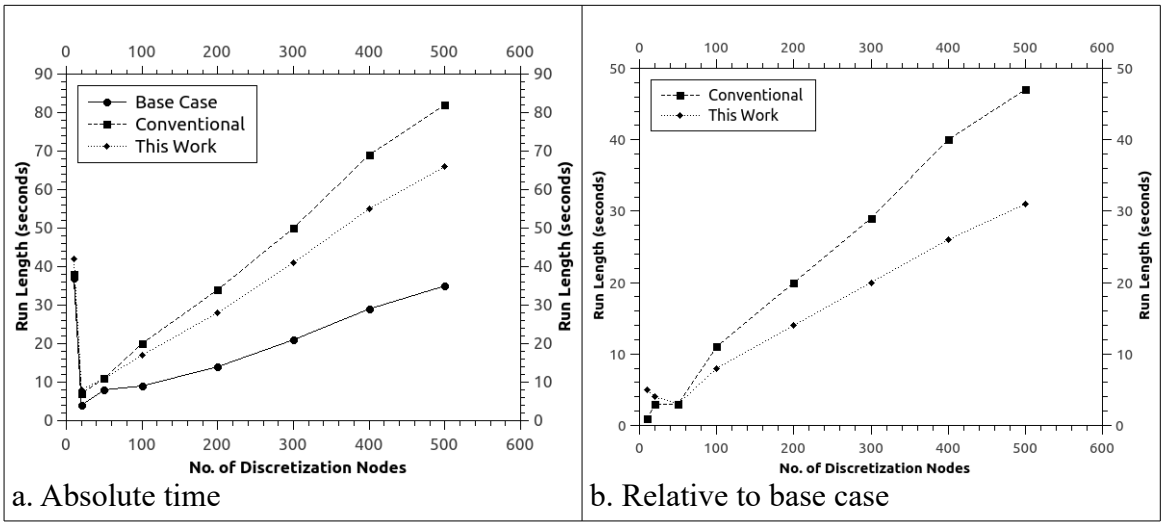


Figure 6.3: Simulation Run Length versus number of internal discretization nodes.

Table 6.1: Reported Simulation Time for several runs using varying discretization nodes

Discretization Nodes	Time (seconds)				
	Base Case	Conventional		This Work	
		Absolute	Above Base	Absolute	Above Base
10	37	38	1	42	5
20	4	7	3	8	4
50	8	11	3	11	3
100	9	20	11	17	8
200	14	34	20	28	14
300	21	50	29	41	20
400	29	69	40	55	26
500	35	82	47	66	31

Table 6.2: Regression results for correlating simulation run length with number of discretization nodes.

	Slope	Intercept	Correlation Coefficient
Conventional	0.15869	3.40857	0.9992
This Work	0.12263	4.78063	0.9993

As illustrated in Figure 6.3a, there is a sudden increase in the reported time when using ten discretization points. This sudden increase in simulation time is mainly attributed to the decline in discretization resolution. As the number of space discretization points decreases, the integrator is forced to take smaller integration steps in order to meet the specified error tolerance criterion for a successful integration step.

6.2. Regularizing Boundary and Initial Conditions of a PSA Column

Pressure Swing Adsorption (PSA) processes are considered among few of the processes that exhibit continuous dynamics from the moment they are started until they are shut down. As discussed in Section 4.2.3, any PSA column undergoes a sequence of steps whereby inlet and exit valves are automatically opened and closed or products are redirected through switch (Motor Operated) valves. Feeds are introduced at some steps and products are collected at either the same step or at different steps. A simplified isothermal set of the PSA model equations, presented in Section 4.2, is used to demonstrate the concept. The PSA cycle is described in Section 4.2.1 is reduced to its simple [Skarström, 1960] form.

Each step undergone by a PSA column possesses differing boundary conditions that uniquely identifies the step from its sister steps as illustrated in Figure 4.9. The switch from one step to the other is either time dependent (e.g. adsorption and desorption steps) or state variable dependent (e.g. pressurization and de-pressurization). Regardless of the solver used, conventional solution of PSA column differential equations requires

reinitialization of the ODE/DAE system at the start of each step in the sequence as outlined in Section 4.2.3. The model repeats the cycles until a desired maximum number of cycles is reached or an error tolerance is reached on exit concentrations between two consecutive cycles at the end of either depressurization or desorption step signifying the reach of a cyclic steady state.

In this work, I regularized the components mass boundary and velocity initial conditions illustrated in equations 4.20-4.24, 4.26-4.30, 4.31-4.35 and 4.36-4.40 for pressurization, adsorption, depressurization and desorption steps, respectively. Regularization is performed through the use of 1D *hermite* interpolating polynomials. One-interval regularization is added between every two consecutive steps as illustrated below for velocity, inlet and exist concentrations composite functions:

$$u|_{z=0 \text{ or } z=L} = f(Time_{Cycle}) = \begin{cases} u|_{z=L} = 0 & 0 \le Time_{Cycle} \le Time_{Pressurization\,Step} \\ Interpolate & Time_{Pressurization\,Step} < Time_{Cycle} < Time_{Pressurization\,Step} + w \\ u|_{z=0} = (u_f) & Time_{Pressurization\,Step} + w \le Time_{Cycle} \le Time_{Adsorption\,Step} \\ Interpolate & Time_{Adsorption\,Step} < Time_{Cycle} < Time_{Adsorption\,Step} + w \\ u|_{z=L} = 0 & Time_{Adsorption\,Step} + w \le Time_{Cycle} \le Time_{Depressurization\,Step} \\ Interpolate & Time_{Depressurization\,Step} < Time_{Cycle} < Time_{Depressurization\,Step} + w \\ u|_{z=L} = -u_p & Time_{Depressurization\,Step} + w \le Time_{Cycle} \le Time_{Desorption\,Step} \\ Interpolate & Time_{Desorption\,Step} < Time_{Cycle} < Time_{Desorption\,Step} + w \end{cases} \tag{6.1}$$

$$\frac{\partial c_i}{\partial z}|_{z=0} = f(Time_{Cycle}) = \begin{cases} u|_{z=0}(c_i^f - c_i|_{z=0}) & 0 \le Time_{Cycle} \le Time_{Pressurization\,Step} \\ Interpolate & Time_{Pressurization\,Step} < Time_{Cycle} < Time_{Pressurization\,Step} + w \\ u|_{z=0}(c_i^f - c_i|_{z=0}) & Time_{Pressurization\,Step} + w \le Time_{Cycle} \le Time_{Adsorption\,Step} \\ Interpolate & Time_{Adsorption\,Step} < Time_{Cycle} < Time_{Adsorption\,Step} + w \\ 0 & Time_{Adsorption\,Step} + w \le Time_{Cycle} \le Time_{Depressurization\,Step} \\ Interpolate & Time_{Depressurization\,Step} < Time_{Cycle} < Time_{Depressurization\,Step} + w \\ 0 & Time_{Depressurization\,Step} + w \le Time_{Cycle} \le Time_{Desorption\,Step} \\ Interpolate & Time_{Desorption\,Step} < Time_{Cycle} < Time_{Desorption\,Step} + w \end{cases} \tag{6.2}$$

$$\frac{\partial c_i}{\partial z}\Big|_{z=L} = f\left(Time_{Cycle}\right) = \begin{cases} 0 & 0 \le Time_{Cycle} \le Time_{Pressurization\,Step} \\ Interpolate & Time_{Pressurization\,Step} < Time_{Cycle} < Time_{Pressurization\,Step} + w \\ 0 & Time_{Pressurization\,Step} + w \le Time_{Cycle} \le Time_{Adsorption\,Step} \\ Interpolate & Time_{Adsorption\,Step} < Time_{Cycle} < Time_{Adsorption\,Step} + w \\ 0 & Time_{Adsorption\,Step} + w \le Time_{Cycle} \le Time_{Depressurization\,Step} \\ Interpolate & Time_{Depressurization\,Step} < Time_{Cycle} < Time_{Depressurization\,Step} + w \\ u\big|_{z=L}\left(c_i^p - c_i\big|_{z=L}\right) & Time_{Depressurization\,Step} + w \le Time_{Cycle} \le Time_{Desorption\,Step} \\ Interpolate & Time_{Desorption\,Step} < Time_{Cycle} < Time_{Desorption\,Step} + w \end{cases} \quad (6.3)$$

Initially, I was planning to demonstrate the concept of two-interval regularization through implementing it in the regularizing interval between desorption and pressurization steps. However, a better modelling of the regularization period through reformulation of the velocity calculation function (Appendix A) allowed the use of one regularization interval between these two steps. Nevertheless, it should be noted that the two-interval regularization can still be used to resolve discontinuities similar (but no exactly the same as I will discuss later) to the one outlined between desorption and pressurization steps.

At each time step, the velocity profile is obtained through solving an ODE equation with one boundary condition. However, the location of the boundary condition is PSA cycle step dependent. So, in order to regularize velocity boundaries, I initially had to calculate the entire velocity profile in the FOI through an independent integration routine provided through GNU Scientific Library [GSL, 2011]. The resulting profile is then passed to gPROMS model. This approach provided the anticipated results. However, since the profile is calculated outside gPROMS solver with no available Jacobian vector, the execution time of every model run tended to take longer time than required. This is presumably because gPROMS solver is trying to construct a Jacobian vector for the velocity by forcing more function calls to the FOI object.

Later on, I eliminated use of the GSL integrator and relied solely on gPROMS integration routine to solve for velocity profile. The FOI only determines the location of velocity initial condition and its value. Both parameters are passed to gPROMS which evaluates

velocity boundary conditions through complex indexing of vector parameters as illustrated below:

$$Velocity\big|_{Velocity\ Location} = Velocity\ Value \tag{6.4a}$$

$$\frac{d(Velocity)}{dx}\Big|_{(1-Velocity\ Location)} = 0 \tag{6.4b}$$

Although results were satisfactory, they were less than acceptable due to a presumed bug in gPROMS solver. Although gPROMS solver accept passing regular expressions as vector indices, it does not reevaluate the regular expression until a discontinuity is encountered, an *if* statement switches branches or the model is reinitialized after a discontinuity.

To resolve the above problem, I had to force evaluation of the regular expression through adding a dummy *if* statement. Only then, the model demonstrated acceptable results within reduced execution time. However, this resolution comes at a cost as I will demonstrate later.

[Borst, 2008] refers to the length of the regularization function with the symbol w as illustrated in Figure 3.1. Since the overlap domain is small enough to apply approach I to discontinuity resolution, one can easily relate w to h through the formula in equation 6.5.

$$w = 3h \tag{6.5}$$

There is always a physical meaning to the length (time span) of the regularizing function. In the PSA example, w refers to the amount of time it takes the valve to move from fully closed (0%) to fully open (100%) or vice versa. The valve travel speed can easily be calculated as:

$$v = \frac{100\%}{w} \tag{6.6}$$

From (6.6), we can easily deduce that $w=0$ (a discretized model) corresponds to a valve exhibiting an infinite speed. This is unrealistic. Moreover, with a regularized model, the modeller can study the effect of valve speed on process performance by varying w and possibly optimizing process performance through manipulating w. Thus, with regularization, we are able to add one more parameter to the PSA unit optimization problem. This addition couldn't have been brought into the optimisation problem had we used a discretized model.

In order to test the directional accuracy of the developed algorithm, I need to compare both the discretized and regularized models to a reference model. I could not locate any literature that discusses or experiments with the effect of valve dynamics on the operation of a PSA unit. So, I added a simplified valve model to the original disretized model. The resulting model (referred to as “reference model” hereafter) is still a discrete model. However, it assumes linear changes (not instantaneous) in flow overtime after each reinitialization between steps. This linear transformation closely mimics the operation of a motor operated valve (MOV) that is normally used in PSA units using conventional PSA modelling techniques. I should also stress that this model has its own flaws since it is still a discretized model. However, the closeness of this model results to one of the predefined models (discretised or regularized) over the other provides confidence in the obtained results. Last, the interval used to apply the linear change in flow for the reference model corresponds to w in the regularized model.

To ensure a unified starting point, I ran regularized and reference models at regularization interval of w=0.001 seconds. This value corresponds to a valve moving from a fully closed to fully open position or vice versa in 0.001 seconds. Although not realistic, it provides confidence that all models' will provide similar, if not exact, outputs at this valve

travel time. It should also be noted that *hermite* tension parameter is set to a value of one in all regularized models. Setting it to a value less than one generates a loose interpolation curve that results in a state variable limit violations. At w=0.001, all models reported almost exact figures. The calculated absolute error between all models was no more than 0.0004.

I then ran all models at w=5. To keep a fixed cycle length for all models, I divided the added regularization period w between adsorption and desorption steps of the discretized model as illustrated in Figure 6.4.

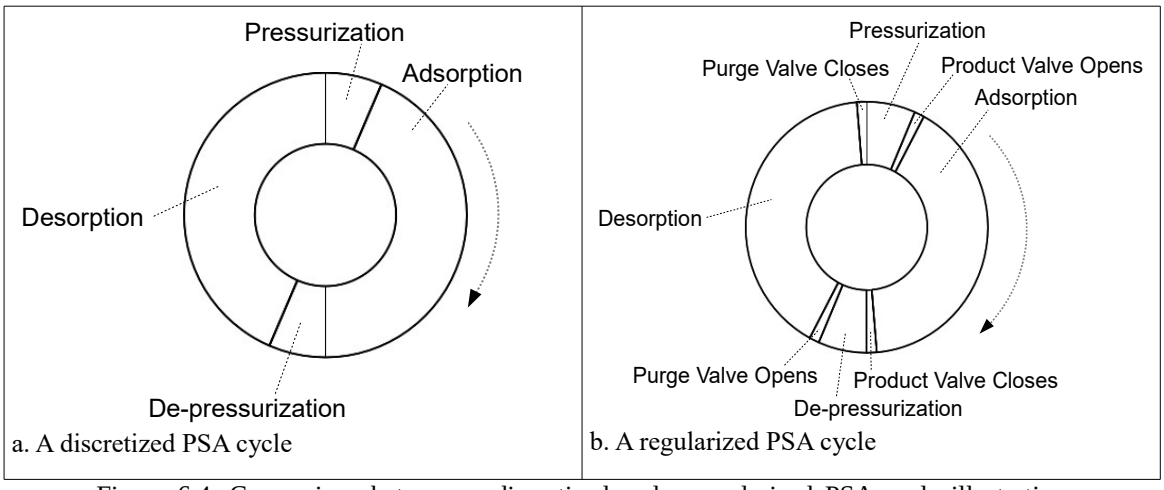
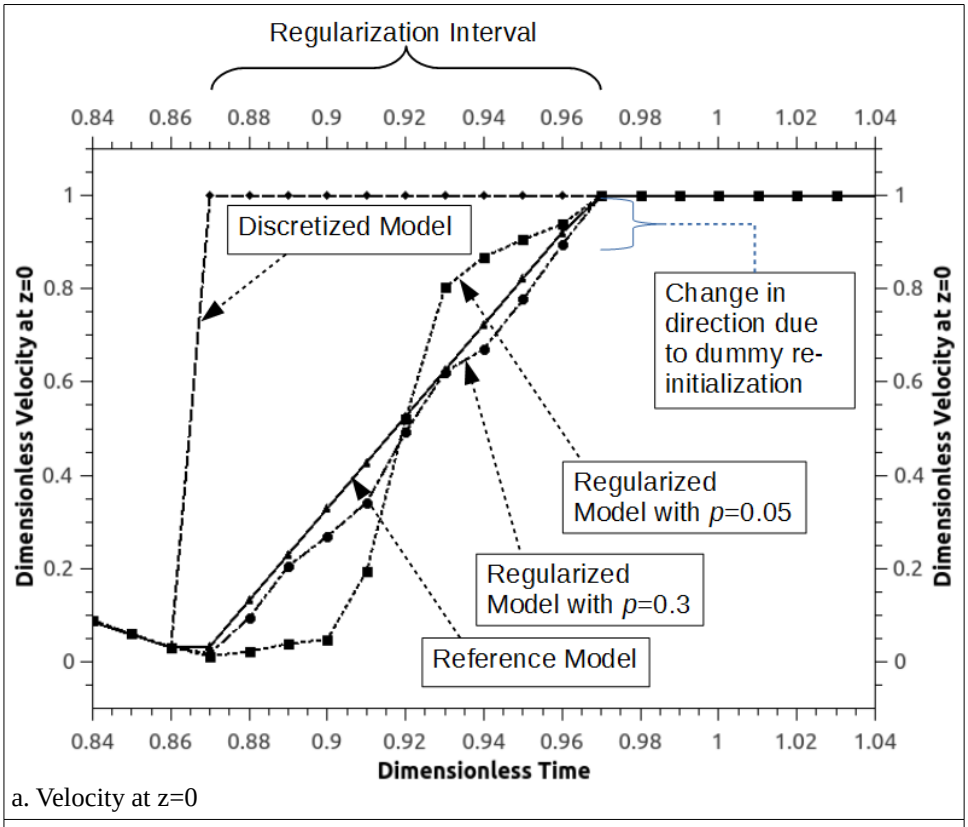
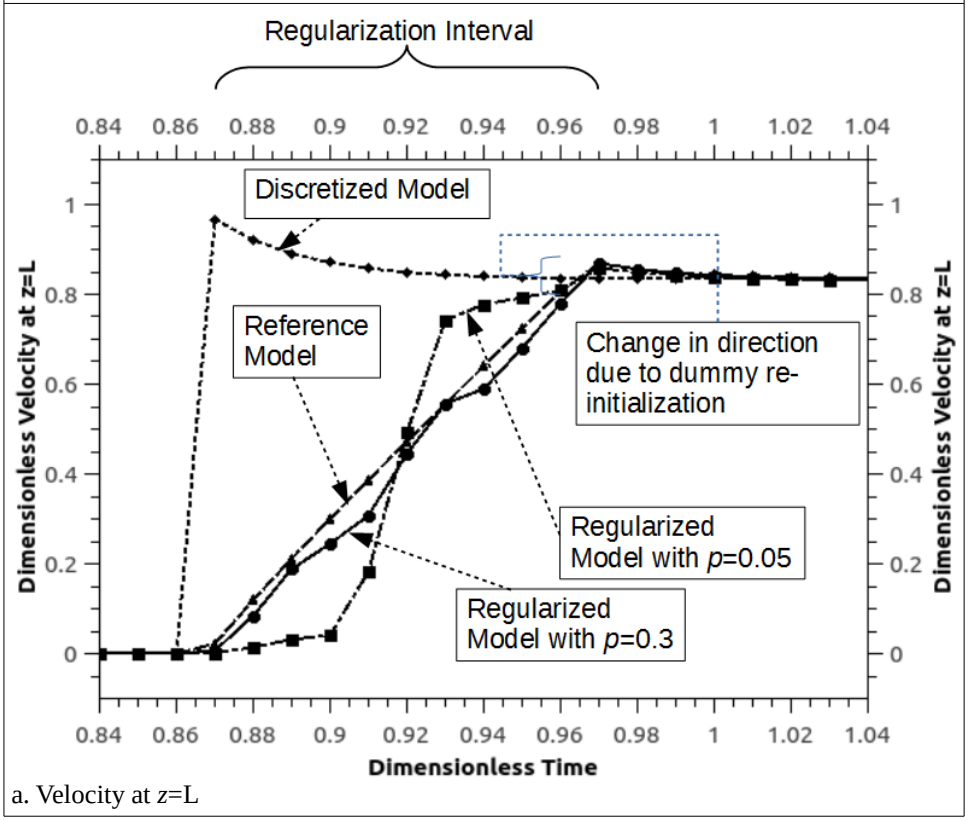


Figure 6.4: Comparison between a discretized and a regularized PSA cycle illustrating relative time span for each of the cycle steps and valve opening/closure span for w=10. The arrows indicate cycle direction.

The vessel velocity at z=0 is plotted in Figure 6.5a. The velocity at z=L is plotted in Figure 6.5b. Two curves representing regularization trends at p=0.05 and at p=0.3 are plotted to illustrate how the value of p changes the shape of the regularization curve. A p=0.3 is selected to closely mimic the reference model although I think a value of p=0.05 more resembles a typical valve behaviour. It should be noted that between Pressurization and Adsorption steps, the valve at z=L moves from 0 to 100% opening. This means that the initial condition for velocity at the interpolation region is set by the velocity at z=L (Figure 6.5b). Thus, the velocity at z=0 is a direct result of the ODE solution.



a. Velocity at z=0



a. Velocity at z=L

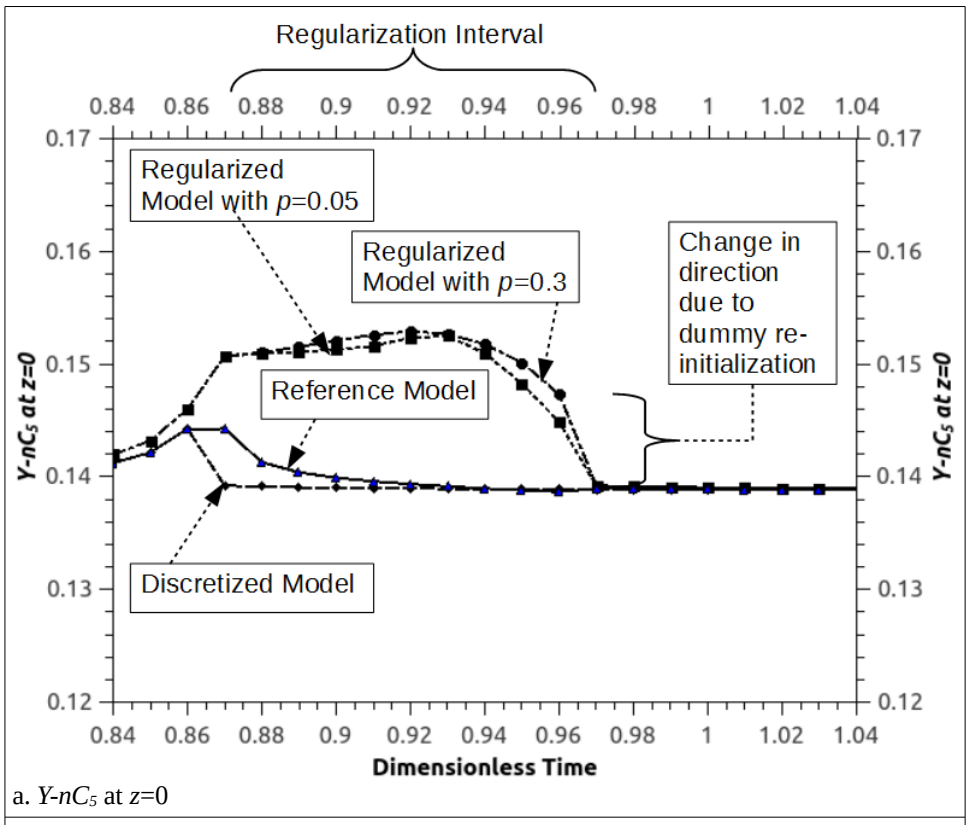
Figure 6.5: Curves representing velocity profiles at the period between Pressurization and Adsorption steps for both ends of the PSA column. The curves represent Reference, Discretized and Regularized models at w=5. For the Regularized model, curves representing p=0.05 and p=0.3 are plotted.

Although regularized models appear to follow the reference model, there is a fundamental difference between the curves. Since, the Reference model is a ramped-discretized model, the model shifts to the adsorption step before opening the valve. Since the velocity initial condition for the adsorption step is set at z=0, the reference model simulates the opening of the valve at z=0. This is a fundamentally flawed concept as the valve at z=0 has already been opened during the previous pressurization step. It should not be opened twice.

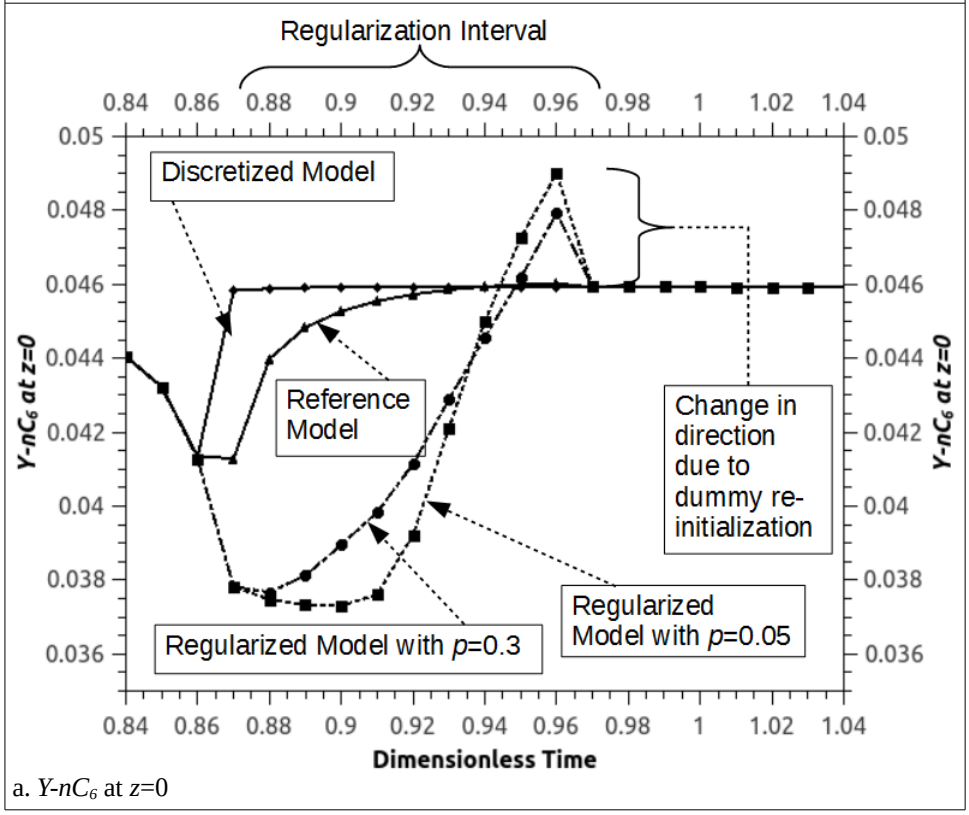
As can be seen from the discretized model, there is an instantaneous change in the velocity at z=0 from 0 to 1 (Figure 6.5a). The velocity maintains a value of 1 afterwards. Since the reference model is a ramped-discretized model, it follows the same path of the discretized model with the exception of the ramp. At the other end of the vessel (z=L), it can be noticed that for the discretized model, the velocity is calculated using the spatial differential equation. Thus, it jumps to an unacceptable value because of reinitialization. Then, the model corrects itself by recalculating subsequent velocity values based on model differential equations as illustrated in Figure 6.5b. On the other hand, the regularized model simulates the opening of the valve at z=L. Thus, it more resembles the actual process. The implications of this fundamental difference are evident in the concentration curves of Figures 6.6a and 6.6b for n-C_5 and n-C_6, respectively.

The sudden change in the direction of the concentration curves is due to the dummy reinitialization code implemented in gPROMS to force it to shift velocity boundaries as discussed earlier and outlined in equations 6.4a and 6.4b. As discussed, this is a bug in gPROMS software that should be addressed by [gPROMS, 2012] development team.

The reader should also note that for concentration profiles, the regularized model is not regularizing concentrations directly. It is rather regularizing their spatial derivatives (continuity of fluxes) as outlined in equations 6.2 and 6.3.



a. $Y\text{-}nC_5$ at z=0



a. $Y\text{-}nC_6$ at z=0

Figure 6.6: Curves representing concentration profiles for $n\text{-}C_5$ and $n\text{-}C_6$ at the period between Pressurization and Adsorption steps at z=0. The curves represent Reference, Discretized and Regularized models at w=5. For the regularized model, curves representing p=0.05 and p=0.3 are plotted.

The changes of concentration fluxes for all models at z=0 and z=L for the regularization interval extending between pressurization and adsorption steps are plotted in Figure 6.7. Note how regularized models inlet concentration flux increases with time until it reaches a maximum. Afterwards it continues declining to a value of zero. This behaviour is expected since opening the product end valve increases velocity across the column and hence allows components to move across the vessel. The spatial flux increases as velocity increases. Once velocity settles at the value corresponding to maximum valve opening, the inlet spatial flux starts dropping until it reaches a value of zero. Such a phenomena is hardly noticeable in the discretized models because of the rapid reintialization.

Figures 6.8a and 6.8b illustrate velocity profiles for the regularization interval between adsorption and depressurization steps. After adsorption step is complete, the valve at z=L is closed. Thus, the initial velocity condition is set at z=L. Velocity changes at z=0 follow the calculated profiles based on the differential equation. All models simulate this behaviour regardless of the regularization interval. Note that the sharp decline in velocity at z=0, to the right of the regularized and reference model curves of Figures 6.8a, is a direct result of the dummy reinitialization that is discussed earlier and outlined in equations 6.4a and 6.4b. At this step, the reinitialization is required to shift the location of the velocity boundaries from z=0 for adsorption step to z=L for depressurization step.

Figure 6.9 demonstrates how concentration profiles for the respective n-C_5 and n-C_6 components change across the transition between adsorption and depressurization steps. Although very small, the effect of the dummy reinitialization is also noticed in the concentrations of both components. The dummy reinitialization will only be evident in the first regularization step between pressurization and adsorption steps and in the second regularization step between adsorption and depressurization. The model changes the

velocity initial condition location from z=L to z=0 in the first regularization step and from z=0 to z=L in the second regularization step. Transitions between other steps do not require dummy reinitialization as their velocity initial condition locations are set at z=L.

Figure 6.10a illustrates the change in inlet spatial concentration derivatives (fluxes) for the period between adsorption and depressurization steps. The peaks of the regularized models are expected. As the valve at z=L closes, the back-end flux reduces. The front-end flux also reduces. However, due to the negative slope of the velocity profile, the inlet flux exhibits an increase. As the valve further closes, the negative slope of the velocity profile decreases resulting in a decrease in inlet flux.

The negative flux represented by the reference model is due to the pre-mature change in concentration boundary conditions. For the reference model, concentration boundary conditions change from those representing adsorption to those representing depressurization before valve closure. This premature change results in concentration flux moving towards the feed end instead of moving towards the product end. The discretized model maintains the same boundary conditions and fluxes throughout the regularization period before switching to depressurization boundary conditions immediately after the regularization period. Thus, no change is noticed in the flux of the discretized model during the regularization period.

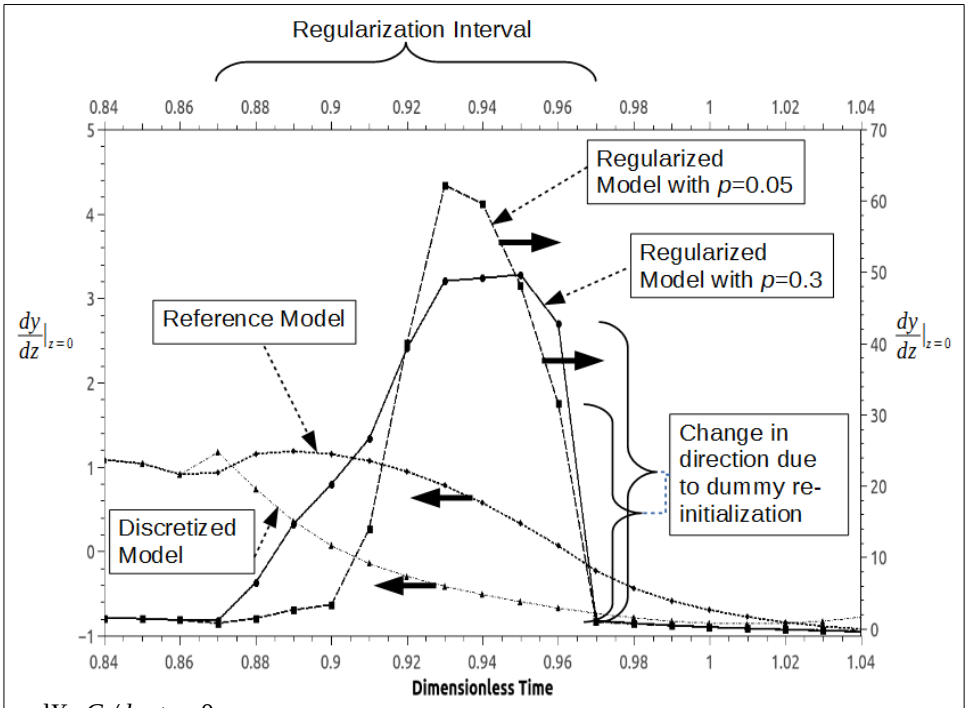
Concentration fluxes at z=L (Figure 6.10b) do not change because the boundary conditions at this location are the same for both adsorption and depressurization steps.

The velocity profiles for the regularization period between de-pressurization and desorption steps are plotted in Figures 6.11a and 6.11b for the respective ends of the vessel at z=0 and z=L. Figure 6.12a illustrates the concentration profile for n-C_5 at z=0 while Figure 6.12b illustrates n-C_6 concentration profile at the same end. Note the

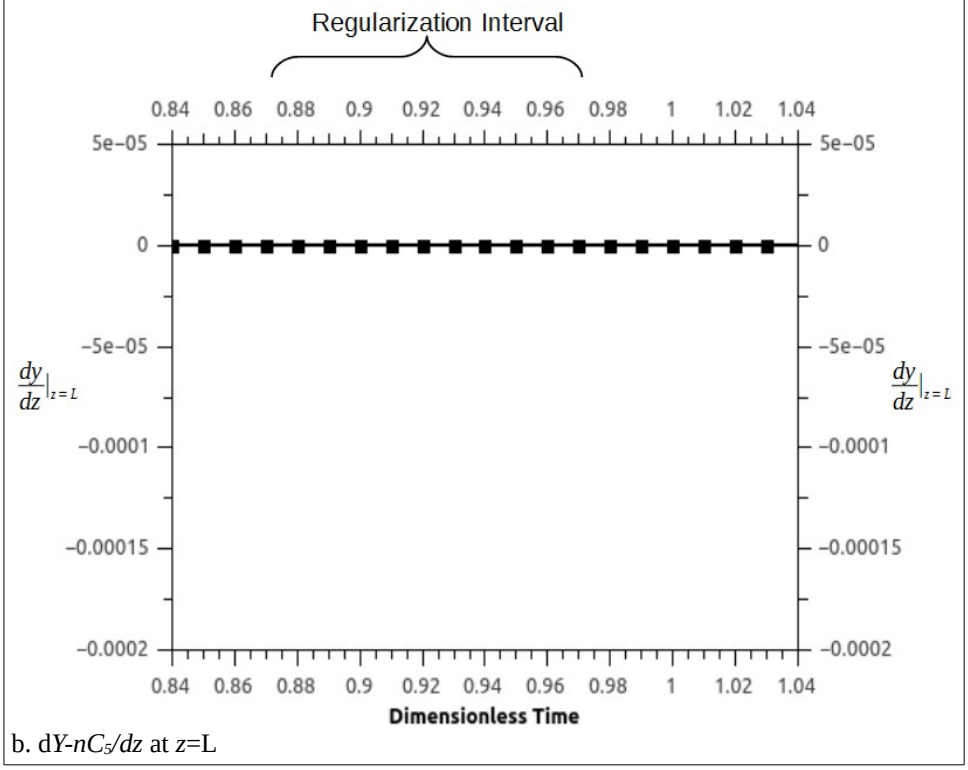
continuity in the profiles for the regularized and reference models because of the absence of reinitialization.

Figures 6.13a and 6.13b illustrate the changes in spatial flux at z=0 and z=L, respectively. The reason behind no observable flux changes at z=0 is because boundary conditions for depressurization and desorption steps at this location are the same. The noticeable jump in flux curves at the end of the regularization period (marked as 1 in the figure) is due to the concentration flux reaching its intended desorption value. Thus, the flux afterwards drops to zero indicating a perfect match between the final value reported by the interpolating polynomial and the destination function (inlet flux of desorption step).

Before discussing regularization curves for the period between desorption and pressurization steps, it is worth shedding some light on how inlet velocity is calculated during pressurization step. For the parabolic profile, this velocity instantaneously changes from a value of 0 to 15 times that of the feed velocity. For the exponential velocity profiles, the initial inlet velocity depends on pressurization rate M_p. However, regardless of the value of M_p, pressurization is almost always instantaneous. The exception is associated with low values of M_p which are not representative of the system.



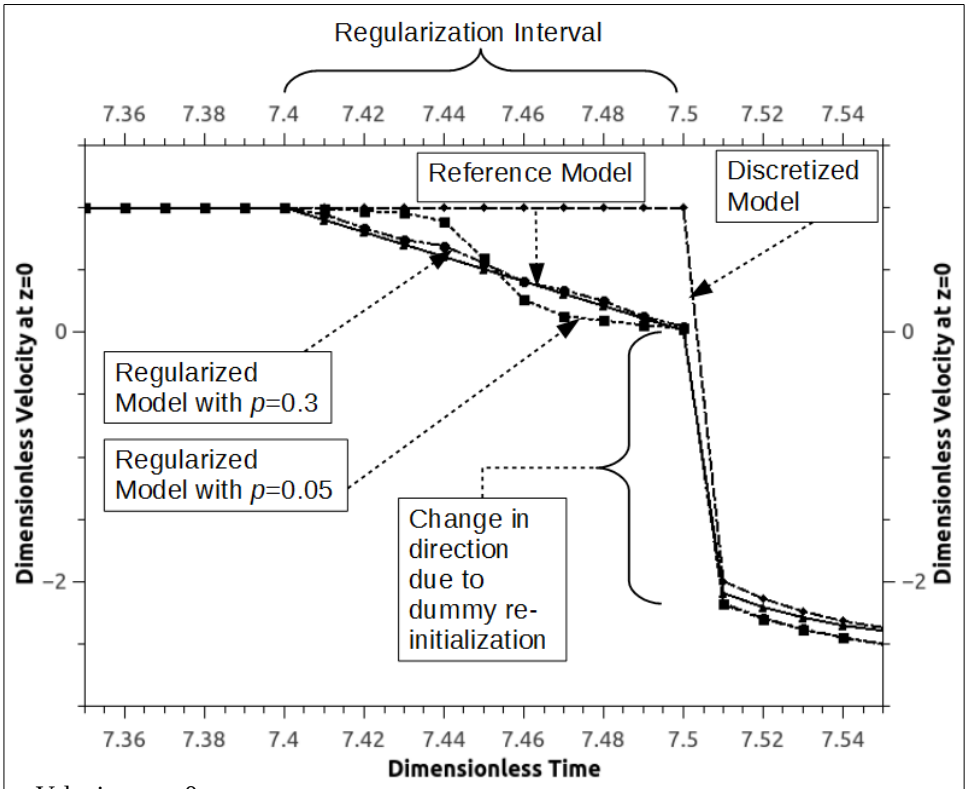
a. d*Y-nC₅/dz* at z=0



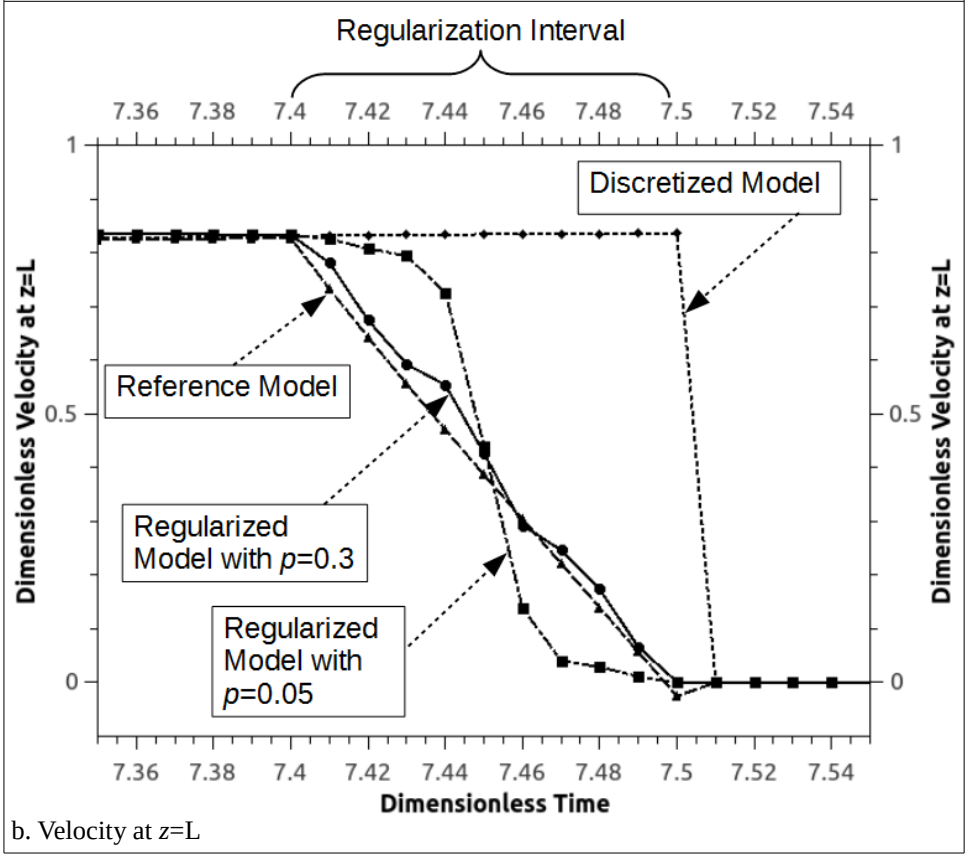
b. d*Y-nC₅/dz* at *z*=L

Figure 6.7: Curves representing the change in concentration spatial derivatives at both ends of the PSA column between pressurization and adsorption steps. The curves represent reference, discretized and regularized models at *w*=5. For the regularized model, curves representing *p*=0.05 and *p*=0.3 are plotted.

For the Figure 6.7b, all curves are superimposed on each other.

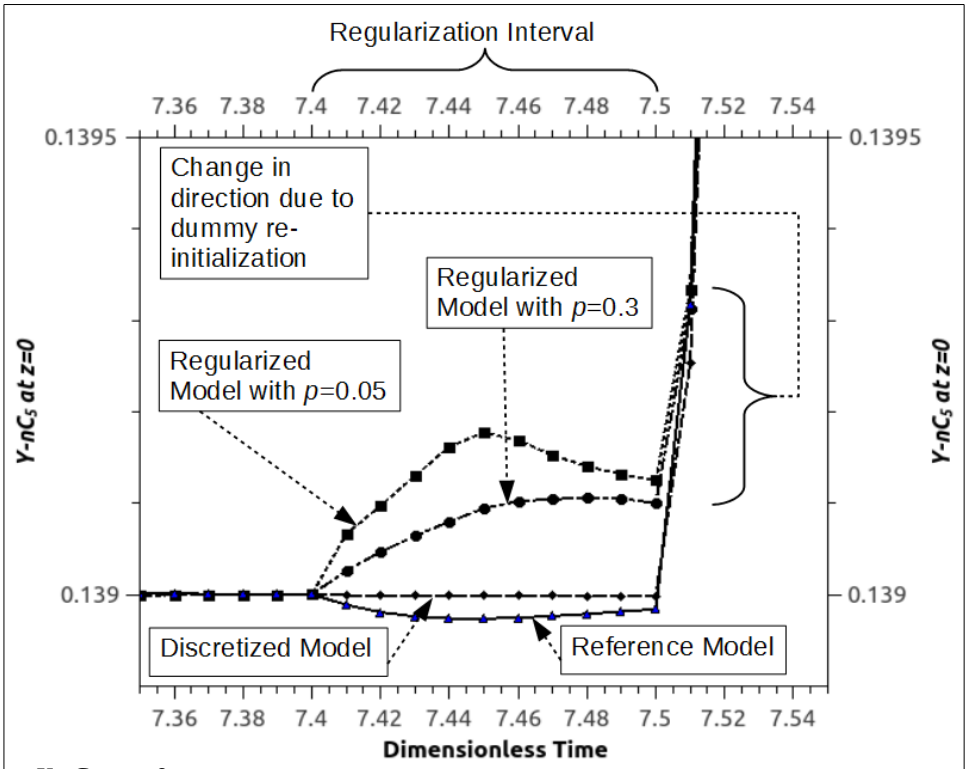


a. Velocity at z=0

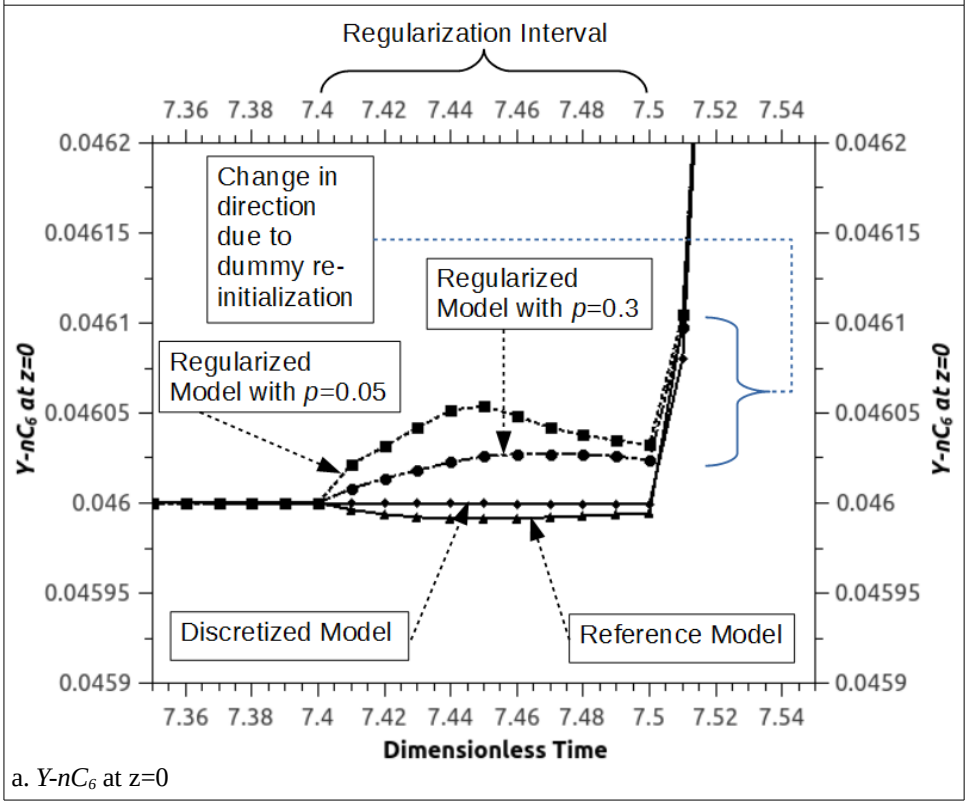


b. Velocity at z=L

Figure 6.8: Curves representing velocity profiles at the period between adsorption and depressurization steps for both ends of the PSA column. The curves represent reference, discretized and regularized models at w=5. For the regularized model, curves representing p=0.05 and p=0.3 are plotted.

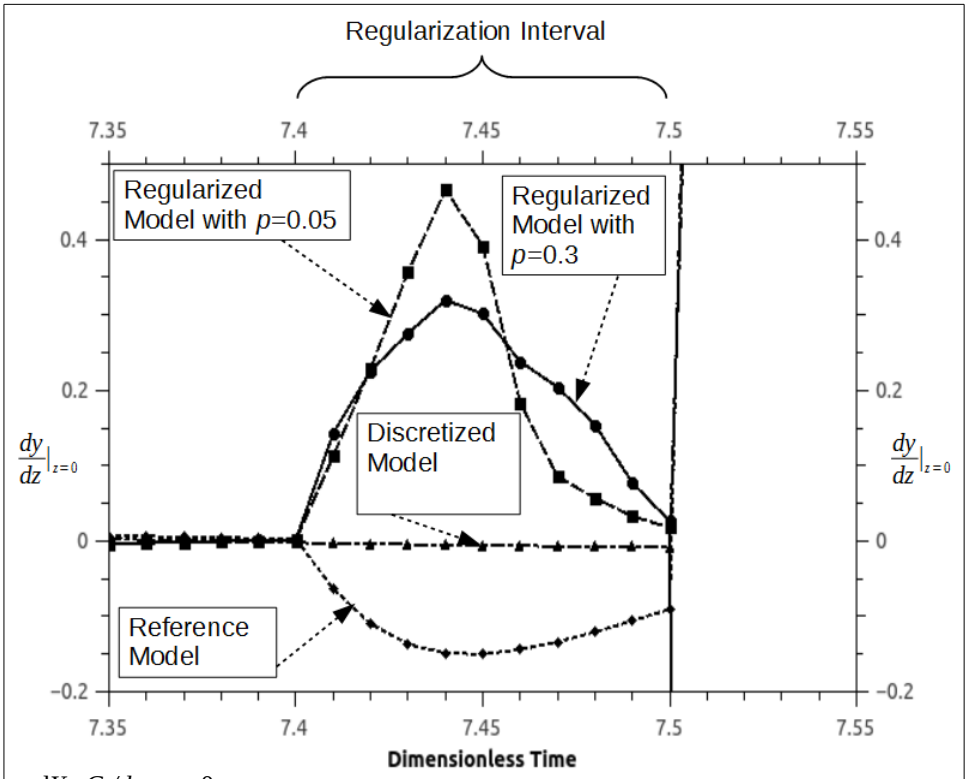


a. *Y-nC*$_5$ at z=0

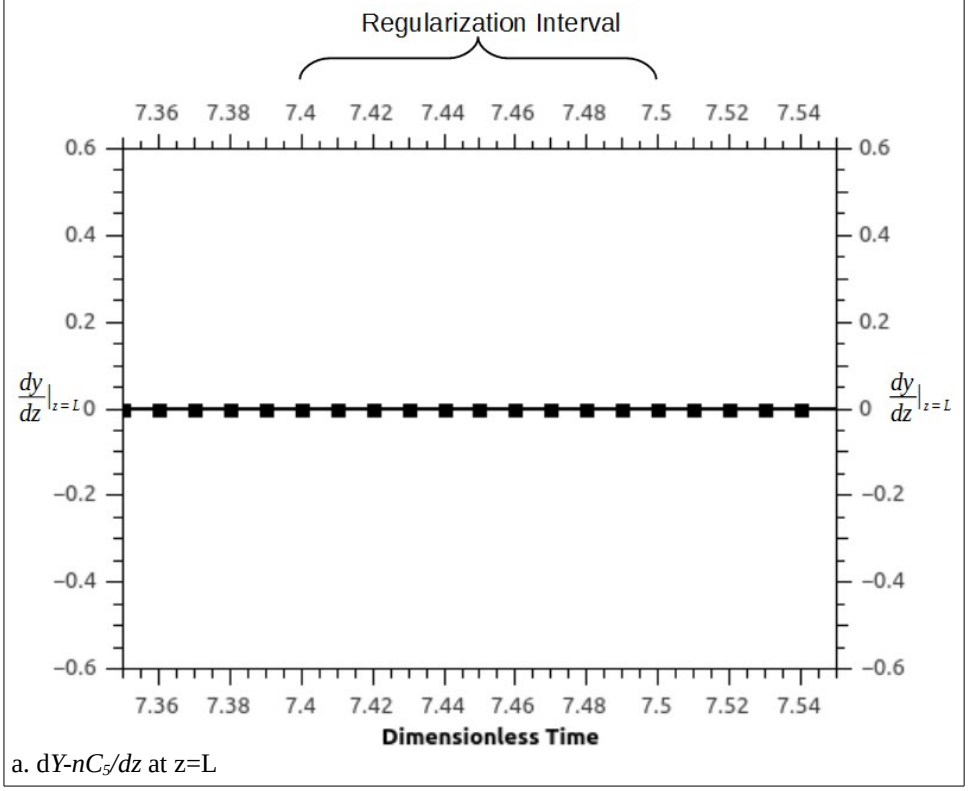


a. *Y-nC*$_6$ at z=0

Figure 6.9: Curves representing concentration profiles for *n*-C_5 and *n*-C_6 at the period between adsorption and de-pressurization steps at z=0. The curves represent Reference, Discretized and Regularized models at w=5. For the Regularized model, curves representing p=0.05 and p=0.3 are plotted.



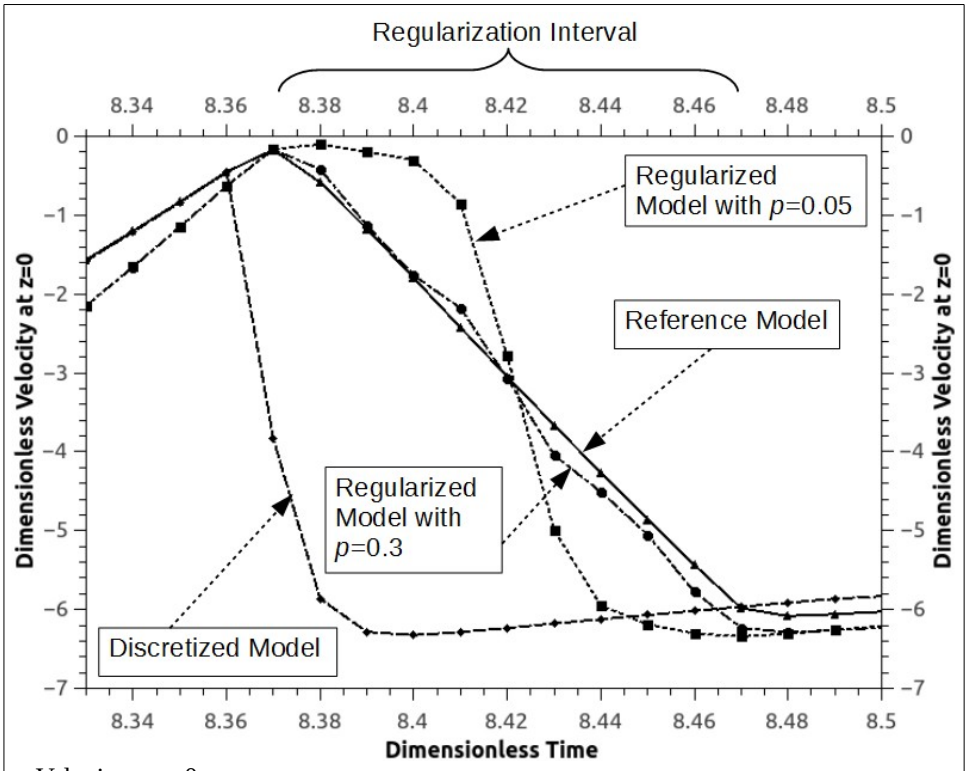
a. d*Y-nC₅/dz* at z=0



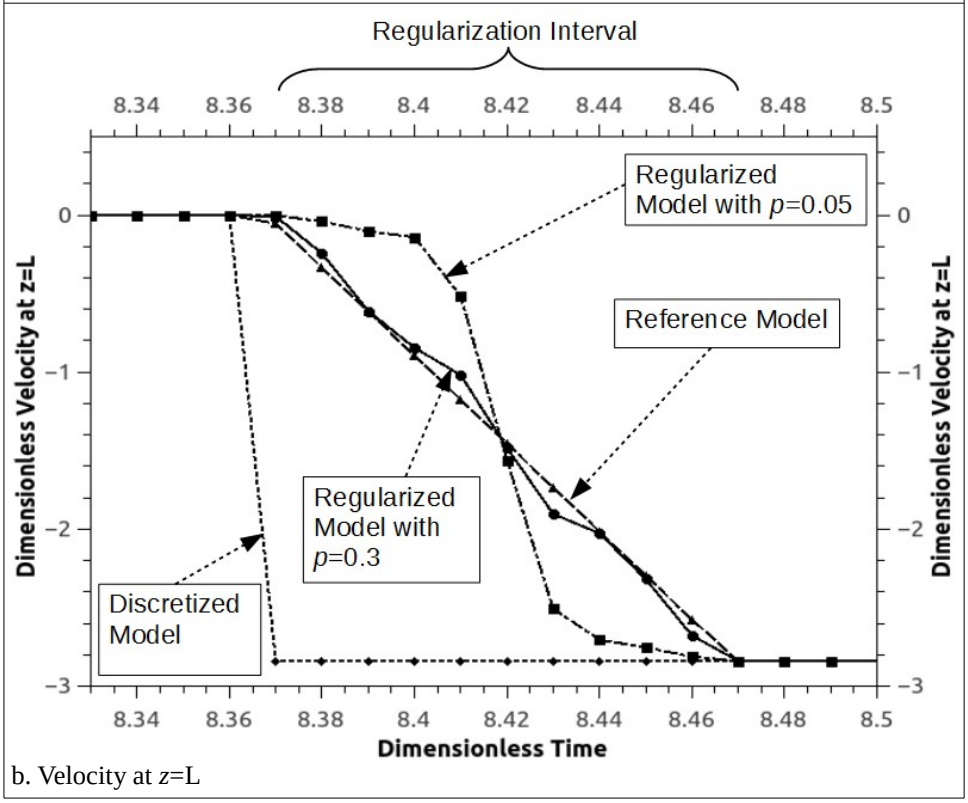
a. d*Y-nC₅/dz* at z=L

Figure 6.10: Curves representing the change in concentration spatial derivatives at both ends of the PSA column between adsorption and depressurization steps. The curves represent reference, discretized and regularized models at w=5. For the regularized model, curves representing p=0.05 and p=0.3 are plotted.

For Figure 6.10b, all curves are superimposed on each other.

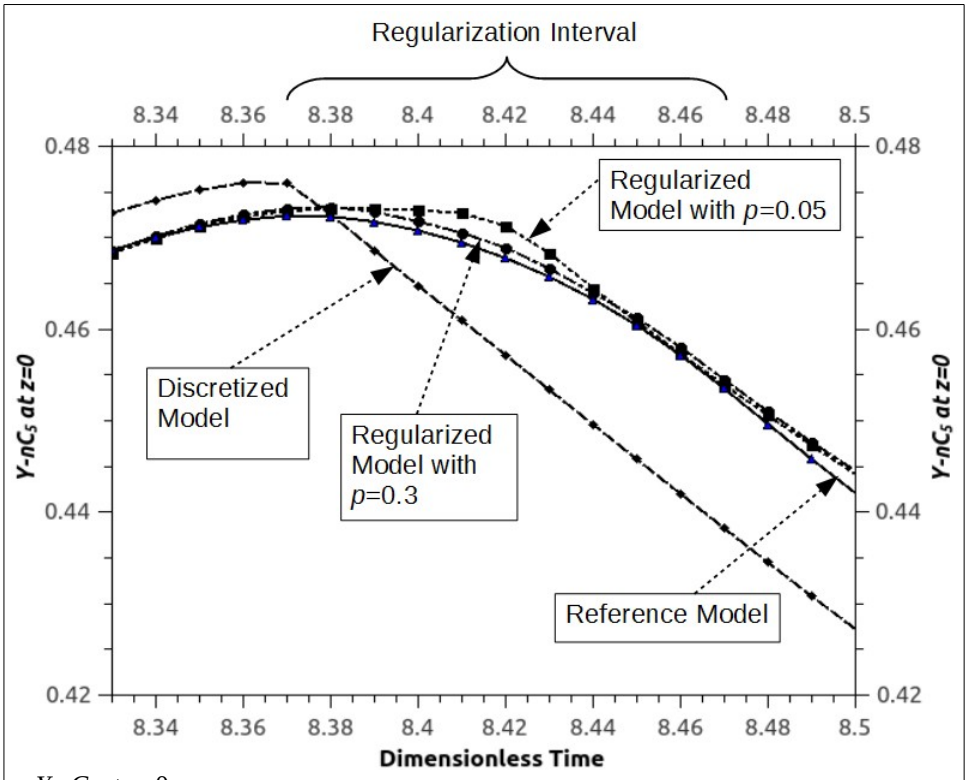


a. Velocity at z=0

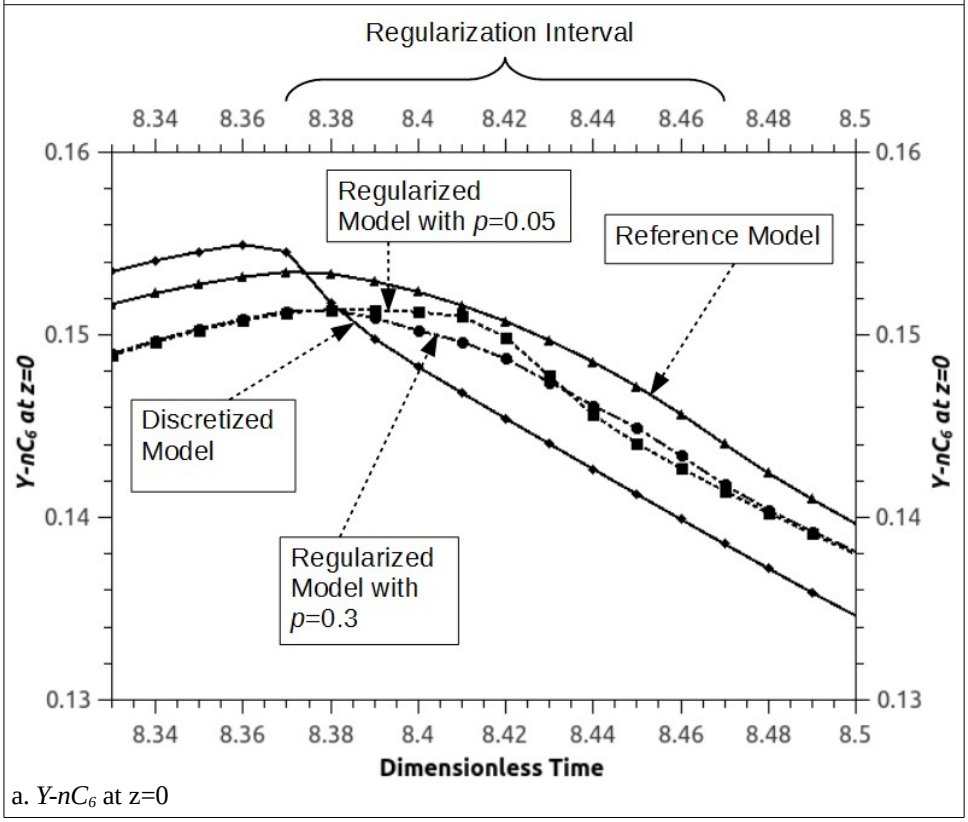


b. Velocity at z=L

Figure 6.11: Curves representing velocity profiles at the period between de-pressurization and desorption steps for both ends of the PSA column. The curves represent reference, discretized and regularized models at w=5. For the Regularized model, curves representing p=0.05 and p=0.3 are plotted.

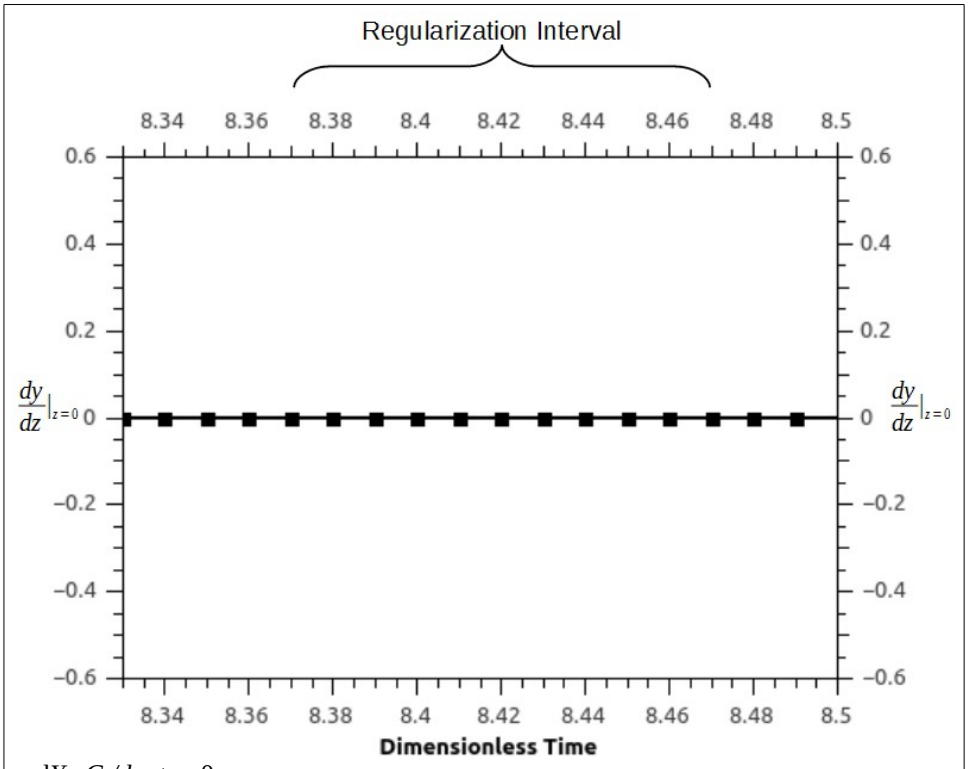


a. $Y\text{-}nC_5$ at z=0

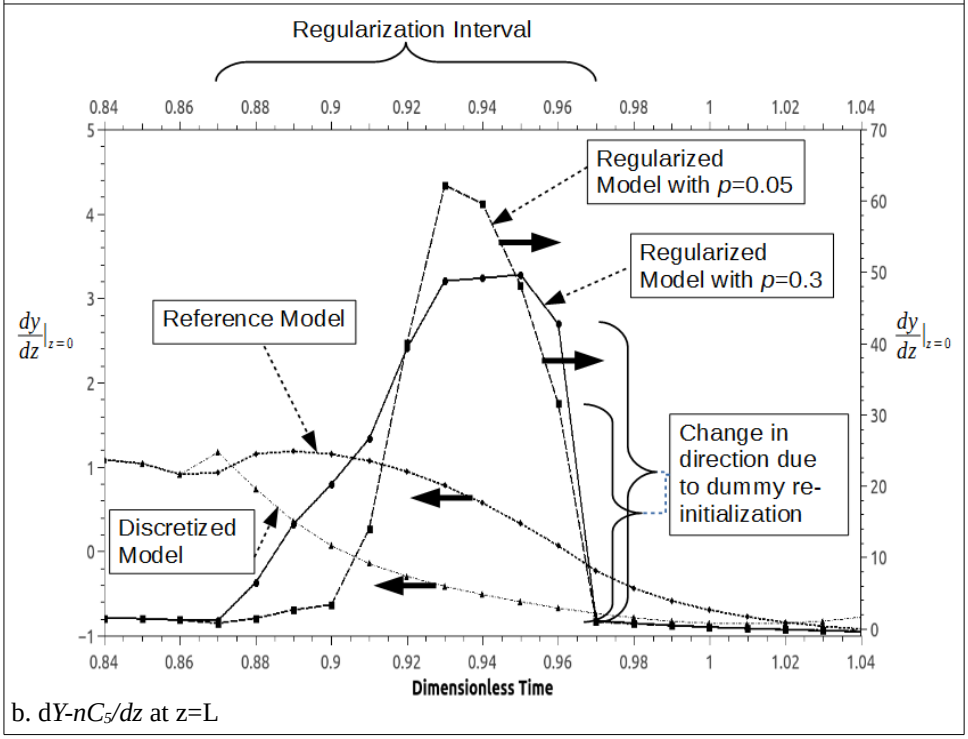


a. $Y\text{-}nC_6$ at z=0

Figure 6.12: Curves representing concentration profiles for $n\text{-}C_5$ and $n\text{-}C_6$ at the period between de-pressurization and desorption steps at z=0. The curves represent reference, discretized and regularized models at w=5. For the Regularized model, curves representing p=0.05 and p=0.3 are plotted.



a. d*Y-nC₅/dz* at z=0



b. d*Y-nC₅/dz* at z=L

Figure 6.13: Curves representing the change in concentration spatial derivatives at both ends of the PSA column between depressurization and desorption steps. The curves represent reference, discretized and regularized models at w=5. For the regularized model, curves representing p=0.05 and p=0.3 are plotted.

For the Figure 6.13a, all curves are superimposed on each other.

Such a sudden change in velocity profile does not correspond to the reality of a continuous process. Also, since this sudden change is hard coded as a change in the constitutive equation value that results from a change in boundary conditions (not as a conditional statement), it becomes hard to detect and regularize. In such cases, if the modeller is not willing to alter the model to a better one that more accurately represents the inherent dynamics, there would be no escape from reinitializing the model between desorption and pressurization steps. Simply stated, there is no substitute for good modelling practices.

Note that there are two sources for this discontinuity. The first source is the switch in boundary conditions between the desorption step that exhibits constant counter-current flow to that of the pressurization step. The second source is the formulation of the pressure profile equation (whether using parabolic or exponential profile equation). Both equations assume an instantaneous change in inlet pressure from P_{low} to P_{high}. The first source can be eliminated through one-interval regularization that was discussed in the previous chapter. The second source requires reformulation of the pressure profile equations. A complete derivation of a novel velocity calculation function is discussed in Appendix A. The novel velocity calculation approach is used to calculate velocity profiles in the constructed PSA column.

Figures 6.14a and 6.14b illustrate the velocity profiles for the period between desorption and pressurization at z=0 and z=L, respectively. Note that at this transition, the active velocity initial condition is located at z=L. Concentration profiles are illustrated in Figures 6.15a and 6.15b for normal pentane and hexane, respectively. To better illustrate the transition, Figure 6.15a is magnified in Figure 6.16. Similarly, 6.15b is magnified in Figure 6.17. The noticeable sudden shifts in concentration profiles trended in Figures

6.15a and 6.15b after the regularization period are due to the introduction of the fresh feed that possesses differing concentrations from those encountered at the end of the desorption step.

Spatial fluxes for the desorption-pressurization regularization periods at *z*=0 and *z*=L are trended in Figures 6.18a and 6.18b, respectively. The sudden changes in fluxes of the discretized and reference models at *z*=L are due to model reinitialization. The values of these fluxes should have stayed at zero due to the restriction imposed by the boundary condition. However, reinitialization deviated the values from their intended path. Note how the regularized models maintain the flux at the value imposed by the boundary condition.

Now, let us shed some light on the accuracy of the developed algorithm when compared to conventional discretization algorithms. I used inlet and exit velocities as basis for the comparison. Inlet and exit concentrations or their respective spatial fluxes cannot be used as a base for comparison because each is dependent on the velocity profile. I used the reference model as a base for the comparison although it has its own flaws. For each of the steps, the cumulative relative error in dimensionless velocity that spans the entire regularization period is calculated as:

$$E_C\big|_{z=0 \vee z=1} = \sum_{i=1}^{n} \left| \frac{\left| v_i - v_{i,ref} \right|}{v_{i,ref}} \right| \tag{6.7a}$$

The cumulative errors calculated for each of the steps at z=0 and z=L are tabulated in Tables 6.3 and 6.4, respectively. It should be noted that the increased accuracy of the regularized model with *p*=0.30 over the one with *p*=0.05 is primarily because the regularized model with *p*=0.30 closely resembles the profile of the reference model. Nevertheless, I think the regularized model with *p*=0.05 more resembles a real valve

operation as the velocity profile starts with a non-linear range between valve opening and flow. It then follows that with a linear range before closing the opening-velocity curve with another non-linear profile.

Table 6.3: Cumulative relative error in velocity at z=0 spanning regularization interval

Regularization Period	Cumulative Relative Error in Velocity at z=0		
	Discretized	Regularized at p=0.05	Regularized at p=0.30
Pressurization-Adsorption	45	4	1
Adsorption-Depressurization	90	5	3
Depressurization-Desorption	41	5	7
Desorption-Pressurization	47	5	1

Table 6.4: Cumulative relative error in velocity at z=L spanning regularization interval

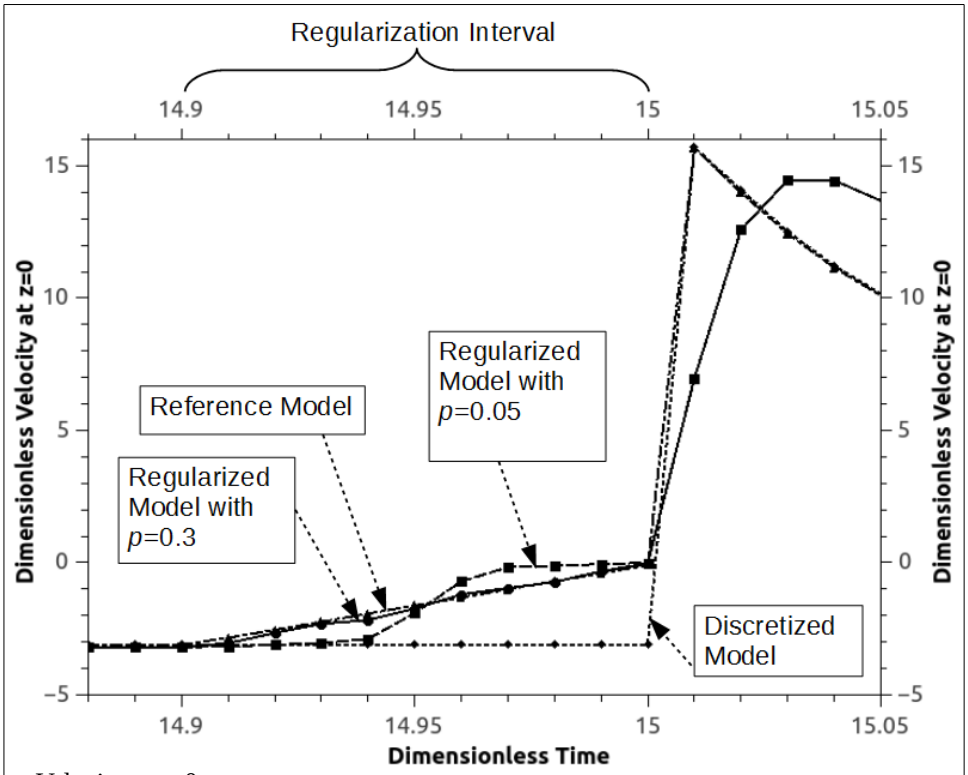
Regularization Period	Cumulative Relative Error in Velocity at z=L		
	Discretized	Regularized at p=0.05	Regularized at p=0.30
Pressurization-Adsorption	58	5	2
Adsorption-Depressurization	60	5	2
Depressurization-Desorption	68	5	1
Desorption-Pressurization	52	5	1

To further illustrate differences between discretized and regularized models, the cumulative difference in n-C_5 and n-C_6 concentrations at z=0 between the discretized model and the reference one and its regularized counterpart (p=0.05) are plotted in Figure 6.19 for values of w=5 and w=10. The x-axis time spans a full PSA cycle. Note how the regularized model always provides better results over the discretized one. It is also arguable that the regularized model provides better results than the reference model itself. The error analysis clearly indicate the substantial increase in accuracy of the developed algorithm over conventional discretization algorithms.

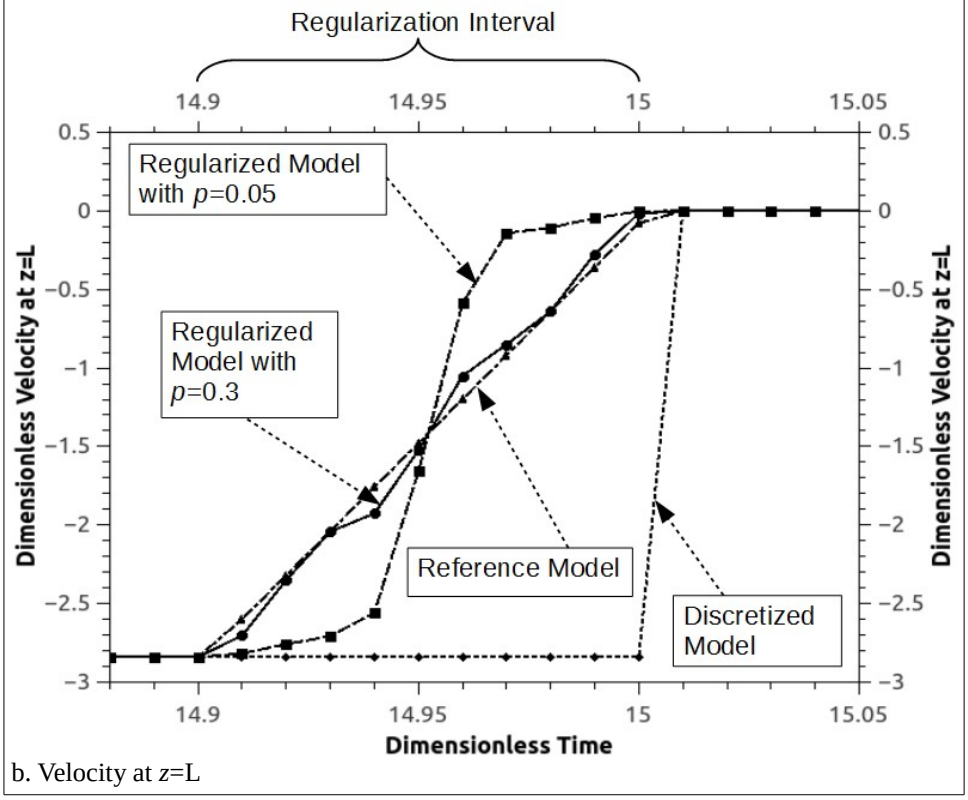
Moreover, what adds to the accuracy of the developed algorithm is the strict adherence of

the interpolating polynomial to the bounds set by the model equations. Figures 6.5b and 6.18b clearly demonstrate how a discretized solution violates model bounds at the reintializatation time. Although the error is corrected by the model equations in subsequent steps, the introduced error resides in the calculation of the cumulative error and alters the subsequent model solution path. We can comfortably conclude that regularization supersedes discretization.

Appendix E demonstrates how the concepts, presented in Chapter 5 and demonstrated by the applications in this chapter, are coded in C++

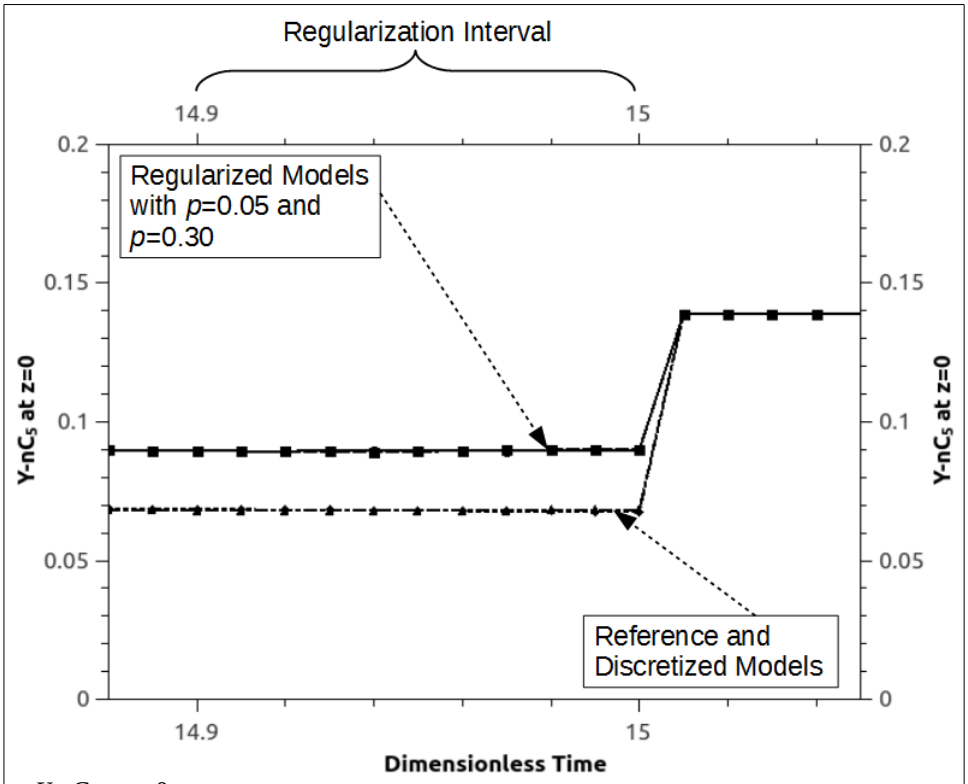


a. Velocity at z=0

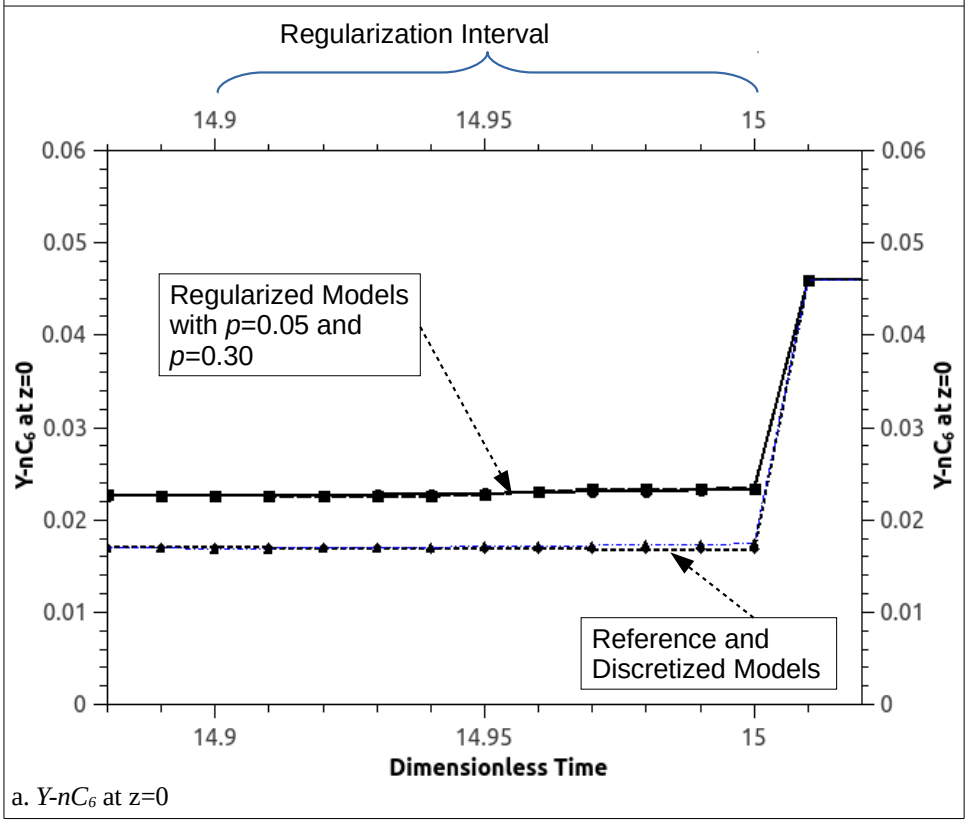


b. Velocity at z=L

Figure 6.14: Curves representing velocity profiles at the period between desorption and pressurization steps for both ends of the PSA column. The curves represent reference, discretized and rregularized models at w=5. For the Regularized model, curves representing p=0.05 and p=0.3 are plotted.

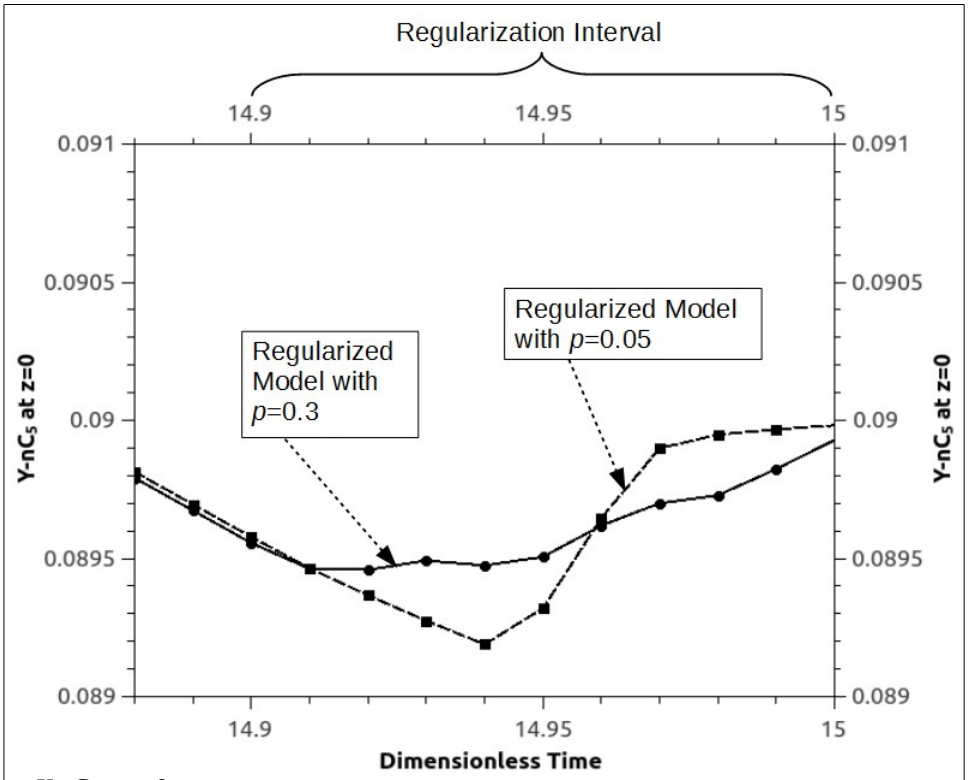


a. *Y-nC*$_5$ at z=0

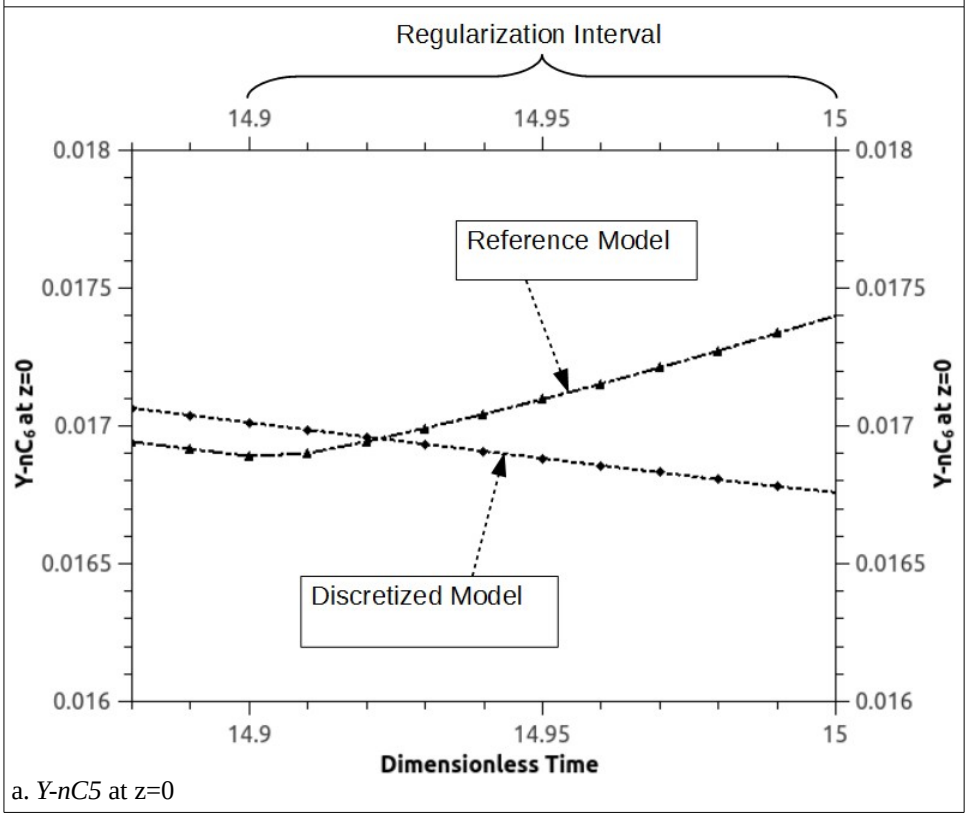


a. *Y-nC*$_6$ at z=0

Figure 6.15: Curves representing concentration profiles for *n*-C_5 and *n*-C_6 at the period between desorption and pressurization steps at z=0. The curves represent reference, discretized and regularized models at w=5. For the Regularized model, curves representing p=0.05 and p=0.3 are plotted.

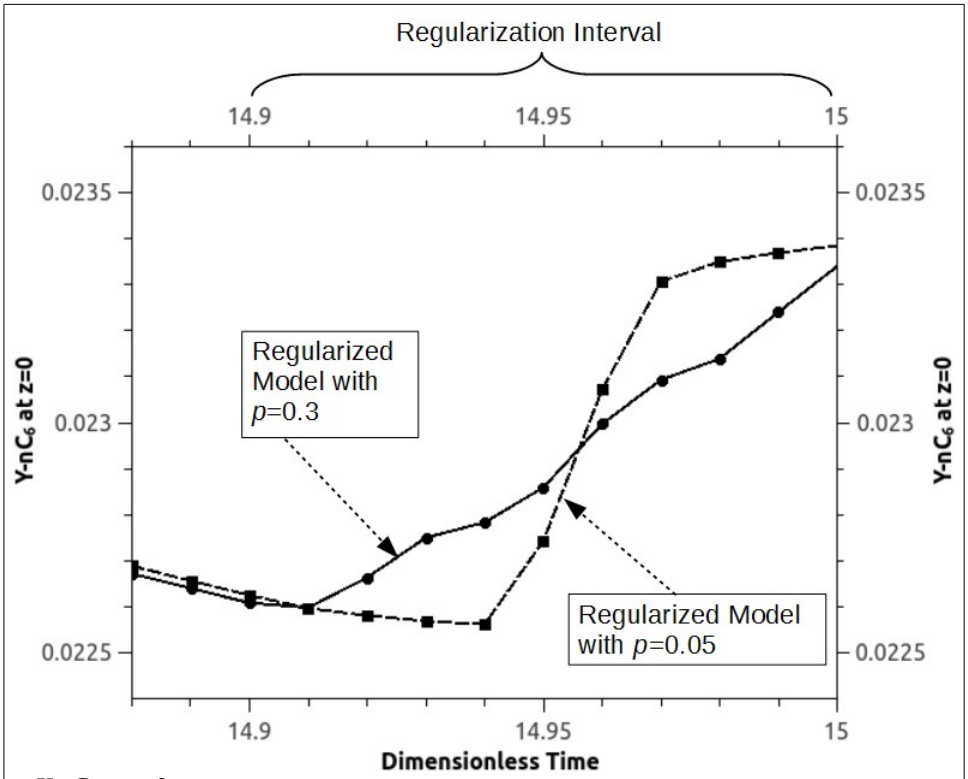


a. *Y-nC₅* at z=0

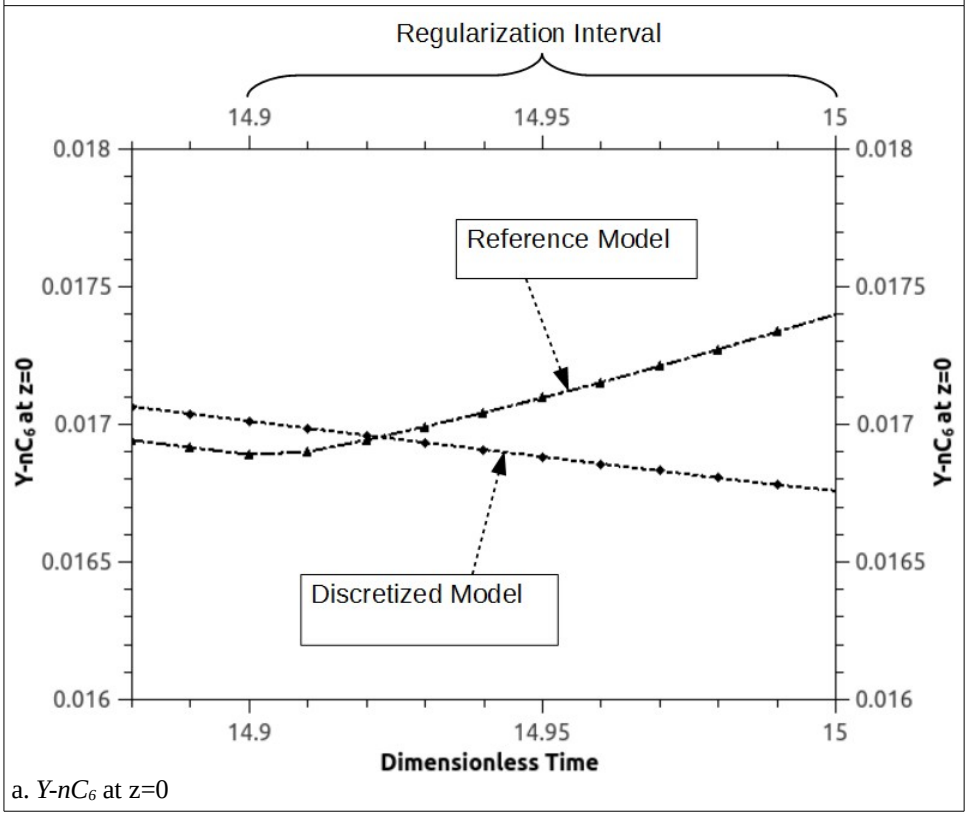


a. *Y-nC5* at z=0

Figure 6.16: Magnified version of the curves presented in Figure 6.15a illustrating concentration profiles for *n*-C_5 at the period between desorption and pressurization steps at z=0. The curves represent reference, discretized and regularized models at *w*=5. For the Regularized model, curves representing *p*=0.05 and *p*=0.3 are plotted.

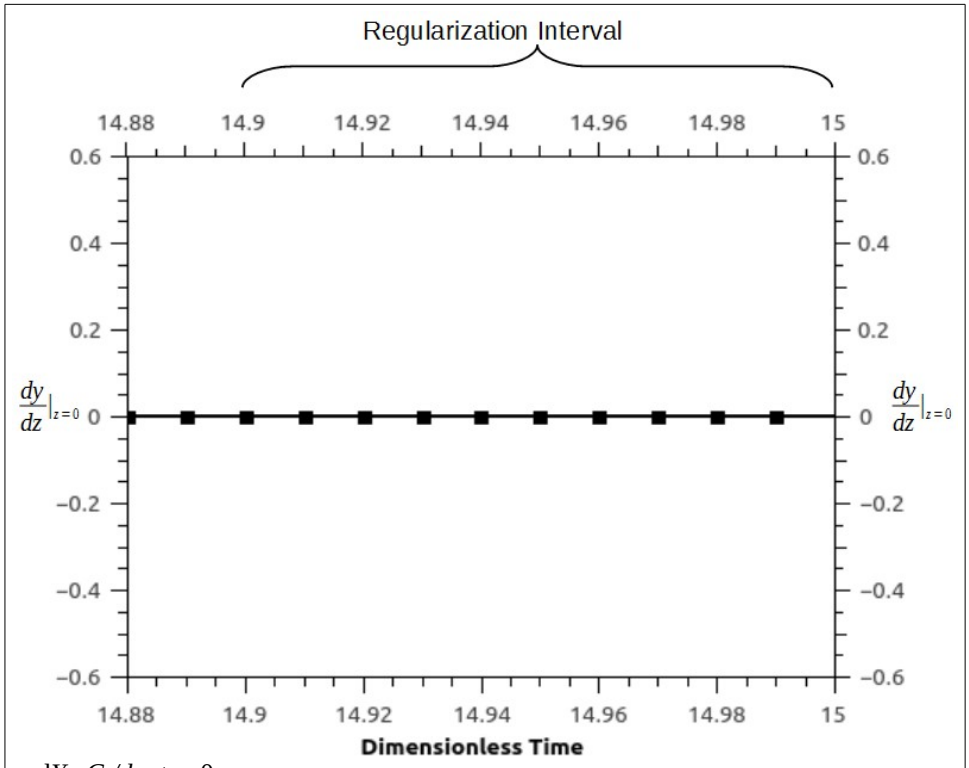


a. *Y-nC₆* at z=0

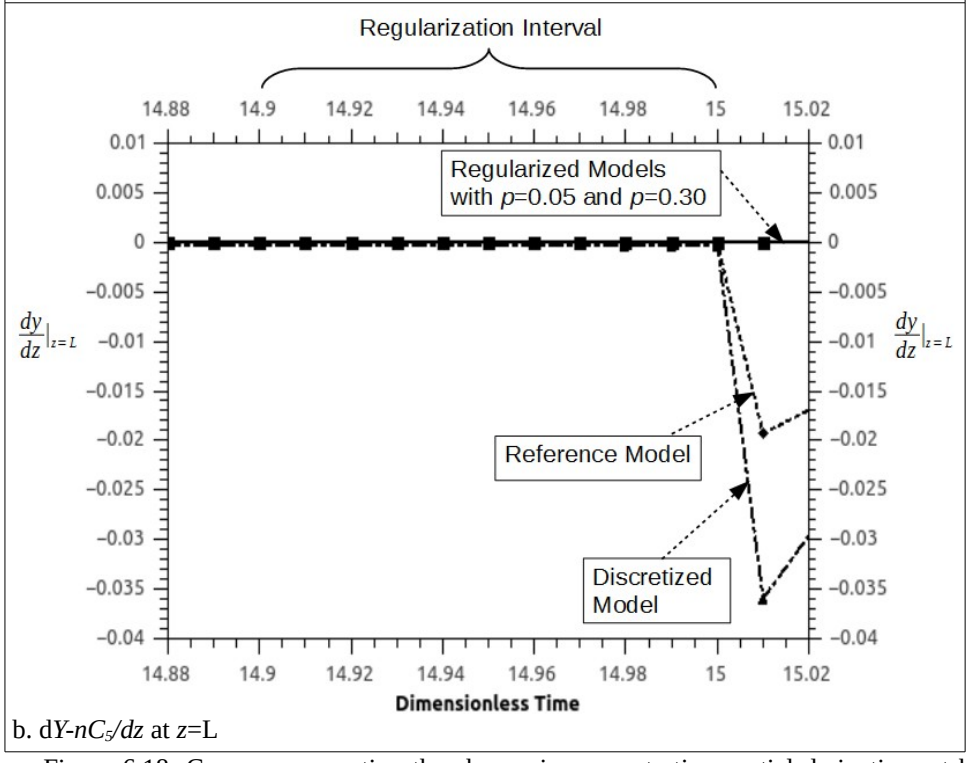


a. *Y-nC₆* at z=0

Figure 6.17: Curves representing concentration profiles for*n-C₆* at the period between desorption and pressurization steps at z=0. The curves represent reference, discretized and regularized models at *w*=5. For the Regularized model, curves representing *p*=0.05 and *p*=0.3 are plotted. Curves are identical for all models. Thus, only one curve appears in each of the figures.



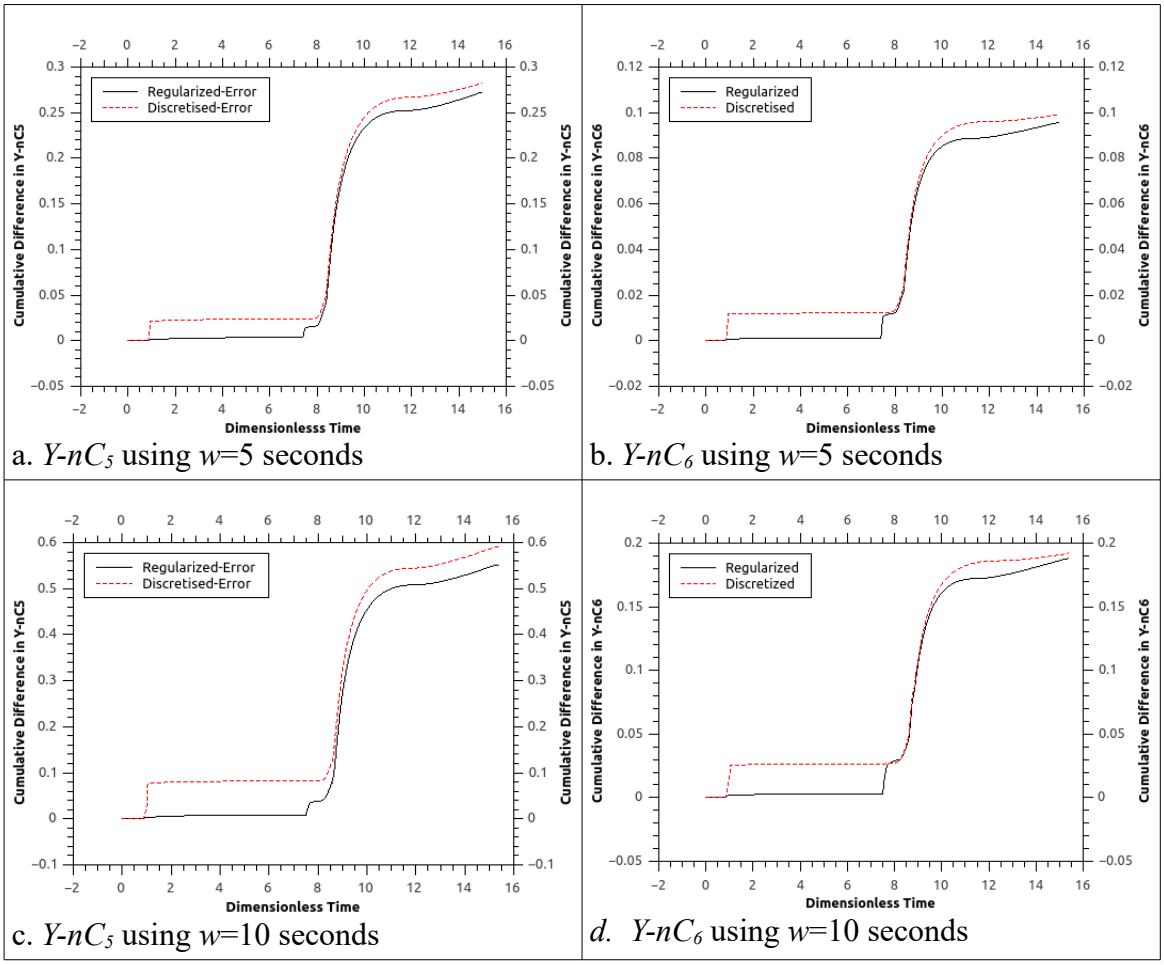
a. d*Y-nC₅/dz* at *z*=0



b. d*Y-nC₅/dz* at *z*=L

Figure 6.18: Curves representing the change in concentration spatial derivatives at both ends of the PSA column between desorption and pressurization steps. The curves represent reference, discretized and regularized models at *w*=5. For the regularized model, curves representing *p*=0.05 and *p*=0.3 are plotted.

For the Figure 6.18a, all curves are superimposed on each other.



a. *Y-nC₅* using *w*=5 seconds

b. *Y-nC₆* using *w*=5 seconds

c. *Y-nC₅* using *w*=10 seconds

d. *Y-nC₆* using *w*=10 seconds

Figure 6.19: The cumulative difference between *Y-nC₅* and *Y-nC₆* inlet concentrations (*z*=0) predicted by the discretized and regularized models (*p*=0.05) compared to the reference model after the first PSA cycle.

6.3. Summary and Concluding Remarks

In this chapter, I demonstrated how the algorithm that was developed in Chapter 5 can be applied to regularize one and two dimensional discontinuous composite functions. I demonstrated the application to one-dimensional models by implementing the algorithm in a PSA column model. The algorithm is used to regularize the change in the boundary conditions between the steps of a [Skarstrom, 1960] cycle. It is also used to regularize the velocity profile initial condition value and location.

To demonstrate the applicability of the algorithm to two-dimensional discontinuous functions, the algorithm is implemented to regularize the transition of heat *Nusselt* number between laminar and turbulent flow regimes. *Nusselt* number is calculated using two separate equations for each of the flow regimes. Since *Nuseelt* is a function of *Reynolds* and *Prandtl* numbers, the discontinuous function is a two dimensional one.

I illustrated how the application of the regularization algorithm reduces simulation runtime by 23% compared to models relying on reinitialization of variables. The main reason behind the reduction in simulation run length is the localized resolution of the discontinuity. This localized resolution eliminates the unnecessary reinitialization of the entire set of model equations.

In addition to increased simulation run efficiency, I also demonstrated how the regularized model provides more accurate results by better resembling what is happening in an actual process.

In the next chapter, I summarize the outcome of this work and introduce possible areas for future research that can further enhance the developed algorithm.

CHAPTER 7:

SUMMARY AND CONCLUSIONS

In the previous chapters, I demonstrated how discontinuities arise when simplifying mathematical models during their construction. I illustrated how jump discontinuities rise through the use of conditional statements. I also demonstrated how sometimes a discontinuity can easily be removed or minimized through altering the limits of the bounds of the conditional statement. I also classified the previous work on resolving discontinuities in mathematical models into two approaches. Approach I to discontinuity resolution relies solely on the integrating routine to resolve a discontinuity once it is encountered. The conventional resolution techniques relied on either generating an interpolating function at the state-variable level or reinitializing model variables. The drawbacks of each approach have been discussed. I also highlighted the situations at which each of these resolution approaches outperform the other.

An algorithm has been developed to automatically detect discontinuities based on the applicability boundaries of the discontinuous functions and to minimize or eliminate them based on the behaviour of the discontinuous functions at the discontinuity. The discontinuity detection algorithm can be programmed to run within a modelling language or to run independently. In both cases, the detection algorithm should be run prior to the start of the simulation to adjust model conditional statements based on the output of the algorithm. It can also be run independently of the discontinuity

resolution algorithm. If a discontinuity is resolved through the detection algorithm without the need for regularization, the resulting model can be directly run without the need to pass it through a discontinuity resolution algorithm.

When the discontinuity detection algorithm fails to resolve a discontinuity (mainly because of the behaviour of the sub-functions around the conditional statement), the discontinuity should be resolved through the discontinuity resolution algorithm. The discontinuity resolution algorithm basically bridges the missing gap between the discontinuous functions lying on adjacent sides of the conditional statement through the use of an interpolating polynomial. I demonstrated that the use of four control points to construct the interpolating polynomial provide a good compromise between accuracy and computational effort.

To bridge the gap, *hermite* interpolating polynomials are used because they offer two advantages over other readily available interpolating polynomials. They are third order polynomials which assist the solver in calculating Jacobian and Hessian matrices of the simulation model even when integrating through an interpolation region. Cubic splines offer such a feature. However, when using Cubic splines, there is no control over the shape of the curve for a given set of control points. This means that the spline is fixed for a fixed set of control points. On the other hand, *hermite* interpolating polynomials provide two extra parameters to adjust the shape of the curve while preserving its continuity, namely tension and bias parameters. In this work, I only made use of the tension parameter. I also introduced the dip parameter to assist in better control over the shape of the curve.

However, the use of *hermite* interpolating polynomials comes at a cost. With cubic splines, only four control points are required per dimension to construct a cubic

interpolating polynomials. With *hermite* interpolating polynomials, two additional points are required resulting in a total of six control points. The additional control points are required to shape the curvature between the interpolating polynomial and the discontinuous functions at the closing ends of the curve. The relationship between the number of dimensions and the required number of control points is exponential.

In addition to resolving jump discontinuities, I demonstrated how removable discontinuities can be resolved by bridging the gap through the use of interpolating polynomials. Although it is always better to close a removable discontinuity gap by adding a properly bounded function representing the gap domain, bridging the gap using an interpolating polynomial will serve when such functions do not exist. The decision on which path to follow is completely left to the modeller discretion.

Discontinuity resolution approaches are demonstrated to work on problems with many dimensions. They are generic enough to be adopted in solving any ODE/DAE system involving discontinuities in either state variables and/or their respective constitutive equations.

For 1D discontinuous functions, it is recommended to run the discontinuity resolution algorithm before running the simulation. The main reason behind this recommendation is that in 1D functions, the interpolating polynomial between two adjacent discontinuous functions is unique. Thus, regularization solution is independent of simulation path. The same argument holds for Type I discontinuity resolution of 2D+ functions where the interpolation mesh covers the entire overlap domain.

For 2D+ discontinuous functions, I demonstrated two resolutions. The first (Type I) resolution relies on covering the entire overlap domain with a single interpolating polynomial. Type I discontinuity resolution is suitable for relatively small overlap

domains. The bigger the overlap domain, the greater is the number of control points required to properly interpolate. Using a fixed low number of control points results in a coarse interpolation mesh.

Type II discontinuity resolution relies on a fixed-size mesh of control points that is instantaneously constructed once the expression leaves a discontinuous branch of the conditional statement. However, unlike Type I discontinuity resolution generated mesh, the generated mesh in Type II discontinuity resolution does not cover the entire overlap domain between the two discontinuous functions. Instead, it covers a small portion of the domain that allows for regularizing the discontinuity while maintaining acceptable mesh resolution. The compromise when using this type of resolution is the steep departure slope that results in a faster transition between the discontinuous functions. Nevertheless, the conducted experiments demonstrated no decline in integrator efficiency due to the fast transition. The main reason behind maintaining a good performance, despite the resolution of the mesh, is mainly attributed to the fast variable-step search algorithm embedded within [gPROMS, 2012].

To eliminate the exhaustive need to generate unnecessary meshes along the entire course of the simulation, both discontinuity resolution approaches rely on storing a vector of the independent variables required to construct the mesh. The mesh is only created once the conditional statement leaves one sub-function and immediately destroyed after it lands on the adjacent discontinuous one to reserve computer memory space. For Type II discontinuity resolution, a proper method to generate an evenly distributed mesh around the tracking vector is also devised.

Few sections are devoted to regularizing boundary conditions because of the nature of their discontinuities. I demonstrated how boundary conditions can be transformed into

conditional statements involving spatial discontinuities. I also provided a generic resolution approach that relies on the same principles outlined earlier.

Discontinuity resolution completely eliminates re-initialization of state variables because it bridges a discontinuity at its localized origin whether the origin is a state variable or a constitutive equation. Elimination of reinitialization reduces simulation run length by 23%. The reduction in simulation run-length is attributed to the localized treatment of the discontinuity at its origin instead of reinitializing the entire model equations to resolve a local discontinuity. Nevertheless, this reduction is not the major achievement of the work. This work achieves two other goals that were not present in previous works in this field:

1. Regularization more resembles reality than mere re-initialisation of variables because it takes into account the time and/or space factors between state changes. States transit through time and space from their initial to final values. Failing to take this fact into account jeopardises model accuracy. This failure is clearly evident in conventional model variables' re-initialization as I presented in PSA unit example.

2. Sticky discontinuities result from the use of interpolating polynomials that are not derived from model equations to bridge model discontinuities as outlined earlier. Even if the integration routine manages to overcome sticky discontinuities, the generated error between the equations representing the actual model and those used by the approximating interpolating polynomial might lead to misleading simulation results. This work completely eliminates the use of integrator-based polynomials to bridge discontinuities by relying on interpolating polynomials that are derived from model equations with strict adherence to bounds that match both ends of interpolating polynomial to its adjacent discontinuous sub-functions.

In order for this generic approach to discontinuity resolution to function, the following is required:

1. When a conditional expression is to be inserted into a mathematical model, domains of all independent variables belonging to each branch of the conditional expression need to be identified by the modeller and fed to the algorithm. This is an essential requirement for the discontinuity detection algorithm to search for the optimum switch point that minimizes the jump between the two branches and to reconstruct the conditional statement based on the supplied domains. It also helps flagging a warning message and continuing or flagging an error message and stopping the simulation when the algorithm that detects the simulation trajectory is stepping out of the bounds provided for each branch of the conditional statement. Some modelling languages such as [gPROMS, 2012] include an option for the modeller to define bounds of model variables during modelling. Then, the integrator ensures integrating variables within these bounds when simulation is running. Such a capability can be extended to bound an independent variable to a sub-domain of its full domain when a branch of a conditional statement is executed.

2. When regularizing a discontinuity in boundary conditions, it also becomes the modeller's responsibility to identify what and how model-embedded constitutive equations are to be regularized along with the boundary conditions. Automating such a task is also a promising area for a continuing research. Changing modelling practices by formulating equations requiring regularization as differential equations and others as algebraic ones can also act as a starting point. However, such a starting point imposes unnecessary restrictions on the modelling task.

Another challenge would be to automatically set the bounds of the interpolating polynomial that will be used to regularize these variables.

3. When regularizing a discontinuity that involves conflicting boundary conditions, the modeller should decide whether to use one or more regularization intervals depending on the physics of the problem. When opting for more than one regularizing interval, the modeller should also specify the location and the conditions of the common interchange point between the two regularization intervals.

   Automating such a task is a promising area for continuing research that requires a person who is equipped with the knowledge of modelling and computer programming. The work can also easily be split into a group of two persons from two disciplines. A starting point would be to realize that only three boundary conditions exist (Dirichlet, Robin and Dankwert). The challenge is to determine which two-combinations of the three known boundary conditions lead to a boundary conflict when regularized. If the automated procedure can detect conflict, it can advise the use of two regularizing intervals or even automatically insert them into the model. The next challenge would be to identify the common interchange point between the two regularizing intervals. In the examples we demonstrated, it just happened that the interchange point is located at a point that shares a common boundary conditions between the two regularizing functions. Whether the same argument holds for all other modelling problems remains a question that requires an answer.

4. When using *hermite* interpolating polynomials, care must be practised when assigning values to the tension parameter. This point particularly holds when the

interpolating polynomial is to be strictly restricted to the bounds assigned by the control points. Setting the tension to 1 ensures proper bounding to the limits set by the control points. Setting lower values result in smother curvature but with a compromise on proper bounding.

Although, in the context of this work, examples have been drawn from the chemical engineering discipline, the approach is generic enough to be applied to modelling practices in all scientific and engineering disciplines. For example, the algorithm can be used to regulate the transition between equations representing elastic and plastic regions of a string.

Multiscale modelling is an area where this approach to modelling might prove useful. The algorithm brings into the modelling problem some information about the behaviour of phenomena that are occurring at a faster time scale or more detailed hierarchical level than that of the model equations without the need to detail the modelling of the high resolution phenomena. For example, the approach was able to provide information about the behaviour of PSA unit valves without the need to model them.

Does this approach to discontinuity handling apply to all problems involving discontinuities? Not entirely. In the context of this work, I am addressing a resolution to naturally occurring continuous processes that are discretized through modelling practices. Naturally occurring discontinuous processes should not be regularized through these approaches. An example would be modelling the fracture of a broken glass. Phase change can also be relatively regarded as discontinuous phenomena.

A very interesting aspect of this approach is that it brings back the intimate relationship between model equations and their solver. It proves that one way to resolve today's integration problems is by allowing the solver to navigate through model equations and

adjust them when appropriate to generate a better simulation path and ultimately lead to better results. However, the question about which equations a solver need to regularize and which are not remains unanswered when the regularization problem involves special kinds of constitutive equations. I have illustrated that it is very difficult for the approach at its current state to detect and resolve discontinuities in the spatial velocity profile without the modeller pin pointing them to the algorithm. A change in modelling practices to distinguish regularizable equations from others might lead to automatic resolution. However, some problems will still be open to mind exploration. Automatically detecting the location and value of the velocity initial condition is an evident example of such problems.

With this work, I hope that I am able to open a door to overcome difficulties associated with reinitialization and hopefully eliminating reinitialization as a whole.

References

1. Abadpour, A. and M. Panfilov, "Method of Negative Saturations for Modeling Two-phase Compositional Flow with Oversaturated Zones.", *Transport in Porous Media*, vol. 79, pp. 197-214, 2009.
2. Achinstein, P., Concepts of Science. A Philosophical Analysis, Baltimore: John Hopkins Press, 1968.
3. Archibald, R., A. Gelb and J. Yoon, "Determining the Locations and Discontinuities in the Derivatives of Functions", *Applied Numerical Mathematic*, vol. 58, pp. 577-592, 2008.
4. Aris, R., Mathematical Theory of Diffusion and Reaction in Permeable Catalysts, London: Oxford University Press, 1975.
5. Aris, R., Mathematical Modelling : A Chemical Engineer's Perspective, Academic Press, 1999.
6. Augustin, D. C., M. S. Fineberg, B. B. Johnson, R. N. Linebarger, F. J. Sansom and J. C. Straus, "The SCI continuous system simulation language (CSSL)", *Simulation*, vol. 9, pp. 281-303, 1967.
7. Avery, W.F. and M.N.Y. Lee, "ISOSIV Process Goes Commercial", *Oil and Gas Journal*, 1962.
8. Banerjee, R., K.G. Narayankhedkar and P. Sukhatme, "Exergy Analysis of Pressure Swing Adsorption Processes for Air Separation", *Chemical Engineering science*, 1990.
9. Bär, M. and M. Zeitz, "A knowledge-based flowsheet oriented user interface for a dynamic process simulator", *Computers & Chemical Engineering*, vol. 14, pp. 1275-1283, 1990.
10. Bartels, R.H., J.C. Beatty and B.A. Barsky, An Introduction to Splines for Use in Computer Graphics and, Morgan-Kaufman, pp. 422-434, 1987.
11. Berlin, N.H., Method for Providing an Oxygen-Enriched Environment, U.S. Patent No. 3,280,536.

12. Black, M., Models and Metaphors: Studies in Language and Philosophy, New York : Cornell University Press, 1962.

13. Bogusch, R., B. Lohmann and W. Marquardt, "Computer-aided process modelling with ModKit", *Computers & Chemical Engineering*, vol. 25, pp. 963-995, 2001.

14. Borst, R., "Challenges in computational materials science: Multiple scales, multi-physics and evolving discontinuities", *Computational Materials Science*, vol. 43, pp. 1-15, 2008.

15. Bourke, 2011: Bourke, Paul, Interpolation Methods, 2011, http://paulbourke.net/miscellaneous/interpolation/

16. Breeuwsma, 2011: Breeuwsma, 2011, Cubic Interpolation, 2011, http://www.paulinternet.nl/?page=home

17. Cameron, Ian T., E. S. Fraga and I.D.L. Bogle, "Process modelling goals: concepts, structure and development", European Symposium on Computer Aided Process Engineering – 15, 2005.

18. Carver, M.B., "Efficient Integration Over Discontinuities in Ordinary Differential Equation Simulations", *Mathematics and Computers in Simulation*, vol. XX, pp. 190-196, 1978.

19. Cassidy R.T. and E.S. Holmes, "Twenty-Five Years of Progress in `Adiabatic` Adsorption Processes", 1984.

20. Cellier, F.E., "Combined Continuous/Discrete System Simulation by Use of Digital Computers ", 1979.

21. Chahbani, M.H. and D. Tondeur, "Predicting the Final Pressure in the Equalization Step of PSA Cycles ", *Separation and Purification Technology*, vol. 71, pp. 225–232, 2010.

22. Chapra, S.C. and R.P. Canale, Numerical Methods for Engineers, McGraw-Hill, 2002.

23. Cheney, W. and D. Kincaid, Numerical Mathematics and Computing, Gary W. Ostedt, 1999.

24. Cheney, Ward and David Kincaid, The American Heritage Dictionary, Brooks/Cole Publishing, 1985.

25. Chiang A.S.T, "An Analytical Solution to Equilibrium PSA Cycles", *Chemical Engineering Science*, vol. 50, 2, , 1996.

26. Crittenden, B.D., J. Guan and W.N. Ng, W.J. Thomas, “Dynamics of Pressurization and Depressurization during Pressure Swing Adsorption”, *Chemical Engineering Science*, vol. 49, issue 16, 1994, pp. 2657-2669.

27. Chung, T.-H, M. Ajlan, L.L. Lee and K.E. Starling, "Application of kinetic gas theories and multiparameter correlation for prediction of dilute gas viscosity and thermal conductivity ", *Ind. Eng. Chem. Fundam.*, vol. 19, p. 186, 1984.

28. Chung, T.-H, M. Ajlan, L.L. Lee and K.E. Starling, "Generalized Multiparameter Corresponding State Correlation for Polymeric, Polar Fluid Transport Properties ", *Ind. Eng. Chem. Res. Design Development,*Submitted 1986.

29. de Montgareuil, P. Guerin and D. Domine, French Patent 1,223,261, 1957.

30. de Waal, A. and T. Richey, "Combining morphological analysis and Bayesian networks for strategic decision support", *Orion*, vol. 23, ed. 2, pp. 105-121, 2007.

31. Delgado, J.A. and A.E. Rodrigues, "Analysis of the Boundary Conditions for the Simulation of Pressure Equalization step in PSA Cycles ", *Chemical Engineering Science*, vol. 63, , pp. 4452 - 4463, 2008.

32. Dixon, A.G. and D.I. Cresswell, "Effective Heat Transfer Parameters for Transient Packed-Bed Models ", *AIChE Journal*, vol. 32, 5, pp. 809-819, 1986.

33. Dömges, R., K. Pohl, M. Jarke, B. Lohmann and W. Marquardt, "PRO-ART/CE – An Environment for Managing the Evolution of Chemical Process Simulation Models", Proceedings of the 10th European Simulation Conference. Geril, P. (Ed.) , pp. 1012-1017, 1996.

34. Ellison, D, "Efficient Automatic Integration of Ordinary Differential Equations with Discontinuities", *Mathematicsand Computers in Simulation*, vol. XXIII,

pp. 12-20, 1981.

35. Evans, R.B., G.M. Watson, E.A. Mason, "Gaseous Diffusion in Porous Media at Uniform Pressure", *Journal of Chemical Physics*, vol. 35, 6, pp. 2076-2083, 1961.

36. Filip, Daniel, Robert Magedson and Robert Markot, "Surface Algorithms Using Bounds on Derivatives", *Computer Aided Geometric Design*, vol. 3, , pp. 296-311, 1986.

37. Frigg, R. and S. Hartman, Models in Science, Stanford Encyclopedia of Philosophy, 2012.

38. Fritsch, Fred and Ralph Carlson, "Monotone Piecewise Cubic Interpolation", *SIAM Journal on Numerical Analysis*, vol. 17, No. 2, pp. 238-246, 1980.

39. Fuller, E.N., K. Ensley and J.C. Giddings, "Diffusion of Halogenated Hydrocarbons in Helium", *J. Phys. Chem.*, 73, , pp. 3679-3685, 1969.

40. Fuller, E.N., P.D. Schettler, and J.C. Giddings, "A New Method for Prediction of Binary Gas Phase Diffusion Coefficients", *Ind. Eng. Chem.*, 58, , pp. 19-27, 1966.

41. Gear, C.W., "The Automatic Integration of Ordinary Differential Equations", *Cmm. ACM* , vol. 14, , pp.176-190, 1970.

42. Gensym, 1995: Gensym Corp., G2 Version 4.0 Beta Release Notes, 1995

43. Gnielinsky, V., "New Equations for Heat and Mass Transfer in Turbulent Pipe Channel Flow", *Int. Chem. Eng.*, vol. 16, p. 359, 1976.

44. gPROMS, gPROMS Modelling Language, Copyright © 1997-2012,, 2012

45. Grossmann, I. E. and J. P. Ruiz, Generalized Disjunctive Programming: A Framework for Formulation and Alternative Algorithms for MINLP Optimization, Springer, pp. 93-115, , 2011.

46. GSL, 2011: GSL, GNU Scientific Library, 2011,

47. Hangos, Katalin and Ian Cameron, Process Modelling and Model Analysis, Academic Press,2001.

48. Helenbrook, B.T., L. Martnelli and C.K. Law, "A Numerical Method for solving Incompressible Flow Problems witha Surface of Discontinuity", *Journal of Computational Physics* , vol. 148, pp. 366-396, 1999.

49. Javey, S., "A Language Construct for the Specification of Discontinuities", *Journal of Systems and Software*, vol. 8, issue 5, pp. 409-417, 1988.

50. Brackbill, J.U., D.B. Kothe and C. Zemach, "A Continuum Method for Modelling Surface Tension", *Journal of Computational Physics*, vol. 100, pp. 335-354, 1992.

51. Kallrath, J., Modelling Languages in Mathematical Optimization (Applied Optimization), Kluwer Academic Publishers, 2004.

52. Kauzmann, W., Kinetic Theory of Gases, Benjamin, , New York, 1966.

53. Keller II, G.E., "Gas Adsorption Processes: State of the Art in Industrial Gas Separations", 1983.

54. King R. and R. Hull, "Semantic database modelling: survey, applications, and research issues", *ACM Comput. Surv.*, vol. 19, pp. 201-260, 1987.

55. Kochanek, D. H. U. and R. H. Bartels, "Interpolating Splines with Local Tension, Continuity and Bias Control", *Computer Graphics*, vol. 18, ed. 3, pp. 33-41, 1984.

56. Kreith, Frank, CRC Handbook of Thermal Engineering, CRC Press, 2000.

57. Lioyd, E.A., The Structure of Confirmation of Evolutionary Theory, Princeton University Press, 1994.

58. Lioyd, E.A., "A Semantic Approach to the Structure of Population Genetics", *Philosophy of Science*, vol. 51, pp. 242-264, 1984.

59. Little, Donald M., Catalytic Reforming, PennWellBooks, 1985.

60. Mao, G. and L.R. Petzold, "Efficient Integration over Discontinuities for Differential-Algebraic Systems", *Computers and Mathematics with Applications*, vol. 43, pp. 65-79, 2002.

61. Marquardt, W., "Trends in Computer Aided Process Modelling", *Computers &*

Chemical Engineering, vol. 20, , pp. 591-609., 1996.

62. Marsh, W.D., F.S. Pramuk, R.C. Hoke and C.W. Skarstrom, “Pressure Equalization Depressuring in Heatless Adsorption”, Patent No. 3,142,547, 1964

63. Mason, E.A. and S.C. Saxena, "Approximate Formula for the Thermal Conductivity of Gas Mixtures", *Phys. Fluids*, vol. 1,p. 361, 1958.

64. mcCabe, W.L., J.C. Smith and P. Harriott, Unit Operations of Chemical Engineering, 2005.

65. Minkkinen, A., L. Mank and S. Jullian, “Process for the Isomerization of C5 and C6 Normal Paraffins with Recycling”, Patent No. 5,233,120.

66. Minkkinen, A., L. Mank, and S. Jullian, Process for the Isomerization of of C5/C6 Normal Paraffins with, U.S. Patent No. 5,233,120,1993.

67. Moler, Cleve B., Numerical Computing with Matlab, SIAM, 2004.

68. Morgan, M., "Models, Stories and Economic World", *Journal of Economic Methodology,* vol. 8, ed. 3, pp. 361-384, 2001.

69. Nilchan, S. and C.C. Pantelides, "On the Optimisation of Periodic Adsorption Processes", *Adsorption*, vol. 4, 2, 1998.

70. Nitta, T., M. Kuro-Oka and T. Katayama, "An Adsorption Isotherm of Multi-site Occupancy Model for Homogeneous Surface", *J. Chem.Eng. Jpn.*, 17, pp. 39-45, 1984.

71. Park, T. and P.I. Barton, "State Event Location in Differential-Algebraic Models", *ACM Transactions on Modelling & Computer Simulation*, vol. 6, issue 2, pp. 137-165, 1996.

72. Pollard, W.G. and R.D. Present, "On Gaseous Self-Diffusion in Long Capillary Tubes", *Phys. Rev.*, vol. 73, pp. 762, 1948.

73. R.C. Reid, J.M. Prausnitz and B.E. Poling, The Properties of Gases and Liquids, McGraw Hill, 1987.

74. Rathke, C., "Object oriented programming and frame-based knowledge representation", 5th International Conference, pp. 95-98, November 1993,

Boston.

75. Ritchey, T., "Outline for a Morphology of Modelling Methods", *Acta Morphologica Generalis*, vol. 1, , pp. 1-20, 2012.

76. Ritchey, T., Wicked Problems/Social Messes : Decision Support Modelling with Morphological Analysis, Springer, 2011.

77. Ruthven, D.M., Principles of Adsorption and Adsorption Processes, John Wiley and Sons, 1984.

78. Ruthven, D.M., S. Farooq and K.S. Knaebel, Pressure Swing Adsorption, John Wiley and Sons, 1994.

79. Satterfield, C.N., Heterogenious Catalysis in Practice, McGraw Hill, 1980.

80. Schichl, H., Models and the History of Modeling, in: Modeling Languages in Mathematical Optimization (J. Kallrath, ed.), Boston: Klower Academic Publishers, pp. 25-36, 2004.

81. Scott, D.S. and F.A.L. Dullien, "The Flux-Ratio for Binary Counter Diffusion of Ideal Gases", *Chemical Engineering Science*, vol. 17, pp. 771-775, 1962.

82. Shah, R.K. and A.L. London, Laminar Flow: Forced Convection in Ducts, Academic Press, , New York, 1978.

83. Shirley, A. I. and N. O. Lemcoff, "Effect of Pressurization Rate on the Performance of Nitrogen Pressure Swing Adsorption Processes", *Fundamentals of Adsorption*, vol. 36, pp. 837-844, 1996.

84. Sieder, E.N. and C.E. Tate, "Heat Transfer and Pressure Drop of Liquids in Tubes", *Ind. Eng. Chem.*, vol. 28, p. 1429, 1936.

85. Silva, J.A. and A.E. Rodrigues, "Separation of n/iso- Paraffins Mixtures by Pressure Swing Adsorption", *Separation and Purification Technology*, vol. 13, pp. 195-208, 1998.

86. Silva, J.A.C., F.A.D. Silva and A.E. Rodrigues, “Separation of n/iso paraffins by PSA”, *Separation and Purification technology*, vol. 20, pp. 97-110, 2000.

87. Skarström, C.W., “Method and Apparatus for Fractionating Gaseous Mixtures

by Adsorption", U.S. Patent No. 2,944,627, 1960.

88. Spivey, J.J. and P.A. Bryant, "Hydroisomerization of n-C_5 and n-C_6 Mixtures on Zeolite Catalysts", *Industrial & Engineering Chemistry Process Design and Development*, 21, pp. 750-780, 1982.

89. Stephik, M. and D.G. Bobrow, "Object-oriented programming: themes and variations", *AI Magazine*, vol. 6, , pp. 40-62, 1986.

90. Swokowski, E.W., Calculus, Brooks/Cole, 1991.

91. Tamura, T., U.S. patent 3,797,201, 1974.

92. Taylor R. and R. Krishna, Multicomponent Mass Transfer, John Wiley & Sons, 1993.

93. Wagner, J.L., Selective Adsorption Process, U.S. Patent No. 3,430,418, 1969.

94. Wakao, N. and T. Funazkri, "Effect of Fluid Dispersion Coefficients on Particle-To-Fluid Mass Transfer Coefficients in Packed Beds", *Chemical Engineering Science*, vol. 33, pp. 1375-1384, 1978.

95. Wakao, N., S. Kaguei and T. Funazkri, "Effect of Fluid Dispersion Coefficients on Particle-to-Fluid Heat Transfer", *Chemical Engineering Science*, vol. 34, pp. 325-336, 1979.

96. Warmuzinski, K., "Effect of Pressure Equalization on Power Requirements in PSA Systems", *Chemical Engineering Science*, vol. 57, pp. 1475-1478, 2002.

97. Warmuzinski,K. and M. Tanczyk, "Calculation of the Equalization Pressurein PSA Systems", *Chemical Engineering Science*, vol. 58, pp. 3285-3289, 2003.

98. Wassiljewa, A., "Warmeleitung in Gasgemischen", *Physik Z.*, vol. 5, p. 737, 1904.

99. Wen, C.Y. and L.T. Fan, Models for Flow Systems and Chemical Reactors, Dekker, 1975.

100. Wilke, C.R., "Diffusional Properties of Multicomponent Gases", *Chem. Eng. Prog.*, vol. 46, pp. 95-104, 1950.

101. Yang, Ralph T., Gas Separation by Adsorption Processes, Imperial College

Press, 1987.

APPENDIX A: A Novel Formula for Calculating Pressurization and De-pressurization Velocity Profiles

The spatial velocity profile during pressurization or depressurization of any vessel is calculated using equation 4.6. Assuming no adsorption at the boundaries, equation 4.6 reduces to:

$$C_t \frac{du}{dz} + \frac{dC_t}{dt} = 0 \tag{A.1}$$

Equation A.1 can be normalized using the following transformations :

$$v = u/U_{max},\ C_T = C_t/C_{t,max},\ x = z/L \text{ and}$$
$$\tau = t/t_{ref} \text{ where } t_{ref} = L/U_{max} \tag{A.2}$$

The normalized equation takes the form:

$$C_T \frac{dv}{dx} + \frac{dC_T}{d\tau} = 0 \tag{A.3}$$

Realizing that:

$$C_t = \frac{P}{RT} \rightarrow C_T = \frac{P/RT}{P_{ref}/RT} = \frac{P}{P_{ref}} \rightarrow \frac{dC_T}{d\tau} = \frac{d\overline{P}}{d\tau} \tag{A.4}$$

Where $\overline{P} = P/P_{ref}$. Equation A.1 can then be written in terms of $\overline{P}$ as :

$$\overline{P} \frac{dv}{dx} + d\frac{\overline{P}}{d\tau} = 0 \tag{A.5}$$

Since $\overline{P}$ is independent of *x*, *v* is independent of *τ and v*(1) =0, equation A.5 can be integrated to yield:

$$v(x, \overline{P}) = \frac{1}{\overline{P}} \frac{d\overline{P}}{d\tau}(1 - x) \tag{A.6}$$

At *x*=0 (inlet velocity), equation A.6 reduces to :

$$v(P) = \frac{1}{\overline{P}} \frac{d\overline{P}}{d\tau} \tag{A.7}$$

The pressure $\overline{P}$ can be calculated using a normalized version of either equation 4.16 or

4.18. Equations 4.16 and 4.18 can be written in their dimensionless form with their respective time derivatives as:

$$\overline{P} = \frac{P_{high}}{P_{ref}} - \left(\frac{P_{low}}{P_{ref}} - \frac{P_{high}}{P_{ref}} \right) \left[\frac{\tau}{\tau_p} - 1 \right]^2 \quad \text{(A.8a)}$$

$$\frac{d\overline{P}}{d\tau} = -\frac{2}{\tau_p} \left(\frac{P_{low}}{P_{ref}} - \frac{P_{high}}{P_{ref}} \right) \left[\frac{\tau}{\tau_p} - 1 \right] \quad \text{(A.8b)}$$

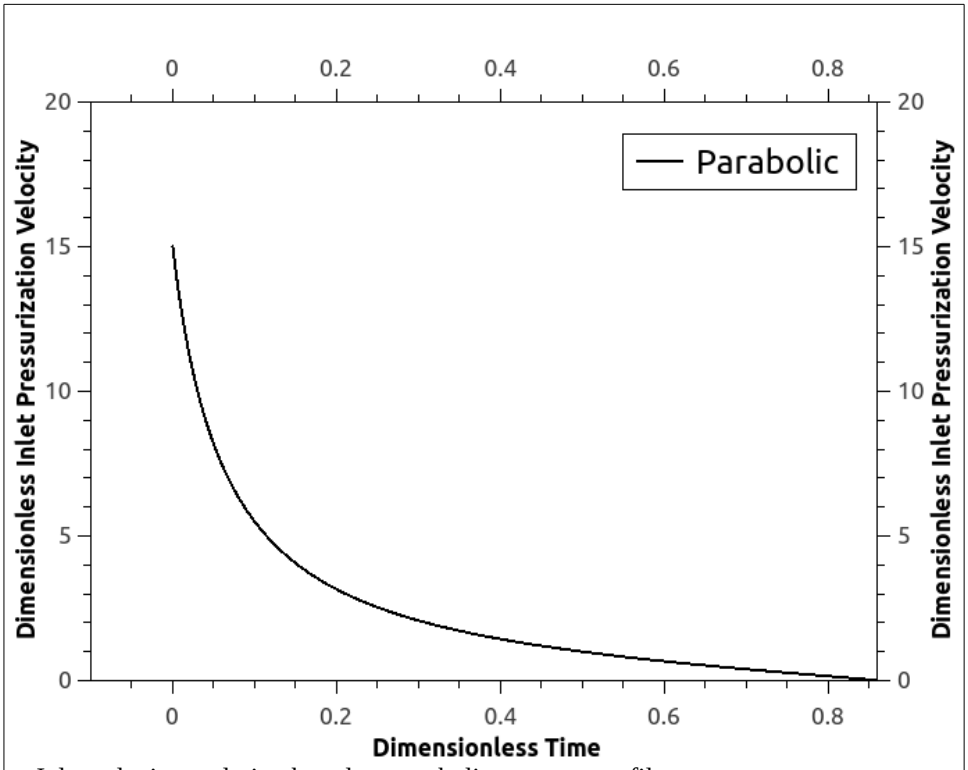
$$\overline{P} = \frac{P_{low}}{P_{ref}} + \left(\frac{P_{high}}{P_{ref}} - \frac{P_{low}}{P_{ref}} \right) \left[1 - e^{(-M_p \tau)} \right] \quad \text{(A.9a)}$$

$$\frac{d\overline{P}}{d\tau} = -M_p \left(\frac{P_{high}}{P_{ref}} - \frac{P_{low}}{P_{ref}} \right) \left[1 - e^{(-M_p \tau)} \right] \quad \text{(A.9b)}$$

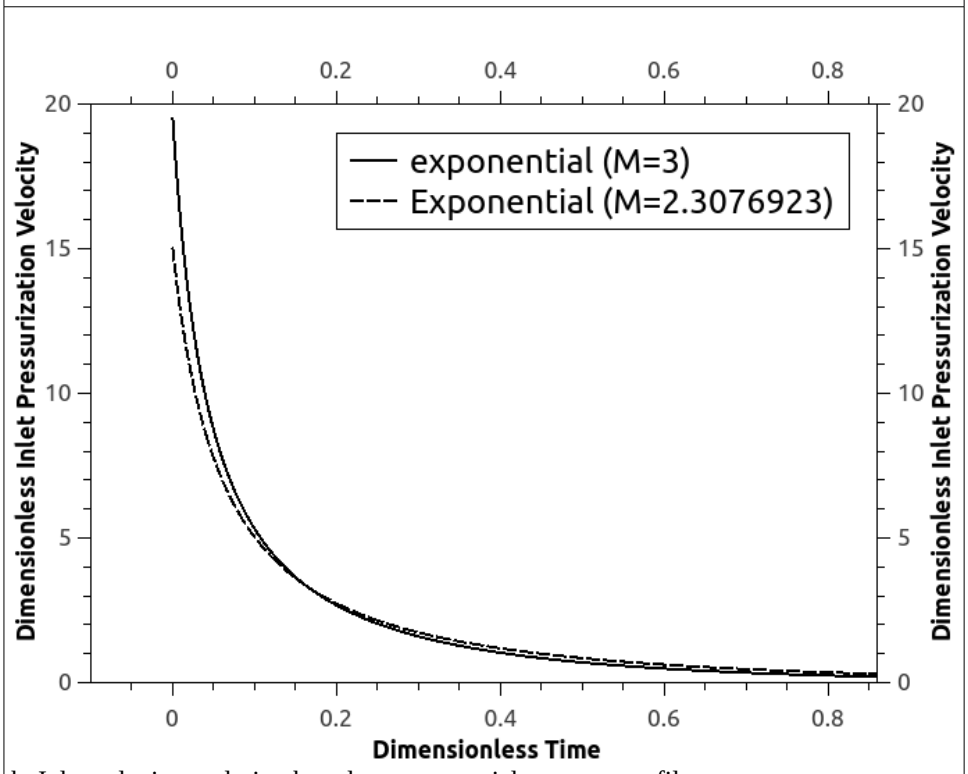
Substituting either equation A.8 or A.9 into A.7 yields an expression for inlet velocity as a function of time. Figure A.1a illustrates the response of inlet velocity to time changes using equations A.8 (parabolic profile). Figure A.1b illustrates the response of inlet velocity to time changes using equations A.9b (exponential profile). The value of M_P = 2.3076923 corresponds to an initial pressurization velocity (at τ=0) that is equivalent to that provided by the parabolic profile.

These two equations are widely adopted in literature. However, they posses a fundamental drawback. They instantaneously change bed initial velocity from a value of zero to a value that corresponds to multiples of feed velocity at adsorption step.

For the parabolic profile, this velocity instantaneously changes from a value of 0 to 15 times that of the feed velocity during adsorption step. For the exponential velocity profiles, the initial inlet velocity depends on pressurization rate M_P. However, regardless of the value of M_P, pressurization is almost always instantaneous.



a. Inlet velocity evolution based on parabolic pressure profile.



b. Inlet velocity evolution based on exponential pressure profile.

Figure A.1: Dimensionless inlet velocity during pressurization step calculated using *a:* parabolic pressure profile, *b:* exponential pressure profile. The value of M=2.3076923 corresponds to an initial velocity value (at t=0) that is equivalent to the one provided by the parabolic profile .

This sudden change in velocity profile does not correspond to the reality of a continuous process. In order to derive a better representing equation, we should realize first that pressure changes in this step are due to introduction of high pressure feed through opening of the feed valve. The opening of the feed valve is a continuous function that is mistakenly modelled as an instantaneous one. The pressure rises downstream of the feed valve before reaching the PSA column. The pressure downstream of the valve is a function of valve opening, In addition, the pressure always rises downstream of the feed valve before reaching the vessel. Such a change is an incremental and not an instantaneous one. It can be modelled by substituting the constant value of P_{high} in equations A.8 or A.9 by an incremental function in pressure that is bounded by pressure limits [P_{low}, P_{high}]. Referring to Figure 4.5, it can be realized that the exponential pressure profile is always leading the parabolic one yet the exponential profile is still an incremental one and bounded by P_{low} *and* P_{high}. Thus, replacing the constant P_{high} value in equation A.8 will result in an incremental pressure profile and simultaneously result in an incremental velocity profile. Thus, equation A.8 becomes:

$$\overline{P}(\tau)=\frac{1}{P_{ref}}\left[P_{high}(\tau)-\left(P_{low}-P_{high}(\tau)\right)\left[\frac{\tau}{\tau_p}-1\right]^2\right] \quad \text{(A.10a)}$$

$$\frac{d\overline{P}(\tau)}{d\tau}=\frac{1}{P_{ref}}\left[T_1+T_2+T_3\right] \quad \text{(A.10b)}$$

Where:

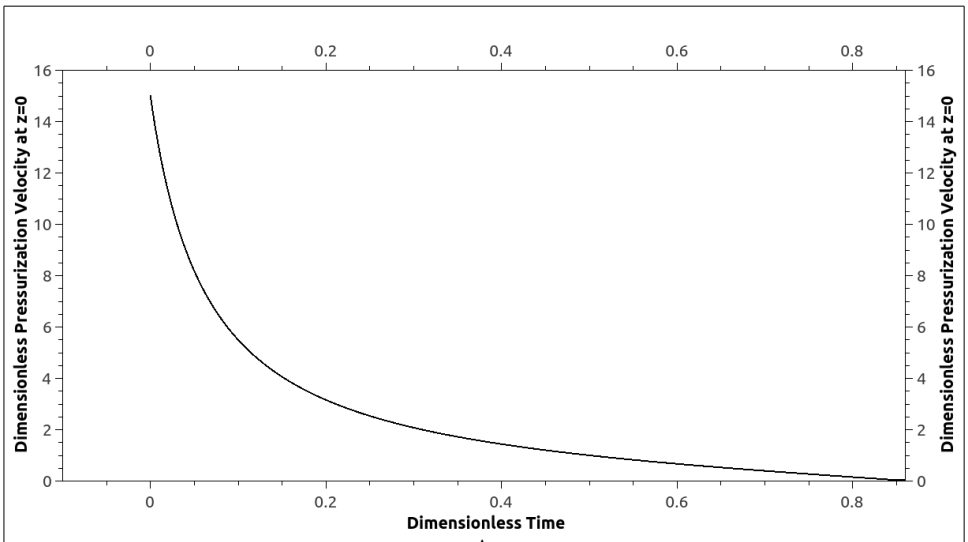
$$P_{high}(\tau)=P_{low}+\left(P_{Feed}-P_{low}\right)\left[1-e^{(-M_p\tau)}\right] \quad \text{(A.10c)}$$

$$T_1=\frac{2}{\tau_p}\left[P_{low}+\left(P_{Feed}-P_{low}\right)\left(1-e^{-M_p\tau}\right)\right]\left(1-\frac{\tau}{\tau_p}\right) \quad \text{(A.10d)}$$

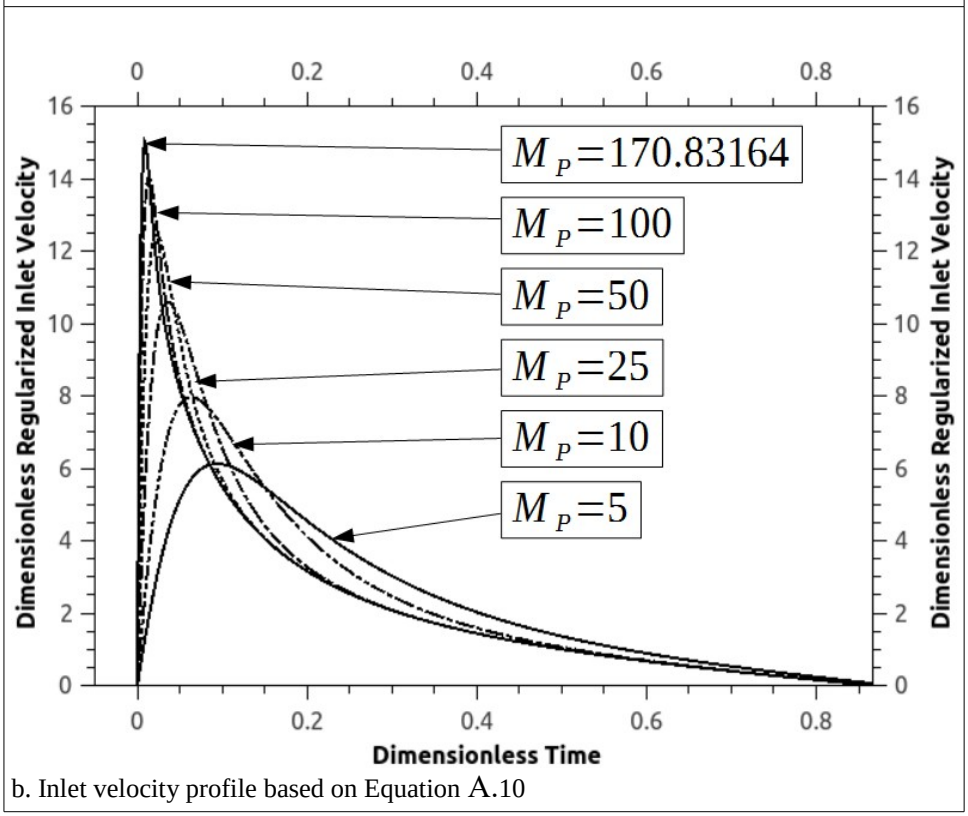
$$T_2=\left[1-\left(\frac{\tau}{\tau_p}-1\right)^2\right]\left[\left(P_{Feed}-P_{low}\right)Me^{(-M\tau)}\right] \quad \text{(A.10e)}$$

$$T_3 = \frac{2P_{low}}{\tau_p}\left[\frac{\tau}{\tau_p} - 1\right] \tag{A.10f}$$

Several trends of equation A.10 with various M_P values are plotted in Figure A.2. The value of M_P=170.83164 corresponds to a dimensionless inlet velocity peaking at a value of 15. This value is exactly the same as that reported by equation A.8. However, the value provided by equation A.10 does not peak at the start of pressurization step. Thus, the value calculated by equation A.10 provides a better regularization.



a. Inlet velocity profile Based on Equation A.8



b. Inlet velocity profile based on Equation A.10

Figure A.2: Dimensionless pressurization step inlet velocity based on a: a fixed value of upstream feed pressure that is equivalent to the high pressure value (parabolic profile based on equation A.8), b: a variable upstream pressure that is based on equation A.10.

APPENDIX B: MODELS' VALIDATIONS WITH THE MINKINNEN PROCESS

I centred the validation of the modelling work around the [Minkkinen et al, 1993] process which hydroisomerizes normal pentane and normal hexane compounds to their branched isomers in a reactor before separating products from reactants using a Pressure Swing Adsorption unit.

B.1 A Brief Description of the Process

The [Minkkinen et al, 1993] process consists principally of:

1. a distillation column (deisopentanizer) to separate isopentane from the feed (not modelled),
2. an isomerization reactor to convert normal alkanes to their branched isomers,
3. a distillation column (product separator) to separate Hydrogen from reactor effluent (not modelled),
4. and two Pressure Swing Adsorption columns to separate normal alkanes from their branched isomers.

In their patent, [Minkkinen et al, 1993] presented an original scheme for the process and also introduced a modified variant. I only focused on the original scheme of the process which is illustrated in Figure B.1. The original Minkkinen process feed composition is outlined in the left column of Table B.1. To simplify calculation and modelling tasks, I approximated the feed composition to that presented at the right column of Table B.1 by averaging concentrations of various *i*-C_6 isomers into one isomer, namely 2-2 Dimethyl Butane. I deliberately averaged the concentrations of *i*-C_6 isomers instead of

agglomerating them. Agglomerating them would lead to an isomer feed composition that is higher than normal hexane composition and hence shifting equilibrium towards producing normal hexane instead of iso-hexane. Process stream flows, composition, temperatures and pressures are outlined in Table B.2. Any missing information is obtained through drawing an overall and individual component material balances around process units. Respective stream numbers are outlined in Figure B.1.

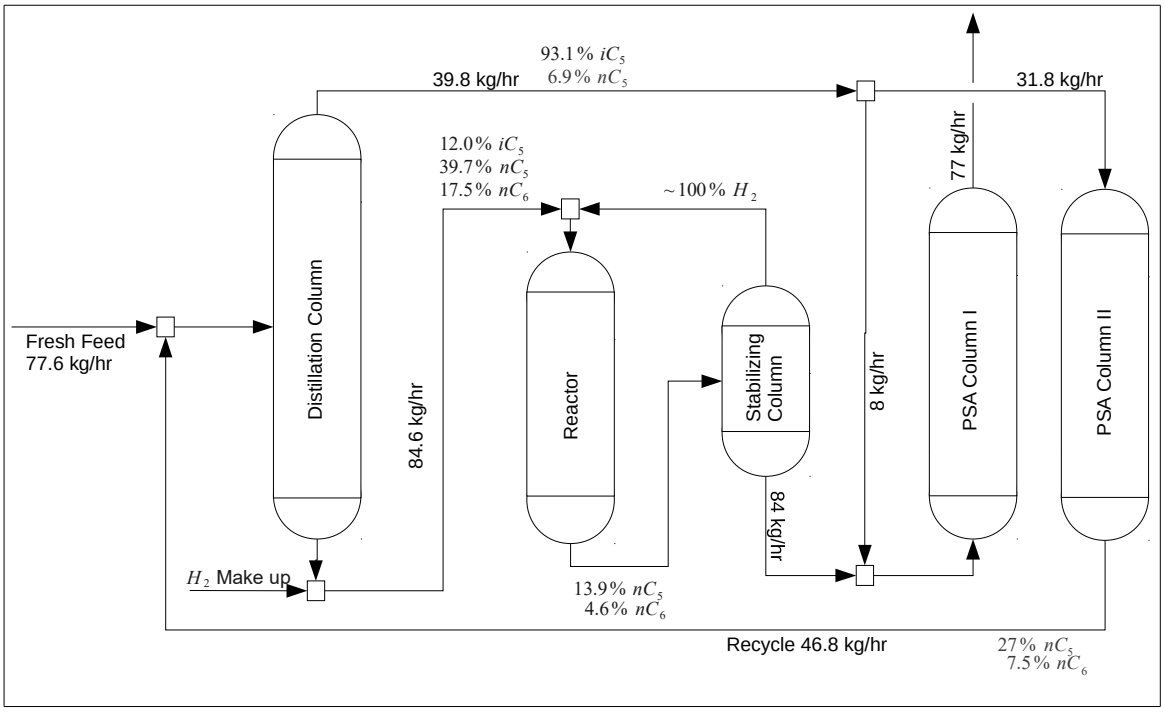


Figure B.1: Simplified process diagram for the [Minkkinen et al, 1993] Process. Individual stream specifications are outlined in Table B.2.

In Minkkinen process, the feed consisting of normal and iso- parafiins is feed to a distillation unit. Lighter components are stripped at the top of the column and the rest of the material is collected as a bottom product and fed to an isomerization reactor. The stripped top product is used as a purge stream during the desorption step of the PSA unit.

The bottom draw of the distillation column is mixed with a recycled hydrogen stream before entering the isomerization reactor. Hydrogen acts as a reaction promoter. More than 60% of *n-C₅* and 73.0% of *n-C₆* are converted to their respective isomers. Reactor effluent is fed to a product separator where essentially all hydrogen is stripped off the

product separator feed and recycled back to the reactor. A Hydrogen feed line is present at the bottom of the distillation column to compensate for any loss in Hydrogen recycle loop.

Table B.1: Original [Minkkinen et al, 1993] and approximated feeds to Minkkinen Process.

Compound	*Original Feed (mol %)*	*Approximated Feed (mol %)*
Isobutane	0.4	25.38
Normal Butane	2.4	2.4
Isopentane	21.0	23.2
Normal Pentane	29.0	29.0
Cyclopentane	2.2	0.0
2-2 Dimethyl Butane	0.5	6.03
2-3 Dimethyl Butane	0.9	0.0
2 Methyl Pentane	12.7	0.0
3 Methyl Pentane	10.0	0.0
Normal Hexane	14.0	14.0
Methyle Cyclopentane	5.0	0.0
Cyclohexane	0.5	0.0
Benzene	1.3	0.0
C7+	0.1	0.0

Since conversion is incomplete, a need arises to separate normal paraffins from the isomers. This separation is performed in a two-bed PSA unit. The bottom of the product separator is mixed with a bleed stream from the top of the distillation column before it is fed to the PSA column undergoing pressurization and adsorption steps (Column I in Figure B.1).

Each PSA column is filled with an adsorbent that is selective to normal paraffins. Iso-paraffins pass, unadsorbed, through the column and are collected as a raffinate product. Simultaneously, PSA Column II is undergoing depressurization and desorption steps to remove normal paraffins that were accumulated as a result of a previous adsorption step. The effluent from PSA Column II (extract) is recycled and mixed with the main feed

before entering the distillation column. Once PSA Column I adsorbent is saturated with normal paraffins, the feed is switched to PSA Column II and PSA Column I is purged with distillation column top product. The cycle between the two PSA columns is repeated indefinitely until the unit is shut down.

B.2 The Reactor Model

Isomerization reactors (commercially known as reformers) are mainly used to convert normal alkanes to their isomers using a catalytic reactor in the presence of Hydrogen. Isomerization is one of the reactions required to raise the octane number of the feed stream by converting normal alkanes to their branched isomers. Other side reactions occurring inside reformers include the desirable Dehydrogenation, Dehydrocyclization, Hydrocracking, and the undesirable Demethylation reaction [Little, 1985]. High octane numbers reduce knocking characteristics and increase the efficiency of combustion engines that are used to power most of today's auto-mobile industry.

[Minkkinen et al, 1993] recommended the use of a high activity catalyst that is based on Chlorinated Alumina and Platinum in order to operate the reactor at temperatures between 130-220ºC and pressures ranging from 20-35 bars in addition to the low Hydrogen to hydrocarbon [H:HC] ratios of 0.1 to 1.0. In their laboratory test unit, they used 52 litres of a η alumina-based isomerization catalyst that contains 7 wt% chlorine and 0.23 wt% Platinum. They also mention the suitability to use Zeolite based catalysts such as Mordenites although they dismissed their use due to the higher activation energies required by such catalysts that eventually require higher reactor inlet temperatures to achieve the required conversion.

Table B.2: Properties of individual streams described by [Minkkinen et al, 1993]. Shaded areas indicate information that is obtained through material balances. Bold-faced figures with white backgrounds refer to information supplied by [Minkkinen et al, 1993] in their patent.

Stream Number	1	2	3	4	5	6	7	8	9	10	11	12	13
Stream Name	Fresh Feed	Recycled Feed	Deisopentanizer feed	Deisopentanizer Top Draw	Deisopentanizer Bottom Draw	Product Separator Recycled Hydrogen	Reactor Feed	Reactor Effluent	Product Separator Bottom Draw	Deisopentanizer top bleed to PSA feed	PSA Feed	PSA Raffinate	PSA Purge
Pressure (bar)	**2.00**	**2.00**	**2.00**	**2.00**	**2.00**	**30.00**	**30.00**	**30.00**	**15.00**	**15.00**	**15.00**	**15.00**	**2.00**
Temperature (°C)							**140.00**	**160.00**	**160.00**	**16.00**	**300.00**	**300.00**	**300.00**
MW (AVG)													
Flow (kg/hr)	**77.60**	**46.80**	124.40	**39.80**	**84.60**	**2.32**	**69.26**	**84.60**	**84.60**	**8.00**	**84.60**	**77.00**	**31.80**
Flow (kgmol/hr)	1.09	0.64	1.73	0.85	1.15	1.15	2.03	1.15	1.15	0.12	1.24	1.00	0.47
Composition (mol%)													
Hydrogen (H2)	0.00	0.00	0.00	0.00	0.00	100.00	56.58	50.65	0.00	0.00	0.00	0.0000	0.00
Isobutane (IC4)	**25.38**	0.00	16.01	32.60	0.00	0.00	0.00	1.16	1.80	32.60	5.24	2.1587	32.60
Normal Butane (nC4)	**2.40**	0.00	1.51	0.65	1.80	0.00	1.02	0.00	0.00	0.65	0.06	0.0000	0.65
Isopentane (iC5)	**23.20**	65.50	38.80	62.79	**12.00**	0.00	6.7898	24.31	37.80	62.79	50.56	51.0884	62.79
Normal pentane (nC5)	**29.00**	**27.00**	28.26	3.95	**39.70**	0.00	22.4630	8.94	**13.90**	3.95	16.76	0.9509	3.95
Isohexane (i-C6)	**6.03**	0.00	3.80	0.00	5.74	0.00	3.2452	12.33	19.18	0.00	22.60	45.8021	0.00
Normal hexane (nC6)	**14.00**	**7.50**	11.60	0.00	**17.50**	0.00	9.9018	2.61	**4.06**	0.00	4.78	0.0000	0.00
TOTAL	100.00	100.00	100.00	100.00	76.73	100.00	100.00	100.00	76.73	100.00	100.00	100.00	100.00
Composition (wt%)													
Hydrogen (H2)	0.00	0.00	0.00	0.00	0.00	100.00	3.34	0.00	0.00	0.00	0.00	0.00	0.00
Isobutane (IC4)	20.79	0.00	12.97	28.08	0.00	0.00	0.00	0.00	0.00	28.08	4.08	1.62	28.08
Normal Butane (nC4)	1.97	0.00	1.23	0.56	1.56	0.00	1.74	0.00	0.00	0.56	0.05	0.00	0.56
Isopentane (iC5)	23.60	64.56	39.01	67.13	12.95	0.00	14.37	0.00	0.00	67.13	48.83	47.65	67.13
Normal pentane (nC5)	29.49	26.61	28.41	4.23	12.95	0.00	47.52	0.00	0.00	4.23	16.19	0.89	4.23
Isohexane	7.15	0.00	4.46	0.00	49.98	0.00	8.01	0.00	0.00	0.00	25.46	49.84	0.00
Normal hexane (nC6)	17.01	8.83	13.93	0.00	22.56	0.00	25.02	0.00	0.00	0.00	5.39	0.00	0.00
TOTAL	100.00	100.00	100.00	100.00	100.00	100.00	100.00	0.00	0.00	100.00	100.00	100.00	100.00

The catalysts used in such processes are usually Platinum based, hence the name noble. [Spivey and Bryant, 1982] classify catalysts used into Mordenite and Faujasite with the former exhibiting the highest activity.

In this work, I modelled the Modernite catalyst that was presented by [Spivey and Bryant, 1982] in their paper as they reported the required reaction rate constants. In their study on Hydroisomerization of *n-C₅* and *n-C₆* mixtures on Zeolite catalysts, they used a 0.5 wt% Platinum H-mordenite (Pt-H-M) with [SiO_2: Al_2O_3] ratio of [14:1] and a 0.5 wt% Palladium H-faujasite type Y (Pd-H-Y) with a [SiO_2:Al_2O_3] ratio of [6.4:1]. Since the catalyst used by Minkkinen is a Platinum based one, I picked the corresponding rate constants from the paper by [Spivey and Bryant, 1982].

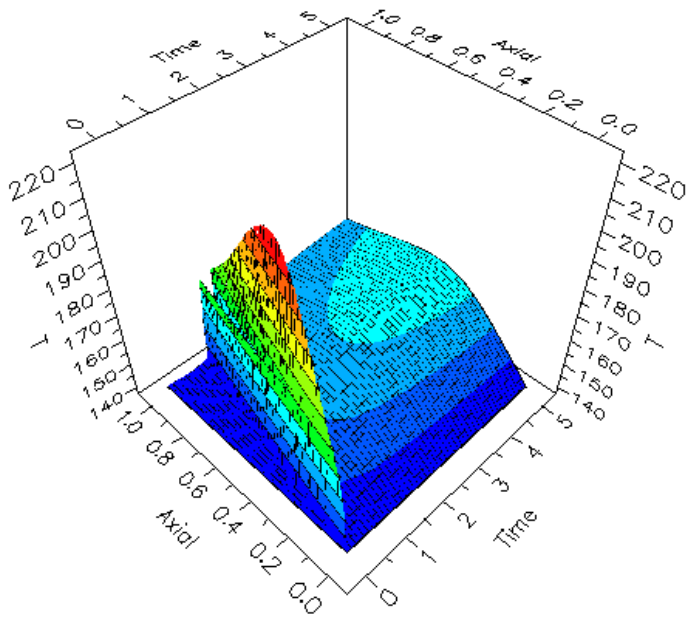


Figure B.2: 3D Temperature profile versus normalized axial distance x and time τ. $x = z/L_R$ and $\tau = t/t_{ref}$ where $t_{ref} = L_R/U_{ref}$. Initial higher temperature profiles are due to the release of heat of adsorption.

B.2.1 Reactor Sizing Calculation

Other than the total volume of the catalyst used (52 litres), [Minkkinen et al, 1993] did not provide any information on reactor geometry. So, I had to perform a simple isothermal reactor design exercise to estimate reactor length and diameter. The n-C_5 Hydroisomerization reaction is a reversible reactions that can be expressed as:

$$C_{nC_5} \underset{k_{iC_5}}{\overset{k_{nC_5}}{\rightleftarrows}} C_{nC_5} \tag{B.1}$$

The rate of the reaction is expressed as:

$$-r_{nC_5} = k_{nC_5} C_{nC_5} - k_{iC_5} C_{iC_5} \tag{B.2}$$

Where:

k_{nC_5} : forward reaction rate constant

k_{iC_5} : reverse reaction rate constant

C_{nC_5} : normal-pentane concentration

C_{iC_5} : iso-pentane concentration

Equation B.2 can also be written based on one of the reaction rate constants and the reaction equilibrium constant:

$$-r_{nC_5} = k_{nC_5}\left(C_{nC_5} - \frac{C_{iC_5}}{K_C}\right) \tag{B.3}$$

Where:

$$K_C = \frac{k_{nC_5}}{k_{iC_5}}$$

[Spivey and Bryant, 1982] discuss temperature and pressure dependency of the forward and reverse reaction rate constants. However, for a simplified design calculation we will assume isothermal operation. Since the reactor is operated at a constant pressure and a relatively fixed feed composition, the assumption of isobaric operation seems a valid one.

Under these conditions, and assuming a plug flow reactor, the reactor design equation can be written as:

$$\tau = \frac{L_R}{u_f} = C_{nC_5} \int_0^{X_{nC_5}} \frac{dX_{nC_5}}{-r_{nC_5}} = C_{nC_5^o} \frac{K_C}{k_{nC_5}} \int_0^{X_{nC_5}} \frac{dX_{nC_5}}{B - C\,X_{nC_5}} \tag{B.4}$$

Where:

$$B = K_C C_{nC_5^o} - C_{iC_5^o}$$

$$C = C_{n\,C_5^o}(1 + K_C)$$

$$L_R : Reactor\ Length$$

Equation B.4 can be analytically integrated and solved for normal pentane conversion (X_{nC_5}):

$$X_{nC_5} = \frac{B}{C}\left[1 - e^{\left(-\frac{L_R}{A}\right)}\right] \tag{B.5}$$

Where:

$$A = \frac{u_f}{k_{nC_5}}\left[\frac{K_C}{1 + K_C}\right]$$

Reactor feed flow and composition can be obtained from the material balance presented in Table B.1 after assuming a reasonable [H:HC] ratio. Equation B.5 still holds two degrees of freedom, namely column length (L_R) and feed velocity u_f. Feed velocity can easily be calculated from feed molar/mass flow rates by assuming a reasonable reactor diameter (d_R). The diameter d_R and length L_R are correlated through reactor volume. For a fixed catalyst volume, total bed volume can be calculated using equation B.6:

$$V_T = \frac{V_C}{(1 - \varepsilon_B)} \tag{B.6}$$

Where:

V_T : Total reactor volume

V_C: Catalyst volume

ε_B: Bed void.

So, for a specified d_R *and* L_R, Equation 9 can be solved to obtain reactor exit conversion X_{nC_5}.

Equilibrium conversion can be calculated by taking the limit of (B.5) as reactor length L_R approaches infinity as illustrated in equation B.7. At 140ºC, respective equilibrium conversions for *n*-C_5 and *n*-C_6 are 0.70 and 0.31. A reactor [H:HC] ratio of [1:1] is adopted in constructing the material balance in Table B.1.

$$X_{nC_5}^{eq} = \lim_{L_R \to \infty} \frac{B}{C}\left[1 - e^{\left(-\frac{L_R}{A}\right)}\right] = \frac{B}{C} = \left[\frac{K_C}{(1+K_C)}\right]\left[\frac{(C_{n\,C_5^o} - C_{i\,C_5^o})}{C_{n\,C_5^o}}\right] \tag{B.7}$$

B.2.2 Reactor Model Validation

The constructed reactor model is validated against experimental exit concentrations and temperatures provided by [Minkkinen et al, 1993] and summarized in Table B.1. Steady state reactants and products axial concentration profiles, along with the temperature profile, are illustrated in Figure B.3. Table B.3 compares reactor effluent concentrations and temperatures reported by [Minkkinen et al, 1993] to those produced in this model. The wall external heat transfer coefficient is used as a tuning parameter to match the exit temperature to that reported by [Minkkinen et al, 1993].

Typical [Minkkinen et al, 1993] Reactor feed and effluent streams' properties are also respectively outlined in streams 7 and 8 of Table B.2.

Figure B.3 illustrates the spatio-temporal profile of reactor temperature. As can be noticed, reactor temperature sharply rises after initial start-up of the reactor and drops as the reactor reaches steady state. The steady-state drop in the temperature profile is due to the saturation of the catalyst pellets. Reactor effluent *n*-C_5 and *i*-C_5 concentrations closely

match those reported by [Minkkinen et al, 1993]. The noticeable difference between *n*-C_6 and *i*-C_6 concentrations reported in this work and those produced by [Minkkinen et al, 1993] is due to averaging the concentrations of hexane isomers at the reactor feed as outlined earlier.

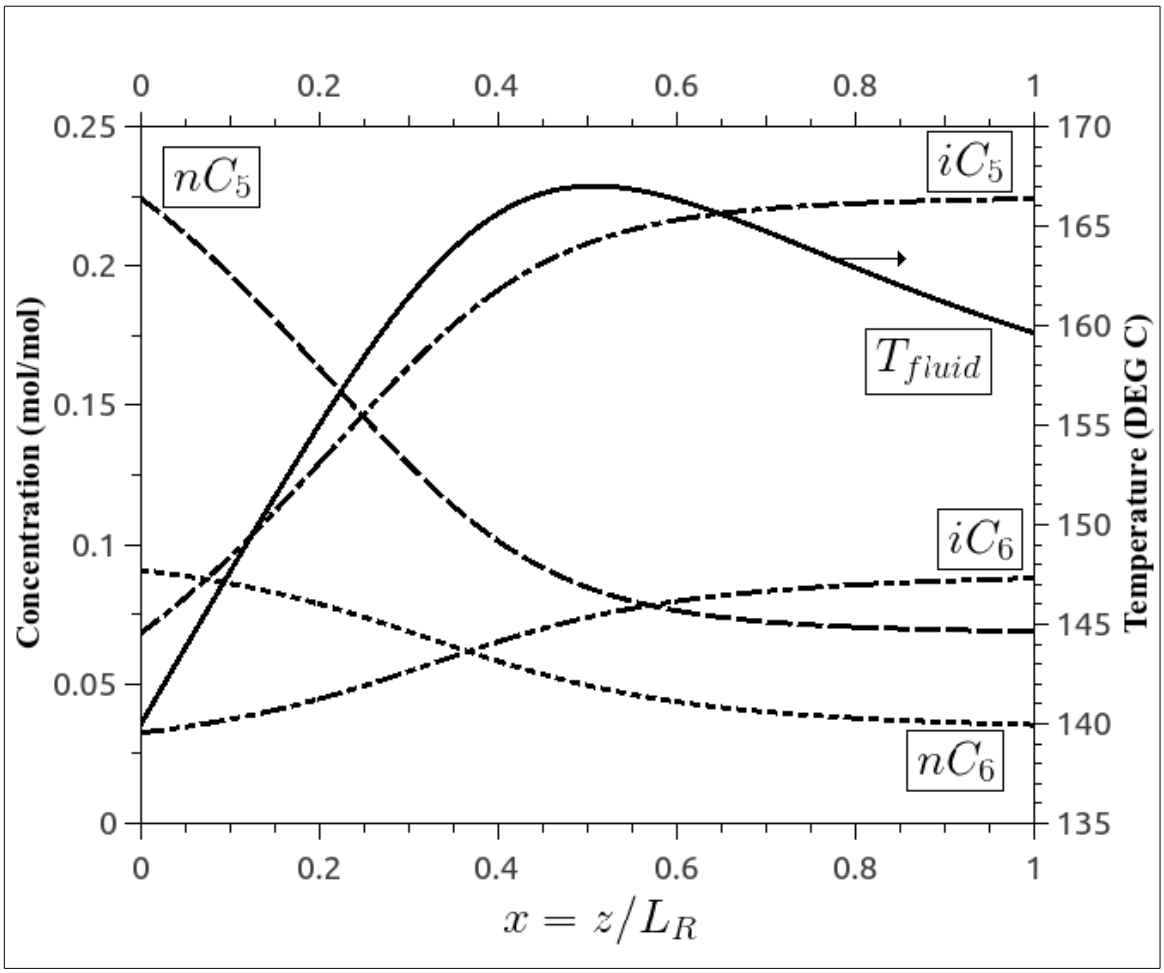


Figure B.3: Steady state reactants and products concentration profiles and temperature profile versus normalized axial distance.

Temperature profile is plotted against the right y-axis while all other profiles are plotted against the left y-axis.

Table B.3: Comparison between reactor effluent concentrations and temperatures reported by [Minkkinen et al, 1993] and those produced in this work.

Variable	Minkkinen	This Work	Absolute Difference	% Difference
n-C_5	0.0894	0.0689	0.0205	23.0
n-C_6	0.0261	0.0354	0.0093	36.0
i-C_5	0.2431	0.2240	0.0191	8.0
i-C_6	0.1233	0.0879	0.0354	29.0
Exit Temperature	160.0	159.6	0.4	0.3

B.3 The PSA Model

B.3.1 Constitutive Equations Used in Constructing the PSA Column Model

In this section, I highlight constitutive relations used in constructing the pressure swing adsorption model discussed in Chapter 4: .

B.3.1.1 *Adsorption Isotherm*

[Nitta et al, 1984] adsorption isotherm is used to calculate solid phase concentration. The isotherm assumes occupation of the adsorbed molecule to multiple sites on the surface of the adsorbent. For a single component adsorption, the isotherm takes the form:

$$nKP = \frac{\theta}{(1-\theta)^n} \tag{B.8}$$

The additional parameter n accounts for non-linearities associated with components exhibiting adsorption behaviours that are not captured by the Langmuir isotherm. Basically, it slows down the decline in adsorption capacity due to the decrease in adsorbate concentration. For n =1, the isotherm reduces to that of Langmuir. Also, when the surface coverage is infinitesimally small, the denominator reduces to unity and the equation reduces to Henry's law. In presence of multicomponent adsorption, Nitta derives the following equation:

$$n_i K_i p_i = \frac{\theta_i}{\left(1 - \sum_{j=1} \theta_j\right)^{n_i}} \tag{B.9}$$

assuming ideal gas behaviour and substituting $p_i = RT < c_i >$ into equation B.9, leads to the form used in our model:

$$n_i < c_i > RT = \frac{1}{K_i^{ads}} \frac{\theta_i}{\left(1 - \sum_{j} \theta_j\right)^{n_i}} \tag{B.10}$$

where the adsorption equilibrium constant K_i^{ads} follows Arrhenius behaviour with respect to changes in temperature.

B.3.1.2 **Gas-Solid Mass Diffusivity**

Calculation of effective diffusivity is required to determine the gas-solid mass transfer coefficient. Effective diffusivity is composed of two terms: molecular (or bulk) diffusivity and Knudsen diffusivity. Molecular diffusivity is evident with dense gases and/or relatively large solid pore sizes. On the other hand, Knudsen diffusivity is dominant in low density gases and/or small pore sizes. The reason behind the distinction between the two diffusivities is related to relative number of collisions between gas molecules to that with the solid surface. In molecular diffusion, collisions between gas molecules are more often than that between a gas molecule and the solid surface. The opposite is true with Knudsen diffusion.

Knudsen diffusivity is calculated using the equation reported by [Kauzmann, 1966] that is derived from kinetic theory of gases:

$$D_k = \frac{2 d_p}{6} \left(\frac{8 RT}{\pi M}\right)^{\frac{1}{2}} \tag{B.11}$$

Since collisions are more often encountered with the gas molecule than with the solid surface and due to the relative small pore sizes, molecular weight is taken as that of the

colliding gas. [Satterfield, 1980]. However, [Ruthven et al, 1994] used a mean molecular weight of the binary diffused substances:

$$\frac{1}{M}=\frac{1}{M_1}+\frac{1}{M_2} \tag{B.12}$$

In this work, we followed the equation by Ruthven et al to calculate *M*.

Binary molecular diffusivity is also derived from kinetic theory of gases and reported as :

$$D_{12}=CT^{(\frac{3}{2})}\frac{\sqrt{\left[\frac{M_1+M_2}{M_1M_2}\right]}}{P\sigma_{12}^2\Omega_D} \tag{B.13}$$

However, because data is scarce on values for collision diameter σ_{12} and collision integral Ω_D, [Fuller et al, 1966] and [Fuller et at, 1969] provided a simplified equation that is based on atomic diffusion volumes:

$$D_{12}=10^{-3}T^{1.75}\frac{\sqrt{\left[\frac{M_1+M_2}{M_1M_2}\right]}}{P\left[\sqrt[3]{\Sigma(v_1)}+\sqrt[3]{\Sigma(v_2)}\right]^2} \tag{1}$$

The noticeable symmetry of the equation implies that $D_{12}=D_{21}$ for both equations.

A simplified form for calculating “ideal” effective diffusivity, based on the assumption of equal but opposing fluxes of components A and B:

$$\frac{1}{D}=\frac{1}{D_m}+\frac{1}{D_k} \tag{B.14}$$

Interestingly, although literature is consistent about the form of the equation , it is not firm about the source of the equation. For example, [Yang et al, 1998] reports that the equation was obtained by Bosanquet [referenced in [Aris, 1975] and [Pollard and Present, 1948]. On the other hand, [Ruthven, 1984] reports that the equation was simultaneously

published by [Evans et al, 1961] and [Scott and Dullien, 1962].

In addition to Knudsen and Molecular diffusivities, we added an additional term that accounts for Poiseuille flow diffusivity that is evident in large pore sizes and/or high pressures:

$$D_p = \frac{d_p^2 P}{16\mu} \tag{B.15}$$

The final equation for "ideal" diffusivity becomes [Ruthven et al, 1994]:

$$\frac{1}{D} = \frac{1}{D_m} + \frac{1}{D_k} + \frac{1}{D_p} \tag{B.16}$$

Since the actual diffusion path is not always equivalent to the radius of the pore, the diffusivity resulting from equation B.16 needs to be corrected. Correction is made through dividing by a factor that accounts for tortuosity effects. Also, to account for the fact that pore diffusion volume is only a fraction of the total pore volume, diffusivity is multiplied by intra-particle void. The resulting diffusivity is called effective diffusivity:

$$D_e = \frac{\varepsilon_p D}{\tau} \tag{B.17}$$

For multicomponent adsorption, [Taylor and Krishna, 1993] discuss the difficulty of obtaining a general formula to calculate mixture diffusivity. They have also indicated the conditions for which assumptions of effective diffusivity would be valid:

1. Binary diffusion coefficients are equal, as we pointed out earlier.
2. The concept of effective diffusivity is also applicable in cases where one component is in large excess of the rest. In this case, effective diffusivity of component *i* that is not in excess reduces to its pure diffusivity D_{ii}.
3. When diffusion occurs through a stagnant gas. In this case the [Wilke, 1950]

approximation holds:

$$D_{i,eff} = \frac{1 - x_i}{\sum_{j=1,\, j \neq i} \frac{x_j}{D_{ij}}} \tag{B.18}$$

The third case is eliminated by default in this work due to the continuous flow of the processes studied. To preserve relative generality, we will be limiting our examples to case 1.

B.2.1.3 **Gas-Solid Overall Mass Transfer Coefficient**

Overall mass transfer coefficient using an equation combining both internal and external mass-transfer coefficients, referenced in [McCabe et al, 2005]:

$$\frac{1}{K_{gl}} = \frac{1}{k_i} + \frac{1}{k_e} \tag{B.19}$$

Where: $k_i = \frac{10 D_e}{d_p}$,

The external mass transfer coefficient is evaluated using the correlation suggested by [Wakao and Funazkri, 1978]:

$$Sh = 2.0 + 1.1 \mathrm{Sc}^{\frac{1}{3}} \mathrm{Re}^{0.6} \tag{B.20}$$

or

$$\frac{k_e d_p}{D_m} = 2.0 + 1.1 \left(\frac{\mu}{\rho_g D_m} \right)^{\frac{1}{3}} \left(\frac{\rho_g u d_p}{\mu} \right)^{0.6} \tag{B.21}$$

The equation is suitable for calculating packed beds axial dispersion coefficient within:

$3 < \mathrm{Re} < 10^4$

B.3.1.4 ***Axial Dispersion Coefficient***

Although dispersion usually occurs in axial and radial directions, radial dispersion is usually neglected when bed diameter is substantially bigger than adsorbent particle

diameter. In our simulations, we will try to hold to a minimum Bed-to-particle diameter ratio of 5 when bed diameter is included as an optimization variable; unless it becomes an optimization constraining variable. For axial dispersion, we used the correlation recommended by [Wen and Fan, 1975]:

$$\frac{1}{Pe}=\frac{0.3}{ReSc}+\frac{0.5}{\left(1+\frac{3.8}{ReSc}\right)} \tag{B.22}$$

or

$$\frac{D_z\rho}{d_p\mu}=\frac{0.3}{\frac{\rho u d_p}{\mu}\frac{\mu}{\rho_g D_m}}+\frac{0.5}{\left(1+\frac{3.8}{\frac{\rho_g u d_p}{\mu}\frac{\mu}{\rho_g D_m}}\right)} \tag{B.23}$$

The readers attention should be drawn to the definition of Pe_m in this equation (that differs from the definition of Pe_m in the rest of the document. The equation is valid in the range of:

$$0.008<Re<400 \text{ and } 0.28<Sc<2.2$$

B.3.1.5 ***Particle-to-Fluid Heat Transfer Coefficient***

Particle-to-Fluid heat transfer coefficient is calculated using the correlation provided by [Wakao et al, 1979]:

$$Nu_p=2.0+1.1\,Pr^{\left(\frac{1}{3}\right)}\mathrm{Re}^{0.6} \tag{B.24}$$

or

$$\frac{h_p d_p}{k}=2+1.1\left(\frac{C_{pg}\mu}{k}\right)^{\left(\frac{1}{3}\right)}\left(\frac{\rho_g u d_p}{\mu}\right)^{0.6} \tag{B.25}$$

This equation is valid in the range of:

$$15<\mathrm{Re}<8500$$

It is also worth noting that this correlation was based on the form that was provided by [Wakao and Funazkri, 1978] and outlined in equations B.19 and B.20.

B.3.1.6 Fluid-to-Wall Heat Transfer Coefficient

For wall heat transfer coefficient, we divided the use of correlations based on the flow regime. Furthermore, whenever applicable, we further divided flow regimes into entrance and fully developed. For entrance region Laminar flow, we used the equation recommended by [Sieder and Tate, 1936]:

$$Nu_d = 1.86\left(Re_d\, Pr\right)^{(1/3)}\left(\frac{d_c}{L}\right)^{(1/3)}\left(\frac{\mu}{\mu_w}\right)^{0.14} \tag{B.26}$$

[Sieder and Tate, 1936] indicate that the properties of this correlation should be evaluated at the arithmetic mean bulk temperature $0.5*(T_{in}+T_{out})$. However, because of the dynamic nature of the process, it is very difficult to estimate (and/or fix) bulk entrance T_{in} and exit T_{out} temperatures. So, we opted for evaluating all properties at unit fresh feed conditions. Evaluating all properties at fresh feed conditions leads to elimination of the viscosity effects, between bulk fluid and wall, appearing at the end of the correlation. The correlation is valid when:

$$\left(Re_d\, Pr\right)\left(\frac{d_c}{L}\right) > 10 \tag{B.27}$$

In addition, [Sieder and Tate, 1936] limited the use of the correlation to Prandtl numbers in the range of $0.48 < Pr < 16{,}700$. Reported errors of this correlation are in the range of $\pm 25\%$. For fully developed laminar flow, I applied the recommendation by [Shah and London, 1978]. Basically, they state that, for fully developed laminar flow, Nusselt number tends to settle at a constant value. For flow through ducts the correlation is simply:

$$Nu_d = 4.364 \tag{B.28}$$

For turbulent flow, I used the correlation proposed by [Gnielinsky, 1976]:

$$Nu_d = \frac{(f/2)(Re-1000)Pr}{1+12.7(f/2)^{(1/2)}(Pr^{(2/3)}-1)}\left[1+\left(\frac{d_c}{L_c}\right)^{(2/3)}\right] \quad \text{(B.29)}$$

$$f = \left[1.58\ln(Re_d) - 3.28\right]^{(-2)} \quad \text{(B.30)}$$

The correlation captures the effects of entrance and fully developed regions. For fully developed turbulent flow, the term (d_c/L_c) is set to zero. It is valid in the following ranges:

$$0.5 < Pr < 2000$$

$$2300 < Re_d < 10^6$$

$$0 < \frac{d_c}{L_c} < 1$$

It should be noted that all these correlations are developed for the case of constant heat flux. Although heat flux might not be uniform in our model, I still think that these correlations are more appropriate than their constant wall temperature counterparts because although the heat flux is not constant, it is evident.

B.3.1.7 ***Pure Component Thermal Conductivity***

Pure component thermal conductivity is estimated using the method of Chung et al ([Chung et al, 1986], [Chung et al, 1984]). The method is tested over wide range of hydrocarbons but not with polar substances. However, the authors indicated that the formula can be used for polar substances if values of parameter β for the polar substances are available. The method was originally established to estimate thermal conductivities at low pressures but, later on, modified to account for high pressures too. As reported by [Chung et al, 1986], error resulting from this formula, at high pressures, is within the range of 5-8%:

$$\lambda = \frac{31.2\,\eta^{o}\psi}{M}\left(G_2^{-1} + B_6\,y\right) + qB_7\,y^2\,T_r^{0.5}\,G_2 \qquad \text{(B.31)}$$

Where:

$$G_1 = \frac{1 - 0.5\,y}{(1-y)^3}$$

$$G_2 = \frac{\left(\frac{B_1}{y}\right)\left[1 - e^{(-B_4 y)}\right] + B_2 G_1 e^{(B_5 y)} + B_3 G_1}{B_1 B_4 + B_2 + B_3}$$

$$B_i = a_i + b_i\,\omega + c_i\,\mu_r^4 + d_i\,\kappa$$

Values of constants a_i, b_i, c_i and c_i are tabulated below:

i	a_i	b_i	c_i	d_i
1	2.4166E+0	7.4824E-1	-9.1858E-1	1.2172E+2
2	-5.0924E-1	-1.5094E+0	-4.9991E+1	6.9983E+1
3	6.6107E+0	5.6207E+0	6.4760E+1	2.7039E+1
4	1.4543E+1	-8.9139E+0	-5.6379E+0	7.4344E+1
5	7.9274E-1	8.2019E-1	-6.9369E-1	6.3173E+0
6	-5.8634E+0;	1.2801E+1	9.5893E+0	6.5529E+1
7	9.1089E+1	1.2811E+2	-5.4217E+1	5.2381E+2

B.3.1.8 ***Mixture Gas-Phase Thermal Conductivity***

As suggested by [Reid et al, 1987], mixture thermal conductivity is estimated using the same equation for pure thermal conductivity but with evaluation of parameters using mixing rules provided by [Wassiljewa, 1904] for equation B.32, and [Mason and Saxena, 1958] for equations B.33 and B.34:

$$\lambda_m = \sum_{i=1}^{n} \frac{y_i \lambda_i}{\sum_{j=1}^{n} y_j A_{ij}} \tag{B.32}$$

$$A_{ij} = \frac{\epsilon \left[1 + \left(\frac{\lambda_{tr_i}}{\lambda_{tr_j}}\right)^{0.5} \left(\frac{M_i}{M_j}\right)^{0.25}\right]^2}{\left[8\left(1 + \frac{M_i}{M_j}\right)\right]^{0.5}} \tag{B.33}$$

$$\frac{\lambda_{tr_i}}{\lambda_{tr_j}} = \frac{\Gamma_j \left[e^{0.0464 Tr_i} - e^{-0.2412 Tr_i}\right]}{\Gamma_i \left[e^{0.0464 Tr_j} - e^{-0.2412 Tr_j}\right]} \tag{B.34}$$

B.3.1.9 ***Gas-Phase Axial Effective Thermal Conductivity***

Axial effective thermal conductivity is estimated using the correlation provided by [Dixon and Cresswell, 1986]. After excluding radial dispersion term (refer to Appendix C), the correlation is simplified to:

$$\frac{1}{Pe_a} = \frac{1}{Pe_{af}} + \frac{k_{as}/k_{af}}{RePr} \tag{B.35}$$

or

$$\frac{k_a}{\dot{m} c_t d_p} = \frac{k_{af}}{\dot{m} c_t d_p} + \frac{k_{as}/k_{af}}{\left(\frac{\rho_g u d}{\mu}\right)\left(\frac{C_{pg}\mu}{k_{af}}\right)} \tag{B.36}$$

As pointed out by the authors, this equation covers ranges of *Re* below and above 100, as opposed to a previous work that only focused on laminar flow. It also accounts for transient heat effects which better suits our model. It should be noted that k_{af} is calculated using the correlations for mixture thermal conductivity outlined earlier.

B.3.1.10 ***Gas-Phase Pure Component Viscosity***

Pure component viscosity is estimated using the method of ([Chung et al, 1984], [Chung et al, 1986]):

$$\eta = 40.875 \frac{F_c (MT)^{0.5}}{V_c^{(2/3)} \Omega_v} \tag{B.37a}$$

$$F_c = 1 - 0.2756\,\omega + 0.059035\,\mu_r^4 + \kappa \tag{B.37b}$$

$$\Omega_v = \left[A (T^*)^{-B}\right] + C\left[e^{-DT^*}\right] + E\,e^{(-FT)} \tag{B.37c}$$

$$T^* = 1.2593\,T_r \tag{B.37d}$$

$$\mu_r = 131.3 \frac{\mu}{(V_c T_c)^{0.5}} \tag{B.37e}$$

$$\begin{aligned} A &= 1.16145 \\ B &= 0.14874 \\ C &= 0.52487 \\ D &= 0.77320 \\ E &= 2.16178 \\ F &= 2.43787 \end{aligned} \tag{B.37f}$$

B.3.1.11 ***Gas Phase Mixture Viscosity***

Gas phase mixture viscosity is calculated using a simplification of the kinetic theory of gases that is proposed by [Wilke, 1950]:

$$\eta_m = \sum_{i=1}^{n} \frac{y_i \eta_i}{\sum_{j=1}^{n} y_j \phi_{i,j}} \tag{B.38}$$

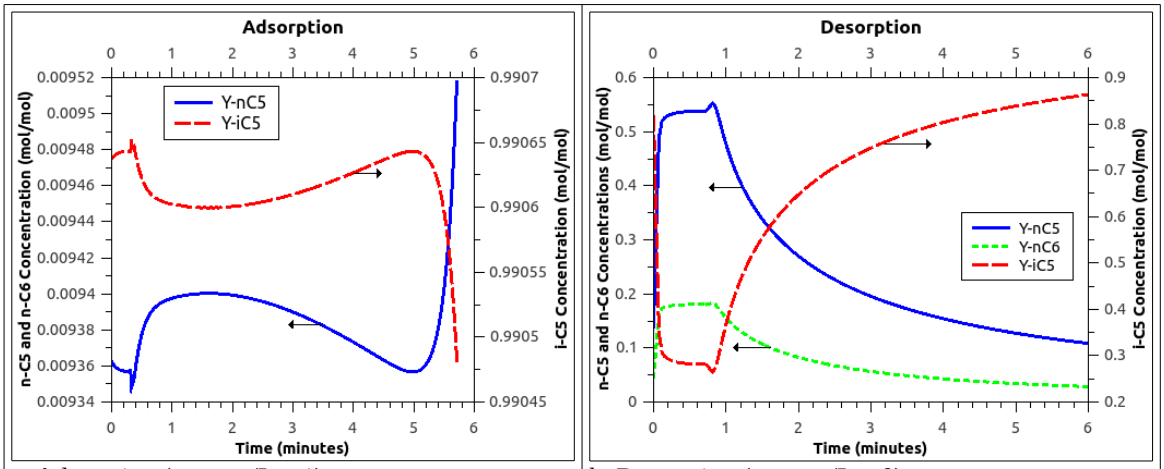
$$\phi_{i,j} = \frac{\left[1 + \left(\frac{\eta_i}{\eta_j}\right)^{0.50} \left(\frac{M_j}{M_i}\right)^{0.25}\right]^2}{\left[8\left(1 + \frac{M_i}{M_j}\right)\right]^{0.5}} \quad \phi_{i,j} = \frac{\left[1 + \left(\frac{\eta_i}{\eta_j}\right)^{0.50} \left(\frac{M_j}{M_i}\right)^{0.25}\right]^2}{\left[8\left(1 + \frac{M_i}{M_j}\right)\right]^{0.5}} \tag{B.39}$$

B.3.2 PSA Model Validation

The validity of the constructed PSA model is tested against the PSA patent for separation of iso- from normal paraffins that was filed by [Minkkinen et al, 1993]. A variant of the PSA section of this process was modelled by [Silva and Rodrigues, 1998]. Silva and Rodrigues have published results for isothermal and non-isothermal cases. Spatial

distributions of normal pentane and normal hexane concentrations (as mole fractions) for the cyclic steady state (CCS) step are reported for the isothermal case. In addition, temperature profiles are reported for the non-isothermal case. Our verification process will target two goals. The first goal is to produce raffinate and extract products concentrations that match those reported by [Minkkinen et al, 1993]. The second goal is to compare CSS concentration and temperature profiles obtained in this work with those reported by Silva and Rodrigues and discuss the sources of bias between reported results.

According to [Minkkinen et al, 1993], the PSA column undergoing Adsorption phase produces iso-pentane with purity greater than 99%. Since the PSA process is totally dynamic, calculation of isopentane purity is only attained through averaging effluent concentration throughout adsorption step.



a. Adsorption (at x = z/L = 1) b. Desorption (at x = z/L = 0)

Figure B.4: Evolution of raffinate and extract concentrations during the Cyclic Steady State (CSS) adsorption and desorption steps.

Raffinate is collected at the back-end of the vessel during Adsorption step. Extract is collected during Desorption step at the front end of the vessel. Normal hexane concentration is omitted from the figure to allow better scaling of axes. The normal hexane exit concentration is always zero as can be realized fromFigures B.5b and B.5d.

Figure B.4a illustrates the exit concentration of normal and iso pentane (molar fractions) against time for the CSS adsorption step. For the first 5 minutes, the curve indicates that the process is producing a nearly steady 99+ mol% pure iso pentane. Purity starts dropping at the end of the step due to a slight breakthrough of normal pentane. The

average isopentane purity throughout the adsorption step is 99.06 mol%. Thus we may comfortably conclude that simulation results coincide with experimental data reported by [Minkkinen et al, 1993]. The exit concentration of normal hexane is omitted from the figure to allow better scaling for the left y-axis where normal pentane concentration is plotted. Normal hexane concentration at product end of the column during adsorption step is always zero. The axial profile plotted in Figure B.5b supports this fact.

Following the same path, [Minkkinen et al, 1993] reports that desorption step effluent consists of 27 mol% normal pentane, 7.5 mol% normal hexane with the balance being iso pentane. The model reports average concentrations of 26.31 and 8.15 mol% for normal pentane and normal hexane, respectively. Differences between reported figures are less than 1 mol%.

Concentration evolution profiles for the depressurization and desorption steps are illustrated in Figure B.4b. The increase in normal pentane and hexane concentrations at the beginning of the step is due to the rapid escape of isopentane from the column and the desorption of normals from adsorbent pellets to the gas phase when depressurizing the vessel from 15 to 2 bars. However, isopentane concentration picks up once the purge stream is introduced during desorption step. Minkkinen does not distinguish between depressurization and desorption steps as the effluent of both steps is combined and recycled back to the distillation column (De-isopentanizer).

Minkkinen also reports that average column temperature is maintained at about 300ºC in both adsorption and desorption steps. The model confirms these results as illustrated in Figures B.5a-B.5d with the exception of the sharp temperature wave that is located close to the product end during adsorption step (Figure B.5b). The sharp temperature wave illustrated in Figure B.5b is due to dynamic adsorption. During pressurization step, the

adsorbate is concentrated at the front end (left) of the vessel with unadsorbable material (inerts) occupying the rest of the vessel. Adsoption requires high pressures . Thus, little adsorption occurs during pressurization step. However, at the start of adsorption step, the bed is already fully pressurized and the product end (right) is open for collection of inert material. Adsorption process is exothermic by nature. Any adsorbed material releases energy that heats up the bed causing a temperature rise. As the bed saturates, no localized adsorption occurs at saturated locations and the temperature at these locations drops to that of the feed due to heat exchange with feed. However, since adsorption is still evident in unsaturated locations of the bed, temperature rises in these locations causing a sharp temperature wave. This consecutive saturation of the bed constructs a temperature wave that starts at feed introduction end when adsorption step starts and moves towards the product end as the front end of the bed is saturated with adsorbates. The wave settles at its final location, illustrated in B.5b, before switching the bed to the depressurization step.

Let us now turn our attention to the results reported by [Silva and Rodrigues, 1998]. Silva and Rodrigues modelled and laboratory tested an exact copy of the PSA unit described by Minkkinen with few modifications. The major difference between both processes lies in the composition of the purge stream. Minkkinen used the top effluent of the de-isopentanizer column to purge the PSA column undergoing desorption step. This scheme, although resulting in a better PSA unit recovery, the purity of the raffinate deteriorated. [Silva and Rodrigues, 1998] opted for recycling part of the pure product stream as a purge stream for the desorption step. This new setup resulted in a high purity product but on the expense of recovery. Purge feed compositions, product purity and recovery of both processes is summarized in Table B.4.

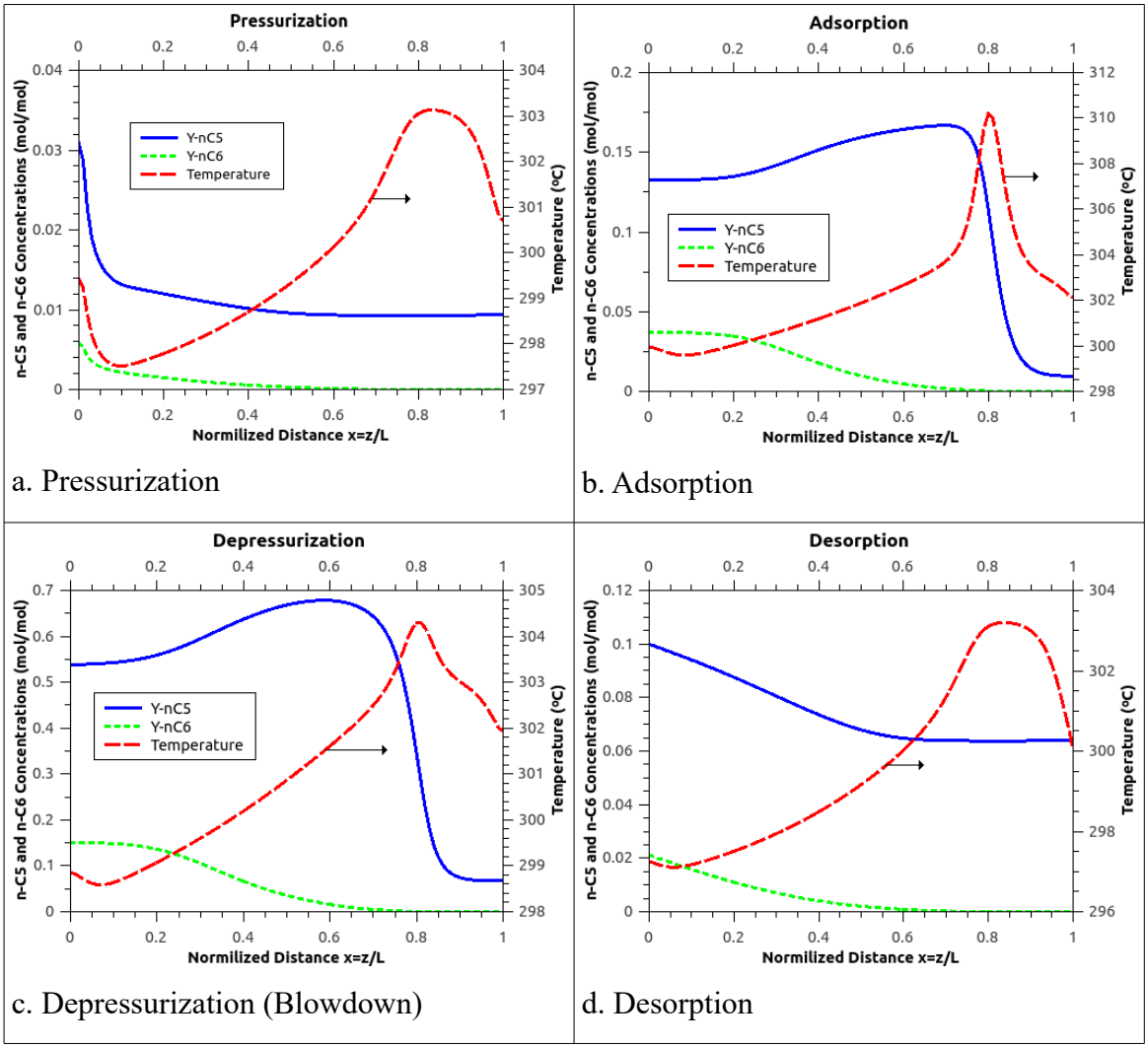


Figure B.5 : Axial concentration and temperature profiles at the end of the Cyclic Steady State.

Plots are generated using the model developed in this work for the case described by [Minkkinen et al, 1993] in his patent. Temperature profiles are plotted against the right y-axis while composition profiles are plotted against the left one.

The high recovery of the Minkkinen process is due to the setup of the process flowsheet. As indicated earlier, Minkkinen uses the stream existing the depentaniser column overhead as a purge to the PSA column undergoing desorption step. This means that all the product stream is recovered since no amount is recycled as a purge stream.

Table B.4: Comparison between Minkkinen and Silva & Rodrigues experiments' recoveries and purities.

	Purge Stream Composition (mol%)			% Recovery	% i-C5 Purity
Process	**n-C5**	**n-C6**	**i-C5**		
Minkkinen	6.9	0.0	93.1	100.00	98.941
Silva and Rodrigues	0.0	0.0	100.0	14.89	99.998

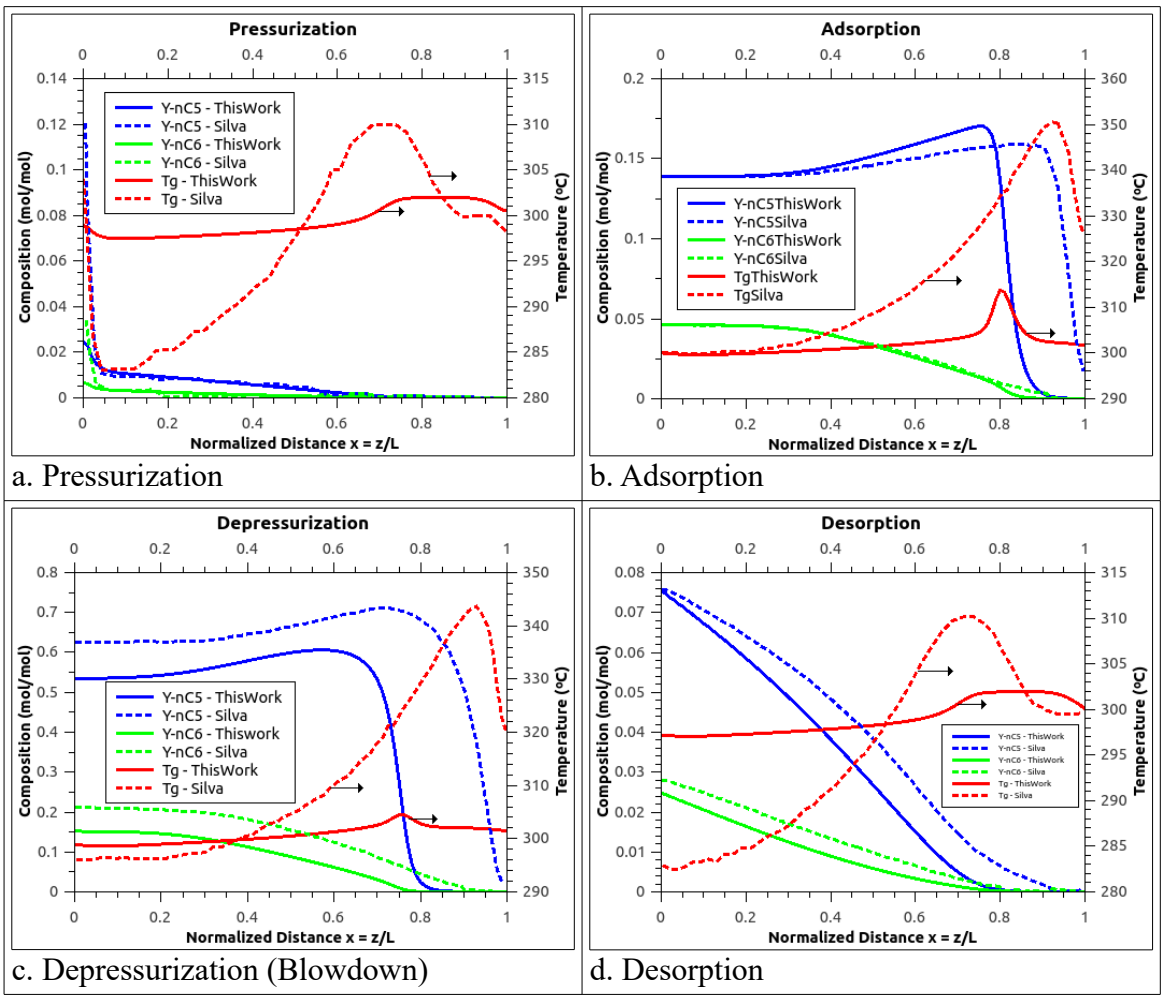


Figure B.6: Comparison of CSS spatial profiles for temperature and composition between results produced in this work and those reported by [Silva and Rodrigues, 1998].

Dotted lines represent results published by [Silva and Rodrigues, 1998]. Continuous lines represent the results produced in this work.

Silva and Rodrigues published CSS axial composition and temperature profiles. Their results formulate good bases to validate the CSS axial profiles produced in this work. Since no tabular data were provided by [Silva and Rodrigues, 1998], I had to digitize their plots before re-plotting them in Figure B.6. For each of the CSS steps, continuous lines in

Figure B.6 represent results obtained from this work while dotted ones represent the work published by Silva and Rodrigues. Temperature profiles are plotted against the right y-axis whereas molar concentrations of normal pentane and normal hexane are plotted against the left y-axis.

The noticeable difference between the two works lies in the temperature profiles. In general, Silva and Rodrigues report higher temperature profiles than those produced in this work. Silva and Rodrigues attribute the rise in the temperature to the use of a parabolic temperature profile to simulate the oven used in their experiments. However, they do not outline the nature of the parabolic profile or how it is incorporated in the simulation model. The higher temperature profile also explains the higher saturation of their PSA bed at the end of the adsorption step compared this work. At higher temperatures, adsorbents saturate at lower concentrations of adsorbates and vice versa. In fact, the influence of the extra oven in the data reported by Silva and Rodrigues explains almost all discrepancies between results. Minkkinen reported an average axial temperature of 300ºC. The results in this simulation work are more aligned with Minkkinen experimental results.

Another noticeable difference is in the concentration front of the pressurization step. Silva and Rodrigues results report higher concentration fronts at the end of the pressurization step. This is probably due to the use of a lower pressurization rate (*M)*. Silva and Rodrigues use exponential function to build up pressure during pressurization step and to depressurize it during depressurization step. The adjustable variable in this exponential function is the pressurization rate *M*. Although they mention the use of the pressurization rate constant *M*, they don't make any notes about the magnitude of that constant. To produce the curves in this work, we used a pressurization rate $M = 1/t_{ref}$ (s^{-1}) where t_{ref} is

the refrence time defined as L/U_{ref}, L being the length of the column and U_{ref} is the refrence velocity. This choice of M corresponds to $\bar{M}=1$ for normalized equations.

To conclude, the profiles produced in this work closely resembles those reported by [Minkkinen et al, 1993] and [Silva and Rodrigues, 1998]. Discrepancies where explained and justified whenever encountered. Thus, the developed model well suits further work on this area.

APPENDIX C: PIECE-WISE CUBIC HERMITE INTERPOLATING POLYNOMIALS

C.1 Introductory

In one dimensional polynomial interpolation, a polynomial $p(x)$ is created from either a function $f(x)$ or a set of $(x_i, f(x_i))$ data representing f. When interpolating, the $(x_i, f(x_i))$ data points are interchangeably called nodes, interpolants or control points. Polynomial interpolation guarantees the existence of $p(x)$ that satisfies:

$$p(x_i) = f(x_i) \tag{C.1}$$

for all nodes. However, the accuracy of the match within the nodes does not imply accuracy of interpolation between them [Cheney and Kincaid, 1999]. In fact, a perfect interpolation match only exists when the interpolating polynomial is an exact replica of $f(x)$. Thus, as x deviates from the control points, the error between $f(x)$ and $p(x)$ increases. One would think that a higher order $p(x)$ might result in a better interpolation. However, this is not the case. As [Cheney and Kincaid, 1999] state it, the news were shocking when the scientific community realized that higher order polynomials deteriorate interpolation accuracy.

To solve this problem, scientists resort to using low order polynomials and dividing the interpolation region into segments. The term spline refers to such segmentation whether even or uneven [Kochanek and Bartels, 1984].

The lowest order polynomial that can be interpolated is the first order polynomial or the straight line. However, straight line interpolation suffers several drawbacks. It is not curved. Thus, it does not follow the intended curvature of the original function. It can

only be differentiated once and the derivative (a constant) is not a function of space, marking it impractical for mathematical or engineering applications requiring the estimate of derivatives to approximate maxima, minima and inflection points of functions among other uses.

The second obvious alternative is second order polynomials (parabolic functions). These functions provide better curvature alignment with original functions. However, they suffer the drawback of constant second derivatives. This drawback prevents scientists from estimating inflection points of the original function.

Third order interpolating polynomials solve the problems encountered in first and second order ones. However, although most third order polynomials well interpolate a given data set, some provide advantageous features over others. Let us start our discussion with a well known theorem:

Theorem A.1:

For a set of distinct points $x_0, x_1,...x_n$ having corresponding y values of $y_0, y_1, \ldots. y_n$, a unique polynomial of a degree ≤ n exists such that $p(x_i) = y_i$ for $0 \leq i \leq n$.

According to Theorem A.1, the shape of the unique polynomial is only a function of the selected data set within the interpolation segment. There are numerous ways to construct an interpolating polynomial from a data set. Newton divided difference and Lagrange interpolating polynomials are among the most popular [Cheney and Kincaid, 1999]. As Newton interpolating polynomials constitute a subset of *hermite* interpolating polynomials, we will start our discussion by demonstrating their construction from a set of data.

Example C.1 Constructing a Newton interpolating polynomial

Let us construct a Newton interpolating polynomial from the below set of scattered data assuming x is the independent variable:

x	-1	1	0	2
y	4	3	6	-11

The final interpolating polynomial is a combination of n-1 sub polynomials where n represents the number of data points. The first polynomial resembles a horizontal line, the second is a sloped line, the third is a parabola and so forth. For this example, since n=4, we will only be able to construct a third degree interpolating polynomial. Using the first point from the left (selection of points is arbitrary as long as each point is utilized once) and realizing that the first polynomial is of degree 0:

$$p_0(x) = y_0 = 4 \tag{C.2}$$

The second polynomial (degree 1) is constructed by adding an extra term to the first one:

$$p_1(x) = p_0(x) + c_1(x - x_0)\text{, where } c_1 \text{ is a constant} \tag{C.3}$$

Substituting equation C.2 in equation C.3:

$$p_1(x) = 4 + c_1(x - x_0) \tag{C.4}$$

Using the second point from the left, we can easily calculate that c=-0.5.

The third polynomial (degree 2) is constructed by adding an additional term to the second polynomial:

$$p_2(x) = p_1(x) + c_2(x - x_0)(x - x_1)\text{, where } c_2 \text{ is a constant} \tag{C.5}$$

Substituting equation C.4 in equation C.5:

$$p_2(x)=4-0.5(x-x_0)+c_2(x-x_0)(x-x_1) \quad \text{(C.6)}$$

Substituting appropriate values from the first three points, c_2 is calculated to be -2.5. Hence,

$$p_2(x)=4-0.5(x+1)-2.5(x+1)(x-1) \quad \text{(C.7)}$$

Following the same pattern, the forth polynomial becomes:

$$p_3(x)=4-0.5(x-x_0)-2.5(x-x_0)(x-x_1)+c_3(x-x_0)(x-x_1)(x-x_2) \quad \text{(C.8)}$$

Inserting appropriate substitutions results in:

$$p_3(x)=4-0.5(x+1)-2.5(x+1)(x-1)-(x+1)(x-1)x \quad \text{(.9)}$$

The combined polynomial is additive. Thus,

$$p(x)=p_3(x)=4-0.5(x+1)-2.5(x+1)(x-1)-(x+1)(x-1)x \quad \text{(C.10)}$$

A plot of the final polynomial is illustrated in Figure C.1.

The reader can easily notice the recursive nature of the procedure. Also, the formula ensures that each added polynomial passes through all the nodes of all previously constructed polynomials. In fact, the adherence to forcing the polynomial into passing through each node is what creates highly oscillatory behaviour when interpolating between nodes using higher order polynomials.

A more convenient way to obtaining these coefficients would be to use Newton divided difference recursive formula [Cheney and Kincaid, 1999] and [Chapra and Cancade, 2002]. The general form of the formula is:

$$f[x_n,x_{n-1},.....x_1,x_0]=\frac{f[x_n,x_{n-1},.....x_{2,}x_1]-f[x_{n-1},x_{n-2},.....x_1,x_0]}{x_n-x_0} \quad \text{(C.11)}$$

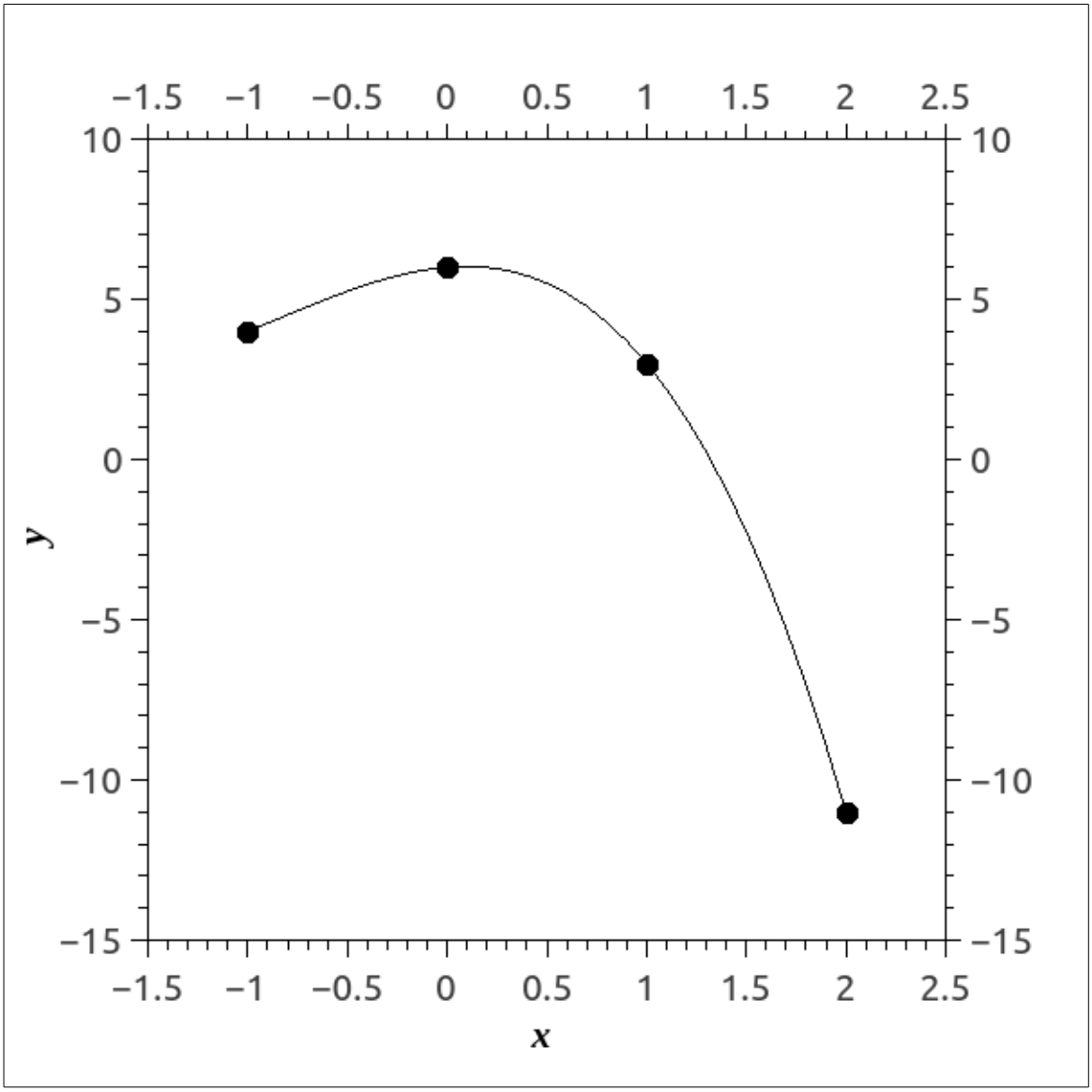


Figure C.1: A plot of the third degree polynomial constructed from Example A.1.

Using the above formula, c_0 is immediately realized to be $c_0 = f[x_0] = y_0$. c_1 is calculated as:

$$c_1 = \frac{f[x_1] - f[x_0]}{x_1 - x_0} = \frac{y_1 - y_0}{x_1 - x_0} \tag{C.12}$$

The coefficient c_2 is calculated using equation (C.13):

$$c_2 = f[x_0, x_1, x_2] = \frac{f[x_2, x_1] - f[x_1, x_0]}{x_2 - x_0} \tag{C.13}$$

The theory is best explained by an Example:

Example C.2: Obtaining Coefficients using Newton divided difference formula

The tabulated data in Example are used to calculate the coefficients of equation (C.10) as illustrated in the table below:

Table C.1: Deriving coefficients of Newton interpolating polynomial

x_i	$f[x_i] = y_i$	$f[x_i,x_j]$	$f[x_i,x_j,x_k]$	$f[x_i,x_j,x_k,x_l]$
-1	$4 = c_0$			
1	3	$-0.5 = c_1$		
0	6	-3	$-2.5 = c_2$	
2	-11	-8.5	-5.5	$-1 = c_3$

C.2 Osculating Polynomials

Note that in the preceding discussion, nothing has been assumed about the conformity of interpolating polynomial derivatives to those of the original function at the nodes. Thus, although the interpolating polynomial nodes' values confirm to the provided function values (or data set), the derivatives of the interpolating polynomial may or may not coincide with those of the original function. If the derivatives of the interpolating polynomial are required to match the derivatives of the original function, Osculating interpolating polynomials should be used.

"Osculating[2] polynomials" is a general term used to describe a set of interpolating polynomials that agree with a set of *n* observation values as well as their *m+1* derivatives. The degree of the resulting osculating polynomial depends on the amount of information available around the nodes.

Osculating polynomials are also called *hermite* polynomials [Moler, 2004]. They are named after the French mathematician Charles Hermite. The maximum degree of an Osculating polynomial can be calculated as:

2 In Latin, "Osculari" means "to kiss".

$$r = n - 1 + \sum_{i=0}^{n} m_i \tag{C.14}$$

Where n represents the number of observations and m_i represents the number of available derivatives per i^{th} observation. The coefficients of the polynomial can be evaluated using Newton divided difference technique that was outlined earlier. The difference between the original Newton divided difference procedure and the one used for osculating polynomials is that in osculating polynomials the same observation is repeated a number of times that are equal to the available derivatives at the observation. The discussion is better understood with an illustration.

Example C.3 Constructing an osculating polynomial

The following table of observations is constructed using the trigonometric sine function and its subsequent first and second derivatives:

x_i	$f(x_i)$	$f^{(1)}(x_i)$	$f^{(2)}(x_i)$
-1	-0.8414710	0.5403023	
0	0	1.0	0.0

The second derivative of the first observation is deliberately omitted to demonstrate that the procedure can always be applied to the available derivatives. For this table of observations, n=2. Also, Since only one derivative is available for the first observation, $m_1 = 1$. Similarly, for the second observation, two derivatives are available. This leads to $m_2 = 2$. Thus, the resulting polynomial is a fourth degree (r = 2-1+1+2) polynomial.

To construct the coefficient matrix using Newton divided differences, each observation should be repeated m_i+1 times. Thus the first observation is repeated

twice and the second observation is repeated thrice. Since it is impossible to use equation (C.11) for repeated observations, equation (C.11) is replaced with the respective available derivatives from the observations table. The resulting table is illustrated below. Note that the z_i column is only used to count the number of used observations to construct the table.

Table C.2: Using Newton divided differences technique to obtain the coefficients of an osculating polynomial for the set of data presented in Example C.3.

z_i	x_i	$f[x_i]$	$f[x_i,x_j]$	$f[x_i,x_j,x_k]$	$f[x_i,x_j,x_k,x_l]$	$f[x_i,x_j,x_k,x_l, x_m]$
1	-1	-0.84147 =c_1				
2	-1	-0.84147	0.54030=c_2			
3	0	0.0	0.8414709	0.30116= c_3		
4	0	0.0	1	0.15852	-0.14263=c_4	
5	0	0.0	1	-0.17255	-0.33108	-0.18844=c_5

The final polynomial, constructed from the above table, is presented in equation (C.15). The equation is split into two rows due to space limitations. Since the resulting polynomial has a match with the original function, at the nodes, up to the second derivative, the function is said to be second degree parametrically continuous, or simply C^2.

$$f(x)=\begin{Bmatrix} -0.84147+0.54030(x+1)+0.30116(x+1)^2 \\ -0.14263(x+1)^2(x-0)-0.18844(x+1)^2(x-0)^2 \end{Bmatrix} \quad \text{(C.15)}$$

The original sine function and its 4th order osculating polynomial are plotted in Figure C.2. using discretisation intervals of h=0.1. Also, error values between the original function and the interpolating polynomial are outlined in Table C.3.

Table C.3: Regression results for correlating simulation run length with number of discretization nodes.

x_i	$f(x_i)$			% Error	
	$sin(x_i)$	$o^2(x_i)$	$h(x_i)$	$o^2(x_i)$	$h(x_i)$
-1.000	-0.841	-0.841	-0.841	0	0
-0.900	-0.783	-0.785	-0.783	0.172	0.023
-0.800	-0.717	-0.722	-0.717	0.595	0.078
-0.700	-0.644	-0.652	-0.643	1.146	0.144
-0.600	-0.565	-0.574	-0.563	1.715	0.208
-0.500	-0.479	-0.490	-0.478	2.201	0.256
-0.400	-0.389	-0.399	-0.388	2.508	0.280
-0.300	-0.296	-0.303	-0.295	2.541	0.271
-0.200	-0.199	-0.203	-0.198	2.204	0.225
-0.100	-0.100	-0.101	-0.100	1.394	0.135
0.000	0.000	0.000	0.000	0	0

It is worthy at this point to discuss some of the continuity aspects of spline functions. Since spline functions use several segments to construct a given curve, the continuity at the points connecting the segments is of critical importance as it determines smoothness of the final curve. If the function only matches observations but not their subsequent derivatives, the function is called a G^0 geometric function. If the resulting function matches the observations and the directions of their first derivatives but not their respective values, the function is called G^1 geometric function. In G^1 functions, the curve leans more toward the tangent of one side of the segment compared to the other. Parametric continuity imposes more restrictions on joints between segments. C^k parametric continuity implies a match between the connecting segments up to the k^{th} derivative of the interpolating function. Thus, by definition, Osculating functions are C^k compliant.

C.3 C^1 Hermite Interpolating Polynomials

A C^1 hermite interpolating polynomial is a *hermite* polynomial were only observations

and their respective first derivatives (tangents) are utilized to construct it. The same methods used in the previous section will be repeated in this section. In addition, we will explore some of the behaviours of C^1 interpolating polynomials. The construction of a polynomial from a data set is illustrated in Example C.4.

Example C.4: Constructing a C^1 *hermite* polynomial from a data set

Let us use the observations in Example C.3 to construct a C^1 hermite interpolating polynomial. Since the coefficients are readily available in Table C.2, it becomes a trivial exercise to construct the polynomial following the technique used earlier. Since no derivatives beyond the first derivative will be required, the resulting polynomial will be of a third degree (r = 2-1+1+1 = 3). The last row and last column of Table C.2 will be omitted since they involve the second derivative of the second observation. The resulting polynomial is outlined in equation A.16:

$$f(x) = -0.84147 + 0.54030(x+1) + 0.30116(x+1)^2 - 0.14263(x+1)^2(x-0) \qquad \text{(C.16)}$$

Note that the polynomial in equation (C.16) is exactly the same as the one in equation (C.15) after omitting the last term involving the second derivative. The behaviour of the original function, its second order (O^2) osculating polynomial and its C^1 hermite equivalent are plotted in Figure C.2 for a discretisation interval h=0.1. Also, interpolation results and error values are tabulated in Table C.3.

Apart from a good fit of both interpolating polynomials to the original function, Figure C.2 does not reveal much. However, the error reported in Table C.3 clearly indicate that the additional term used in the osculating polynomial is a source of noise rather than an error reduction term. Error is reduced by about 10 orders of magnitude just by omitting the extra term.

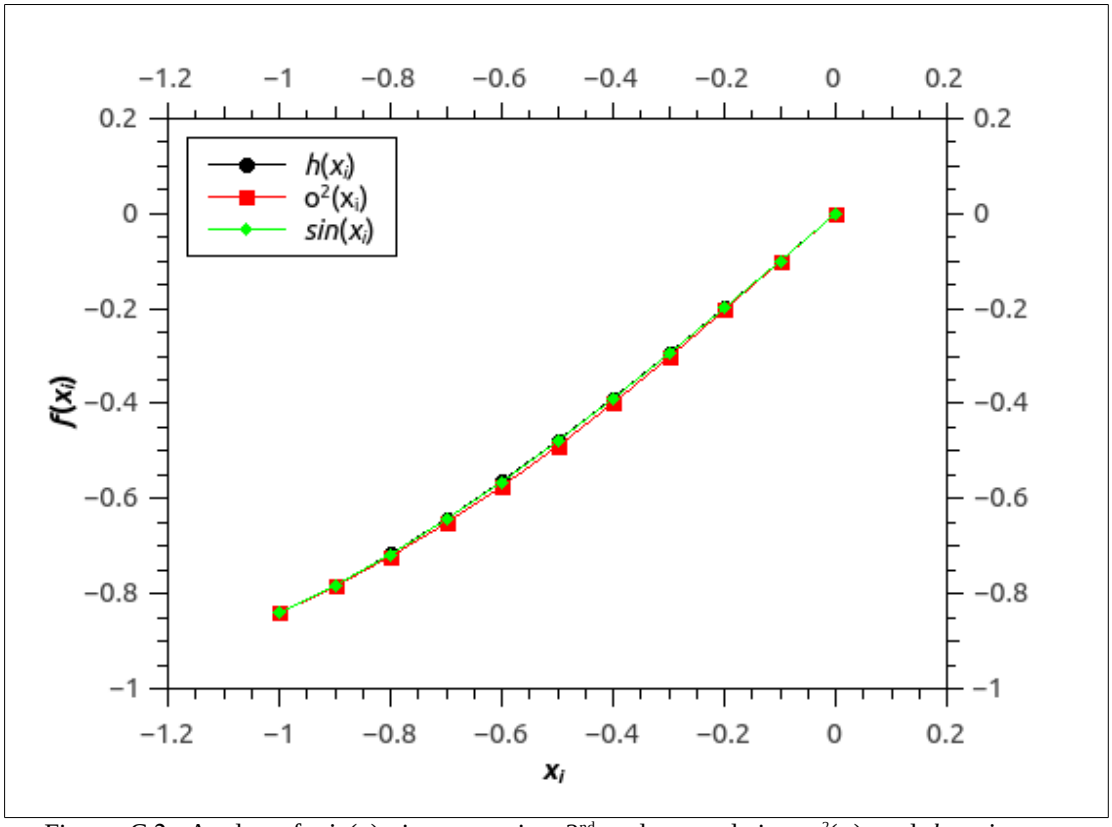


Figure C.2: A plot of *sin*(x), its respective 2nd order osculating $o^2(x_i)$ and *hermite* polynomials over the interval [-1,0] and with segment discretisation of h=0.1.

Although C^1 *hermite* interpolating polynomials can be constructed from Newton divided difference formula, a more convenient (and widely used) method to construct them is to to think of the polynomial as a piece-wise polynomial. Piece-wise polynomials are complex polynomials that are constructed from a set of known elemental polynomials. Since we are dealing with cubic *hermite* polynomials, we will restrict the discussion to this class of polynomials. However, the concepts apply to any *hermite* polynomial with a lower or higher degree. The concept is better illustrated in a matrix form. Also, since we are dealing with spatial coordinates, it is better to use parametric notation instead of explicit coordinate notation. This means that any dimensional curve will be defined using a parameter t to denote its location. The coordinates $x(t)$, $y(t)$ and $z(t)$ are functions of the parametric variable t. We will limit our discussion in this appendix to one dimensional polynomials. Appendix D covers interpolation in multi-dimensional space.

A *hermite* curve can be described as a matrix multiplication of a basis functions matrix *M*, a geometry vector *g* and a polynomial vector *p*. Also, to simplify derivations, both the parametric variable *t* and the geometric coordinate (e.g. *x*) will be scaled to extend between [0,1] The polynomial vector *p* can be written in a parametric form as:

$$p_3 = [t^3 \;\; t^2 \;\; t \;\; 1] \tag{C.17}$$

The geometric vector *g* is a vector holding the basic properties of the curve. For cubic *hermite* polynomial, this vector translates into a four-elements vector. Two of the elements hold the coordinates of the control points and the other two hold the values of their respective first derivatives. Thus, *g* can be expressed as:

$$g = [P_o \;\; P_1 \;\; R_o \;\; R_1] \tag{C.18}$$

The matrix *M* is the coefficient matrix of all base polynomials. Since we are dealing with a third order polynomial, the matrix *M* will have to contain four columns, each column corresponding to a coefficient of the p_3 vector. Also, since the curve is constrained, by two control points and their respective derivatives, the number of rows corresponding to the constrains should be equal to 2 control points + 2 derivatives = 4 columns. The final matrix is thus a 4x4 matrix:

$$M = \begin{bmatrix} m_{11} & m_{12} & m_{13} & m_{14} \\ m_{21} & m_{22} & m_{23} & m_{24} \\ m_{31} & m_{32} & m_{33} & m_{34} \\ m_{41} & m_{42} & m_{43} & m_{44} \end{bmatrix} \tag{C.19}$$

Thus, the parametric form for any of the coordinates (e.g. *x*) can be expressed as:

$$x(t) = p_3 M g_x^T = [t^3 \;\; t^2 \;\; t \;\; 1] \begin{bmatrix} m_{11} & m_{12} & m_{13} & m_{14} \\ m_{21} & m_{22} & m_{23} & m_{24} \\ m_{31} & m_{32} & m_{33} & m_{34} \\ m_{41} & m_{42} & m_{43} & m_{44} \end{bmatrix} \begin{bmatrix} g_{1x} \\ g_{2x} \\ g_{3x} \\ g_{4x} \end{bmatrix} \tag{C.20}$$

Using the four constraints defining the curve, we can construct a matrix A satisfying all constraints as follows:

$$A=\begin{bmatrix}0 & 0 & 0 & 1\\ 1 & 1 & 1 & 1\\ 0 & 0 & 1 & 0\\ 3 & 2 & 1 & 0\end{bmatrix} \tag{C.21}$$

Note that the first row of the matrix corresponds to the first control point having a value of 0 (x(0) = [0 0 0 1]Mg_x) since at *x(0)* all t^k *(k=1,2,3)* terms will evaluate to 0. The second row corresponds to *x*(1) = 1. The third term corresponds to *x*'(0) = *Ro* and so forth. The goal of this exercise is to evaluate the coefficients of the basis functions. This goal is achieved through solving :

$$g^*=\begin{bmatrix}0 & 0 & 0 & 1\\ 1 & 1 & 1 & 1\\ 0 & 0 & 1 & 0\\ 3 & 2 & 1 & 0\end{bmatrix} M g^* \tag{C.22}$$

Of course, the only way to satisfy this equation is to conclude that $AM=I$ *or* $M = A^{-1}$. Solving for *M* yields the coefficients of the elemental functions in a matrix form as outlined in (C.23). Note that the coefficients of each elemental function are arranged column wise because of the order of the *M* matrix in equation (C.23).

$$M=\begin{bmatrix}2 & -2 & 1 & 1\\ -3 & 3 & -2 & -1\\ 0 & 0 & 1 & 0\\ 1 & 0 & 0 & 0\end{bmatrix} \tag{C.23}$$

From (C.23), the elemental functions can be written as:

$$b_1(t)=2t^3-3t^2+1 \tag{C.24}$$

$$b_2(t)=-2t^3+3t^2$$

$$b_3(t)=t^3-2t^2+t$$

$$b_1(t)=2t^3-3t^2+1$$
$$b_4(t)=t^3-t^2 \qquad \text{(C.24)}$$

Figure C.3 illustrates the individual behaviour of each of the base functions. The final hermite polynomial is written as:

$$h_x(t)=M\,g=(3t^3-3t^2+1)\,P_{0x}+(-2t^3+3t^2)\,P_{1x}+(t^3-2t^2+t)\,R_{0x}+(t^3-t^2)\,R_{1x} \qquad \text{(C.25)}$$

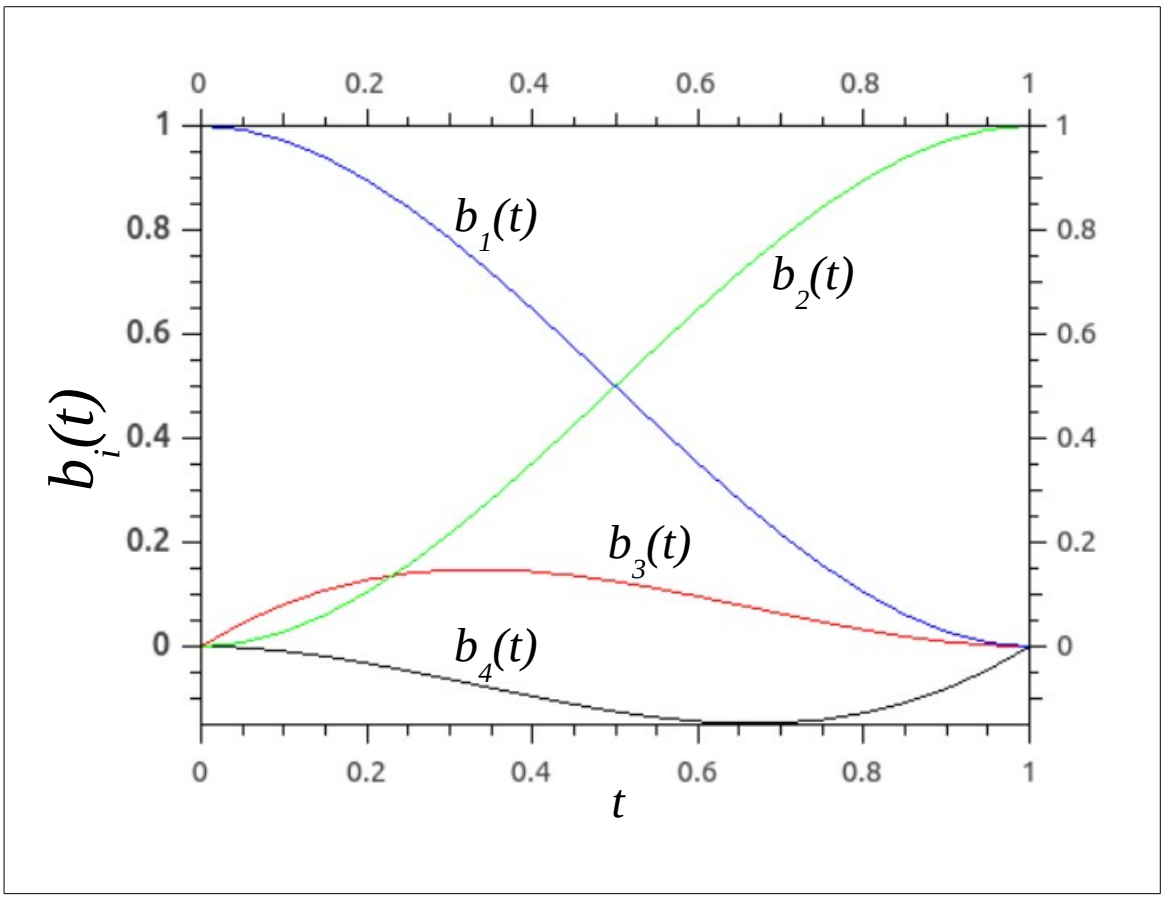


Figure C.3: The basic functions of a *hermite* interpolating polynomial

The *hermite* polynomial form presented in equation (C.25) is more convenient to program and reduces computational power when compared to Newton divided difference formula. Nevertheless, more computationally efficient formulas can also be derived out of equation (C.25).

If the derivatives (tangents) at the first and second control points (R_{0x} and R_{1x}, respectively) are readily available, they can be directly substituted into equation (C.25).

However, in many applications, this is not the case. If tangents are not available, they can be approximated using additional point or points before and after the control points to estimate the derivatives.

Several techniques are available to evaluate tangents based on additional points. The simplest is to use a three-point data set, two of which are control points. This technique results in equal tangents (i.e. $R_{0x} = R_{1x}$) which implies C^1 continuity:

$$R_i = R_{i-1} = \frac{1}{2}\left[\frac{(P_{i+1} - P_i)}{(t_{i+1} - t_i)} + \frac{(P_i - P_{i-1})}{(t_i - t_{i-1})}\right] \tag{C.26}$$

For a data set consisting of a number of points that is greater than three, the central difference formula in equation (C.26) can be used for all points except for one of the end points where either a forward difference or a backward difference formula can be used to estimate the first or last tangent, respectively. [Kochanek and Bartels, 1984] Use this technique to demonstrate their Kochanek-Bartel spline. Their spline is essentially a *hermite* interpolating polynomial.

A better estimate of tangents is achieved by using two additional points instead of one as illustrated in equation (C.27).

$$R_i = (1 - \tau)\left[\frac{P_{i+1} - P_{i-1}}{t_{i+1} - t_{i-1}}\right] \tag{C.27}$$

Splines that use equation (C.27) are called canonical splines. Note the appearance of the tension parameter ($\tau \in [-1,1]$) in the equation. The tension parameter controls the sharpness of the curve when it bends based on the position of the control points [Kochanek and Bartels, 1984]. When the value of the tension parameter is zero, the resulting spline is called Catmull–Rom spline**.**

Bias ($\beta \in [-1,1]$) is the second parameter that affects the direction and value of the

derivative. It can be used to control the direction of the path as it passes through a control point. Equation (C.27) is written assuming a zero bias and a uniform distribution of points. When the value of the bias differs from zero and the points are not evenly distributed, the general form in equation (C.28) replaces that in equation (C.27) :

$$R_i=(1-\tau)\left[(1+\beta)\frac{(P_{i+1}-P_i)}{t_{i+1}-t_i}+(1-\beta)\frac{(P_i-P_{i-1})}{t_i-t_{i-1}}\right] \tag{C.28}$$

Similarly, equation (C.29) presents the general form of equation (C.26) when the value of the bias deviates from zero.

$$R_i=R_{i-1}=\frac{1}{2}\left[(1+\beta)\frac{(P_{i+1}-P_i)}{(t_{i+1}-t_i)}+(1-\beta)\frac{(P_i-P_{i-1})}{(t_i-t_{i-1})}\right] \tag{C.29}$$

[Bourke, 2011] provided a C++ code representing a one-dimensional hermite interpolating polynomial. His code is presented in Appendix E.

APPENDIX D: APPROACH II 3-D VECTOR TRACKING AND MESH GENERATION EQUATIONS

This appendix details the vector tracking procedures that is used in Approach II for discontinuity resolution. The first section details how a vector is tracked during the simulation run. The second section details how a mesh is constructed at the discontinuity location once a discontinuity is reached. Although the discussion is illustrated using a 3D function, the approach is applicable to functions of any dimension.

D.1 Three-D Vector Tracking

Let us assume that at time t_o, the 3D function f initializes at x_o and y_o coordinates of their respective axes in a region bounding f_1. The resulting starting point is $P_o(x_o, y_o, z_o, f(x_o, y_o, z_o))$. Since $f(x_o, y_o, z_o)$ can be calculated at any $P(x, y, z)$, we do not need to track function values. As the simulation advances by one step to t_1, the coordinates of another point $P_1(x_1, y_1, z_1)$ are identified. The locations of these two points are sufficient to determine the trajectory vector v_1 that is accurate to time t_1 only. Using linear algebra notation, vector v_1 can be written as:

$$\vec{v}_1 = \vec{P}_o P_1 = \begin{bmatrix} x_1 - x_o \\ y_1 - y_o \\ z_1 - z_o \end{bmatrix} \tag{D.1}$$

Now, let us transform the conditional statement into a discontinuity plane. A plane can be uniquely identified through either:

1. a point inside the plane and a vector orthogonal to that plane,
2. or through three non-collinear points inside the plane. In this case the vector in case 1 is calculated using the three non-collinear points.

We will define the plane using the second case. To start, we need to locate arbitrary points $P_A(x_A,y_A,z_A)$, $P_B(x_B,y_B,z_B)$ and $P_C(x_C,y_C,z_C)$ located inside the discontinuity plane. We will demonstrate the procedure for the discontinuity plane cutting the x dimension. Since the plane is cutting the x dimension at $x=x_n$, the x-coordinates of the three points will take the value of x_n. The discontinuity plane is extending infinitely in all coordinates. This extension allows us to select arbitrary values for the y coordinates y_A ,y_B and y_C and the z coordinates z_A, z_B and z_C . So, the coordinates of the points become:

$$\begin{matrix} P_A\left(x_A,y_A,z_A\right) & P_A\left(x_A,y_A,z_A\right) \\ P_B\left(x_B,y_B,z_B\right) & P_B\left(x_B,y_B,z_B\right) \\ P_C\left(x_C,y_C,z_C\right) & P_C\left(x_C,y_C,z_C\right) \end{matrix} \tag{D.2}$$

A check for non-co-linearity needs to be performed before proceeding to the next step. If the points are identified as collinear, then another set of arbitrary values for y_A, y_B and y_C needs to be assumed and the above procedure is to be repeated. Once points pass the non-collinearity test, $\vec{v}_p$ that is orthogonal to the discontinuity plane is obtained via multiplying vectors $\vec{P}_A P_B$ with $\vec{P}_A P_C$ (or any similar combination) as vector cross product. Thus,

$$v_p=\vec{P}_A P_B x \vec{P}_A P_C=\begin{bmatrix} x_B-x_A \\ y_B-y_A \\ z_B-z_A \end{bmatrix} x \begin{bmatrix} x_C-x_A \\ y_C-y_A \\ z_C-z_A \end{bmatrix}=\begin{bmatrix} \left(y_B-y_A\right)\left(z_C-z_A\right)-\left(z_B-z_A\right)\left(y_C-y_A\right) \\ \left(z_B-z_A\right)\left(x_C-x_A\right)-\left(x_B-x_A\right)\left(z_C-z_A\right) \\ \left(x_B-x_A\right)\left(y_C-y_A\right)-\left(y_B-y_A\right)\left(x_C-x_A\right) \end{bmatrix}=\begin{bmatrix} a_{vp} \\ b_{vp} \\ c_{vp} \end{bmatrix} \tag{D.3}$$

Since the general equation of any plane passing through point $P_o(x_o, y_o, z_o)$ and orthogonal to $\vec{v}\begin{pmatrix} a \\ b \\ c \end{pmatrix}$ is:

$$a\left(x-x_o\right)+b\left(y-y_o\right)+c\left(z-z_o\right)=0\,, \tag{D.4}$$

we could easily formulate the equation of the discontinuity plane as one of the equations

in (D.5):

$$a_{vp}\left(x-x_A\right)+b_{vp}\left(y-y_A\right)+c_{vp}\left(z-z_A\right)=0 \tag{D.5a}$$

$$a_{vp}\left(x-x_B\right)+b_{vp}\left(y-y_B\right)+c_{vp}\left(z-z_B\right)=0 \tag{D.5b}$$

$$a_{vp}\left(x-x_C\right)+b_{vp}\left(y-y_C\right)+c_{vp}\left(z-z_C\right)=0 \tag{D.5c}$$

using points $P_A(x_A, y_A, z_A)$, $P_B(x_B, y_B, z_B)$ or $P_C(x_C, y_C, z_C)$ as an example.

Next, we need to find the intersection point of the line, directed by v_1 that is passing through P_o and P_1, with the discontinuity plane defined by equation (D.5). To do this, we need to write the equation for this line in the form:

$$x=x_o+\left(x_1-x_o\right)\tau \tag{D.6a}$$

$$y=y_o+\left(y_1-y_o\right)\tau \tag{D.6b}$$

$$z=z_o+\left(z_1-z_o\right)\tau \tag{D.6c}$$

Substituting (D.6) into (D.5), we get:

$$a_{vp}\left(x_o+\left(x_1-x_o\right)\tau-x_A\right)+b_{vp}\left(y_o+\left(y_1-y_o\right)\tau-y_A\right)+c_{vp}\left(z_o+\left(z_1-z_o\right)\tau-z_A\right)=0 \tag{D.7}$$

Equation (D.7) has only one unknown (τ). Solving for τ and substituting the resulting value into (D.6a), (D.6b) and (D.6c), we obtain the x, y, and z coordinates of the intersecting point between the line (vector) P_oP_1 with the discontinuity plane. Since the vector will intersect the plane at time t_n, we will call the intersection point $P_n(x_n, y_n, z_n)$. The discussion is illustrated in Figure D.1.

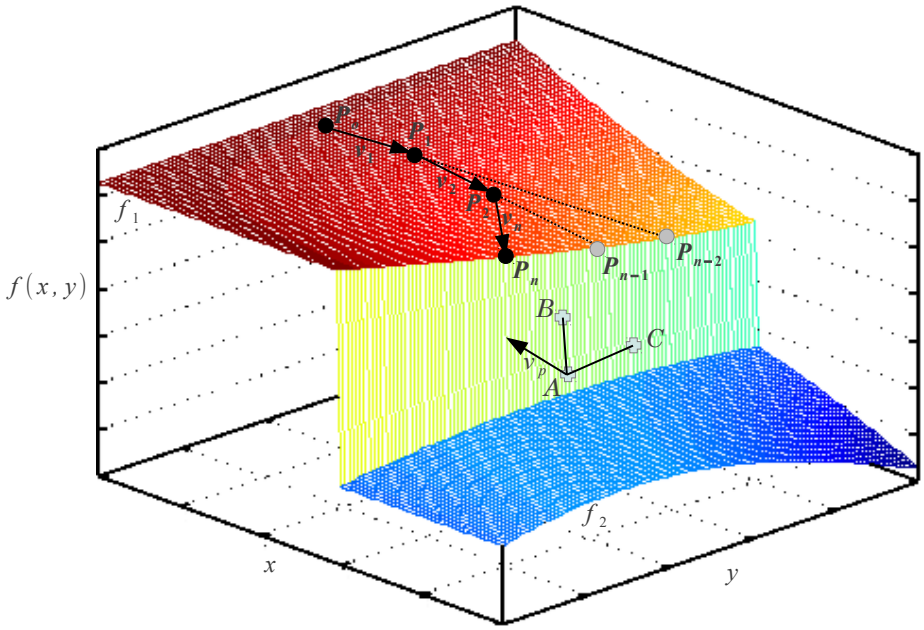


Figure D.1: Progression of $\vec{v}_i$ towards a discontinuity plane.

D.2 Mesh Generation Using Approach II

Next, we need to construct the coordinates of the 64-point interpolating polynomial. To do so, we will rely on the direction of the $\vec{P}_o P_1$ vector. The idea is to generate 4 planes that are parallel to the discontinuity dimension and separated by a distance h along the discontinuous dimension as illustrated in Figure 5.12b for an intersection at z plane. Since we assumed intersection at x-plane, the planes will be separated by a distance h_x. Hence, the x dimensions of the 4 discontinuous planes become: x_n, x_n+h_x, x_n+2h_x and x_n+3h_x if $\vec{v}_n$ is entering the overlap domain from the left end. If $\vec{v}_n$ is entering the overlap domain from the right end, the x dimensions of the 4 discontinuous planes become: x_n, x_n-h_x, x_n-2h_x and x_n-3h_x. Since we are aiming for a symmetrical distribution of control points around the $\vec{v}_n$ vector, we need to calculate the coordinates of the other dimensions (y and z) for the points lying on $\vec{v}_n$ vector and having the 4 x-coordinates mentioned above. To do so, we will calculate a new τ for each of the newly generated x-values:

$$\tau_{x_n+1h_x} = \frac{(x_n+1h_x - x_o)}{(x_1 - x_o)} \quad (a)$$
$$\tau_{x_n+2h_x} = \frac{(x_n+2h_x - x_o)}{(x_1 - x_o)} \quad (b)$$
$$\tau_{x_n+3h_x} = \frac{(x_n+3h_x - x_o)}{(x_1 - x_o)} \quad (c)$$

or

$$\tau_{x_n-1h_x} = \frac{(x_n-1h_x - x_o)}{(x_1 - x_o)} \quad (a)$$
$$\tau_{x_n-2h_x} = \frac{(x_n-2h_x - x_o)}{(x_1 - x_o)} \quad (b)$$
$$\tau_{x_n-3h_x} = \frac{(x_n-3h_x - x_o)}{(x_1 - x_o)} \quad (c)$$

(D.8)

Next we substitute the newly obtained τ values into equation D.6 to get the other coordinates of the points at which $\vec{v}_n$ intersects with other planes. Last, we construct a mesh of sixteen points surrounding each of the four newly calculated points on $\vec{v}_n$. Figure D.2 illustrates the concept when applied to 2D discontinuous functions.

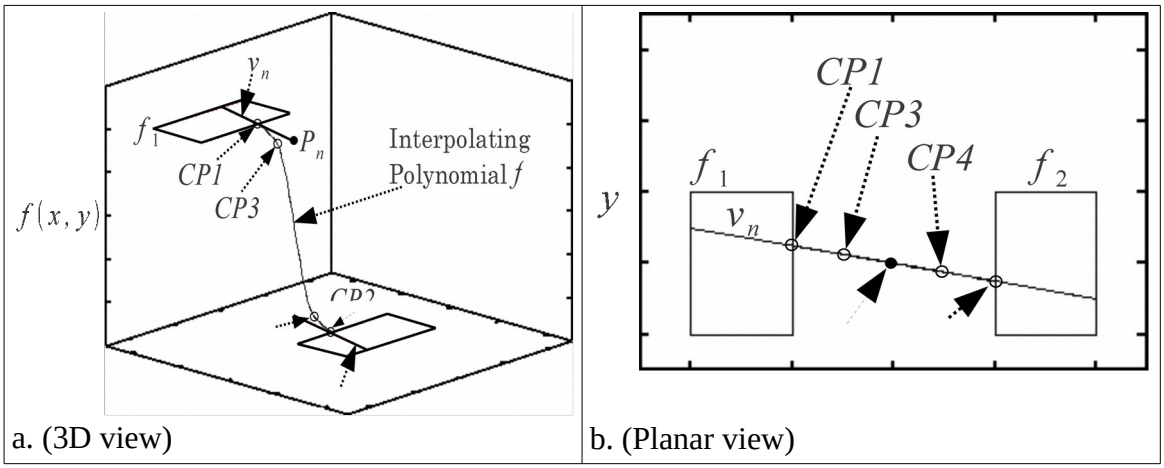


Figure D.2: The behaviour of a 2D interpolating polynomial demonstrating the continuity of the polynomial along the continuous coordinate while interpolating along the discontinuous axis. (*CP* = Control Point)

APPENDIX E: A BRIEF ON THE DEVELOPED CODE

This appendix is meant to serve as a starting point for researchers who would like to have a look at the developed code, copy it or copy some parts of it, mimic it in another environment or even develop a better code. Thus, I don't claim perfection in the written code. I am just presenting a working code.

Some of the concepts are verified using C++. Others are verified using GNU Octave. I will write a brief introductory to each code before presenting it.

E.1 One-Dimensional *Hermite* interpolation

Below is the C++ implementation of the one-dimensional *hermite* interpolation polynomial that is presented by [Bourke, 2011]. The code uses four function values to interpolate between two points. Thus, the function only interpolates between y1 and y2. Nevertheless, it uses y0 and y3 to determine the appropriate slope of the interpolating polynomial at the points y1, y2 and any intermediate point within [y1,y2]. Note that since the x-coordinates of the points are equally spaced, their absolute values are irrelevant to the interpolating function. Instead, the fraction $mu \in [0,1]$ is used to reflect the x position of the point at which the interpolating polynomial should report its y value. The code also reflects how tension and bias parameters (explained in Appendix C) are used.

```
double HermiteInterpolate( double y0,double y1, double y2,double y3, double mu,
                   double tension, double bias) {

  double m0,m1,mu2,mu3, a0,a1,a2,a3;
  mu2 = mu * mu;
  mu3 = mu2 * mu;
  m0  = (y1-y0)*(1+bias)*(1-tension)/2;
  m0 += (y2-y1)*(1-bias)*(1-tension)/2;
  m1  = (y2-y1)*(1+bias)*(1-tension)/2;
  m1 += (y3-y2)*(1-bias)*(1-tension)/2;
  a0 =  2*mu3 - 3*mu2 + 1;
  a1 =    mu3 - 2*mu2 + mu;
  a2 =    mu3 -   mu2;
  a3 = -2*mu3 + 3*mu2;
  return(a0*y1+a1*m0+a2*m1+a3*y2);   }
```

Since the above function is *x*-dimension independent, a separate function should be written to calculate the value of *mu*. An example implementation of such function is presented below:

```
double interpolate(double *xv, double *yv, double x, double tension, double bias) {

 // mu is the scaled location of point x relative to the two bounding points.
 double mu;

 int i = -1;
 while (xv[++i] <= x);
 if (xv[i-1] == x)
        --i;
 else
        i-=2;

 mu = (x - xv[i+1]) / fabs(xv[i+2] - xv[i+1]);

 return hermite_interpolate( yv[i], yv[i+1], yv[i+2], yv[i+3], mu, tension, bias);  }
```

Note that two vectors of six points (**xv* and **yv*) are passed to the above function along with the point *x* at which the value of the interpolation polynomial is to be computed. The function searches for the location of *x* within the provided **xv* vector, calculates *mu* and passes four of the six interpolation points to the *HermiteInterpolate* function. The conditional statement is used to insure using the same set of interpolation points even when x is at the border of the interpolation interval.

E.2 Two-Dimensional Interpolation

[Breeuwsma, 2011] presented a general C++ and Java codes for multidimensional interpolation that can be used in conjunction with any one-dimensional interpolation method. His C++ code is presented below:

```
double bi_interpolate ( double *xv,  double *yv, zm[MAX_POINTS][MAX_POINTS],
        double x,       double y, double tension, double bias ) {
        double arr[MAX_POINTS];

        for (int i=0; i<MAX_POINTS;++i)
                arr[i] = interpolate(yv, zm[i], y, tension, bias);

        return interpolate(xv, arr, x, tension, bias); }
```

Looking at the interpolation mesh as a squared one (zm), the idea behind the code is to interpolate each of the six-point row vectors in the *x*-dimension to produce the six-point interpolation vector for the *y*-dimension. Thus, the two-dimensional interpolation is treated as a double one-dimensional interpolation. The code can easily be extended to cover multi-dimensional interpolation by nesting additional *for* loops or using dynamic arrays.

E.3 Past Interpolation to Determine the Value of the missing *hermite* Point when Regularizing Boundary Conditions

Past Interpolation is to find the value of the passed *hermite* point is discussed in section 5.1.5. A spline interpolating polynomial is used to perform the interpolation. The spline interpolation code is taken from the GNU Scientific C++ Library [GSL, 2011]. The calling function is presented below.

```
double past_interpolate (int n, double t[], double y[], double tval, double *yval) {
        int         ibcbeg = 0, ibcend = 0;

        double  ybcbeg=0, ybcend=0, ypval, yppval, *ypp;

        ypp = spline_cubic_set ( n, t, y, ibcbeg,  ybcbeg, ibcend, ybcend );

        *yval = spline_cubic_val ( n, t, y, ypp, tval, &ypval, &yppval );

        return *yval;       }
```

E.4 Regularizing Initial and Boundary Conditions

The code below is used to regularize velocity initial and concentration boundary conditions.

```
double present_interpolate (double *inputs) {

 double
        g                       = inputs[0],
        h                       = inputs[1],
        Tau                     = inputs[2],
        norm_dip                = inputs[3],        // dip-parameter
        tension                 = inputs[4],
        bias                    = inputs[5],
        t                       = inputs[6],        // current simulation time
        initial_bound           = inputs[7],
        final_bound             = inputs[8],
        // Past interpolated value (Calculated using past_interpolate function)
        initial_bound_mh        = inputs[9],
        // Magnitude of Jump
        AB                      = fabs(final_bound-initial_bound);

        double          tv[MAX_POINTS],   // time interpolation vector
                        yv[MAX_POINTS];   // velocity or BC interpolation vector

        // initializing t interpolation vector
        tv[0] = (g-h)/Tau;      / (tp-h)/tau/
        for (int i=1; i<MAX_POINTS; ++i)
                tv[i] = ( Tau*tv[i-1]+h )/Tau;

        // initializing boundary velocity interpolation vector
        yv[0] = initial_bound_mh;                       // @(tp-h)/Tau
        yv[1] = initial_bound;                          // @(tp)/Tau
        if (initial_bound > final_bound) {
                yv[2] = initial_bound-norm_dip*AB;      // @(tp+h)/Tau
                yv[3] = final_bound+norm_dip*AB;        // @(tp+2h)/Tau
        }
        else {
                yv[2] = initial_bound+norm_dip*AB;      // @(tp+h)/Tau
                yv[3] = final_bound-norm_dip*AB;        // @(tp+2h)/Tau
        }
        yv[4] = final_bound;                            // @(tp+3h)/Tau
        yv[5] = final_bound;                            // @(tp+4h)/Tau

        return interpolate(tv, yv,  t, tension, bias);          }
```

E.5 Generating a Two-Dimensional Interpolation Mesh based on Approach II to Discontinuity Resolution

The code below is a C++ implementation to the concepts provided in Appendix C.

```
void mesh_grid(dim_info *dim, double *inputs, int discont_dim,
        double norm_dip,        // Normalized Dip [0-->1]
        double zpm[MAX_POINTS][MAX_POINTS],
        double f (int, double, double, double*) ) {
        // Calculating interp_loc for discontinuous dimension
        double shift = - 0.5*dim[discont_dim].h*(MAX_POINTS-1);
        for (int j=0; j<MAX_POINTS;++j) {
                dim[discont_dim].interp_loc[j] = dim[discont_dim].v[2] + shift;
                shift+=dim[discont_dim].h;
        }

         // Calculating interpolation location for other dimensions
        double slope;
        for (int i=0; i<DIMENSIONS;++i)
                if ( i!=discont_dim ) {
                        shift = -0.5*dim[i].h*(MAX_POINTS-1);
                        for (int j=0; j<MAX_POINTS; ++j) {
                        slope = calc_slope(dim[discont_dim].v[0], dim[discont_dim].v[1],
                                dim[i].v[0], dim[i].v[1]);
                        dim[i].interp_loc[j] = slope*(dim[discont_dim].interp_loc[j] -
                                dim[discont_dim].v[0] ) + dim[i].v[0] + shift;
                        shift+=dim[i].h;
                }
        }

        // Generating z mesh points
        for (int i=0; i<MAX_POINTS;++i)
                for (int j=0; j<MAX_POINTS;++j) {
                        if (in_range(dim[0].interp_loc[i], dim[1].interp_loc[j], dim[0].v[2],
                                dim[1].v[2], 1) )
                                zpm[i][j] = f(1, dim[0].interp_loc[i], dim[1].interp_loc[j], inputs);
                        else
                                zpm[i][j] = f(2, dim[0].interp_loc[i], dim[1].interp_loc[j], inputs);
        }

        // Dipping intermediate points using p parameter
        int index1, index2, diprow1 = MAX_POINTS/2-1, diprow2 = diprow1+1;
        double
        A = f(1, dim[0].v[2], dim[1].v[2], inputs),
        B = f(2, dim[0].v[2], dim[1].v[2], inputs),
        AB = A-B;

        for (int i=0; i<MAX_POINTS; ++i) {
                // dipping first row/column
                if (dim[0].discontinuous) { index1 = diprow1; index2 = i; }
                else { index1 = i; index2 = diprow1; }

        if ( in_range(dim[0].interp_loc[index1], dim[1].interp_loc[index2], dim[0].v[2], dim[1].v[2],
```

```
                    1) ) {
                    if (A < B)
                            zpm[index1][index2] -= norm_dip*AB;
                    else
                            zpm[index1][index2] += norm_dip*AB;
            }
            else if ( in_range(dim[0].interp_loc[index1], dim[1].interp_loc[index2], dim[0].v[2],
                    dim[1].v[2], 2) ) {
                    if (A < B)
                            zpm[index1][index2] += norm_dip*AB;
                    else
                            zpm[index1][index2] -= norm_dip*AB;
            }

             // dipping second row/column
             if (dim[0].discontinuous) { index1 = diprow2; index2 = i; }
            else { index1 = i; index2 = diprow2; }

            if ( in_range(dim[0].interp_loc[index1], dim[1].interp_loc[index2], dim[0].v[2], dim[1].v[2],
                    1) ) {
                    if (A > B)
                            zpm[index1][index2] += norm_dip*AB;
                    else
                            zpm[index1][index2] -= norm_dip*AB;
            }
            else if ( in_range(dim[0].interp_loc[index1], dim[1].interp_loc[index2], dim[0].v[2],
                    dim[1].v[2], 2) ) {
                    if (A > B)
                            zpm[index1][index2] -= norm_dip*AB;
                    else
                            zpm[index1][index2] += norm_dip*AB;
                    }
            }
}
```

E.6 Determining the location of the cutting planes for *Nu=f(Re,Pr)*

To determine the best location for the cutting planes, corresponding to *minimum(e)*, I used a simple sequential search algorithm. This algorithm is used only to prove the concept. For practical applications, a faster and more rigorous search algorithm should be used.

```
void get_cut_plains(    double *Re1_limit,
                        double *Re2_limit,
                        double *Pr1_limit,
                        double *Pr2_limit,
                        double *x_plain,
                        double *y_plain,
                        double *inputs,
```

```
                        double f(int , double , double , double* ) ) {

        double  x=Re1_limit[1],
        y=Pr1_limit[1],
        min_x=x,
        min_y=y;

        // initializing error function to one of the corners of the overlap domain
        double  error = fabs( f(1, x, y, inputs) - f(2, x, y, inputs) ), min_error = error, step = 0.1;

        // Searching Re-Pr space for an optimum jump location
        for ( x=Re2_limit[0]; x<Re1_limit[1]; x+=step) {
                for ( y=Pr2_limit[0]; y<Pr1_limit[1]; y+=step) {
                        error = fabs( f(1, x, y, inputs) - f(2, x, y, inputs) );
                        if (error < min_error) {
                                min_error = error;
                                min_x = x;
                                min_y = y;
                        }
                }
        }

        *x_plain = min_x;
        *y_plain = min_y;
}
```

E.7 The regularized *Nu=f(Re,Pr)* Function

The below code represents the regularized Nu=f(Re,Pr) function I used to interpolate between the values of the heat transfer coefficient corresponding to laminar and turbulent flow regimes. In practical implementations, this function should be generated by the language compiler. Note the use of C++ static function to track the first entry to the overlap region. This detection facilitates a one-time generation of the interpolation mesh. Also, note how the composite function well-encapsulates the boundaries of the its sub-functions leading to the “illegal extrapolation” message if the simulation crosses the boundaries that are set by the domains of the sub-functions.

```
double Nud_interp (double *inputs) {
        static bool first_entry_to_Nud_interp=true, first_entry = true;

        static double zpm[MAX_POINTS][MAX_POINTS];

        // Renolds limits per function
        double  Re1_limit[] = { inputs[2], inputs[3]}, Re2_limit[] = {inputs[4], inputs[5] },
```

```
// Prandtl limits for each function
Pr1_limit[] = {inputs[6], inputs[7]}, Pr2_limit[] = {inputs[8], inputs[9]},

// Current Values of Re and Pr
Re = inputs[15], Pr = inputs[16];
double  tension = inputs[10], bias = inputs[11], norm_dip = inputs[13];
int discont_dim = inputs[14]-1;

// declaring and initializing dim structure
static dim_info dim[DIMENSIONS];
if (first_entry_to_Nud_interp) {
        fisrt_entry_to_Nud_interp = false;
        for (int i=0; i<DIMENSIONS; ++i) {
                // all dimensions should be continuous except one
                dim[i].discontinuous = false;
                 for (int j=0; j<BOUNDARIES;++j)
                        for (int k=0; k<PROPERTIES;++k)
                                dim[i].boundary[j][k] = 0;
                dim[i].h = 0;
                dim[i].cut_plain = 0;
                for (int j=0; j<MAX_POINTS;++j)
                        dim[i].interp_loc[j]=0;
        }

        get_cut_plains(Re1_limit, Re2_limit, Pr1_limit, Pr2_limit, &dim[0].cut_plain,
        &dim[1].cut_plain, inputs, Nud);

        for (int i=0; i<DIMENSIONS; ++i) {
                // Initial values of h in each dimension
                dim[i].h = inputs[12];

                // Assigning discontinuity
                (i == discont_dim ) ? dim[i].discontinuous = true : dim[i].discontinuous =
                        false;
        }

        // copying respective arrays' limits
        // for discontinuous dimension, check against absolute low and high of both
                functions
        dim[0].boundary[0][0] = inputs[2];          // Re2_low_limit
        dim[0].boundary[1][0] = inputs[5];          // Re1_high_limit
        // for continuous dimension, check against overlap violations.
        dim[1].boundary[0][0] = inputs[8];          // Pr2_low_limit
        dim[1].boundary[1][0] = inputs[7];          // Pr1_high_limit
}

// updating moving vector
for (int i=0; i<DIMENSIONS;++i) {
        // pushing old v vector values to the back of the array
        for (int j=0; j<VECTOR_LENGTH-2;++j)
        dim[i].v[j] = dim[i].v[j+1];
        // updating the v vector with new array values
        dim[i].v[VECTOR_LENGTH-2] = inputs[i+15];
}

double h = dim[ discont_dim ].h,
interp_span = (MAX_POINTS-1-2)*h;
```

```
        // Laminar
        if ( (Re > Re1_limit[0]) && (Re < dim[0].cut_plain - 0.5*interp_span) ) {

                first_entry = true;
                // For uniform heat flux (Taken from Holman, p. 291)
                return Nud(1, Re, Pr, inputs);
        }
        // Interpolation Region
        else if ( fabs( Re - dim[0].cut_plain ) <= 0.5*interp_span)  {
                // Generating mesh points at first entry only
                // ensuring that mesh generation is executed only once per entry to interpolation
                        region
                if (first_entry) {
                        first_entry = false;
                        // locating intersection point of moving vector with cutting plain
                        find_i_point(dim, discont_dim);
                        // resizing (reducing) h if necessary
                        get_gaps(dim, discont_dim);
                        // generating interpolation matrix
                        mesh_grid(dim, inputs, discont_dim, norm_dip, zpm, Nud);
                }
                // Interpolating
                return bi_interpolate ( dim[0].interp_loc, dim[1].interp_loc, zpm, Re, Pr, tension,
                        bias );
        }
        // Turbulent
        else if ( (Re < Re2_limit[1]) && (Re > dim[0].cut_plain + 0.5*interp_span) ) {
        /*      Gnielinski correlation: Gnielinski is a correlation for turbulent flow in tube.
                taken from CRC Handbook of thermal engineering ( p. 3-49) */
                return Nud(2, Re, Pr, inputs);
        }
        else
                cout << "Illegal Extrapolation\n";
}
```

E.8 The discretized *Nu=f(Re,Pr)* Function

The code in this section represents the discretized *Nu=f(Re,Pr)* function that is written by the modeller. I coded each function separately and then coded the composite function as a separate one calling either laminar or turbulent functions depending on the domain. The composite discretized function is called by the regularized one to determine the values of the composite function outside the interpolation region.

```
// Nud in Laminar Regime
double NudL(double Re, double Pr, double *param) {

 return 4.36;
}
```

```
// Nud in Turbulent Regime
double NudT(double Re, double Pr, double *param) {
        double  Lc = param[0],
                        dci = param[1],
                        f = pow( (1.58*log(Re) - 3.28),-2);

        return ( (0.5*f)*(Re-1000.0)*Pr ) / ( 1+12.7*pow(0.5*f,0.5)*(pow(Pr,2.0/3)-1 ) ) * ( 1 + pow
                (dci/Lc, 2.0/3) );
}

double Nud(int domain, double Re, double Pr, double *param) {
        switch (domain) {
                case 1:  {
                        // Laminar regime
                        return NudL(Re,Pr, param);
                        break;
                }
                default: {
                        // Turbulent regime [ case 2 ]
                        return NudT(Re,Pr, param);
                }
        }
}
```

Nomenclature

a_p	Specific area of pellet	h_p	Particle heat transfer coefficient
a_{we}	Wall external specific area	h_{we}	Wall external heat transfer coefficient
a_{wi}	Wall internal specific area	h_{wi}	Wall internal heat transfer coefficient
c_i	Concentration of component i	k	Thermal conductivity
$<c_i>$	Average concentration of component i	k_e	External mass transfer coefficient
C_{pg}	Fluid heat capacity	k_{gl}	Overall mass transfer coefficient
C_{ps}	Solid heat capacity	$k_{i\text{-}C5}$	Reverse reaction constant
C_{pw}	Wall heat capacity	$k_{n\text{-}C5}$	Forward reaction constant
C_t	Total gas phase concentration	k_w	Wall thermal conductivity
$C_{t,max}$	Maximum total concentration	$K_{i,ads}$	Adsorption equilibrium isotherm constant for component i
C_T	Dimensionless total concentration	K_L	Axial thermal conductivity
d_c	Column diameter	L	Column Length (cm)
d_p	Particle diameter	L_R	Reactor length (cm)
D	Ideal diffusivity	m	Mass (grams)
D_{eff}	Effective diffusivity	$\dot{m}$	Mass flowrate (grams/second)
D_k	Knudsen diffusion coefficient	M	Molecular weight
D_L	Axial mass dispersion coefficient	M_{dp}	De-pressurization rate (1/seconds)
D_M	Molecular diffusion	M_p	Pressurization rate (1/seconds)
D_z	Axial dispersion coefficient	n	Polynomial Order or n^{th} derivative
E_{max}	Maximum allowed error in a single integration step	$n\text{-}C_5$	Normal pentane
f	Friction factor	$n\text{-}C_6$	Normal hexane
g	Optimum transition point between two discontinuous functions	N_E	Number of equalization steps
h	Integration step or distance between mesh control points	n_F	Number of feed moles

n_i	[Nitta et al, 1984] Isotherm exponent
n_P	Number of purge moles
Nu_d	Nusselt number based on vessel diameter
P	Pressure (bar)
$\overline{P}$	Dimensionless pressure
Pe	Peclet number
P_{eq}	Equalization pressure
P_F	Feed Pressure
P_P	PSA column purge pressure
T_F	Feed Temperature
T_g	Fluid Temperature
T_P	Purge Temperature
u	velocity
v	Dimensionless velocity
v_i	Dimensionless velocity at time indexed interval i
w	Regularization interval
x	Dimensionless distance
$\bar{x}$	Independent variable scaled value
x_i	Value of the independent variable at i^{th} integration interval
$X_{n\text{-}C5}$	N-pentane conversion
x_O	Initial condition of independent variable
y_O	Initial condition of dependent variable
y_b	Value of dependent variable at end of integration step
$y_{i,j}$	The value of the j^{th} element in the y vector at the i^{th} integration step
$\bar{y}$	Dependent variable scaled value
z	Absolute axial distance (cm)

Greek Letters

α	Ratio between two equilibrium constants
α_A	Equilibrium constant of strongly adsorbed component
α_B	Equilibrium constant of weakly adsorbed component
Δx_i	Change in independent variable x at i^{th} integration step
$\Delta y_{i,j}$	Change in the j^{th} value of the dependent vector y at i^{th} integration step
$-\Delta H$	Component i heat of adsorption
ϵ	Maximum acceptable error in an integration step
ε_b	Bed void
η	Pure component Dynamic viscosity
η_m	Mixture Dynamic viscosity
λ	Fluid pure component thermal conductivity

λ_m	Fluid mixture thermal conductivity
μ	Dynamic viscosity
μ_w	Dynamic viscosity at wall
Ω_D	Collision integral
ρ_g	Fluid density
ρ_s	Solid (adsorbent) density
ρ_w	Vessel wall density
σ_{12}	Collission diameter
τ	Dimensionless time or tortuosity
τ_p	Dimensionless pressurization time
θ_i	Surface coverage of component *I*