Analysis of Heteroazeotropic Systems

by

John Eugene Tolsma

Submitted to the Department of Chemical Engineering
in partial fulfillment of the requirements for the degree of

Doctor of Philosophy in Chemical Engineering

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

June 1999

© Massachusetts Institute of Technology 1999. All rights reserved.

Author .....
Department of Chemical Engineering
April 29, 1999

Certified by .....
Paul I. Barton
Assistant Professor of Chemical Engineering
Thesis Supervisor

Accepted by .....
Robert Cohen
St. Laurent Professor of Chemical Engineering
Chairman, Department Committee on Graduate Students

Analysis of Heteroazeotropic Systems

by

John Eugene Tolsma

Submitted to the Department of Chemical Engineering
on April 29, 1999, in partial fulfillment of the
requirements for the degree of
Doctor of Philosophy in Chemical Engineering

Abstract

Separation processes are used extensively in the chemical process industries and by far the most common of these is distillation. Although several alternative strategies have been developed, distillation will likely remain dominant particularly for the large-scale separation of non-ideal liquid mixtures. A topic of particular interest in recent years has been heterogeneous azeotropic distillation or heteroazeotropic distillation. This technique is commonly employed to separate azeotropic mixtures by introducing a heterogeneous entrainer that causes liquid-liquid phase separation. Although the design and simulation of heteroazeotropic systems is far more complicated than its homogeneous counterpart, heteroazeotropic distillation is often preferred due to the ease of recovery of the entrainer and the crossing of distillation boundaries due to the liquid-liquid phase split in the decanter.

The topic of this thesis is the analysis of heteroazeotropic systems. Specifically, an algorithm has been developed which, under reasonable assumptions, will compute all homogeneous and heterogeneous azeotropes present in a multicomponent mixture predicted by the phase equilibrium model employed. The approach is independent of both the particular representation of the nonideality of the mixture and the topology of the liquid-liquid region. Furthermore, the approach can be readily extended to handle any number of liquid and/or solid phases in equilibrium. Moreover, the heteroazeotrope finding algorithm can be extended to explore the phase equilibrium structure of a multicomponent mixture under system and/or property model parameter variation, including the detection of incipient homogeneous and heterogeneous azeotopes and the determination of the bifurcation values of the parameters where they appear, disappear, or switch between each other. The ability to predict the incipient homogeneous and heterogeneous azeotropes that may appear under different conditions or property parameter values can be incorporated into design algorithms to expand the number of alternative designs. Furthermore, the ability to systematically and efficiently explore the phase equilibrium structure is a valuable tool when fitting property model parameters, allowing the experimentalist to rapidly explore the capabilities and limitations of the phase equilibirum model.

The techniques mentioned above are useful when analyzing heteroazetropic sys-

tems for design purposes. The second product of this thesis improves the efficiency of the actual simulation of the heteroazeotropic system (or any system for that matter). Specifically, a new class of automatic differentiation methods, known as 'subgraph reduction methods', have been developed that offer substantial improvement over existing techniques both in the increase in speed of the derivative evaluation and the reduction in memory required to store and evaluate the Jacobian matrix of a sparse system of equations. Furthermore, a variant of the subgraph reduction approach has been custom-tailored for use within an interpretive simulator architecture that dramatically increases speed and reduces memory requirements compared to other techniques commonly employed in this environment.

Thesis Supervisor: Paul I. Barton
Title: Assistant Professor of Chemical Engineering

To My Family

Acknowledgments

A doctoral thesis is very rarely an individual effort as was the case with this endeavor. Unfortunately, a single page acknowledging those who have contributed to this thesis can not do justice.

I would first like to thank my thesis advisor, Professor Paul I. Barton, whose guidance, advice, and support made this all possible. The countless brainstorming sessions in his office led to the exploration of new ideas every time I thought I had reached an impasse. It has been a true pleasure working with Paul over the past five years.

I'd also like to thank all the members of Paul's research group, Berit Ahmad, Russell Allgor, David Collins, Bill Feehery, Arvind Mallik, Taeshin Park, and Wade Martinson, for their support and ability to put up with me, particulary during the writing of this thesis. It was also a pleasure working with Santos Galan and Lars Kruel. I'd especially like to thank David Collins who's inquisitive demeanor also helped me to explore new avenues in this research.

My thanks also goes out to my good friends in Boston, Talid Sinno, Shahin Ali, Howard Covert, and Scott Dildyne who helped to make my time here at MIT enjoyable as well as fruitful. I'd like to thank Ann Park, who was a wonderful T.A. and an even better friend throughout my entire time here at MIT. I also thank my good buddies in California, Hop and Bao Nguyen, who have remained dear friends even with a continent separating us. I'd especially like to thank Sophia Chong who over the last few months during the writing of this thesis has provided invaluable support and friendship.

I'd especially like to thank my family, to whom this thesis is dedicated. They provided constant support and love, particularly during times when I was most discouraged. My visits home did more to restore my motivation and determination than they could imagine or that I could ever hope to describe here.

Finally, I'd like to thank the Department of Energy for funding.

Contents

List of Figures

List of Tables

Part I

Analysis of Heteroazeotropic Systems

Chapter 1

Heteroazeotropic Distillation

This thesis is divided into three parts. The first part, Analysis of Heteroazeotropic Systems (chapters one through four), presents topics associated with the analysis of heteroazeotropic systems. In particular, a new homotopy continuation based approach for the computation of the heteroazeotropes present in a multicomponent mixture and the efficient analysis of changes in phase equilibrium structure under system and/or property model parameter variation is discussed. The second part, Computational Differentiation (chapters five through eight), presents topics associated with the simulation of heteroazeotropic systems, in particular, the generation of analytical derivatives. This thesis concludes with a chapter describing a technique for the optimization of problems exhibiting multiple local optima, based on the combined application of computational differentiation and homotopy continuation.

1.1 Overview

Separation processes are used extensively in the chemical process industries. Virtually every chemical process involves the separation of products from byproducts and unreacted raw materials, the recovery of solvent from waste streams, and possibly the purification of feed stocks prior to processing. By far the most common of these processes is distillation, and although many new technologies are being developed, distillation is likely to remain dominant, especially for the large-scale separation of

thermodynamically nonideal liquid mixtures. This argument in convincingly presented by Fair [40].

Azeotropy is the condition where a vapor and liquid in equilibrium have the same composition. Azeotropes are characterized by extrema in the equilibrium surfaces of the mixture. Figure 1-1 contains a schematic of a Txy-diagram for a binary mixture exhibiting a maximum boiling azeotrope. As will be shown later in this chapter, the

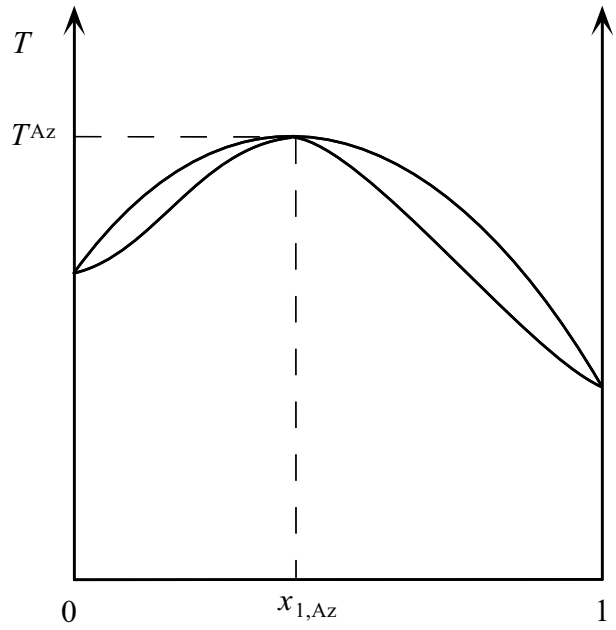

Figure 1-1: Schematic of a binary mixture exhibiting a maximum boiling azeotrope.

presence of azeotropes effectively divides the composition space into different regions characterized by the separations possible with distillation.

A topic of particular interest in recent years has been heterogeneous azeotropic distillation or heteroazeotropic distillation. This technique is commonly employed to separate azeotropic mixtures by introducing a heterogeneous entrainer that causes liquid-liquid phase separation. Heteroazeotropy is the condition where the temperature, pressure, and overall liquid composition of a mixture of two or more immiscible liquid phases is equal to that of the equilibrium vapor phase. Heteroazeotropy is characterized by strong positive deviations from Raoult's law and occurs when the azeotropic point on the vapor-liquid equilibrium surface intersects the multiple liquid region. Figure 1-2 contains a schematic of a Txy-diagram for a binary mixture

exhibiting a heteroazeotrope. Heteroazeotropic distillation is often preferred over the

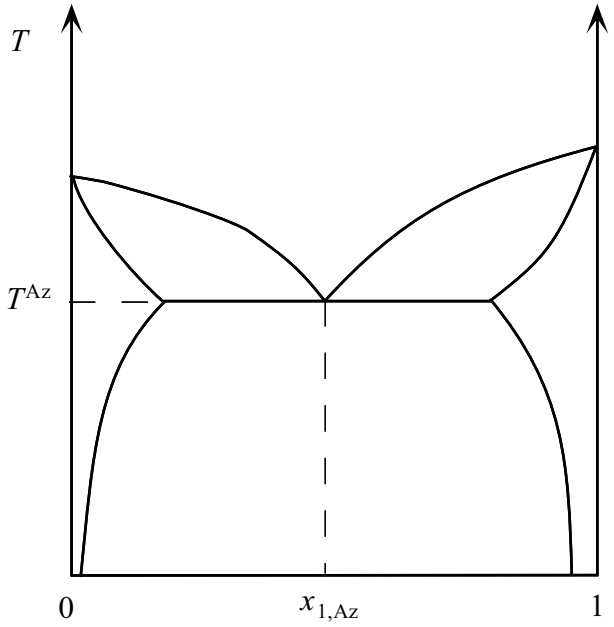

Figure 1-2: Schematic of a binary mixture exhibiting a heteroazeotrope.

more traditional homogeneous distillation due the ease of recovery of the entrainer and the crossing of distillation boundaries due to the liquid-liquid phase split in the decanter (see Figure 1-5). Although heteroazeotropic systems are primarily studied with regard to methods for separating azeotropic mixtures, these systems are also important from the viewpoint of separation system integration [3] within a chemical plant where various process streams may form heterogeneous liquid mixtures when combined, and also from the viewpoint of design and simulation of heterogeneous reactive distillation systems.

This chapter discusses some of the key issues associated with the analysis, design, and simulation of heteroazeotropic systems.

1.1.1 Computation of Homogeneous and Heterogeneous Azeotropes

Obviously an important task when analyzing homogeneous and heterogeneous azeotropic systems is the a priori determination of the homogeneous and heterogeneous

azeotropes present in the mixture of interest. Several approaches have been developed in the past for the computation of homogeneous azeotropes. A few of them are discussed below.

Teja and Rowlinson [111] developed an algorithm based on corresponding states and an equation of state to compute the binary azeotropes present in a mixture. Given a binary mixture, the following equations are satisfied at the azeotropic conditions:

$$A^v(V) - A^l(V) = 0 \tag{1.1}$$

$$A^v(x_1) - A^l(x_1) = 0 \tag{1.2}$$

$$A^v + x_2 A^v(x_1) - V^v A^v(V) - A^l - x_2 A^l(x_1) + V^l A^l(V) = 0, \tag{1.3}$$

where A^v and A^l are the Helmholtz free energy of the vapor and liquid phases, respectively, V, V^v, and V^l are the total molar volume, the vapor molar volume, and the liquid molar volume, respectively, and (x_1, x_2) is the composition of the liquid and vapor phase, equal at the azeotrope. The authors use Powell's method [90] to minimize the sum of the squares of the right-hand-side of equations (1.1)-(1.3) with respect to the partial molar volumes of each phase, V^v and V^l, and composition, $x_2 = x_2^v = x_2^l$. At constant temperature and volume, the azeotrope corresponds to a global minimum of the Helmholtz free energy of the mixture, which is not guaranteed using a local search strategy such as Powell's method. Therefore, solutions obtained will have to be checked for stability (see following section).

Wang and Whiting extended the work of Teja and Rowlinson to multicomponent mixtures [116]. The authors compute azeotropes at constant pressure (or temperature) using a nested iteration. The outer iteration adjusts the composition by performing a secant update with the following system of equations,

$$\ln\left(\hat{\phi}_i^v / \hat{\phi}_i^l\right) = 0 \qquad i = 1, \ldots, n \tag{1.4}$$

where $\hat{\phi}_i^v$ and $\hat{\phi}_i^l$ are the mixture fugacity coefficients of the vapor and liquid phases, respectively, for an n component mixture. The inner iteration adjusts temperature

(or pressure) by performing a secant update with the following equation,

$$A^v - A^l = P(V^l - V^v). \tag{1.5}$$

Equation (1.5) is simply an expression for the equality of the Gibbs energy between the vapor and liquid phases. This approach has been applied successfully to several binary and one ternary mixture. Like the approach of Teja and Rowlinson, the solutions obtained satisfy only the necessary conditions for azeotropy and must be checked for stability. Although the approach can be applied to multicomponent mixtures, there are no guarantees that all azeotropes, if any, will be computed.

An approach developed by Fidkowski *et al.*[45] computes the homogeneous azeotropes present in a multicomponent mixture using homotopy continuation. Higher dimensional azeotropes are obtained through a series of bifurcations from lower dimensional homotopy branches. This approach provides the basis for an algorithm developed in this thesis for the computation of heteroazeotropes and is thus described in more detail in the following chapter. Fidkowski *et al.* conjecture that all azeotropes present in a multicomponent mixture will be computed but present no proof. A detailed analysis in this thesis provides conditions under which all azeotropes will be computed using this method. Implementation improvements also result from this analysis.

A second approach guaranteeing the computation of all homogeneous azeotropes has been developed by Harding *et al.* [55]. The authors enclose all solutions to the necessary conditions for azeotropy using global optimization. The use of convex underestimators within a branch and bound framework partitions the search space into rectangles of decreasing size containing the solutions. In this work, the vapor phase is treated as an ideal gas and the liquid phase nonideality is modeled with activity coefficients computed through the NRTL, UNIQUAC, and UNIFAC equations. These activity coefficient models satisfy the convexity requirements of the global optimization algorithm employed. By employing the αBB-method [2], this method can in principle be extended to any twice continuously differentiable model for phase equilibrium. As with Fidkowski's method, this approach only computes solutions

satisfying the necessary conditions for azeotropy, which must be checked for stability.

Less attention has been given to the more difficult heterogeneous case. One approach has been developed by Chapman and Goodwin [22]. In this work, the authors use the Levenberg-Marquardt method to find multiple solutions to the necessary conditions for azeotropy. Solutions are then checked for stability using the Gibbs tangent plane analysis (see following section). Unstable solutions are used as starting points for a new search to find the heteroazeotropes. There are several problems with this approach. First, the solution procedure used to find homogeneous azeotropes does not guarantee any, let alone all, solutions will be obtained. Second, an unstable homogeneous solution corresponding to an actual heteroazeotrope does not necessarily exist in the physical composition region (i.e., the regular simplex defined by $\{x \in \mathbb{R}^n \mid \sum_{i=1}^{n} x_i = 1 \text{ and } x_i \geq 0 \quad i = 1, \ldots, n\}$). In this thesis, it is shown that homogeneous azeotropes corresponding to heteroazeotropes (referred to as *spurious* homogeneous azeotropes) will exist and very often lie outside the physical region, and are thus of little use to the algorithm described above.

Another approach has recently been briefly described by Harding *et al.*[54] for the computation of heteroazeotropes. This approach is very similar to the global optimization approach described above.

A new approach has been developed in this thesis for the computation of the homogeneous and heterogeneous azeotropes present in a multicomponent mixture. The approach is independent of the liquid-liquid region topology and phase equilibrium model. Furthermore, this approach is capable of predicting incipient homogeneous and heterogeneous azeotropes and computing the bifurcation values of system and/or property model parameters at which they appear, disappear, or switch between each other.

All of the approaches described above find solutions satisfying the necessary conditions for homogeneous and heterogeneous azeotropy. The necessary and sufficient conditions as well as several phase stability tests are described in the following section. Once the homogeneous and heterogeneous azeotropes have been computed, the set can be tested for topological consistency [124, 45]. As will be shown later in this

chapter, the pure components and homogeneous and heterogeneous azeotropes are the fixed-points of a certain dynamical system. Furthermore, the compositions on the trajectories associated with these dynamical systems are confined to lie within the compact physical composition region. Consequently, the fixed-points of an n component mixture are subject to the Poincaré-Hopf Theorem and their indices must satisfy the following constraint,

$$\sum_{k=1}^{n} 2^k (I_k^+ - I_k^-) = (-1)^{n-1} + 1 \tag{1.6}$$

where I_k^+ and I_k^- are the number of fixed-points with k nonzero mole fraction elements that have indices +1 and -1, respectively. If the set of computed homogeneous and heterogeneous azeotropes do not satisfy this constraint, then either one or more solutions were not computed or one or more additional spurious solutions were computed.

1.1.2 Phase Stability Analysis

Necessary conditions for a nonreacting mixture of n components and π phases to be in equilibrium are

$$\begin{aligned} T^1 = T^2 &= \cdots = T^\pi, \\ P^2 = P^2 &= \cdots = P^\pi, \text{ and} \\ \mu_i^1 = \mu_i^2 &= \cdots = \mu_i^\pi \qquad i = 1, \ldots, n, \end{aligned}$$

where μ_i^j denotes the chemical potential of species i in phase j. For a mixture satisfying the conditions above to be a stable equilibrium state, at constant temperature and pressure, the Gibbs free energy of the mixture must be at a global minimum. Determining solutions satisfying the necessary conditions is equivalent to finding a tangent plane to the Gibbs energy of mixing surface of the mixture. Sufficient conditions correspond to finding a supporting hyperplane, that is, a tangent hyperplane that lies completely below the Gibbs energy surface (see Figure 1-3). This equiv-

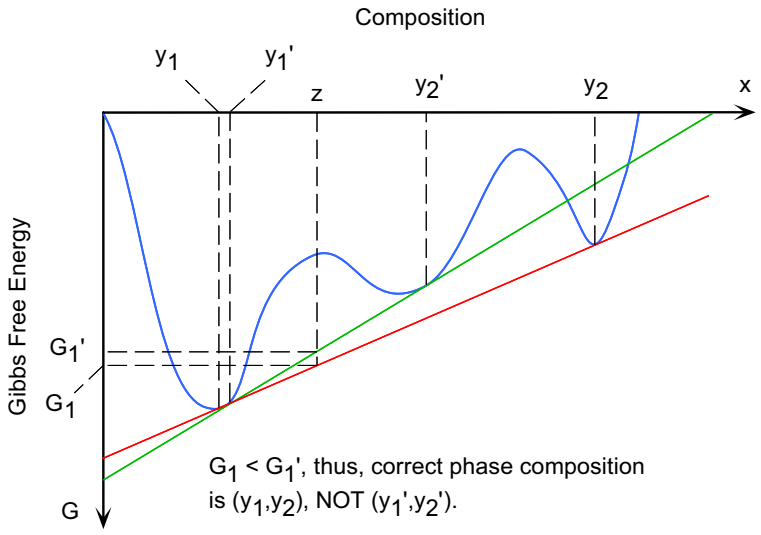

Figure 1-3: Schematic of hyperplanes to the Gibbs energy of mixing surface for a binary mixture with overall composition z.

alence between a global minimum of the Gibbs free energy of the mixture and the tangent plane criteria was presented by Baker *et al.* [8] and an algorithm for phase stability testing based on this criteria was subsequently developed by Michelsen [77]. Michelsen derives a tangent plane distance function,

$$F(x) = \sum_{i=1}^{n} x_i(\mu_i(x) - \mu_i^o), \tag{1.7}$$

which is equal to the vertical distance from the supporting hyperplane at composition x^o, (where $\mu_i^o = \mu_i(x^o)$), to the Gibbs energy surface. Hence, a mixture at conditions (x^o, T, P) is unstable if for any x,

$$F(x) < 0. \tag{1.8}$$

At such a point, the tangent plane lies above the Gibbs energy surface. It is shown in [77] that $F(x)$ will be non-negative for all x in the physical composition space if it is non-negative at all stationary points in the physical composition space. Stationary

points of (1.7) are defined by the following n equations:

$$\begin{gathered}\left(\frac{\partial m}{\partial x_i}\right) - \left(\frac{\partial m}{\partial x_i}\right)_o - \left(\frac{\partial m}{\partial x_n}\right) + \left(\frac{\partial m}{\partial x_n}\right)_o = 0 \qquad i = 1, \ldots, n-1 \\ \sum_{i=1}^{n} x_i - 1 = 0,\end{gathered} \tag{1.9}$$

where $m(x) = \Delta g^M / RT$ is the Gibbs free energy of mixing (scaled by RT) and the subscript o denotes the quantity is evaluated at the test composition x^o. The number of stationary points indicates the potential number of phases the liquid will split into and provide a good initial guess to the composition of each of the resulting phases.

It is important that any phase stability test distinguish an absolutely stable state from a metastable or unstable state. Metastable states are effectively unstable from the viewpoint of heteroazeotropic distillation. The tangent plane criteria correctly classifies metastability as being unstable.

Several phase stability tests have been developed in the past that attempt to solve equation system (1.9) for *all* stationary points and then check to make sure non-negativity of $F(x)$ is satisfied at all solutions. Unless the test is capable of computing, with certainty, all stationary points or at least the stationary point corresponding to the global minimum of (1.7), stability cannot be guaranteed. Of course, only one stationary point satisfying $F(x) < 0$ is required to correctly conclude the phase is unstable (with respect to the phase equilibrium model assumed to formulate $F(x)$). Solution procedures based on sophisticated initialization strategies and homotopy continuation have been developed to make the solution procedure robust, however, only few have been developed that can guarantee either all stationary points or the one corresponding to the minimum value for $F(x)$ will be obtained. One such approach, developed by McDonald and Floudas [76], applies to a certain class of models used to compute the Gibbs free energy and computes the global minimum value for $F(x)$ using global optimization. Obviously, if $F(x) \geq 0$ at the global minimum, the test phase is stable. Another approach, developed by Stadtherr *et al.* [105], computes all stationary points using interval Newton/generalized bisection techniques. This approach applies to any model used to compute the Gibbs energy. If $F(x)$ is non-negative at every

stationary point then it is non-negative everywhere within the physical composition space and the test phase is stable. Both McDonald's and Stadtherr's approaches guarantee correct determination of the stability of the phase, however, they are very computationally costly.

When the nonideality in multiple phases is represented with the same model, one possible solution to the phase equilibrium problem is the *trivial solution*. At the trivial solution, the composition, temperature, and pressure in all phases (associated with the same model) are equal. For the vapor-liquid-liquid equilibrium (VLLE) problem or the liquid-liquid equilibrium (LLE) problem, the trivial solution is the correct solution outside the liquid-liquid region and incorrect within the liquid-liquid region. Unfortunately, this trivial solution has a large region of convergence and unless a phase stability test is employed, there is no way to determine if it is the correct solution. Pham and Doherty [86] developed an algorithm for a limited class of mixtures that provides conditions whether or not the trivial solution should be rejected or not. The algorithm applies to liquids that may potentially split into at most *two* immiscible liquid phases and have liquid-liquid binodals characterized by either an upper critical solution temperature (UCST) *or* a lower critical solution temperature (LCST). For the UCST case, the algorithm is based on finding a maximum temperature, T_{max}, for which a given liquid of composition x^o and at pressure P will split into two stable liquid phases. This temperature is used during a phase equilibrium calculation to determine if a VLE calculation or a VLLE calculation should be performed. If a VLLE calculation is performed, the trivial solution can be confidently rejected. In the case of mixtures exhibiting a LCST, a minimum temperature, T_{min}, is computed and a similar procedure is applied. The disadvantage with this approach, aside from being applicable to a limited class of problems[1], is that the test simply indicates whether or not the trivial solution should be rejected but not how to obtain the actual solution. Furthermore, due to the possibility of multiple nontrivial liquid-liquid solutions, stability is not guaranteed.

[1]The class of problems may be limited, but many liquid mixtures, particularly those of industrial importance, fall into this category.

1.1.3 Residue Curve Maps and Distillation Lines

As mentioned above, azeotropes limit the separation possible with distillation. The presence of azeotropes effectively divides the composition space into regions characterized by the separations possible using a single distillation column. The analysis of these regions is very often carried out through the use of residue curve maps. Residue curve maps have been used since the turn of the century to characterize the behavior of binary distillation [101, 102]. The analysis of residue curve maps was greatly extended in the work of Matsuyama and Nishimura [75] and Doherty and Perkins [35, 36, 37], including the application of the analysis to the separation of homogeneous mixtures. In addition, residue curve maps have been the basis for many column sequencing algorithms [38, 87] and entrainer selection rules [34, 106].

The residue curves of an n component homogeneous mixture are defined by the following system of $n-1$ ODEs:

$$\frac{dx_i}{d\xi} = x_i - y_i(x) \qquad i = 1, \ldots, n-1, \tag{1.10}$$

where ξ is a dimensionless "warped time" (see Appendix A for derivation of (1.10)). According to the Gibbs phase rule, at a specified, constant pressure (or temperature), the equilibrium vapor composition, y, is uniquely defined by $n-1$ liquid mole fractions, x_i. It is shown in [35] that the pure components and azeotropes are exactly fixed points of the dynamical system (1.10) and can either be stable or unstable nodes, saddle points, or non-elementary arm-chair fixed-points. Moreover, they are subject to the topological constraints described above. The presence of azeotropes often introduces stable and unstable separatices and the projection of these separatrices onto the physical composition space defines simple distillation regions; a residue curve starting in one simple distillation region remains in this region for all ξ. These separatrices define simple distillation boundaries. It is generally assumed that the trajectories of (1.10), the residue curves, approximate the composition profiles of an actual distillation column at total reflux. Consequently, the simple distillation boundaries mentioned above are assumed to restrict the separations possible with a single

distillation column. There are two problems with this. First, the residue curves only approximate the column profiles at total reflux. Second, it is possible for a column profile to cross the convex side of a curved simple distillation boundary due to the fact that near the boundary, the vapor composition associated with the residue curve lies in the simple distillation region adjacent to the boundary [115]. This has led several authors in the past to report distillation 'anomalies'. The correct tool to employ in this type of analysis are the distillation lines which are the column operating lines at total reflux [119]. The behavior of both the residue curves and distillation lines are identical in the immediate vicinity of the fixed-points of the system (pure components and homogeneous and heterogeneous azeotropes) and, like the residue curves, the distillation lines often introduce distillation-line boundaries that divide the composition space into distinct regions. Even though the residue curve maps are only approximations to the distillation lines and column profiles at finite reflux, they have proven to be invaluable in the analysis of distillation and have been successfully applied in several column sequencing and entrainer selection algorithms.

Residue curve map analysis is extended to heterogeneous systems in [74] and [88]. The residue curves of an n component heterogeneous mixture are defined by

$$\frac{dx_i^o}{d\xi} = x_i^o - y_i(x^o) \qquad i = 1, \ldots, n-1, \tag{1.11}$$

where x^o denotes the overall liquid composition (see Appendix A for derivation of (1.11)). For a heterogeneous mixture with n_L liquid phases in equilibrium, there are n_L additional liquid compositions and $n_L - 1$ phase fractions embedded in the equilibrium calculation used to compute y. The heteroazeotropes are fixed points of (1.11), which like the homogeneous case, are restricted to stable and unstable nodes and saddles. Non-elementary arm-chair fixed points are found in systems exhibiting positive and negative deviations from Raoult's Law. Since heteroazeotropy is characterized by strong positive deviations from Raoult's Law over a range of composition, the non-elementary fixed-points are not likely to be found in heterogeneous mixtures. Furthermore, it is shown in [74] that the presence of multiple liquid phases further

restricts the number of different types of fixed points of (1.11) to $n - n_L + 1$. The residue curves of a heterogeneous mixture are smooth and continuous as they move through the boundary of the heterogeneous liquid boiling surface, at which point (1.10) correctly describes their behavior. The crossing of the heterogeneous residue curve into the homogeneous liquid region is illustrated in the schematic in Figure 1-4. Figure 1-5 contains a schematic of a two column distillation sequence used to

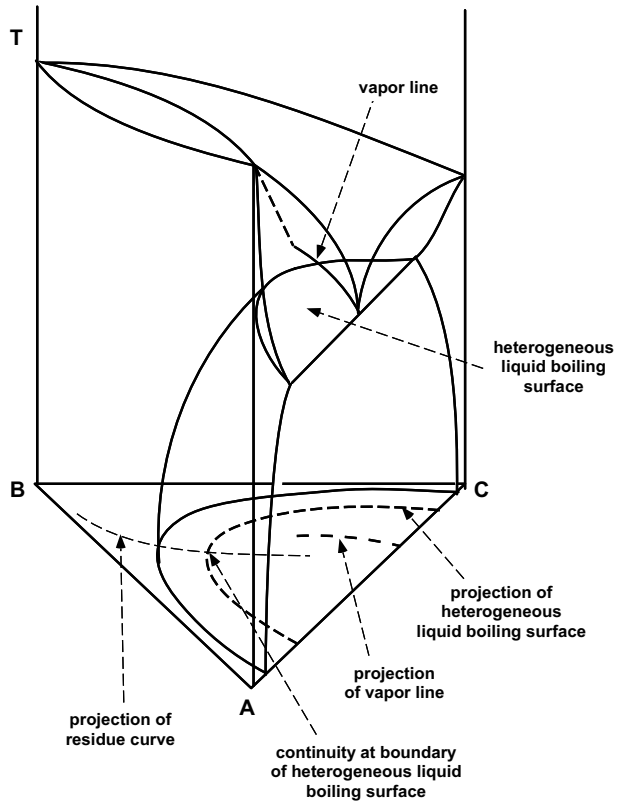

Figure 1-4: Schematic of a heterogeneous Txy-diagram showing the residue curve moving into the homogeneous liquid region.

separate components A and B (which form a binary homogneous azeotrope) using C as a heterogeneous entrainer. The addition of C causes a liquid-liquid phase split and introduces an additional binary BC homogenous azeotrope, a binary AC heterogeneous azeotrope, and a ternary heterogeneous azeotrope. This figure illustrates the partitioning of the composition space into different regions by separatrices and the use of the liquid-liquid phase split in the decanter to move the feed compositions into different distillation regions. Heterogeneous residue curve maps have been employed

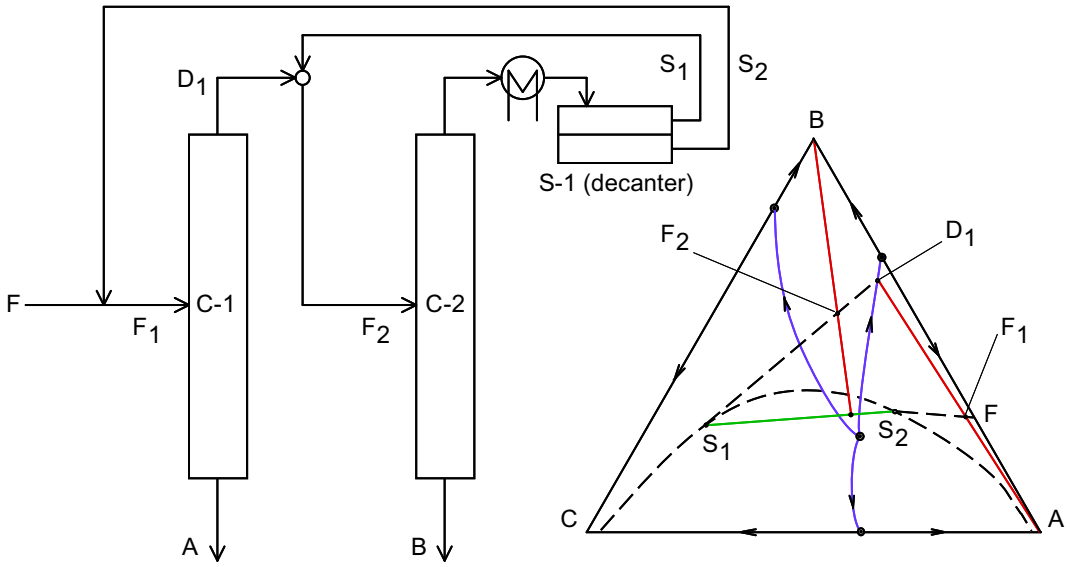

Figure 1-5: Schematic of a two column distillation sequence used to separate components A and B using C as an entrainer and the associated residue curve map. Mass balance lines associated with the columns, decanter, and mixing, as well as distillation boundaries are illustrated in the Gibbs composition triangle.

in the column sequencing algorithm described in [87].

1.1.4 Simulation of Heteroazeotropic Distillation Columns

The discussion above presents topics that are important when analyzing heteroazeotropic systems for design purposes. Once the preliminary designs for a distillation system have been developed (e.g., entrainer selection, column sequencing, etc.) the next step is to simulate the process. Several approaches have been developed for the steady-state and dynamic simulation of heteroazeotropic distillation systems, some of which are discussed below. Heteroazeotropic distillation is far more complicated than its homogeneous counterpart. Often, the liquid-liquid phase is not restricted to the decanter and can appear on as much as seventy percent of the trays. The appearace of a second liquid phase on the trays results in singularities and multiple solutions in the model [120]. Furthermore, a large temperature gradient is associated with the movement of the liquid-liquid front within the column. Heteroazeotropic columns are also characterized by multiple steady-states and extreme parametric sensitivity. Like a homogeneous azeotropic column, a heterogeneous tower exhibits a maximum reflux

ratio, above which separation deteriorates.

Steady-state Simulation

The steady-state simulation of heteroazeotropic columns has been studied extensively in the past [15, 96, 44, 62, 109, 7, 17, 18, 19].

An algorithm employing homotopy continuation has been developed by Kovach and Seider [62] for the steady-state simulation of a three phase distillation tower and associated phase separator. This approach uses homotopy continuation to avoid limit points when multiple liquid phases appear on some of the trays and to compute the liquid-liquid phase equilibrium. This approach uses homotopy continuation to avoid the trivial solution (improving the robustness of the LLE calculation), however, it does not employ a proper phase stability test and thus correct column profiles cannot be guaranteed.

Another algorithm for the steady-state simulation of heteroazeotropic towers has been developed by Cairns and Furzer [17, 18, 19]. The authors employ the modified Naphthali-Sandholm approach of Furzer [46] to express the MESH (material balance, equilibrium, summation of mole fractions, and heat balance) equations and solve them using Newton's method. A phase stability test is employed each time the activity coefficients are computed to ensure the phase is stable, however, the authors do not employ an implementation that guarantees correct results (e.g., the two approaches described in the phase stability section above). Consequently, very little can be said on the correctness of the column profiles.

Dynamic Simulation

Several approaches have been developed for the dynamic simulation of heteroazeotropic distillation columns [93, 122, 121]. In contrast to the steady-state models above which simply contain the MESH equations, the dynamic models of heteroazeotropic columns must be very detailed (e.g. including tray hydraulics, geometry, etc.) in order to accurately predict the dynamic behavior under various conditions such as the response to process disturbances.

Rovaglio and Doherty [93] developed a dynamic model for a heteroazeotropic distillation column and examined the effect of disturbances in a column used to dehydrate ethanol using benzene as an entrainer. The model predictions are consistent with previous work in that these columns exhibit multiple steady-states, complex behavior, extreme parametric sensitivity and that the liquid-liquid region is not confined to the decanter and can be found on as much as sixty percent of the trays. Furthermore, they show that small perturbations in pressure can lead to separation failure over the course of a 24-hour period. Rather than performing a phase stability test at each time step and on every tray, the authors fit a spline to the boundary of the heterogeneous liquid boiling surface in order to check whether or not the overall liquid composition lies inside or outside the liquid-liquid region. As shown in [121], the location of the liquid-liquid region within the column is sensitive to small perturbations. Furthermore, sharp temperature gradients are associated with the movement of the liquid-liquid region through the column. Consequently, accurate determination of the point at which a second liquid phase appears or disappears is crucial for accurately predicting the column dynamics. Although the approach based on the spline fit of the heterogeneous liquid boiling surface boundary is a very rapid way of checking whether or not the overall liquid composition is stable, a better approach may be to use the spline to determine if the overall liquid composition is 'near' the liquid-liquid boundary (based on the interpolation error of the spline) and if it is, use a more robust, but computationally expensive, phase stability test to accurately locate the point at which liquid-liquid phase splitting occurs.

In [122], Wong *et al.*solve a dynamic model using a semi-implicit Runge-Kutta integrator to simulate a tower to dehydrate ethanol using benzene as an entrainer. At each time step, a phase stability test is performed on every tray to determine stability. The authors employ the phase stability algorithm based on the tangent plane criteria, but do not describe how they compute all stationary points or the global minimum of the tangent plane distance function. Judging from the computational times reported, they are probably not using a method such as that of McDonald or Stadtherr and, thus, stability cannot be guaranteed.

A similar approach is described in [121]. Here, the authors solve a dynamic model using DASSL [16], a differential-algebraic equation (DAE) integrator based on the backwards differentiation formula (BDF) method. After each successful time step and on every tray, a phase stability test is employed. The authors address the problem of accurately detecting the real bifurcation occurring at the point where a second liquid phase appears or disappears and develop a branch switching algorithm to correctly reinitialize the system on the correct branch, thereby avoiding the problem of converging to the trivial solution outside the heterogeneous liquid region. The authors employ the phase stability test of Michelsen as implemented in the UNIFLASH program [78, 79]. Unfortunately, this algorithm does not guarantee stability, leaving the computed column profiles in question.

1.2 Challenges

Although the topic of heteroazeotropic distillation has been studied by many researchers in the past, there are still several issues that remain to be addressed. First, there are limited tools available for the systematic analysis of heteroazeotropic systems. Robust and efficient procedures are required for the computation of all homogeneous and heterogeneous azeotropes present in a multicomponent mixture. These tools should be independent of the model used to represent the nonideality of the system and should be capable of dealing with complex liquid-liquid topologies, as well as systems containing three or more liquid phases in equilibrium. Furthermore, operation of a heteroazeotropic column is very sensitive to perturbations in process parameters, leading to complex dynamics and multiple steady-states. Greater understanding of the phase equilibrium structure will improve interpretation of simulation results as well as improve the understanding of how the column should be operated.

Several models and algorithms have been developed for both steady-state and dynamic simulation of heteroazeotropic columns. None of the approaches described apply a phase stability test that guarantees stability. However, the phase stability tests employed are far better than not performing any test at all (i.e., assuming if

a solution to the VLLE model can be found then the overall liquid composition is unstable and if no solution can be found the liquid is stable). These approaches are fast (relative to the approaches of McDonald or Stadtherr) and provide satisfactory results, particularly if the system is well understood (e.g., number of possible phases that can form and which components they are rich in), however, conclusions based on the results of such simulations should bear in mind the results may be incorrect.

A better approach would be to employ a hybrid phase stability test such as that described above. With the developement of appropriate tools, the liquid-liquid region can be rapidly determined a priori and enclosed within a collection of convex sets. During the simulation, the location of the temperature, pressure, and overall liquid composition relative to these sets can be used as a basis for deciding whether or not a robust phase stability test should be performed.

The remainder of this part of the thesis addresses the first deficiency described above. Chapter 2 describes a new approach for the computation of homogeneous and heterogeneous azeotropes present in a multicomponent mixture. Theoretical analysis is performed, resulting in conditions under which *all* homogeneous and heterogeneous azeotropes will be computed as well as several algorithmic improvements. The following chapter describes how this approach can be extended to compute efficiently changes in phase equilibrium structure under system and/or property model parameter variation, including the bifurcation values of the parameters where homogeneous and heterogeneous azeotropes appear, disappear, and switch between each other. This part of the thesis is concluded with a chapter containing several numerical examples illustrating the approaches developed in chapters 2 and 3.

Chapter 2

Computation of Heteroazeotropes

2.1 Introduction

Azeotropes and heteroazeotropes of an n component system satisfy the following system of nonlinear equations:

$$x - y = 0 \tag{2.1}$$

$$f_{eq}(x, y, T, P) = 0 \tag{2.2}$$

$$\sum_{i=1}^{n} y_i - 1 = 0 \tag{2.3}$$

$$x, y \geq 0 \tag{2.4}$$

$$T, P > 0 \tag{2.5}$$

where $x \in \mathbb{R}^n$ is the liquid composition, $y \in \mathbb{R}^n$ is the vapor composition, and f_{eq} is some equilibrium relationship between the liquid and the vapor. If the system is heterogeneous (two or more liquid phases in equilibrium), x is the overall liquid composition and there are N_L additional liquid compositions and $N_L - 1$ liquid phase fractions embedded within the equilibrium relationship, where N_L is the number of distinct liquid phases present. It should be noted that the equations above are a necessary but not sufficient condition for azeotropy. As described in the previous chapter, the solution to equations (2.1)-(2.5) corresponds to a tangent hyperplane to

the Gibbs energy of mixing surface at a temperature T and pressure P where the liquid composition is equal to the vapor composition (overall liquid composition in the heterogeneous case). The sufficient condition for stability is that the composition, temperature, and pressure minimize the Gibbs free energy of the mixture or equivalently, the tangent hyperplane supports the entire Gibbs free energy surface. Figures 2-1 and 2-2 contain schematics of the supporting hyperplanes to the Gibbs free energy surface for a binary homogeneous azeotrope and a binary heterogeneous azeotrope, respectively. Solutions satisfying the necessary conditions must be further examined

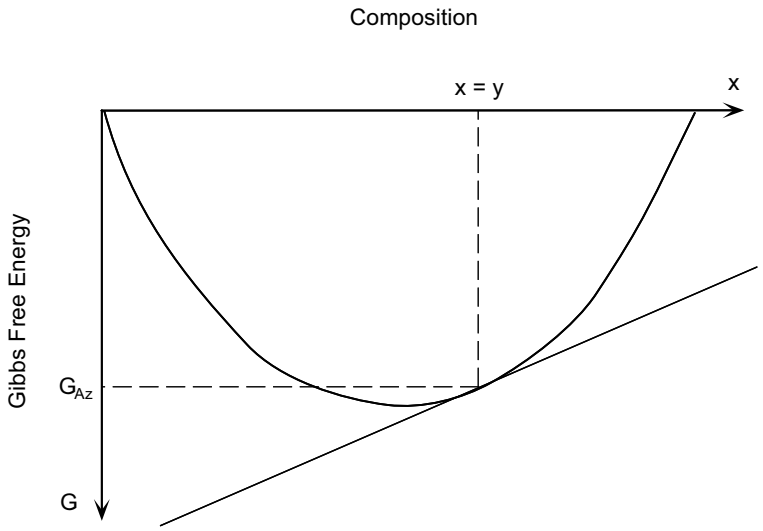

Figure 2-1: Schematic of supporting hyperplane to the Gibbs free energy surface for a binary homogeneous azeotrope.

to determine if they are stable using a phase stability test such as those described in the previous chapter [77, 76, 105].

This chapter describes a systematic approach developed in this thesis for computing the homogeneous and heterogeneous azeotropes present in a multicomponent mixture containing any number of liquid phases in equilibrium and with any liquid-liquid region topology. For example, liquid-liquid binodals exhibiting an upper critical solution temperature (UCST), or a lower critical solution temperature (LCST), or both an UCST and a LCST, disjoint liquid-liquid regions, etc. The approach is an extension of a method developed by Fidkowski *et al.* [45] for the computation of homogeneous azeotropes. Other approaches for the computation of homogeneous

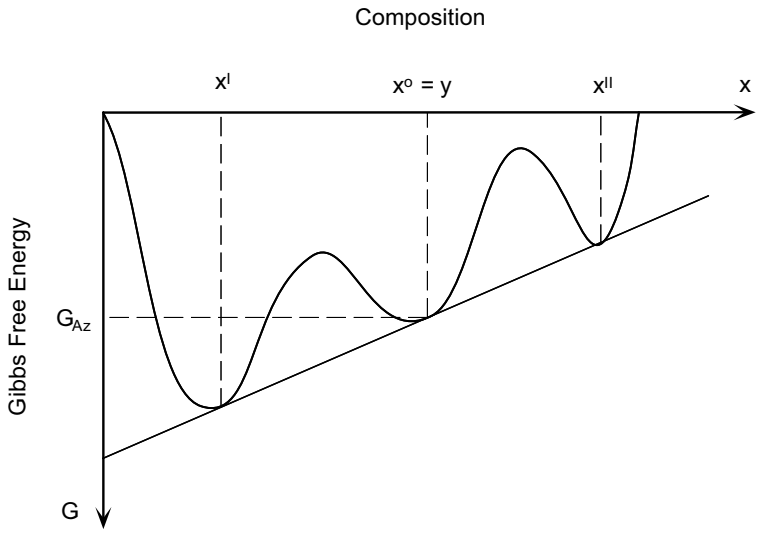

Figure 2-2: Schematic of supporting hyperplane to the Gibbs free energy surface for a binary heterogeneous azeotrope.

and heterogeneous azeotropes are reviewed in the previous chapter. Our approach has the advantage of being able to compute efficiently changes in the phase equilibrium structure under system and/or property model parameter variation including the capability of detecting incipient homogeneous azeotropes and heterogeneous azeotropes that may exist under different conditions. This extension of the algorithm is discussed in detail in chapter 3 of this thesis.

The first section of this chapter describes Fidkowski's approach followed by some previously unreported analysis of the method. The following section describes our extension for the computation of heteroazeotropes. This section also includes an analysis of our approach. The effectiveness of the method is illustrated through several numerical examples contained in chapter 4. In the remainder of this part of the thesis, azeotropes will refer to homogeneous azeotropes and heteroazeotropes will, obviously, refer to heterogeneous azeotropes. The homogeneous qualifier will only be used where necessary to avoid confusion.

2.2 Computation of Homogeneous Azeotropes

Fidkowski's approach for computing homogeneous azeotropes is based on solving the necessary conditions for azeotropy using homotopy continuation. The homotopy map is given by

$$h_i(x,\lambda) = \lambda(x_i - y_i(x)) + (1-\lambda)(x_i - y_i^{id}(x)) \qquad i = 1,\ldots,n-1 \qquad (2.6)$$

where $x \in \mathbb{R}^n$ is the liquid composition, $y^{id} \in \mathbb{R}^n$ is the vapor composition in equilibrium with the liquid computed using *Raoult's Law*, $y \in \mathbb{R}^n$ is the vapor in equilibrium with the liquid computed using some nonideal vapor-liquid equilibrium model, and $\lambda \in \mathbb{R}$ is the homotopy parameter. The summation of mole fraction constraint is handled implicitly by removing x_n, y_n, and y_n^{id}. At a given pressure, system (2.6) is $n-1$ equations in terms of n unknowns ($n-1$ independent mole fractions and λ). By setting this underdetermined system equal to zero, a homotopy path is defined that can be tracked numerically using standard continuation methods [4, 117, 95]. At $\lambda = 0$, the homotopy map reduces to $h(x,0) = x - y^{id}(x) = 0$ and since Raoult's Law cannot predict azeotropy, there are precisely n solutions to this system of equations, the pure components. Provided the pure component boiling temperatures are distinct at the specified pressure, these n solutions correspond to n pure component branches. If the pure component boiling temperatures are not distinct, then more than n branches may exist at $\lambda = 0$. This is discussed in detail in chapter 3. The basic idea of the approach is to start with λ initialized to zero and x initialized to each of the pure components and track these n paths to $\lambda = 1$ where the homotopy map reduces to $h(x,1) = x - y(x) = 0$, the necessary conditions for azeotropy. Along some of these pure component branches bifurcation points appear that correspond to intersections with binary branches (branches with two nonzero mole fraction elements). These binary branches result in points satisfying the necessary conditions for azeotropy for a binary mixture at $\lambda = 1$. Similarly, along some of the binary branches, bifurcation points are identified that correspond to intersections with ternary branches from which solutions satisfying the necessary conditions for azeotropy of ternary mixtures

are obtained at $\lambda = 1$. In general, k-ary azeotropes are obtained from branches that bifurcate off $(k-1)$-ary branches. Figure 2-3 contains a bifurcation diagram (T versus λ) for the acetone, chloroform, methanol, ethanol, and benzene system at a pressure of one bar. The six binary azeotropes, two ternary azeotropes, and one quarternary azeotrope present in the mixture are computed.

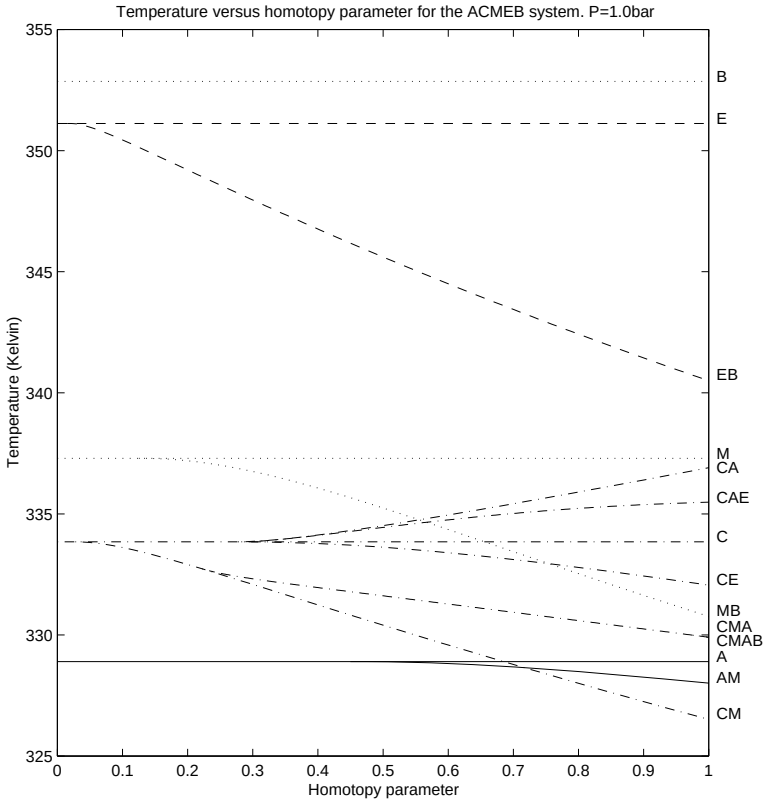

Figure 2-3: Bifurcation diagram for the acetone, chloroform, methanol, ethanol, and benzene system at one bar.

Although the bifurcation points correspond to intersections between branches of different dimension, the term intersection point will refer to a specific intersection discussed in section 2.4.

2.2.1 Analysis

The homogeneous homotopy map, equation (2.6), can be expressed in the following alternative form:

$$F(x, y^*, T, \lambda) = \begin{pmatrix} x - y^* \\ y_1^* - [\lambda K_1 + (1-\lambda)P_1^s/P]\, x_1 \\ \vdots \\ y_n^* - [\lambda K_n + (1-\lambda)P_n^s/P]\, x_n \\ \sum_{i=1}^n y_i^* - 1 \end{pmatrix} \tag{2.7}$$

where $y^* \in \mathbb{R}^n$ is the *perturbed* vapor composition and the term $\bar{K}_j \equiv \lambda K_j + (1-\lambda)P_j^s/P$ is a pseudo K-value. In this form, the summation of mole fraction constraint is handled explicitly. The perturbed vapor composition can be removed from (2.7) using the first n equations:

$$\bar{F}(x, T, \lambda) = \begin{pmatrix} x_1\,(1 - [\lambda K_1 + (1-\lambda)P_1^s/P]) \\ \vdots \\ x_n\,(1 - [\lambda K_n + (1-\lambda)P_n^s/P]) \\ \sum_{i=1}^n x_i - 1 \end{pmatrix}. \tag{2.8}$$

Setting system (2.8) to zero defines a 1-manifold in (x, T, λ)-space. Alternatively, temperature can be fixed in which case setting (2.8) to zero defines a 1-manifold in (x, P, λ)-space. This curve will be referred to as a homogeneous homotopy path or branch or as simply a homogeneous branch. The Jacobian matrix of (2.8) can be expressed as:

$$\nabla \bar{F}(x,T,\lambda) = \begin{pmatrix} \alpha_1 - \lambda x_1 K_{1,1} & -\lambda x_1 K_{1,2} & \cdots & -\lambda x_1 K_{1,n} & -x_1\beta_1 & -x_1\phi_1 \\ -\lambda x_2 K_{2,1} & \alpha_2 - \lambda x_2 K_{2,2} & \cdots & -\lambda x_2 K_{2,n} & -x_2\beta_2 & -x_2\phi_2 \\ \vdots & \vdots & \ddots & \vdots & \vdots & \vdots \\ -\lambda x_n K_{n,1} & -\lambda x_n K_{n,2} & \cdots & \alpha_n - \lambda x_n K_{n,n} & -x_n\beta_n & -x_n\phi_n \\ 1 & 1 & \cdots & 1 & 0 & 0 \end{pmatrix} \tag{2.9}$$

where

$$\alpha_j = 1 - \left[\lambda K_j + (1-\lambda) P_j^s / P\right] = 1 - \bar{K}_j, \tag{2.10}$$

$$\beta_j = \lambda \left(\frac{\partial K_j}{\partial T}\right)_{x,y,P} + \frac{(1-\lambda)}{P}\frac{dP_j^s}{dT}, \tag{2.11}$$

$$\phi_j = K_j - \frac{P_j^s}{P}, \text{ and} \tag{2.12}$$

$$K_{i,j} = \left(\frac{\partial K_i}{\partial x_j}\right)_{x_i[j],y,T,P}, \tag{2.13}$$

for $i, j = 1, \ldots, n$ and the subscript $x_{i[j]}$ denotes all mole fraction elements are held constant except x_j.

Without loss of generality, a k-ary branch of an n component mixture satisfies the following

$$x_j \neq 0 \quad \forall j = 1, \ldots, k, \tag{2.14}$$

$$\alpha_j = 0 \quad \forall j = 1, \ldots, k, \tag{2.15}$$

$$x_j = 0 \quad \forall j = k+1, \ldots, n, \text{ and} \tag{2.16}$$

$$\alpha_j \neq 0 \quad \forall j = k+1, \ldots, n. \tag{2.17}$$

Constraints (2.14) and (2.17) may be simultaneously violated at isolated points along the homotopy branch. As shown below, there violations are of particular interest. Let $\bar{c}(\xi) \in \bar{F}^{-1}(0)$ denote a homotopy branch where ξ is some suitable parameterization (e.g., arclength) and suppose we are currently on a k-ary branch. On $\bar{c}(\xi)$ the following holds:

$$\alpha_j = 0 \text{ for all } j = 1, \ldots, k.$$

Thus,

$$\nabla \bar{F}(\bar{c}(\xi))=\begin{pmatrix} -\lambda x_1 K_{1,1} & -\lambda x_1 K_{1,2} & \cdots & -\lambda x_1 K_{1,k+1} & \cdots & -\lambda x_1 K_{1,n} & -x_1\beta_1 & -x_1\phi_1 \\ \vdots & \vdots & \ddots & \vdots & \ddots & \vdots & \vdots & \vdots \\ -\lambda x_k K_{k,1} & -\lambda x_k K_{k,2} & \cdots & -\lambda x_k K_{k,k+1} & \cdots & -\lambda x_k K_{k,n} & -x_k\beta_k & -x_k\phi_k \\ 0 & 0 & \cdots & \alpha_{k+1} & \cdots & 0 & 0 & 0 \\ \vdots & \vdots & \ddots & \vdots & \ddots & \vdots & \vdots & \vdots \\ 0 & 0 & \cdots & 0 & \cdots & \alpha_n & 0 & 0 \\ 1 & 1 & \cdots & 1 & \cdots & 1 & 0 & 0 \end{pmatrix}. \quad (2.18)$$

The following lemma provides useful information for identifying bifurcation points and is used in the discussion of when all azeotropes will be computed, presented later in this section.

Lemma 1 *A necessary condition for a transcritical bifurcation from a k-ary branch onto a $(k+1)$-ary branch at a point $\bar{c}(\tilde{\xi})$ is*

$$\alpha_j(\tilde{\xi}) = 0 \text{ for some } j \in \{k+1, \ldots, n\}. \quad (2.19)$$

A necessary condition for a transcritical bifurcation from a $(k+1)$-ary branch to a k-ary branch at a point $\bar{c}(\tilde{\xi})$ is

$$x_j(\tilde{\xi}) = 0 \text{ for some } j \in \{1, \ldots, k+1\}. \quad (2.20)$$

Condition (2.20) becomes necessary and sufficient for the bifurcation from a $(k+1)$-ary branch onto a k-ary branch by adding the following:

1. $dx_j/d\xi\big|_{\xi=\tilde{\xi}} \neq 0$,
2. *rank* $\nabla \bar{F}(\tilde{\xi}) = N - 1$ *where* $N = n + 1$, *and*
3. $\partial F/\partial x_j\big|_{\xi=\tilde{\xi}} \in \mathcal{R}\left(\nabla_{[j]}\bar{F}(\tilde{\xi})\right)$ *where* $\nabla_{[j]}$ *denotes partial derivatives with respect to all variables except* x_j.

Proof. See Appendix B.

The bifurcation points on the pure component branches can be identified a priori without any branch tracking. First, define the following quantity:

$$\begin{aligned} \lambda_{i,j} &= \frac{1 - P_j^s(T_i^s)/P}{K_j(e_i, e_i, T_i^s, P) - P_j^s(T_i^s)/P} \\ &= \frac{1 - P_j^s(T_i^s)/P}{(\psi_{i,j}^\infty - 1)P_j^s(T_i^s)/P} \qquad (i \neq j) \end{aligned} \tag{2.21}$$

where T_i^s is the boiling point of pure component i at pressure P and

$$\psi_{i,j}^\infty = \lim_{x_i \to 1} \frac{\gamma_j \hat{\nu}_j}{\nu_j^s} \exp\left[-\frac{1}{RT}\int_{P_j^s}^{P} V_j dP\right]$$

($\hat{\nu}_j$ is the fugacity coefficient for species j in a mixture and ν_j^s is the fugacity of saturated pure j). At low to moderate pressure, the quantity $\psi_{i,j}^\infty$ is approximately equal to the infinite dilution activity coefficient, γ_j^∞ in a binary mixture of i and j. There will be a bifurcation point on the pure component i branch corresponding to a binary branch of components i and j if $0 < \lambda_{i,j} < 1$. Actually, the bifurcation point will exist regardless of the value of $\lambda_{i,j}$ provided $\psi_{i,j}^\infty \neq 1$. However, as explained later in this section, it is branches associated with bifurcation points between $\lambda = 0$ and $\lambda = 1$ that lead to solutions satisfying the necessary conditions for azeotropy at $\lambda = 1$. Assume that component 2 forms an azeotrope with component 1 and this is not an isolated azeotrope. An isolated azeotrope is an azeotrope with the xy-diagram shown in Figure 2-4. According to Fidkowski [45], although isolated azeotropes cannot be ruled out on thermodynamic grounds, there are no known physical examples of them and it is extremely unlikely they even exist. The authors cite only one known case of multiple azeotropy, formed in a mixture of benzene and hexaflourobenzene. In this mixture both azeotropes are computed with this method: a minimum boiling binary azeotrope bifurcates from the lower boiling species' homotopy branch and the maximum boiling azeotrope bifurcates from the higher boiling species' homotopy branch. When computing homogeneous azeotropes, we are interested in bifurcations

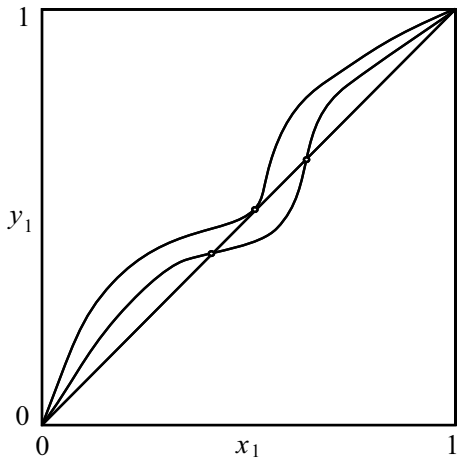

Figure 2-4: Binary mixture exhibiting an isolated azeotrope and a pair of azeotropes (at another pressure) that bifurcate from the isolated azeotrope.

in $0 < \lambda < 1$. Criteria (2.21) will be satisfied if

$$0 < 1 - \frac{P_2^s(T_1^s)}{P} < (1 - \psi_{1,2}^{\infty})\frac{P_2^s(T_1^s)}{P} \tag{2.22}$$

or

$$(1 - \psi_{1,2}^{\infty})\frac{P_2^s(T_1^s)}{P} < 1 - \frac{P_2^s(T_1^s)}{P} < 0. \tag{2.23}$$

Thus,

$$\frac{P_2^s(T_1^s)}{P} > 1 \text{ and } \psi_{1,2}^{\infty}\frac{P_2^s(T_1^s)}{P} < 1 \tag{2.24}$$

or

$$\frac{P_2^s(T_1^s)}{P} < 1 \text{ and } \psi_{1,2}^{\infty}\frac{P_2^s(T_1^s)}{P} > 1. \tag{2.25}$$

Now, if a binary azeotrope actually exists, it can be readily seen from the xy-plot that if the azeotrope is minimum boiling,

$$\psi_{2,1}^{\infty}\frac{P_1^s(T_2^s)}{P} > 1 \text{ and } \psi_{1,2}^{\infty}\frac{P_2^s(T_1^s)}{P} > 1, \tag{2.26}$$

or if the azeotrope is maximum boiling,

$$\psi_{2,1}^{\infty}\frac{P_1^s(T_2^s)}{P} < 1 \text{ and } \psi_{1,2}^{\infty}\frac{P_2^s(T_1^s)}{P} < 1, \tag{2.27}$$

provided the azeotrope is not isolated. Finally, if the azeotrope is maximum boiling and component 1 is less volatile than component 2 (i.e., $T_1^s > T_2^s$) then $P_2^s(T_1^s)/P > 1$ and the bifurcation point will appear on the pure component 1 branch (the higher boiling component). If, however, the azeotrope is minimum boiling (and component 1 is still less volatile than component 2) then the bifurcation appears on the pure component 2 branch. Thus, for the binary case, if the azeotrope actually exists, the bifurcation point will appear at some $0 < \lambda < 1$ and a minimum boiling azeotrope will be obtained through a bifurcation on the lower boiling component branch and a maximum boiling azeotrope will be obtained through a bifurcation on the higher boiling component branch (assuming the phase equilibrium model employed accurately represents the physical behavior). Unfortunately, the exact values for the bifurcation λ's cannot be predicted a priori for $k > 1$ since $\alpha_k = \alpha_k(\bar{c}(\xi))$ and $\bar{c}(\xi)$ is not known a priori.

The exact condition under which a bifurcation point occurs is important because it allows for a more aggressive stepsize strategy to be used during the numerical continuation. If the bifurcation points were identified by monitoring the sign of the determinant of the Jacobian matrix, there is a possibility of jumping over multiple bifurcation points in which case it would possible to miss an even number of them due to a cancellation of sign changes or incorrectly conclude there is only one when there may be an odd number of them greater than one. By using the explicit necessary conditions for the existence of a bifurcation point, it is possible to use more sophisticated approaches for detecting them, thereby increasing the efficiency and robustness of the algorithm. This is described in the implementation section later in this chapter.

The remainder of this section discusses under what conditions all homogeneous azeotropes will be computed using the homotopy method. First, a brief outline of the proof is given. A bounded region, $\mathcal{S}$, will be constructed in (x, T, λ)-space. If an

azeotrope exists, it will be located on a side of $\mathcal{S}$ and the homogeneous path through this azeotrope will move into $\mathcal{S}$. If zero is a regular value (i.e., $\nabla \bar{F}$ has full rank $n+1$) inside this region, it will be shown that, except under extremely pathological cases, the homogeneous path will leave $\mathcal{S}$ at a bifurcation point associated with a lower dimensional homogeneous branch. Once on this lower dimensional branch, the same reasoning above is applied. This process is continued until a pure component branch is obtained and the computation of the original azeotrope is guaranteed. The section concludes with a discussion of the conditions under which zero will be a regular value in this region.

The bounded, connected set mentioned above is defined as follows:

$$\mathcal{S} = \mathcal{C} \times [T_{min}, T_{max}] \times (0, 1] \tag{2.28}$$

where

$$\mathcal{C} = \{x \in \mathbb{R}^n \ \mid \ 0 \leq x_j \leq 1, \ \ j = 1, \ldots, n, \text{ and } \textstyle\sum_{i=1}^{n} x_i = 1\} \tag{2.29}$$

is the physical composition space and $0 < T_{min} < T_{max}$. The bounds on the temperature, T_{min} and T_{max} are based on the fact that $\bar{K}_j$ is a convex combination of two smooth functions, K_j and P_j^s/P in $\mathcal{S}$. Suppose an n-ary azeotrope exists. Denote this point on the homotopy path as $\bar{c}(\tilde{\xi}) = (x(\tilde{\xi}), T(\tilde{\xi}), 1) \in \mathcal{S}$. Assuming that $d\bar{c}/d\xi$ is not tangent to the level set $\mathcal{C} \times [T_{min}, T_{max}] \times \{1\}$ at $\tilde{\xi}$ (this exception will be discussed later), $\bar{c}(\xi)$ will point into $\mathcal{S}$ (defining the positive ξ direction as the direction of decreasing λ at $\tilde{\xi}$). If zero is a regular value of $\bar{F}$ in the interior of $\mathcal{S}$ (i.e., $\nabla \bar{F}$ has maximal rank $n+1$ along $\bar{c}(\xi) \in \text{int}(\mathcal{S})$) then after finite ξ, $\bar{c}(\xi)$ will leave $\mathcal{S}$. Obviously, zero will not be a regular value of $\bar{F}$ on the boundary of $\mathcal{S}$ since this is where bifurcations onto lower dimensional branches occur. This homotopy path will leave $\mathcal{S}$ in one of two ways: through a side of $\mathcal{S}$ where $x_i = 0$ for some $1 \leq i \leq n$ or turn back around and exit through the side where $\lambda = 1$ (see Figure 2-5). A branch with $n > 1$ cannot, in general, leave through the side where $\lambda = 0$ since as a consequence of Raoult's Law, the pure components are the only solutions at this point.

It is possible, however, for the homotopy branch to leave $\mathcal{S}$ at $\lambda = 0$. This special case, discussed in detail in chapter 3, occurs only at specific values of pressure and does not present any problem for this algorithm. If $\bar{c}(\xi)$ leaves through a side of $\mathcal{S}$ where $x_i = 0$ then this point corresponds to a transcritical bifurcation point onto an $(n-1)$-ary branch provided the conditions of lemma 1 are satisfied. The case where $\bar{c}(\xi)$ leaves through the side defined by $\lambda = 1$ corresponds to the case of multiple azeotropes bifurcating from an isolated azeotrope (see Figure 2-4). As stated above, this has never been observed experimentally. Nevertheless, the method will fail in this case. Furthermore, the case where $d\bar{c}/d\xi$ is tangent to $\mathcal{C} \times [T_{min}, T_{max}] \times \{1\}$ at $\tilde{\xi}$ corresponds to the isolated azeotrope. If the homotopy branch passes through a side defined by $x_i = 0$, we are able to jump onto a lower dimensional branch. The set $\mathcal{S}$ is redefined for this new lower dimensional space by reducing the dimensionality of the composition space $\mathcal{C}$. If zero is a regular value of $\bar{F}$ in the interior of the new $\mathcal{S}$ and the homotopy path does not leave this new $\mathcal{S}$ through the side defined by $\lambda = 1$, we will be able to bifurcate onto a branch of even lower dimension. This reasoning is continued until we reach a pure component branch. If we are able to do this then the original n-ary azeotrope can be obtained through the homotopy method. The next question to examine is under what conditions zero will be a regular value of $\bar{F}$.

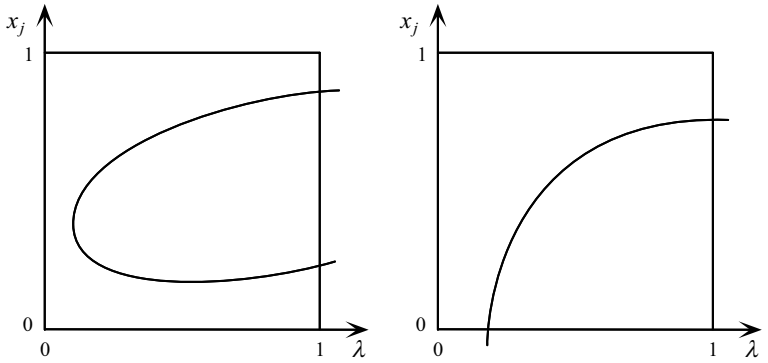

Figure 2-5: Two possible ways $\bar{c}(\xi)$ can leave $\mathcal{S}$.

Zero will be a regular value of $\bar{F}$ along a path $\bar{c}(\xi)$ if the Jacobian matrix (2.18) has rank $n+1$. To facilitate the analysis, the vapor is treated as an ideal gas and the

Jacobian matrix is rewritten as

$$
\begin{aligned}
\nabla \bar{F}(\bar{c}(\xi)) &= \begin{pmatrix} -\lambda x_1 K_{1,1} & -\lambda x_1 K_{1,2} & \cdots & -\lambda x_1 K_{1,n} & -x_1\beta_1 & -x_1\phi_1 \\ \vdots & \vdots & \ddots & \vdots & \vdots & \vdots \\ -\lambda x_n K_{n,1} & -\lambda x_n K_{n,2} & \cdots & -\lambda x_n K_{n,n} & -x_n\beta_n & -x_n\phi_n \\ 1 & 1 & \cdots & 1 & 0 & 0 \end{pmatrix} \\
&= \begin{pmatrix} -\lambda x_1 K_1 g_{1,1} & -\lambda x_1 K_1 g_{1,2} & \cdots & -\lambda x_1 K_1 g_{1,n} & -x_1 K_1\omega_1 & -x_1 K_1\theta_1 \\ \vdots & \vdots & \ddots & \vdots & \vdots & \vdots \\ -\lambda x_n K_n g_{n,1} & -\lambda x_n K_n g_{n,2} & \cdots & -\lambda x_n K_n g_{n,n} & -x_n K_n\omega_n & -x_n K_n\theta_n \\ 1 & 1 & \cdots & 1 & 0 & 0 \end{pmatrix} \\
&= \begin{pmatrix} -x_1 K_1 & 0 & \cdots & 0 \\ 0 & -x_2 K_2 & \vdots & \vdots \\ \vdots & \cdots & \ddots & \vdots \\ 0 & \cdots & \cdots & 1 \end{pmatrix} \begin{pmatrix} \lambda g_{1,1} & \cdots & \lambda g_{1,n} & \omega_1 & \theta_1 \\ \vdots & \ddots & \vdots & \vdots & \vdots \\ \lambda g_{n,1} & \cdots & \lambda g_{n,n} & \omega_n & \theta_n \\ 1 & \cdots & 1 & 0 & 0 \end{pmatrix} \\
&= \begin{pmatrix} -x_1 K_1 & 0 & \cdots & 0 \\ 0 & -x_2 K_2 & \vdots & \vdots \\ \vdots & \cdots & \ddots & \vdots \\ 0 & \cdots & \cdots & 1 \end{pmatrix} \begin{pmatrix} \lambda G & \omega & \theta \\ e^T & 0 & 0 \end{pmatrix} \qquad (2.30)
\end{aligned}
$$

where

$$\omega_i = \lambda \left(\frac{\partial \ln \gamma_i}{\partial T} \right)_{x,P} + \left(\lambda + \frac{1-\lambda}{\gamma_i} \right) \frac{d \ln P_i^s}{dT}, \qquad (2.31)$$

$$\theta_i = 1 - \frac{1}{\gamma_i}, \qquad (2.32)$$

$$(G)_{i,j} = g_{i,j} = \left(\frac{\partial \ln \gamma_i}{\partial x_j} \right)_{x_{k[j]},T,P}, \text{ and} \qquad (2.33)$$

$$e^T = (1\ 1\ \ldots\ 1)\,. \qquad (2.34)$$

Note that G is the Hessian matrix of the excess Gibbs free energy of mixing (scaled by RT) and thus,

1. $G = G^T \in \mathbb{R}^{n \times n}$ and

2. By the Gibbs-Duhem relation, the rank of G is at most $n-1$.

The first term in the factored Jacobian matrix, equation (2.30), is a diagonal matrix which is nonsingular in the interior of $\mathcal{S}$. Thus, it is sufficient to look only at the second term to examine the rank of the overall Jacobian matrix.

Theorem 1 *Zero will be a regular value of $\bar{F}$ in the interior of $\mathcal{S}$ if*

1. *rank$(G) \geq n-2$,*
2. *$\bar{H}_i^E < \Delta H_i^{vap}$ for $i = 1, \ldots, n$ ($\bar{H}_i^E$ and ΔH_i^{vap} are the excess partial molar enthalpy and the molar heat of evaporation of component i, respectively), and*
3. *$x^T(\theta - \kappa\omega) \neq 0$ for any constant κ where x is the current composition at which θ and ω are evaluated (this constraint is only necessary if rank$(G) = n-2$)*

(assuming the vapor may be treated as an ideal gas).

Proof. Zero will be a regular value of $\bar{F}$ in the interior of $\mathcal{S}$ if

$$\text{rank} \begin{pmatrix} \lambda G & \omega & \theta \\ e^T & 0 & 0 \end{pmatrix} = n+1$$

Suppose $u \in \mathcal{R}(G) = \mathcal{R}(G^T)$. Then $u = \sum_{i=1}^n c_i g_i$ where $G = [g_1 \; g_2 \; \cdots \; g_n]$ and

$$g_k = \begin{pmatrix} g_{1,k} \\ \vdots \\ g_{n,k} \end{pmatrix} \quad \forall k = 1, \ldots, n.$$

By the Gibbs-Duhem relation, $x^T u = \sum_{i=1}^n c_i x^T g_i = 0$ since $x^T g_i = 0$ for all i where x is the current liquid composition on the curve . However, $x^T e \neq 0$ and if $\bar{H}_i^E < \Delta H_i^{vap}$ for all i then $\omega > 0$ in the interior of $\mathcal{S}$ and $x^T\omega \neq 0$. This implies the following

$$\text{rank} \begin{pmatrix} \lambda G \\ e^T \end{pmatrix} = \text{rank}(G) + 1$$

$$\text{rank} \begin{pmatrix} \lambda G & \omega \\ e^T & 0 \end{pmatrix} = \text{rank}(G) + 2$$

This completes the proof if $\text{rank}(G) = n - 1$. If condition (3) also holds,

$$\text{rank} \begin{pmatrix} \lambda G & \omega & \theta \\ e^T & 0 & 0 \end{pmatrix} = \max\{\text{rank}(G) + 3, n + 1\}.$$

Corollary 1 *There will be no isolated azeotropes nor multiple azeotropes that bifurcate from isolated azeotropes if the rank of G is equal to $n-1$ and condition (2) holds in the interior of $\mathcal{S}$.*

There must be a turning point in the interior of $\mathcal{S}$ for there to be an isolated branch (i.e., a branch not connected to a lower dimensional branch) connecting a pair of azeotropes that bifurcate off an isolated azeotrope (see Figure 2-4). The corollary above excludes turning points in λ in the interior of $\mathcal{S}$ and thus, these isolated branches and corresponding azeotropes are not possible.

The condition for non-negativity of ω is derived as follows:

$$\begin{aligned} \omega_i &= \lambda \left(\frac{\partial \ln \gamma_i}{\partial T} \right)_{x,P} + \left(\lambda + \frac{1-\lambda}{\gamma_i} \right) \frac{d \ln P_i^s}{dT} \\ &= -\lambda \frac{\bar{H}_i^E}{RT^2} + \left(\lambda + \frac{1-\lambda}{\gamma_i} \right) \frac{\Delta H_i^{vap}}{RT^2 \Delta Z_i^{vap}} \end{aligned}$$

where $\bar{H}_i^E$ is the excess partial molar enthalpy of mixing for species i, ΔH_i^{vap} is the molar heat of evaporation for species i, and ΔZ_i^{vap} is the change in the compressibility factor associated with evaporation for species i (which is very close to unity at low to moderate pressure). Since, in the interior of $\mathcal{S}$, $0 < \lambda < \lambda + (1-\lambda)/\gamma_i$, $\omega_i > 0$ if $\bar{H}_i^E < \Delta H_i^{vap}$. This is not at all an unreasonable assumption.

The exact conditions under which the rank of G is less than $n-2$ in the interior of $\mathcal{S}$, if this may occur at all, have not been determined. However, the only physically meaningful case where the rank of G is equal $n-2$ is at a non-elementary arm-chair azeotrope (of which the isolated azeotrope is an example), which are extremely rare and will occur only at specific values of pressure. Furthermore, arm-chair azeotropes which are not isolated azeotropes are readily computed with this method. Within the interior of $\mathcal{S}$, the phase equilibrium is modeled using pseudo K-values which

are convex combinations of the nonideal and ideal K-values. Thus, the "pseudo equilibrium" is not likely to be significantly different from the actual phase equilibrium except for the most pathological cases.

If the approach described above is applied to systems containing heteroazeotropes, solutions are found at $\lambda = 1$ that either don't satisfy the necessary conditions for azeotropy due to a violation of the non-negativity of the mole fractions constraint or they satisfy the necessary conditions, but a subsequent phase stability test indicates the liquid at the computed composition, temperature, and pressure will split into multiple liquid phases. These *spurious* homogeneous solutions, discussed in detail in the following section, correspond to actual heteroazeotropes present in the mixture.

The spurious solutions themselves do not assist in the computation of the corresponding heteroazeotropes: the composition and temperature of these spurious solutions, even when within their respective bounds, are significantly different from the actual heteroazeotrope composition and temperature and there is no systematic way to initialize the additional variables associated with the heterogeneous model. However, sections 2.3 and 2.4 describe how two additional homotopy maps can be constructed so that heteroazeotropes can be computed using the spurious homogeneous azeotropes and branches.

2.3 Spurious Homogeneous Azeotropes

In this section, the existence of spurious homogeneous azeotropes is considered. It will be shown in the following section that the spurious azeotropes need not be computed (only the spurious branches are required to obtain the heteroazeotropes), however, the following analysis provides some useful insights and an alternative, very efficient mechanism for computing certain heteroazeotropes. The existence of spurious homogeneous azeotropes will be analyzed by examining the following homotopy map:

$$h_i^s(x,\lambda) = \lambda(x_i - y_i(x)) + (1-\lambda)(x_i - y_i^o(x)) \quad i = 1,\ldots,n-1 \tag{2.35}$$

where $x \in \mathbb{R}^n$ is the liquid composition, $y(x) \in \mathbb{R}^n$ is the vapor composition in equilibrium with the liquid computed using an appropriate vapor-liquid equilibrium (VLE) model, $y^o(x) \in \mathbb{R}^n$ is the vapor composition in equilibrium with the liquid computed using an appropriate vapor-liquid-liquid equilibrium (VLLE) model (x is the overall liquid composition in this case), and $\lambda \in \mathbb{R}$ is the homotopy parameter. The equilibrium vapor composition and temperature as well as the additional liquid compositions and phase fraction in the heterogeneous case are fully determined by specifying x and pressure. System (2.35) is $n-1$ equations in terms of n unknowns ($n-1$ independent liquid compositions and λ). The summation of mole fraction constraint was used to eliminate a mole fraction element, rather than treating the n compositions as independent and explicitly handling the summation constraint as in the other homotopies described in this chapter. At $\lambda = 0$, the homotopy map reduces to $h_i^s(x,0) = x_i - y_i^o(x)$, $i = 1, \ldots, n-1$, the necessary conditions for heteroazeotropy. At $\lambda = 1$, the homotopy map reduces to $h_i^s(x,1) = x_i - y_i(x)$, $i = 1, \ldots, n-1$, the necessary conditions for homogeneous azeotropy. This homotopy map is related to the residue curves of homogeneous and heterogeneous mixtures. The homogeneous residue curves are defined by the dynamical system

$$\frac{dx_i}{d\xi} = x_i - y_i(x) \qquad i = 1, \ldots, n-1 \tag{2.36}$$

and the heterogeneous residue curves are defined by

$$\frac{dx_i}{d\xi} = x_i - y_i^o(x) \qquad i = 1, \ldots, n-1. \tag{2.37}$$

The notation is slightly changed from that in chapter 1 to remain consistent in this chapter. As stated in chapter 1, fixed-points of these dynamical systems are the pure components, azeotropes, and heteroazeotropes and can only be stable or unstable nodes, saddles, or non-elementary arm-chair fixed-points [35, 74]. Furthermore, it is shown in [74] that the number of different types of fixed-points is further restricted in the heterogeneous case by the number of components present and number of liquid phases in equilibrium. The homotopy curve defined by $h^s(x,\lambda) = 0$ describes how

the fixed-points of the following dynamical system vary with λ:

$$\frac{dx_i}{d\xi} = \lambda(x_i - y_i(x)) + (1-\lambda)(x_i - y_i^o(x)) \qquad i = 1, \ldots, n-1. \tag{2.38}$$

Figure 2-6 contains a schematic of the relationship between the fixed-points of the three dynamical systems above.

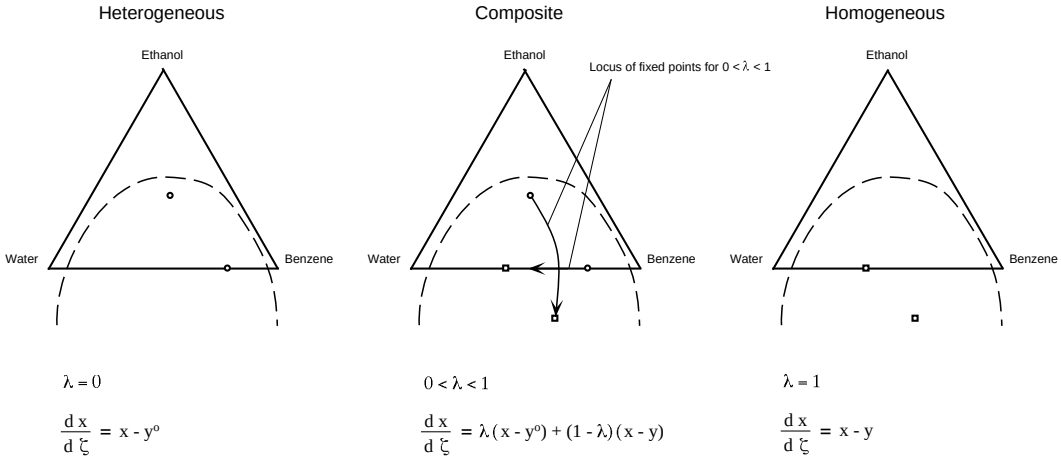

Figure 2-6: Schematic of relationship between the fixed-points of the dynamical systems (2.36), (2.37), and (2.38)

.

Suppose $\tilde{x}^o$ satisfies the necessary and sufficient conditions for heteroazeotropy and $\tilde{x}$ satisfies the necessary conditions for homogeneous azeotropy (and is thus, the composition of a spurious homogeneous azeotrope). We are interested in under what conditions a smooth path defined by $h^s(x, \lambda) = 0$ connects $(\tilde{x}^o, 0)$ and $(\tilde{x}, 1)$. The homotopy map can be expressed as:

$$h_i^s(x, \lambda) = x_i \left[\lambda(1 - K_i) + (1-\lambda)(1 - K_i^o)\right] \quad i = 1, \ldots, n-1 \tag{2.39}$$

where K_i is the normal VLE K-value and

$$\frac{1}{K_i^o} = s\frac{1}{K_i^I} + (1-s)\frac{1}{K_i^{II}} \tag{2.40}$$

is the overall K-value for the heterogeneous system.

The Jacobian of (2.39) is

$$\nabla h^s(x,\lambda) = \left(\nabla_x h^s \;\middle|\; \frac{\partial h^s}{\partial \lambda}\right) \tag{2.41}$$

$$\tag{2.42}$$

where

$$\begin{aligned}
(\nabla_x h^s)_{i,j} &= \Delta_i \delta_{i,j} - x_i \left[\lambda \frac{\partial K_i}{\partial x_j} + (1-\lambda)\frac{\partial K_i^o}{\partial x_j}\right], && (2.43)\\
\Delta_i &= \lambda(1-K_i) + (1-\lambda)(1-K_i^o), \text{ and} && (2.44)\\
\left(\frac{\partial h^s}{\partial \lambda}\right)_i &= -x_i\left(K_i - K_i^o\right). && (2.45)
\end{aligned}$$

Let $c(\xi) \in (h^s)^{-1}(0)$ denote the homotopy path where ξ is some suitable parameterization. As before, define a k-ary branch, denoted by $c_{(k)}(\xi)$, as a connected component of $(h^s)^{-1}(0)$ such that $x_i \neq 0$ for $1,\ldots,k$ and $x_i = 0$ for $i = k+1,\ldots,n-1$. On $c_{(k)}(\xi)$, $\Delta_i = 0$ for all $i = 1,\ldots,k$. On this path, the Jacobian can be written as

$$\nabla_x h^s(c_{(k)}(\xi)) = \begin{pmatrix}
-x_1\mathbb{K}_{1,1} & \cdots & -x_1\mathbb{K}_{1,k+1} & \cdots & -x_1\mathbb{K}_{1,n-1} \\
\vdots & \ddots & \vdots & \ddots & \vdots \\
-x_k\mathbb{K}_{k,1} & \cdots & -x_k\mathbb{K}_{k,k+1} & \cdots & -x_k\mathbb{K}_{k,n-1} \\
0 & \cdots & \Delta_{k+1} & \cdots & 0 \\
\vdots & \ddots & \vdots & \ddots & \vdots \\
0 & \cdots & 0 & \cdots & \Delta_{n-1}
\end{pmatrix}$$

and

$$\frac{\partial h^s}{\partial \lambda}\Big|_{c_{(k)}(\xi)} = \begin{pmatrix}
-x_1(K_1 - K_1^o) \\
\vdots \\
-x_k(K_k - K_k^o) \\
0 \\
\vdots \\
0
\end{pmatrix}$$

where $\mathbb{K}_{i,j} = \lambda K_{i,j} + (1-\lambda)K^o_{i,j}$. As with the other homotopy branches discussed in this paper, $c(\xi)$ exhibits transcritical bifurcation points, corresponding to intersections between k-ary and $(k+1)$-ary branches, where a mole fraction element on the $(k+1)$-ary branch and the corresponding Δ_i on the k-ary branch cross zero. (The proof is very similar to the one shown in Appendix B for the bifurcation points on the homogeneous branches.) This observation will be shown to be particularly useful in the heteroazeotrope finding algorithm.

Let $s(x)$ denote the liquid phase fraction computed through a VLLE calculation with a fixed pressure and overall liquid composition x. Define the following set:

$$\Omega = \{z \in \mathbb{R}^{n-1} \mid x^T = (z^T, 1 - \textstyle\sum_{i=1}^{n-1} z_i) \text{ and } 0 \leq s(x) \leq 1\}.$$

At a specified pressure, this set is constant and does not change along the homotopy path (it is simply the region where a liquid-liquid phase split is predicted by the necessary conditions for vapor-liquid-liquid equilibrium). Let $\tilde{c}(0) = (\tilde{x}^o, 0) \in \Omega \times \mathbb{R}$ where $\tilde{x}^o$ satisfies the necessary and sufficient conditions for heteroazeotropy. We are interested in under what conditions a spurious homogeneous azeotrope will exist when a corresponding heteroazeotrope exists. Thus, we can assume $\tilde{c}(\xi) \subset \Omega \times \mathbb{R}$ for $-\infty < \xi < +\infty$. This is due to the fact that if $\tilde{c}(\xi)$ leaves $\Omega \times \mathbb{R}$ at a point $\tilde{c}(\bar{\xi}) = (\bar{x}, \bar{\lambda})$, $0 < \bar{\lambda} < 1$, then $s(\bar{x}) = 0$ (or 1) and $\bar{x}$ will satisfy the necessary conditions for azeotropy and thus, be the spurious homogeneous azeotrope we are looking for. Sufficient conditions for the homotopy path to cross the level set $\Omega \times \{1\}$ at a point $(\tilde{x}, 1)$ (where $\tilde{x}$ is a spurious homogeneous azeotrope) are:

1. Zero is a regular value of h in $(\Omega - \mathcal{Z}) \times [0,1]$ and

2. Multiple heteroazeotropy does not occur

where $\mathcal{Z} = \{z \in \mathbb{R}^{n-1} \mid x^T = (z^T, 1 - \sum_{i=1}^{n-1} z_i)$ and $x_i = 0$ for at least one $i\}$ (this set is removed from Ω to exclude the case of transcritical bifurcations onto other branches when a mole fraction element crosses zero). If the first condition holds then $\tilde{c}(\xi)$ will be diffeomorphic to a circle or the real line and will either cross $\Omega \times \{1\}$

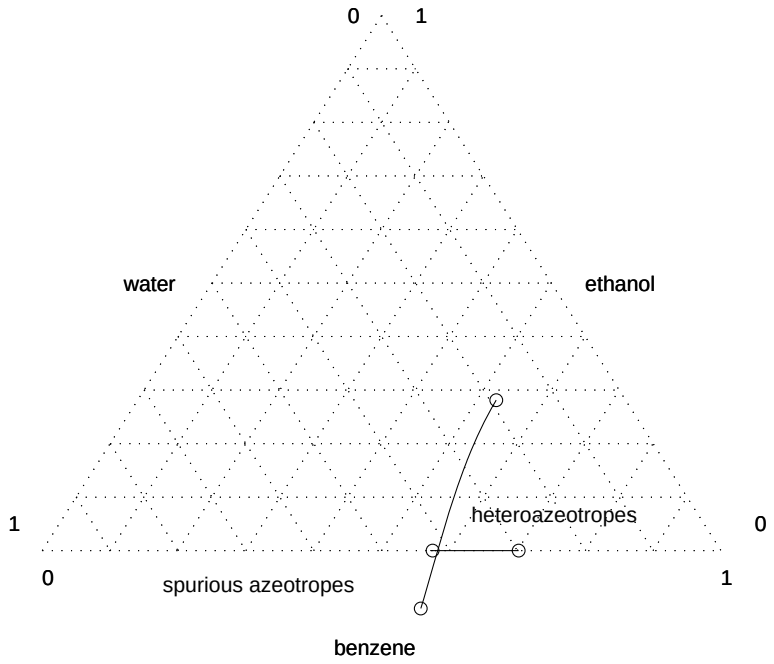

Figure 2-7: Homotopy paths connecting heteroazeotropes to spurious homogeneous azeotropes for the benzene, ethanol, and water system at 1.0 bar.

or turn around before the λ-component of $\tilde{c}(\xi)$ reaches unity. Condition (2) prevents the latter from occurring.

Figure 2-7 contains a plot of the homotopy paths connecting the heteroazeotropes to the corresponding spurious homogeneous azeotropes in the benzene, ethanol, and water system at a pressure of one bar. In this figure, the ternary branch intersects the binary branch where the ethanol mole fraction crosses zero. The binary heteroazeotrope is a saddle when the system is viewed as a three component mixture and the corresponding spurious azeotrope is an unstable node (they are both unstable nodes when the system is viewed as a binary benzene-water mixture). This difference in stability is due to the fact that a ternary homotopy branch experiences a transcritical bifurcation through the binary branch. Suppose the two conditions listed above hold on all homotopy paths connecting heteroazeotropes to spurious homogeneous azeotropes present in an n-component mixture[1]. In addition, suppose we are currently on a k-ary branch. At any value of λ, the matrix $\nabla_x h^s$ has $n-k-1$

[1]Actually, condition (1) above can be relaxed: zero need only be a regular value for the homotopy map, $h^s(x,\lambda)$, in the set $(\Omega \cap \mathcal{C}) \times [0,1)$.

eigenvalues equal to Δ_i, $i = k+1, \ldots, n-1$. A sufficient condition for the existence of a higher dimensional branch crossing the k-ary branch is

$$\Delta_i(\lambda = 0) \cdot \Delta_i(\lambda = 1) \leq 0 \text{ for some } i = k+1, \ldots, n. \tag{2.46}$$

(The Δ corresponding to the component removed using the summation of mole fractions constraint must also be examined.) This observation can be exploited in the heteroazeotrope finding algorithm. Given a k-ary heteroazeotrope and its spurious homogeneous azeotrope, compare the signs of $\Delta_i(\lambda = 0)$ and $\Delta_i(\lambda = 1)$ for $i = k+1, \ldots, n$. If any of these quantities differ in sign then a $(k+1)$-ary heteroazeotrope may exist. In addition, we know with which additional component this new heteroazeotrope is formed and that the spurious homogeneous azeotrope lies outside $\mathcal{C}$. Furthermore, this provides an efficient means of computing the higher dimensional heteroazeotrope. Suppose (2.46) is satisfied for some $k+1 \leq j \leq n$. The homotopy path is tracked from the k-ary heteroazeotrope to determine where $\Delta_j(\xi)$ crosses zero. A point on the higher dimensional homotopy branch is then computed by solving the following system of equations:

$$\begin{pmatrix} x_1\Delta_1 \\ \vdots \\ x_k\Delta_k \\ x_j - \varepsilon \end{pmatrix} = 0$$

for some sufficiently small $\varepsilon > 0$. This branch is then tracked in the direction of decreasing λ to the heteroazeotrope at $\lambda = 0$. Obviously, we will need to compute binary heteroazeotropes in a different manner in order to obtain the higher dimensional heteroazeotropes using the approach described above. The following section describes another approach for the computation of the heteroazeotropes.

2.4 Computation of Heterogeneous Azeotropes

This section describes our extension of Fidkowski's approach for the computation of the heteroazeotropes present in a multicomponent mixture. The description of the approach is followed by some analysis. The case of two liquid phases in equilibrium with vapor is discussed below. However, the approach can be readily extended to handle any number of liquid phases in equilibrium.

The first step is to derive a heterogeneous homotopy map. Since there is no equivalent to Raoult's Law for vapor-liquid-liquid equilibrium, equation (2.6) cannot be used as a starting point. However, starting with equation (2.7) the following homotopy map can be derived:

$$F^o(x, y^*, x^I, x^{II}, T, s, \lambda) = \begin{pmatrix} x - y^* \\ y_1^* - \left[\lambda K_1^I + (1-\lambda) P_1^s/P\right] x_1^I \\ \vdots \\ y_n^* - \left[\lambda K_n^I + (1-\lambda) P_n^s/P\right] x_n^I \\ \lambda\left[\gamma_1^I x_1^I - \gamma_1^{II} x_1^{II}\right] + (1-\lambda)\left[x_1^I - x_1^{II}\right] \\ \vdots \\ \lambda\left[\gamma_n^I x_n^I - \gamma_n^{II} x_n^{II}\right] + (1-\lambda)\left[x_n^I - x_n^{II}\right] \\ x - s x^I - (1-s) x^{II} \\ \sum_{i=1}^n y_i^* - 1 \\ \sum_{i=1}^n x_i^I - 1 \end{pmatrix} \tag{2.47}$$

where $x \in \mathbb{R}^n$ is the overall liquid composition, $y^* \in \mathbb{R}^n$ is the perturbed vapor composition, $x^I, x^{II} \in \mathbb{R}^n$ are the liquid compositions of liquid phases I and II, respectively, s is the liquid phase fraction (the fraction of the total number of moles of liquid in liquid phase I), T is temperature, P is pressure, and $\lambda \in \mathbb{R}$ is the homotopy parameter. As in the homogeneous case, the perturbed vapor composition

can be eliminated from (2.47) using the first n equations:

$$\bar{F}^o(x, x^I, x^{II}, T, s, \lambda) = \begin{pmatrix} x_1 - \left[\lambda K_1^I + (1-\lambda)P_1^s/P\right] x_1^I \\ \vdots \\ x_n - \left[\lambda K_n^I + (1-\lambda)P_n^s/P\right] x_n^I \\ \lambda\left[\gamma_1^I x_1^I - \gamma_1^{II} x_1^{II}\right] + (1-\lambda)\left[x_1^I - x_1^{II}\right] \\ \vdots \\ \lambda\left[\gamma_n^I x_n^I - \gamma_n^{II} x_n^{II}\right] + (1-\lambda)\left[x_n^I - x_n^{II}\right] \\ x - sx^I - (1-s)x^{II} \\ \sum_{i=1}^n x_i - 1 \\ \sum_{i=1}^n x_i^I - 1 \end{pmatrix}. \qquad (2.48)$$

Similar to the homogeneous case, at constant pressure setting expression (2.48) to zero defines a 1-manifold in $(x, x^I, x^{II}, T, s, \lambda)$-space. This curve will be referred to as a heterogeneous homotopy path or branch or simply as a heterogeneous branch. This form of the heterogeneous homotopy map has the following property: when the liquid phase fraction s on the heterogeneous branch crosses zero or one, a *projection* of the heterogeneous branch will intersect the corresponding spurious homogeneous branch (see Figure 2-8). As stated earlier in this chapter, a spurious homogeneous azeotrope is a solution to the necessary conditions of azeotropy that fails a stability test or lies outside the physical bounds of the variables (i.e., mole fractions outside the range of zero and unity). The basic heteroazeotrope finding algorithm can be summarized in the following steps:

1. Compute the homogeneous azeotropes using the homogeneous homotopy map,

2. Test all homogeneous azeotropes for stability (this also indicates which azeotropes are spurious),

3. Retrace all spurious branches and search for the points of intersection with a projection of a heterogeneous branch,

4. From the intersection points, track the heterogeneous branches to the het-

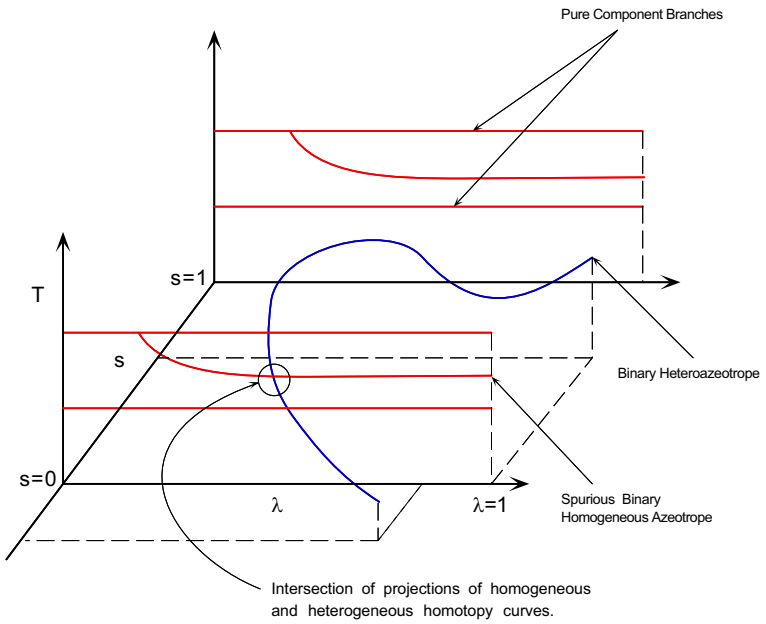

Figure 2-8: Schematic of the intersection of spurious homogeneous branch and projection of heterogeneous branch.

eroazeotropes using the heterogeneous homotopy map, and

5. Test all heterogeneous solutions for stability.

Since only the necessary conditions are satisfied at $\lambda = 1$, a phase stability test must be performed on all solutions. The homogeneous azeotropes are easily obtained through a series of bifurcations from some of the pure component branches. Criteria (2.46) should be checked at each heteroazeotrope computed. If this condition is satisfied, the approach described in the previous section provides an alternative, independent means of computing the higher dimensional heteroazeotrope.

At the intersection points, the common components of the homogeneous and heterogeneous branches are equal. The following additional constraints are satisfied on the heterogeneous branch (when $s = 1$):

$$\lambda(\gamma_j x_j - \gamma_j^{II} x_j^{II}) + (1-\lambda)(x_j - x_j^{II}) = 0 \quad j = 1, \ldots, n \tag{2.49}$$

$$\sum_{i=1}^{n} x_i^{II} = 1. \tag{2.50}$$

The heterogeneous homotopy map was constructed so that it can be assumed the liq-

uid phase fraction s crosses one. As the spurious homogeneous branches are retraced, equations (2.49)-(2.50) are monitored to determine if a root exists in the current continuation step. A root exclusion test based on interval arithmetic [81] is employed to make this search robust and efficient. This algorithm is described in detail in the implementation section later in this chapter. If a root is identified in a neighborhood of the current continuation point, a bounded Newton's method is used to compute the missing components associated with the heterogeneous branch (x^{II}). After computing the intersection point, the heterogeneous homotopy map is used to track the path from the intersection point to the heteroazeotrope at $\lambda = 1$.

2.4.1 Analysis

There are two classes of bifurcations that are important in the heterogeneous case: bifurcations onto higher dimensional heterogeneous branches and intersections with spurious homogeneous branches. Both of these classes of bifurcations are analyzed in this section. This section is concluded with a discussion of under what conditions all heteroazeotropes will be computed using this approach.

In order to make the following Jacobian derivation and analysis clearer,

$$\bar{F}^o(x, x^I, x^{II}, T, s, \lambda)$$

is partitioned as follows:

$$\bar{F}^o(x, x^I, x^{II}, T, s, \lambda) = \begin{pmatrix} \bar{F}_1^o \\ \bar{F}_2^o \\ \bar{F}_3^o \\ \bar{F}_4^o \\ \bar{F}_5^o \end{pmatrix} \tag{2.51}$$

where

$$
\begin{aligned}
(\bar{F}_1^o)_i &= x_i - \left[\lambda K_i^I + (1-\lambda)P_i^s/P\right] x_i^I \quad i = 1, \ldots, n, && (2.52)\\
(\bar{F}_2^o)_i &= \lambda\left[\gamma_i^I x_i^I - \gamma_i^{II} x_i^{II}\right] + (1-\lambda)\left[x_i^I - x_i^{II}\right] \quad i = 1, \ldots, n, && (2.53)\\
\bar{F}_3^o &= x - sx^I - (1-s)x^{II}, && (2.54)\\
\bar{F}_4^o &= \sum_{i=1}^{n} x_i - 1, \text{ and} && (2.55)\\
\bar{F}_5^o &= \sum_{i=1}^{n} x_i^I - 1. && (2.56)
\end{aligned}
$$

The Jacobian matrix can then be expressed as follows

$$
\nabla \bar{F}^o(x,x^I,x^{II},T,s,\lambda) = \begin{pmatrix}
I & \partial \bar{F}_1^o/\partial x^I & 0 & \partial \bar{F}_1^o/\partial T & 0 & \partial \bar{F}_1^o/\partial \lambda \\
0 & \partial \bar{F}_2^o/\partial x^I & \partial \bar{F}_2^o/\partial x^{II} & \partial \bar{F}_2^o/\partial T & 0 & \partial \bar{F}_2^o/\partial \lambda \\
I & -sI & (s-1)I & 0 & x^{II}-x^I & 0 \\
e^T & 0 & 0 & 0 & 0 & 0 \\
0 & e^T & 0 & 0 & 0 & 0
\end{pmatrix} \qquad (2.57)
$$

where

$$
\begin{aligned}
\left(\frac{\partial \bar{F}_1^o}{\partial x^I}\right)_{i,j} &= -\left[\lambda K_i^I + (1-\lambda)P_i^s/P\right]\delta_{i,j} - \lambda x_i^I K_{i,j}^I, && (2.58)\\
\left(\frac{\partial \bar{F}_2^o}{\partial x^I}\right)_{i,j} &= \lambda\left[x_i^I \frac{\partial \gamma_i^I}{\partial x_j^I} + \gamma_i^I \delta_{i,j}\right] + (1-\lambda)\delta_{i,j}, && (2.59)\\
\left(\frac{\partial \bar{F}_2^o}{\partial x^{II}}\right)_{i,j} &= -\lambda\left[x_i^{II} \frac{\partial \gamma_i^{II}}{\partial x_j^{II}} + \gamma_i^{II} \delta_{i,j}\right] - (1-\lambda)\delta_{i,j}, && (2.60)\\
\left(\frac{\partial \bar{F}_1^o}{\partial T}\right)_{i} &= -\left[\lambda \frac{\partial K_i^I}{\partial T} + \frac{(1-\lambda)}{P}\frac{dP_i^s}{dT}\right] x_i^I, && (2.61)\\
\left(\frac{\partial \bar{F}_2^o}{\partial T}\right)_{i} &= \lambda\left[x_i^I \frac{\partial \gamma_i^I}{\partial T} - x_i^{II} \frac{\partial \gamma_i^{II}}{\partial T}\right], && (2.62)\\
\left(\frac{\partial \bar{F}_1^o}{\partial \lambda}\right)_{i} &= -\left[K_i^I - \frac{P_i^s}{P}\right] x_i^I, && (2.63)\\
\left(\frac{\partial \bar{F}_2^o}{\partial \lambda}\right)_{i} &= \left[\gamma_i^I x_i^I - \gamma_i^{II} x_i^{II}\right] - \left[x_i^I - x_i^{II}\right], && (2.64)
\end{aligned}
$$

and the i and j indices above vary from 1 to n. Similar to the homogeneous case, define

$$\begin{aligned}
\bar{K}_j^I &= \lambda K_j^I + (1-\lambda)P_j^s/P = 1 - \alpha_j, && (2.65)\\
\bar{\gamma}_j^I &= \lambda \gamma_j^I + (1-\lambda), && (2.66)\\
\bar{\gamma}_j^{II} &= \lambda \gamma_j^{II} + (1-\lambda), && (2.67)\\
\beta_j &= \lambda \left(\frac{\partial K_i^I}{\partial T}\right)_{x,y,P} + \frac{(1-\lambda)}{P}\frac{dP_i^s}{dT}, \text{ and} && (2.68)\\
\phi_j &= K_i^I - \frac{P_i^s}{P}. && (2.69)
\end{aligned}$$

Intersections with other Heterogeneous Branches

Let $\bar{c}^o_{(k)}(\xi) = (x(\xi), x^I(\xi), x^{II}(\xi), T(\xi), s(\xi), \lambda(\xi)) \in (\bar{F}^o_{(k)})^{-1}(0)$ denote a heterogeneous branch where $x(\xi)$, $x^I(\xi)$, $x^{II}(\xi)$ each have k nonzero elements. Suppose we are currently on a k-ary branch of an n component system such that $x_i \neq 0$, $i = 1, \ldots, k$, and $x_i = 0$, $i = k+1, \ldots, n$. Without loss of generality, suppose for some $\tilde{\xi}$, $x_k(\tilde{\xi}) = 0$. Then $x_k^I(\tilde{\xi}) = x_k^{II}(\tilde{\xi}) = 0$ and

$$\bar{F}^o_{(k-1)}(\mathbb{P}_{(k-1)}\bar{c}^o_{(k)}(\tilde{\xi})) = \mathbb{P}_{(k-1)}\bar{F}^o_{(k)}(\bar{c}^o_{(k)}(\tilde{\xi})) = 0$$

where $\mathbb{P}_{(k-1)}$ is a projection matrix that removes all elements associated with x_k, x_k^I, and x_k^{II} from $\bar{c}^o_{(k)}$ and $\bar{F}^o_{(k)}$ (i.e., element k, $x_k - \bar{K}_k^I x_k^I$, element $2k$, $\bar{\gamma}_k^I x_k^I - \bar{\gamma}_k^{II} x_k^{II}$, and

element $3k$, $x_k - sx_k^I - (1-s)x_k^{II}$). Furthermore,

$$\left(\frac{\partial \bar{F}_1^o}{\partial x^I}\right)_{k,j} = -\left[\lambda K_k^I + (1-\lambda)P_k^s/P\right]\delta_{k,j} = -\bar{K}_k^I \delta_{k,j}, \tag{2.70}$$

$$\left(\frac{\partial \bar{F}_2^o}{\partial x^I}\right)_{k,j} = \left[\lambda\gamma_k^I + (1-\lambda)\right]\delta_{k,j} = \bar{\gamma}_k^I \delta_{k,j}, \tag{2.71}$$

$$\left(\frac{\partial \bar{F}_2^o}{\partial x^{II}}\right)_{k,j} = -\left[\lambda\gamma_k^{II} + (1-\lambda)\right]\delta_{k,j} = -\bar{\gamma}_k^{II} \delta_{k,j}, \tag{2.72}$$

$$\left(\frac{\partial \bar{F}_1^o}{\partial T}\right)_k = 0, \tag{2.73}$$

$$\left(\frac{\partial \bar{F}_2^o}{\partial T}\right)_k = 0, \tag{2.74}$$

$$\left(\frac{\partial \bar{F}_1^o}{\partial \lambda}\right)_k = 0, \text{ and} \tag{2.75}$$

$$\left(\frac{\partial \bar{F}_2^o}{\partial \lambda}\right)_k = 0. \tag{2.76}$$

This point may correspond to a transcritical bifurcation point where the k-ary branch intersects a $(k-1)$-ary branch. The first step in proving this is a transcritical point is to show that rank $\nabla \bar{F}^o_{(k)}(\bar{c}^o_{(k)}(\tilde{\xi})) = 3n+1$ (i.e., rank deficient by one). When $x_k = 0$, there are three rows in the Jacobian matrix that have entries that become identically zero (as opposed to other entries that may happen to equal zero coincidentally), rows k, $2k$, and $3k$. Removing all zero entries common to all three of these rows, we have the following 'reduced' rows:

$$\begin{pmatrix} 1 & -\bar{K}_k^I & 0 \\ 0 & \bar{\gamma}_k^I & -\bar{\gamma}_k^{II} \\ 1 & -s & s-1 \end{pmatrix}.$$

If these rows are not independent then the original matrix will be rank deficient and

$$\det \begin{pmatrix} 1 & -\bar{K}_k^I & 0 \\ 0 & \bar{\gamma}_k^I & -\bar{\gamma}_k^{II} \\ 1 & -s & s-1 \end{pmatrix} = 0. \tag{2.77}$$

Expanding the determinant above,

$$(s-1)\bar{\gamma}_k^I - (s-\bar{K}_k^I)\bar{\gamma}_k^{II}. \tag{2.78}$$

On the k-ary branch, the following holds

$$\begin{aligned} \bar{K}_k^I &= x_k/x_k^I & (2.79)\\ s &= \frac{x_k - x_k^{II}}{x_k^I - x_k^{II}} = \frac{\bar{K}_k^I - \eta_k}{1-\eta_k}. & (2.80) \end{aligned}$$

where $\eta_k = x_k^{II}/x_k^I$. Furthermore, $\bar{K}_k^I(\xi)$, $s(\xi)$, and $\eta_k(\xi)$ remain smooth and continuous as ξ passes through $\tilde{\xi}$. In addition, rearranging equation (2.53) (with $i = k$) we see that $\bar{\gamma}_k^I = \bar{\gamma}_k^{II}\eta_k$ ($\eta_k \neq 1$ if $x^I \neq x^{II}$). Substituting these quantities into the expression for the determinant, we have

$$\begin{aligned} \bar{\gamma}_k^I(s-1) - \bar{\gamma}_k^{II}(s-\bar{K}_k^I) &= \bar{\gamma}_k^I(\frac{\bar{K}_k^I - \eta_k}{1-\eta_k} - 1) - \bar{\gamma}_k^{II}(\frac{\bar{K}_k^I - \eta_k}{1-\eta_k} - \bar{K}_k^I)\\ (1-\eta_k)(\bar{\gamma}_k^I(s-1) - \bar{\gamma}_k^{II}(s-\bar{K}_k^I)) &= \bar{\gamma}_k^I(\bar{K}_k^I - \eta_k - 1 + \eta_k) -\\ &\quad \bar{\gamma}_k^{II}(\bar{K}_k^I - \eta_k - \bar{K}_k^I + \bar{K}_k^I\eta_k)\\ &= \bar{\gamma}_k^I(\bar{K}_k^I - 1) - \bar{\gamma}_k^{II}\eta_k(\bar{K}_k^I - 1)\\ &= \bar{\gamma}_k^I(\bar{K}_k^I - 1) - \bar{\gamma}_k^I(\bar{K}_k^I - 1)\\ &= 0 \end{aligned}$$

and, thus, the original matrix is rank deficient (rank deficient by at least one by inspection). The derivation above simply shows that the dimension of the null-space of the Jacobian matrix is at least two. The following lemma contains necessary and sufficient conditions for a transcritical bifurcation point onto another heterogeneous branch.

Lemma 2 *A necessary condition for a bifurcation from a $(k-1)$-ary branch onto a k-ary branch at a point $\bar{c}^o(\tilde{\xi})$ is*

$$\left[s(\tilde{\xi}) - 1\right]\bar{\gamma}_j^I(\tilde{\xi}) - \left[s(\tilde{\xi}) - \bar{K}_j^I(\tilde{\xi})\right]\bar{\gamma}_j^{II}(\tilde{\xi}) = 0 \text{ for some } j \in \{k, \ldots, n\}. \tag{2.81}$$

A necessary condition for a bifurcation from a k-ary heterogeneous branch onto a $(k-1)$-ary branch at a point $\bar{c}^o(\tilde{\xi})$ is

$$x_j(\tilde{\xi}) = 0 \text{ for some } j \in \{1, \ldots, k\}. \tag{2.82}$$

Condition (2.82) becomes necessary and sufficient for a bifurcation from a k-ary branch onto a $(k-1)$-ary branch by adding the following:

1. $dx_j/d\xi\big|_{\xi=\tilde{\xi}} \neq 0$,
2. *rank* $\nabla \bar{F}^o(\tilde{\xi}) = N^o - 1$ *where* $N^o = 3n + 2$, *and*
3. $\partial F^o/\partial x_j\big|_{\xi=\tilde{\xi}} \in \mathcal{R}\left(\nabla_{[j]}\bar{F}^o(\tilde{\xi})\right)$ *where* $\nabla_{[j]}$ *denotes partial derivatives with respect to all variables except* x_j.

Proof. See Appendix C.

Intersections with Homogeneous Branches

In this section, the subscript (k) will be dropped since we are dealing with intersections with branches with the same number of nonzero elements in the mole fraction vectors (which will be greater than or equal to two). Suppose for some $\tilde{\xi}$, $e_s^T \bar{c}^o(\tilde{\xi}) = s(\tilde{\xi}) = 1$, then $x(\tilde{\xi}) = x^I(\tilde{\xi})$ (in general, $x^I(\tilde{\xi}) \neq x^{II}(\tilde{\xi})$) and

$$\nabla \bar{F}^o(x,x^I,x^{II},T,s,\lambda) = \begin{pmatrix} I & \partial \bar{F}_1^o/\partial x^I & 0 & \partial \bar{F}_1^o/\partial T & 0 & \partial \bar{F}_1^o/\partial \lambda \\ 0 & \partial \bar{F}_2^o/\partial x^I & \partial \bar{F}_2^o/\partial x^{II} & \partial \bar{F}_2^o/\partial T & 0 & \partial \bar{F}_2^o/\partial \lambda \\ I & -I & 0 & 0 & x^{II}-x & 0 \\ e^T & 0 & 0 & 0 & 0 & 0 \\ 0 & e^T & 0 & 0 & 0 & 0 \end{pmatrix}. \tag{2.83}$$

Furthermore, $\bar{F}_1^o(\bar{c}^o(\tilde{\xi}))$ and $\bar{F}_4^o(\bar{c}^o(\tilde{\xi}))$ are identical to the corresponding homogeneous homotopy map (with the same nonzero mole fraction elements), and thus, this point is an intersection of the projection of the heterogeneous branch onto (x, T, λ)-space and a spurious homogeneous branch. It can be readily seen that the last $n + 2$ rows

of (2.83) can be combined linearly to form a zero row and thus, the matrix is rank deficient by at least one at the intersection point. We only need to treat the case where $s = 1$ since the designation of liquid phases I and II is arbitrary. If from a given starting point the branch crossed $s = 0$ then if the two liquid phase compositions were swapped and the continuation was performed again from the original starting point the branch would cross $s = 1$. The homogeneous homotopy curve, $\bar{c}(\xi)$, can be thought of as a hypersurface in $(x, x^I, x^{II}, T, s, \lambda)$-space, where x^I, x^{II}, and s are free to take any value. An intersection point is a point where the heterogeneous homotopy curve, $\bar{c}^o(\xi)$ intersects this hypersurface.

The remainder of this section discusses under what conditions all heteroazeotropes will be computed. The proof is similar to the homogeneous case. A bounded region, $\mathcal{S}^o$, will be constructed in $(x, x^I, x^{II}, T, s, \lambda)$-space containing the heteroazeotrope on the side defined by $\lambda = 0$. The heterogeneous branch will move into $\mathcal{S}^o$ and if zero is a regular value of the heterogeneous map in this region, the branch will either leave $\mathcal{S}^o$ at a point corresponding to an intersection with a lower dimensional heterogeneous branch (a bifurcation point as defined in this paper) or at an intersection with a spurious homogeneous branch. If a bifurcation point is identified then the reasoning is continued in this lower dimensional space. If an intersection point is identified then the analysis described in the homogeneous azeotrope section is evoked.

If the heteroazeotrope exists and the Jacobian has maximal rank at this point then a smooth path $\bar{c}^o(\xi)$ will exist. Let $\bar{c}^o(\tilde{\xi})$ denote the heteroazeotrope. Similar to the homogeneous case, the following bounded, connected set can be defined:

$$\mathcal{S}^o = \mathcal{C} \times \mathcal{C} \times \mathcal{C} \times [T_{min}, T_{max}] \times [0, 1] \times (0, 1]\,. \tag{2.84}$$

The heteroazeotrope,

$$\bar{c}^o(\tilde{\xi}) = (x(\tilde{\xi}), x^I(\tilde{\xi}), x^{II}(\tilde{\xi}), T(\tilde{\xi}), s(\tilde{\xi}), \lambda(\tilde{\xi})),$$

is located on the side of $\mathcal{S}^o$ where $\lambda = 1$. If zero is a regular value of $\bar{F}^o$ in the interior of $\mathcal{S}^o$, the heterogeneous branch passing through $\bar{c}^o(\tilde{\xi})$ will leave $\mathcal{S}^o$ after finite ξ

(provided $d\bar{c}^o/d\xi$ is not tangent to $\mathcal{S}^o$ at $\tilde{\xi}$). Again, the positive ξ direction is the direction of decreasing λ at $\tilde{\xi}$. Furthermore, the heterogeneous branch will leave $\mathcal{S}^o$ in one of three ways:

1. $\bar{c}^o(\xi)$ will turn around and leave through the side of $\mathcal{S}^o$ where $\lambda = 1$,
2. $\bar{c}^o(\xi)$ will leave through a side of $\mathcal{S}^o$ where s equals zero or unity, or
3. $\bar{c}^o(\xi)$ will leave through a side of $\mathcal{S}^o$ where $x_i = 0$ for some $1 \leq i \leq n$.

Figure 2-9 contains a diagram of the three possible cases. The branch cannot cross

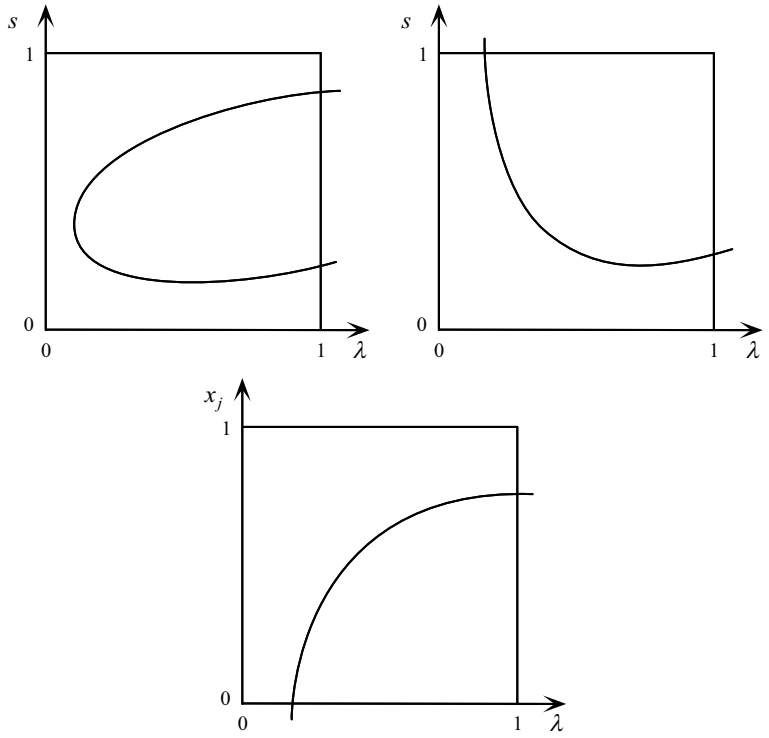

Figure 2-9: Three possible ways $\bar{c}^o(\xi)$ can leave $\mathcal{S}^o$.

$\lambda = 0$ due to the fact that there is no solution at this point for $n > 1$ and pure component heterogeneous branches are physically meaningless and degenerate. For the case where $n = 1$, the only solution is the trivial solution and the liquid phase fraction is indeterminant. The first case listed above corresponds to multiple heteroazeotropy. Although multiple heteroazeotropy is not physically possible[2], there may arise situations where one of the two heteroazeotropes is not physical. In this case, the

[2]In the homogeneous case, multiple azeotropy occurs when there are both positive and negative

heterogeneous branch will be isolated and the corresponding heteroazeotrope will not be computed using our approach. This is an extremely pathological case and it is very unlikely that it will occur. If the second case above occurs, the point at which the heterogeneous branch leaves $\mathcal{S}^o$ will correspond to an intersection with a spurious homogeneous branch. Provided this spurious homogeneous branch is obtainable from a pure component branch (see discussion on the computation of homogeneous azeotropes above), the corresponding heteroazeotrope will be computed using our approach. If the third case above occurs, the point at which the heterogeneous branch leaves $\mathcal{S}^o$ will correspond to an intersection with a lower dimensional heterogeneous branch provided the conditions in lemma 2 are satisfied. As with the homogeneous case, if a lower dimensional heterogeneous branch is obtained the same analysis is applied within this lower dimensional space. If the original heterogeneous branch is obtained through a series of bifurcations on lower dimensional heterogeneous branches then we must eventually reach a branch that is obtainable from a pure component branch. Appendix D contains a proof that, under reasonable assumptions, all binary heteroazeotropes will be obtained from pure component homogeneous branches. Appendix E contains a discussion of the rank of the Jacobian of the heterogeneous homotopy map.

Thus, there are three independent mechanisms by which heteroazeotropes may be obtained: through intersections with spurious homogeneous branches, through bifurcations from lower dimensional heterogeneous branches, and, if the spurious azeotrope lies outside $\mathcal{C}$, through the approach described in section 2.3. Numerical examples and implementation details are presented in chapter 4. In nearly every case examined, the heteroazeotropes ($k > 2$) were obtained through all three mechanisms.

---

deviations from Raoult's Law at various compositions. Since heteroazeotropy is associated with strong positive deviations from Raoult's Law an analogous situation is not likely to occur.

2.5 Algorithm

This chapter describes and analyzes a new approach for the computation of azeotropes and heteroazeotropes present in a multicomponent mixture. Azeotropes are obtained through a series of bifurcations on the homotopy paths defined by (2.8). The starting points for these homogeneous branches are $(x, T, \lambda) = (e_k, T_k^s, 0)$, $k = 1, \ldots, n$, where e_k denotes pure component k with boiling temperature T_k^s at the specified pressure. Heteroazeotropes are obtained using the heterogeneous homotopy map, (2.48). Starting points for the heterogeneous branches are obtained by two mechanisms, through intersections with spurious homogeneous branches and through bifurcations on lower dimensional heterogeneous branches. As a consequence of the analysis in this chapter, bifurcation and intersection points of interest for the computation of azeotropes and heteroazeotropes will occur in the range $0 < \lambda < 1$. Heteroazeotropes are also obtained through a third, independent mechanism described in section 2.3.

The algorithm for computing the homogeneous and heterogeneous azeotropes is described below. The algorithm described computes the azeotropes at a fixed pressure, however, computing the azeotropes at a fixed temperature is simply a matter of replacing temperature with pressure in the description below.

Three lists are employed in this algorithm. The first list, $\mathcal{AL}$, holds the homogeneous azeotropes, both stable and spurious. The elements of this list are data structures holding the value of the bifurcation points, (x^b, T^b, λ^b), the value of the azeotrope, (x, T), and a status field. These are referred to as *azeotrope data structures.* The second list, $\mathcal{HL}$, holds the heterogeneous azeotropes. The elements of this list are data structures holding the value of the intersection point, $(x^i, x^{II,i}, T^i, \lambda^i)$, the value of the bifurcation point, $(x^b, x^{I,b}x^{II,b}, T^b, s^b, \lambda^b)$, and the value of the heteroazeotrope, (x, x^I, x^{II}, T, s). These are referred to as *heteroazeotrope data structures.* The third list, $\mathcal{SL}$ contains the values of the thermodynamically stable homogeneous and heterogeneous azeotropes identified. These data structures are referred to as homogeneous and heterogeneous *solution data structures.*

As described earlier in this chapter, there are three independent mechanisms by

which the heteroazeotropes are obtained: 1) bifurcations on lower dimensional heterogeneous branches, 2) through the use of lower dimensional heteroazeotropes and spurious homogeneous azeotropes and the spurious homotopy map, equation (2.39), and 3) through intersections on spurious homogeneous branches. These three mechanisms are summarized in Table 2.1.

Table 2.1: The three mechanisms for computing heteroazeotropes.

Mechanism	Starting Point	Homotopy Map
1	Bifurcations on lower dimensional heterogeneous branches	Heterogeneous homotopy map, equation (2.48)
2	Bifurcation points on the homotopy branches defined by (2.39)	Spurious homotopy map, equation (2.39)
3	Intersection on homogeneous homotopy branches	Heterogeneous homotopy map, equation (2.48)

Computation of Stable and Spurious Homogeneous Azeotropes

The first step in the heteroazeotrope algorithm is to construct a list containing all homogeneous azeotropes, both stable and spurious. This is performed in the following steps.

1. Compute the quantities $\lambda_{i,j}$, defined by equation (2.21), for all $i, j = 1, \ldots, n$.

2. For each $0 < \lambda_{i,j} < 1$ compute a point on the binary branch (see following section) and store these bifurcation points in the *azeotrope data structures* and set status field equal to `unprocessed`. Append each of these data structures to the end of $\mathcal{AL}$.

3. For each element of $\mathcal{AL}$, track the homogeneous branch from the bifurcation point to $\lambda = 1$. For each new bifurcation point identified along the branch, compute a point on the new branch, store in a *azeotrope data structure* as above, and append to the end of $\mathcal{AL}$. At $\lambda = 1$, set the azeotrope value field of the *azeotrope data structure* containing the bifurcation point from which the azeotrope was obtained equal to the value of the computed azeotrope.

Step 3 above will be repeated as many times as there are bifurcation points on the homogeneous branches (within the set $\mathcal{S}$). Upon completion of the steps above, the list $\mathcal{AL}$ will contain all homogeneous azeotropes predicted by the model (provided the conditions described in section 2.2.1 are satisfied) and a set of spurious homogeneous azeotropes.

Computation of Heteroazeotropes

Having computed the list of homogeneous azeotropes, the next step is to compute the heteroazeotropes. The procedure described below attempts to locate the heteroazeotropes through mechanisms 1 and 2 first, using mechanism 3 as a last resort (although it is necessary for binary branches).

For each entry in the list $\mathcal{AL}$, perform the following steps:

1. Check stability.
2. If azeotrope is stable, remove entry from list and place in $\mathcal{SL}$.
3. If azeotrope is unstable or lies outside the physical composition space (and is thus a spurious homogeneous azeotrope), search the list $\mathcal{HL}$ for a corresponding entry to determine if the corresponding heteroazeotrope may be obtained through mechanisms 1 or 2.
4. If an entry is found, set the status field equal to `possibly_found`, move to the next entry in $\mathcal{AL}$, and go to step 1.
5. If an entry is not found, retrace the corresponding spurious homogeneous branch and search for the point of intersection with the projection of a heterogeneous branch. When found, store the intersection point in the *heterogeneous data structure* and append to the list $\mathcal{HL}$. It may be necessary to search the branch both forward and backward from the bifurcation point, however, the search is confined to the set $\mathcal{S}$.
6. Using the heterogeneous map, track the heterogeneous branch from the intersection point identified in the step above. During the tracking of the heterogeneous

branch, identify bifurcation points along the heterogeneous branch, compute points on the new branch, store in *heterogenous data structures*, append to end of $\mathcal{HL}$, and continue tracking the original heterogeneous branch to $\lambda = 1$. Store the computed heteroazeotrope in the data structure holding the intersection point from which this heteroazeotrope was obtained.

7. At this point, all bifurcations associated with the current heterogeneous branch (obtained through the intersection point identified in step 5) are explored. Preferably, all heteroazeotropes reachable from this branch should be computed through this mechanism. For all entries in list $\mathcal{HL}$ associated with with the heteroazeotrope computed in the previous step, track the heterogeneous branch to $\lambda = 1$ and store the heteroazeotrope value in the appropriate entry of $\mathcal{HL}$. As the heterogeneous branches are tracked, identify and compute additional bifurcation points and append to $\mathcal{HL}$. Continue this step until all heteroazeotropes reachable from the branch associated with the heteroazeotrope computed in step 6 have been computed.

8. Next, attempt to compute a higher dimensional heteroazeotrope through mechanism 2 as follows. Remove the spurious homogeneous azeotrope corresponding to heteroazeotrope computed in step 6 from $\mathcal{AL}$. Check the criteria (2.46) and if satsified, track the spurious homotopy map, equation (2.39), from the heteroazeotrope to the spurious homogeneous azeotrope. If a bifurcation point is identified on this homotopy branch, scan the heteroazeotrope list, $\mathcal{HL}$, to determine if it was previously computed through a bifurcation on a lower dimensional heterogeneous branch in step 7. If not, switch to the higher dimensional branch and track this new branch to $\lambda = 0$ (the location of the heteroazeotrope on the homotopy branches associated with (2.39). Store this new heteroazeotrope in a heteroazeotrope data structure and append to end of $\mathcal{HL}$.

Upon completion of the steps above, the azeotrope list, $\mathcal{AL}$, will contain spurious homogeneous azeotropes with status `possibly_found`, indicating that their corresponding azeotropes may have been obtained through mechanisms 1 or 2. The het-

eroazeotrope list, $\mathcal{HL}$, will contain values for the heteroazeotropes computed through mechanisms 1, 2, or 3.

According to theory developed in this chapter, on a given heterogeneous branch, either an intersection point with a spurious homogeneous branch or a bifurcation point on a lower dimensional heterogeneous branch will occur within the set $\mathcal{S}^o$. Consequently, the intersection point search can be limited to this region. In many of the systems examined, the heterogeneous branch leaves $\mathcal{S}^o$ through an intersection on a spurious homogeneous branch then crosses a lower dimensional heterogeneous branch (a bifurcation point) outside $\mathcal{S}^o$. Obtaining the heterogeneous branches through bifurcation points is somewhat more efficient than the intersection search and thus, it is worthwhile to perform the branch tracking outside $\mathcal{S}^o$ in an attempt to locate additional heterogeneous bifurcation points. How far the continuation is performed outside this set depends on the dimensionality of the current branch (the cost of the intersection search increases with size faster than the bifurcation point search, which is extremely efficient for systems containing several components, thus, higher dimensional branches warrant a more exhaustive bifurcation point search).

The next step is to determine the stability of the heteroazeotropes. For each entry in list $\mathcal{HL}$, perform the following steps:

1. Compute stability if heteroazeotrope is physical (i.e., variable are within their respective bounds).

2. If stable, append heteroazeotrope to list $\mathcal{SL}$ and remove corresponding spurious azeotrope from $\mathcal{AL}$.

Upon completion of the steps above, the sets $\mathcal{AL}$ and $\mathcal{HL}$ should be empty if the conditions described in sections 2.2 and 2.4 are satisfied. If $\mathcal{HL}$ is nonempty then it contains nonphysical solutions. If the set $\mathcal{AL}$ is nonempty then every entry with a corresponding entry in $\mathcal{HL}$ also corresponds to a nonphysical solution predicted by the equilibrium model. All other entries may correspond to heteroazeotropes that were missed in the steps above (theoretical robustness does not imply computational robustness). For each of these entries, an intersection search is performed. Each

resulting heteroazeotrope is tested for stability and stored in $\mathcal{SL}$ if stable.

The set $\mathcal{SL}$ contains all homogeneous and heterogeneous azeotropes computed through the approach described in this chapter. The final step of the heteroazeotrope finding algorithm is to check for topological consistency using the constraint, equation (1.6), described in chapter 1.

The following chapter considers the case of bifurcation and intersection points outside the range $0 < \lambda < 1$ and describes how the approach above can be used to explore the phase equilibrium structure under system and/or property model parameter variation, including the detection of incipient azeotropes and heteroazeotropes (i.e., azeotropes and heteroazeotropes that do not appear under current conditions but may exist if parameters, such as pressure, are perturbed).

Chapter 4 contains several numerical examples illustrating the approach described in this chapter. In nearly every system examined, the heterogeneous branches of dimensionality greater than two were obtained through mechanisms 1, 2, and 3. One exception was the benzene-isopropanol-water system where the ternary spurious branch is isolated from the other homogeneous branches. This ternary heteroazeotrope was obtained, however, through mechanisms 1 and 2. Another exception was the water-acetone-chloroform system where the ternary heterogeneous branch does not intersect the binary heterogeneous branch. In this case, the ternary heteroazeotrope was obtained through mechanisms 2 and 3.

2.6 Implementation

The basic algorithm for the computation of azeotropes and heteroazeotropes described in this thesis can be summarized in the following steps: (1) compute the homogeneous azeotropes using the homogeneous homotopy map, (2) perform a phase stability test on all homogeneous solutions, (3) identify the spurious homogeneous branches, (4) retrace spurious branches and search for points of intersection with a projection of the heterogeneous branches, (5) track the heterogeneous branches to the heteroazeotropes using the heterogeneous homotopy map, and (6) perform a phase stability test on all

heterogeneous solutions. There are four key numerical calculations in this algorithm:

1. Branch tracking,

2. Bifurcation point identification and branch switching,

3. Intersection point identification and computation of missing components, and

4. Phase stability test.

These items are discussed below.

2.6.1 Branch Tracking

The main numerical calculation performed in this algorithm is the tracking of the homogeneous and heterogeneous branches. In every multicomponent mixture analyzed, no turning points in any of the parameters was exhibited by the homogeneous branches inside the set $\mathcal{S}$, however, several turning points were found in the more complicated heterogenous branches (see examples in chapter 4). As a result, a continuation procedure capable of dealing with turning points must be employed. All branch tracking in the numerical examples in this thesis were performed using the continuation code PITCON [95] which uses a locally parameterized continuation method. Alternatively, a continuation procedure based on arclength continuation [117] can be used, however, the algorithm used in PITCON was sufficiently robust and efficient. The remainder of this subsection briefly describes this algorithm.

Consider the following underdetermined system of nonlinear equations:

$$f(z) = 0 \tag{2.85}$$

where $f : \mathcal{D} \subset \mathbb{R}^n \longrightarrow \mathbb{R}^{n-1}$. The following assumptions on f are made:

1. f is continuously differentiable in $\mathcal{D}$,

2. The derivative ∇f of f is locally Lipschitzian on $\mathcal{D}$, and

3. The regularity set of f, $R(f) = \{x \in \mathcal{D} \mid \nabla f(x) \text{ has full rank } n-1\}$ is nonempty.

The connected component $c(\xi) \in f^{-1}(0)$ where $c : \xi \in \mathbb{R} \longmapsto \mathbb{R}^n$ is computed by first selecting an appropriate variable z_i to parameterize the system above. The appropriate choice is based on making $\nabla_{[i]} f$ nonsingular and as well-conditioned as possible ($\nabla_{[i]}$ denotes the Jacobian matrix with partial derivative entries with respect to all variables except z_i). The appropriate selection of z_i at any given point on the curve is based on looking at the local curvature of $c(\xi)$, hence the name locally parameterized continuation method. Given an appropriate parameter, the following system is solved at the current point on the the path, $z^{(k)}$, for $v^{(k)} \in \mathbb{R}^n$:

$$\begin{pmatrix} \nabla f(z^{(k)}) \\ e_i^T \end{pmatrix} v^{(k)} = e_n. \tag{2.86}$$

The matrix on the left-hand-side of the equation above will be referred to as the augmented Jacobian matrix. The direction to step on the curve is then given by

$$d^{(k)} = \sigma \frac{v^{(k)}}{\|v^{(k)}\|_2} \tag{2.87}$$

where

$$\sigma = \text{sgn}((v^{(k)})^T e_i)\text{sgn det} \begin{pmatrix} \nabla f(z^{(k)}) \\ e_i^T \end{pmatrix}. \tag{2.88}$$

Provided conditions (1) through (3) above hold, $d^{(k)}$ will be uniquely determined. The next point on the curve is

$$z^{(k+1)} = z^{(k)} + h^{(k)} d^{(k)} \tag{2.89}$$

where $h^{(k)}$ is the current stepsize selected by looking at the local curvature.

If a simple transcritical bifurcation point exists between $z^{(k-1)}$ and $z^{(k)}$ then

$$\text{sgn det} \begin{pmatrix} \nabla f(z^{(k-1)}) \\ e_{i_{k-1}}^T \end{pmatrix} \text{sgn det} \begin{pmatrix} \nabla f(z^{(k)}) \\ e_{i_k}^T \end{pmatrix} = -1 \tag{2.90}$$

where i_{k-1} and i_k are the parameter indices at step $k-1$ and k, respectively. A better approach for the identification of the bifurcations of interest in the heteroazeotrope finding algorithm is described below.

2.6.2 Bifurcation Point Identification

As shown in section 2.2 for the homogeneous case, while moving along a k-ary branch, a necessary condition for a bifurcation point corresponding to an intersection with a $(k+1)$-ary branch is

$$\alpha_j(\xi) = 0 \tag{2.91}$$

for some $j \in \{k+1, \ldots, n\}$. If the bifurcation point is identified by monitoring the sign of the determinant of the augmented Jacobian matrix there is a risk that an even number of bifurcation points might missed due to a cancellation in the sign change of the determinant or we may incorrectly conclude there is one bifurcation point when there may acually be an odd number greater than one. The occurrence of a bifurcation along the homotopy branch is analogous to a state event in a differential/algebraic equation (DAE) system, that is, a point at which the functional form of the DAE changes during the course of a simulation. Typically, the state events are identified by zero crossings of an appropriate *discontinuity function*. Park and Barton describe an algorithm for state event location that not only guarantees the identification of the state event, but also correctly identifies the first zero crossing of the discontinuity function [84]. Condition (2.91) is analogous to the discontinuity function of a hybrid discrete/continuous DAE model. By constructing an interpolating polynomial for each α_j, $j = k+1, \ldots, n$, using l previously computed points, the algorithm described in [84] can be used to identify efficiently the bifurcation points along the path.

Once the bifurcation point has been identified, a point on the $(k+1)$-ary branch is predicted by stepping in the direction of the eigenvector associated with the zero eigenvalue of the augemented Jacobian matrix [100]. This predicted point is then moved onto the $(k+1)$-ary branch by solving the following system of equations:

$$\begin{pmatrix} x_1(1-\bar{K}_1(x,T,\lambda)) \\ \vdots \\ x_k(1-\bar{K}_k(x,T,\lambda)) \\ \sum_{i=1}^{k+1} x_i - 1 \\ \alpha_j(x,T,\lambda) \\ x_j - \varepsilon \end{pmatrix} = 0 \tag{2.92}$$

where $k+1 \leq j \leq n$ and ε is some sufficiently small positive constant. The exact direction for stepping onto the new branch, as well as necessary and sufficient conditions for the existence of a transcritical bifurcation point, can be derived from first and second order derivative information of $\bar{F}$ at the singular point [100]. However, experience has shown the approach described above to be more than adequately robust and efficient. Once this new point is computed, it is saved so that this branch may be tracked later. The branch tracking is then continued on the original k-ary branch to $\lambda = 1$.

In the case of a k-ary heterogeneous branch, the bifurcation criteria is

$$\delta_j(\xi) = [s(\xi) - 1]\,\bar{\gamma}_j^I(\xi) - \left[s(\xi) - \bar{K}_j^I(\xi)\right]\bar{\gamma}_j^{II}(\xi) = 0 \tag{2.93}$$

for some $j \in \{k+1, \ldots, n\}$. As with the homogeneous case, polynomials can be fitted to the δ_j's and the state event location algorithm mentioned above can be used to locate the bifurcation points. Once the points are identified, the following system of equations is solved for a point on the new $(k+1)$-ary branch (with a starting point obtained from stepping in the direction of the eigenvector associated with the zero

eigenvalue of the augemented Jacobian matrix):

$$\begin{pmatrix} x_1 - \left[\lambda K_1^I + (1-\lambda)P_1^s/P\right] x_1^I \\ \vdots \\ x_n - \left[\lambda K_n^I + (1-\lambda)P_n^s/P\right] x_n^I \\ \lambda\left[\gamma_1^I x_1^I - \gamma_1^{II} x_1^{II}\right] + (1-\lambda)\left[x_1^I - x_1^{II}\right] \\ \vdots \\ \lambda\left[\gamma_n^I x_n^I - \gamma_n^{II} x_n^{II}\right] + (1-\lambda)\left[x_n^I - x_n^{II}\right] \\ x - sx^I - (1-s)x^{II} \\ \sum_{i=1}^n x_i - 1 \\ \sum_{i=1}^n x_i^I - 1 \\ \delta(x, x^I, x^{II}, T, s, \lambda) \\ x_j - \varepsilon \end{pmatrix} = 0 \tag{2.94}$$

where $k+1 \leq j \leq n$ and ε is some sufficiently small positive constant. This point is saved so that that this branch may be tracked later.

2.6.3 Intersection Point Identification

At the intersection of an n component spurious homogeneous branch and a projection of a heterogeneous branch the following system of equations is satisfied:

$$\begin{pmatrix} x_1(1-\bar{K}_1) \\ \vdots \\ x_n(1-\bar{K}_n) \\ \bar{\gamma}_1 x_1 - \bar{\gamma}_1^{II} x_1^{II} \\ \vdots \\ \bar{\gamma}_n x_n - \bar{\gamma}_n^{II} x_n^{II} \\ \sum_{i=1}^n x_i^{II} - 1 \\ \sum_{i=1}^n x_i - 1 \end{pmatrix} = 0 \tag{2.95}$$

Equation (2.95) was formed from (2.48) by simply setting $s = 1$ and $x = x^I$ and removing the redundant equations. Elements 1 through n and element $2n + 2$ of (2.95) are satisfied on the spurious homogeneous branch. The intersection finding problem amounts to moving along the spurious homogeneous branch and looking for solutions to (2.95). Since the cost of this root finding problem increases with the number of variables, locating roots of the following subset of equations has been found to be computational more efficient and just as robust:

$$\Lambda = \begin{pmatrix} \bar{\gamma}_1 x_1 - \bar{\gamma}_1^{II} x_1^{II} \\ \vdots \\ \bar{\gamma}_n x_n - \bar{\gamma}_n^{II} x_n^{II} \\ \sum_{i=1}^{n} x_i^{II} - 1 \end{pmatrix} = 0 \tag{2.96}$$

The intersection finding problem is divided into two steps: 1) intersection point identification and 2) intersection point computation. The identification problem is handled using a root exclusion test.

A root exclusion test performed on a system of equations $f(z) = 0$, $f : \mathcal{D} \subset \mathbb{R}^n \longrightarrow \mathbb{R}^n$, provides the following information about $X \subseteq \mathcal{D}$:

1. There is no root in X,
2. There is a unique root in X, or
3. There may or may not be one or more roots in X.

For a multidimensional problem, a root exclusion test based on interval arithmetic is ideal. Interval arithmetic is a branch of mathematics concerned with operations with intervals or closed subsets of the real line [81]. Let $X_0 \equiv \left[X_0^l, X_0^u\right] = \{x \in \mathbb{R} \mid X_0^l \leq x \leq X_0^u\}$ denote an interval. The set of all intervals in $\mathbb{R}$ is denoted by $\mathbb{IR}$. Similarly, the set of all n-dimensional intervals in $\mathbb{R}^n$ is denoted by $\mathbb{IR}^n$ (e.g., the set of all rectangles in $\mathbb{R}^2$ is denoted by $\mathbb{IR}^2$). The familiar operations associated with real numbers can also be defined for intervals. For example, let $\circ$ denote some binary operation defined for real numbers. The associated interval operation is defined as

$X_0 \circ Y_0 = \{x \circ y \mid x \in X_0, y \in Y_0\}$. If $x \circ y$ is undefined for any $x \in X_0$ or $y \in Y_0$, the corresponding interval operations is undefined. For example, interval division is undefined if the denominator contains zero. Similarly, elementary functions can be extended to intervals. Let $func$ denote some function (e.g., sin, cos, etc.). The interval value for $func$, denoted by $Func$, is defined by $Func(X_0) = \{func(x) \mid x \in X_0\}$. Again, $Func(X_0)$ is undefined if $func(x)$ is undefined for any $x \in X_0$. An *interval extension* of a general nonlinear function $f : \mathcal{D} \subset \mathbb{R}^n \longrightarrow \mathbb{R}^n$, denoted by $F : \mathbb{IR}^n \longrightarrow \mathbb{IR}^n$, can be constructed by replacing all real operators and elementary functions in f with the corresponding interval operations. The interval extension of f has the important property that given $X \in \mathbb{IR}^n$ ($X \subseteq \mathcal{D}$), $F(X)$ contains the image set of X under f, that is, $f(X) \subseteq F(X)$. The cost of evaluating the interval extension of a function is a small multiple of the cost of evaluating the function using real operations. This provides a rapid way of answering question (1) above: if $0 \notin F(X)$ then there is no solution to $f(x) = 0$ for any $x \in X$. Unfortunately, since the interval extension of a function over-approximates the image set, $0 \in F(X)$ does not provide any information about solutions within X. Question (2) above can be answered by a theorem due to Moore [80]. First, the *Krawczyk operator* is defined:

$$\mathbb{K}(X) \equiv y - Yf(y) + [I - Y\nabla F(X)](X - y) \tag{2.97}$$

where $y \in X \subseteq \mathcal{D}$, $Y \in \mathbb{R}^{n \times n}$ is an arbitrary nonsingular, real matrix, and $\nabla F(X) \in \mathbb{IR}^{n \times n}$ is the interval extension of the Jacobian matrix of f.

Theorem 3 *(Moore). If*

$$\mathbb{K}(X) \subset int(X) \tag{2.98}$$

then there exists a unique solution $x^* \in X$ to $f(x) = 0$.

In practice, $y = \mathrm{m}(X)$ and $Y = (\mathrm{m}(\nabla F(X)))^{-1}$ where $\mathrm{m}(\cdot)$ denotes the midpoint

of an interval vector, i.e.,

$$\mathrm{m}(X) = \begin{pmatrix} (X_1^u + X_1^l)/2 \\ \vdots \\ (X_n^u + X_n^l)/2 \end{pmatrix}$$

Figure 2-10 contains a pseudo-code description of the intersection finding algorithm. The input to this procedure is the current point on a spurious homogeneous curve, $(x^{(k)}, T^{(k)}, \lambda^{(k)})$, and an interval in λ over which to search for the intersection point, $\delta\lambda^{(k)}$, based on the size of the previous step taken during the continuation. The return value of this procedure is a subdomain $X \in \mathbb{IR}^N \times \mathbb{IR}$ that contains a unique solution to (2.96) (in which case procedure argument `solution` is set to `contains_solution`), or a subdomain that may contain a solution if `status` is set to `possible_solution`, or the empty set is returned and `status` is set to `no_solution`, indicating there is no solution to (2.96) near the current continuation point. Upon entering LocateIntersection, the nontrivial index set, $\mathcal{I}$, is constructed. This set contains the indices of the nonzero mole fraction elements (the dimensionality of the root finding problem is equal to the dimensionality of the spurious branch plus one). Next, the current search domain, $\mathcal{D}^k$, is constructed. Solutions are sought where $(x^{II}, \lambda) \in [0,1]^N \times [\lambda^{(k)} - \delta\lambda^{(k)}, \lambda^{(k)} + \delta\lambda^{(k)}]$ where N is the number of nonzero mole fraction elements on the current branch. Although the intersection point may occur at a point where x^{II} is outside the physical region (where the elements are bounded between zero and one), these branches will be obtained through bifurcations on lower dimensional heterogeneous branches. Since the trivial solution ($x^{II} = x^{(k)}$) satisfies (2.96) at each continuation point, $x^{(k)}$ must be excluded from the search domain. Thus, 2^N subdomains are constructed from $\mathcal{D}^k$ by bisecting through $x^{(k)}$. These subdomains are stored in a list $\mathcal{L}$. For each domain in this list, the interval extension of (2.96), denoted by $\bar{\Delta}(X)$, is evaluated and a check is made to see if $0 \in \bar{\Delta}(X)$. If zero is not contained within this set, the X is deleted from $\mathcal{L}$ and the next subdomain in the list is examined. If zero is contained within $\bar{\Delta}(X)$ then X may contain one or

more solutions. If the diameter of X, denoted by $\mathrm{diam}(X)$ and defined as

$$\mathrm{diam}(X) \equiv \max_{1 \le i \le n} \{X_i^u - X_i^l\}, \tag{2.99}$$

is less than $\beta > 0$ then the Krawczyk operator is evaluated. If $\mathbb{K}(X) \subseteq \mathrm{int}(X)$ then X contains a unique solution and X is returned with `status` set to `contains_solution`. Otherwise, if the diameter of X is less than ε where $0 < \varepsilon < \beta$, then X is very small and may contain a solution. This set is then returned with `status` equal to `possible_solution` and the problem is handled by the intersection point computation routine. Finally, if $\mathrm{diam}(X) > \varepsilon$ and $\mathbb{K}(X) \not\subseteq \mathrm{int}(X)$ then X is bisected through its largest edge, forming two sets, X_1 and X_2. X is then removed from $\mathcal{L}$, X_1 and X_2 are appended to $\mathcal{L}$, and the next subdomain in the list is examined. This is repeated until the list is empty, in which case, the empty set is returned with `status` set to `no_solution` indicating there is no solution near the current continuation point.

If `status` is returned with a value of `contains_solution` or `possible_solution`, a bounded Newton's method is employed to compute the solution. The search is limited to the region X returned from LocateIntersection.

2.6.4 Phase Stability Test

Solutions satisfying only the necessary conditions for azeotropy and heteroazeotropy must be tested for thermodynamic stability. A test based on the tangent plane criteria [77] was chosen for this work. As described in chapter 1, the solution procedure must be guaranteed to compute either all stationary points the global minimum of the tangent plane distance function, equation (1.7). For the the phase stability test used in this thesis, all stationary points were computed using an interval Newton/generalized bisection approach, similar to that described in [105]. This option was chosen for its simplicity and computational robustness. It is important to note that the approaches developed in this thesis minimize the number of times the highly costly stability test has to be performed.

2.7 Conclusion

This chapter describes a new approach for the computation of the homogeneous and heterogeneous azeotropes present in a mulicomponent mixture. The approach, which is an extension of an approach developed by Fidkowski *et al.*for the computation of homogeneous azeotropes, is independent of both the topology of the liquid-liquid region and the model used to represent the nonideality of the system. Theoretical analysis of the method provides conditions under which all azeotropes and heteroazeotropes will be computed. This analysis has also led to several algorithmic improvements. It is important for one to recognize the difference between theoretical robustness and computational robustness of an algorithm. Theoretical robustness results from the determination of exact conditions under which the algorithm will not fail. Algorithmic robustness has to do with the actual implementation of the approach within a computer. Theoretical robustness in this approach is addressed with the developement of the theories in this chapter. Algorithmic robustness results from the capability of computing the heteroazeotropes through several independent mechanisms.

The following chapter describes an extension of this algorithm for computing efficiently the changes in phase equilibirium structure under system and/or property model parameter variation. The analysis of the changes in phase equilibrium structure include the determination of bifurcation values of the parameters where homogeneous and heterogeneous azeotropes appear, disappear, and switch between each other.

```
LocateIntersection(x^(k),T^(k),λ^(k), δλ^(k),status)
 1  ▷ Determine the index set of nonzero mole fractions.
 2  I = { j | x_j^(k) > 0 and 1 ≤ j ≤ n} ;
 3  N = dim I ;
 4  ▷ Construct current search domain.
 5  D^k = {(x^II, λ) ∈ R^N × R | 0 ≤ x^II ≤ 1 , λ^(k) − δλ^(k) ≤ λ ≤ λ^(k) + δλ^(k)} ;
 6  ▷ Construct subdomains by bisecting through x^(k) and store in list.
 7  L = {D_1^k, D_2^k, ... , D_{2^N}^k} ;
 8  while L ≠ ∅ do
 9     X ∈ L ;
10     ▷ Evaluate natural interval extension of intersection equations.
11     if 0 ∈ Δ̄(X) then
12        if diam(X) < β and K(X) ⊆ int(X) then
13           ▷ X contains unique solution to intersection equations.
14           status  = contains_solution ;
15           return X ;
16        elseif diam(X) < ε then
17           ▷ Current subdomain very small and may contain a solution.
18           ▷ Return X and status set accordingly.
19           status = possible_solution ;
20           return X ;
21        else
22           ▷ Refine subdomain by bisecting through the midpoint of the
23           ▷ longest edge of X. Replace X in list with the refinement.
24           X → (X_1, X_2) ;
25           L = (L − {X}) ∪ {X_1, X_2} ;
26        end
27     else
28        ▷ Delete X from list.
29        L = L − {X} ;
30     end
31  end
32  ▷ No solution in neighborhood of current point.
33  status  = no_solution ;
34  return ∅ ;
```

Figure 2-10: Intersection location algorithm.

Chapter 3

Analysis of Phase Equilibrium Structure

3.1 Introduction

Computation of all azeotropes and heteroazeotropes present in a multicomponent mixture is a necessary task when analyzing and designing separation systems. In addition to computing the azeotropes and heteroazeotropes at a given pressure (or temperature), it is often valuable to know how the azeotropic composition and temperature vary with pressure (or, equivalently, since the azeotrope is a univariate state, how the azeotropic composition and pressure vary with temperature). A systematic approach for analyzing such changes can be incorporated directly into design algorithms, sometimes dramatically increasing the space of alternative designs.

As described in chapter 1, the operation of a heteroazeotropic distillation column exhibits multiple solutions, high parametric sensitivity, and a sharp temperature gradient associated with the movement of the liquid-liquid front through the column. In addition, as shown by Rovaglio and Doherty [93], small pressure perturbations can lead to separation failure over a relatively short period of time. Increased knowledge of the phase equilibrium structure under system parameter variation can improve the interpretation of simulation results as well as improve the understanding of how the column should be operated. Furthermore, tools capable of improving the modeler's

understanding of the phase equilibrium behavior can be used to develop more robust and efficient simulation methods.

Most activity coefficient models are based on binary interaction parameters, obtained through regression with experimental equilibirum data. Ideally, when the activity coefficient models are applied to multicomponent mixtures ($n > 2$), the computed values of higher dimensional azeotropes and heteroazeotropes are extrapolated from the parameters fit to the binary pairs contained in the systems. In some cases, these higher dimensional azeotropes and heteroazeotropes are not predicted or the computed values are significantly different than the experimental values, due to the limitations of using binary interaction parameters. Often, the solutions not predicted by the phase equilibrium model actually exist albeit outside the physical composition space, $\{x \in \mathbb{R}^n \mid \sum_{i=1}^{n} x_i = 1 \text{ and } x_i \geq 0 \quad i = 1, \ldots, n\}$, where they are of little practical use. In the event of such failures, a systematic analysis of the effect of parameter perturbation on the prediction of azeotropes and heteroazeotropes provides useful insights into the capabilities of the phase equilibrium model. Furthermore, tools assisting the user in the systematic exploration of the phase equilibrium structure can be used as the property model parameters are being estimated in order to judge the quality of the fit.

Given the composition, temperature, and pressure of an azeotrope or heteroazeotrope, standard continuation methods [95, 117] can be applied to the necessary conditions for azeotropy or heteroazeotropy to determine how the state variables change with a given system parameter (e.g., temperature or pressure) or property model parameter. By applying a phase stability test during the continuation, it is possible to determine at what conditions an azeotrope becomes a heteroazeotrope or vice versa. However, there often arises situations where at the specified temperature or pressure, the azeotrope or heteroazeotrope does not physically exist or does exist, but is not predicted within the physcial composition space by the phase equilibrium model either due to the limitations of the actual model or the model parameter values.

This chapter describes how the approach for computing azeotropes and heteroazeotropes developed in this thesis can be used to compute, systematically and ef-

ficiently, changes in phase equilibrium structure under system and property model parameter variation. The approach has the capability of predicting *incipient* azeotropes and heteroazeotropes, that is, azeotropes and heteroazeotropes that do not exist under current conditions but may appear at a different temperature and pressure or different values for the property model parameters.

The following section briefly summarizes the algorithm described in the previous chapter. This is followed by a description of the extension to the exploration of the phase equilibrium structure. Numerical examples of this approach are provided in chapter 4.

3.2 Summary of Homogeneous and Heterogeneous Azeotrope Finding Algorithm

The previous chapter in this thesis describes an algorithm for the computation of the azeotropes and heteroazeotropes present in a multicomponent mixture. The procedure is based on using two homotopy maps. The homogeneous map is

$$\bar{F}(x,T,P,\lambda) = \begin{pmatrix} x_1\left(1-[\lambda K_1(T,P,x)+(1-\lambda)P_1^s(T)/P]\right) \\ \vdots \\ x_n\left(1-[\lambda K_n(T,P,x)+(1-\lambda)P_n^s(T)/P]\right) \\ \sum_{i=1}^{n} x_i - 1 \end{pmatrix} = 0 \qquad (3.1)$$

where $x \in \mathbb{R}^n$ is the liquid composition, T is temperature, P is pressure, and $\lambda \in \mathbb{R}$ is the homotopy parameter. The heterogeneous map is

$$\bar{F}^o(x, x^I, x^{II}, T, P, s, \lambda) = \begin{pmatrix} x_1 - \left[\lambda K_1^I(T,P,x^I) + (1-\lambda) P_1^s(T)/P\right] x_1^I \\ \vdots \\ x_n - \left[\lambda K_n^I(T,P,x^I) + (1-\lambda) P_n^s(T)/P\right] x_n^I \\ \lambda\left[\gamma_1^I(T,P,x^I) x_1^I - \gamma_1^{II}(T,P,x^{II}) x_1^{II}\right] + (1-\lambda)\left[x_1^I - x_1^{II}\right] \\ \vdots \\ \lambda\left[\gamma_n^I(T,P,x^I) x_n^I - \gamma_n^{II}(T,P,x^{II}) x_n^{II}\right] + (1-\lambda)\left[x_n^I - x_n^{II}\right] \\ x - s x^I - (1-s) x^{II} \\ \sum_{i=1}^n x_i - 1 \\ \sum_{i=1}^n x_i^I - 1 \end{pmatrix} = 0 \quad (3.2)$$

where $x \in \mathbb{R}^n$ is the *overall* liquid composition, $x^I, x^{II} \in \mathbb{R}^n$ are the composition of liquid phases I and II, respectively, and $s \in \mathbb{R}$ is the liquid phase fraction (the fraction of the total number of moles of liquid in liquid phase I).

As described in the previous chapter, at constant temperature or pressure, the homogenous azeotropes are obtained through a series of bifurcations on the homongeneous branches, $\bar{c}(\xi) \in \bar{F}^{-1}(0)$, originating from the pure components. The heteroazeotropes are computed with the heterogeneous branches, $\bar{c}^o(\xi) \in (\bar{F}^o)^{-1}(0)$, from starting points obtained either through intersections with spurious homogeneous branches or bifurcations on lower dimensional heterogeneous branches. As shown in previous chapter, bifurcation and intersection points of interest when computing the azeotropes and heteroazeotropes will occur within $0 < \lambda < 1$. However, if the homotopy branches are tracked outside this range, bifurcation and intersection points are often identified that correspond to nonphysical branches, that is, branches that either do not cross $\lambda = 1$ (the only physically meaningful point on the branch) or cross $\lambda = 1$ at a point where one or more mole fraction vector elements or the phase fraction (in the heterogeneous case) are outside their physical range of zero and unity. For example, if a pure component homogeneous branch is tracked backwards from $\lambda = 0$ (backwards defined as the direction of decreasing λ) and a bifurcation point is

identified, then the corresponding binary branch is not physical because it is unable to cross $\lambda = 0$ in order to reach $\lambda = 1$. (Recall that, as a consequence of Raoult's Law, the only solutions to the homotopy map at $\lambda = 0$ are the pure components and thus, branches of dimension two or greater can only cross $\lambda = 0$ under very specific conditions.) If a bifurcation or intersection point is identified at some $\lambda > 1$ then the corresponding branch either does not cross $\lambda = 1$ or does so at a point where the composition or phase fraction is outside the physical bounds of zero and unity.

These nonphysical branches themselves are not of any use, however, as will be shown in section 3.3.1, the nonphysical bifurcation and intersection points (i.e., points occurring outside $0 < \lambda < 1$) are useful in examining the phase equilibrium structure of the mixture.

3.3 Analysis of Phase Equilibrium Structure

The condition of azeotropy occurs when there is an extremum in the equilibrium surface of a mixture. At such a point, the composition of the equilibrium vapor and liquid are equal, and thus, this point acts as a barrier to separations exploiting composition gradients (or, more strictly, chemical potential gradients) as described in chapter 1. Azeotropic points are univariate; specifying one intensive property uniquely defines the intensive state of the system. Under certain conditions, changing a system parameter (e.g., pressure) results in the appearance or disappearance of an azeotrope. For example, Figure 3-1 contains a schematic of a Txy diagram for a binary mixture where the azeotropic state does not exist at low pressure, but emerges from a pure component vertex as pressure is increased. The pressure at which the azeotrope appears or disappears is referred to as a bifurcation pressure. This is illustrated for the acetone-ethanol system and the tetrahydrofuran-water system in sections 4.2.2 and 4.2.6, respectively.

Heteroazeotropy occurs when the azeotropic composition on the vapor-liquid equilibrium surface intersects the liquid-liquid binodal. The locus of azeotropic and heteroazeotropic points as a function of pressure are shown in the schematic in Figure

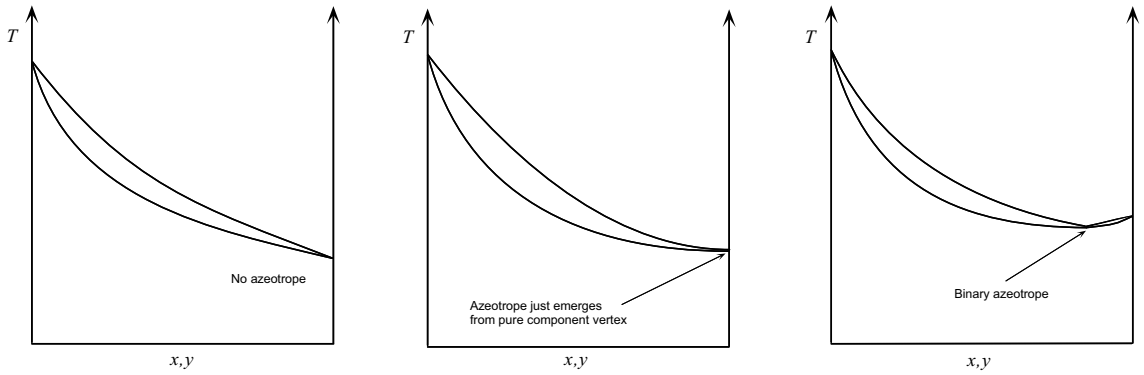

Figure 3-1: Binary homogeneous T-xy diagram.

3-2 where, in this case, the liquid-liquid binodal exhibits an UCST.

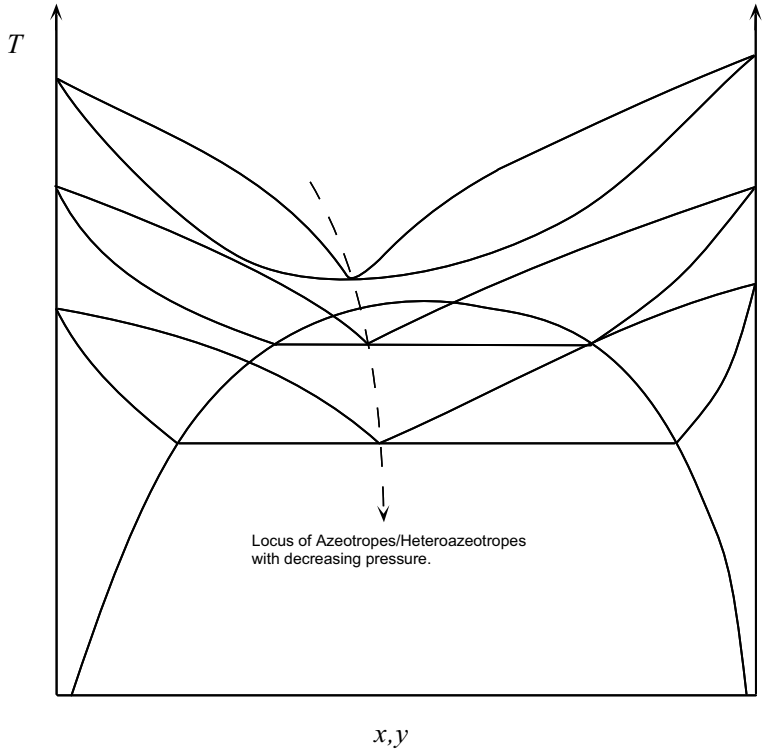

Figure 3-2: Binary heterogeneous T-xy diagram.

Pressure does not strongly effect the liquid-liquid region, however, decreasing pressure reduces the boiling temperatures of the components present in the mixture, thereby causing the vapor-liquid equilibrium surface to move into the liquid-liquid region. As a result, the azeotrope becomes a heteroazeotrope. The pressure where the azeotropic point on the vapor-liquid surface just touches the liquid-liquid binodal

is also referred to as a bifurcation pressure.

In many cases, the azeotropic and heteroazeotropic conditions are highly sensitive to changes in the system parameters. Knowing how the conditions of the system change is important because in some cases avoiding a liquid-liquid phase split is desirable whereas in other cases, it may be advantageous to exploit this condition. This chapter describes an efficient procedure for analyzing the changes in phase equilibrium structure with system and/or property model parameter variation. The approach is an extension of an algorithm for the computation of the azeotropes and heteroazeotropes present in a multicomponent mixture described in the previous chapter. Thus, it is possible to compute simultaneously the azeotropes and heteroazeotropes present under one set of conditions (e.g., specified temperature or pressure) as well as predicting the effect on the azeotropic states of the system under parameter variation.

3.3.1 Examination of the Phase Equilibrium Structure

There are three types of points of interest on the bifurcation diagrams generated using the approach described in chapter 2: solution points, bifurcation points, and intersection points. Solution points are points on the branches where the λ component equals unity. Bifurcation points are intersections of k-ary and $(k+1)$-ary branches (branches are either both homogeneous or both heterogeneous). Intersection points are intersections of heterogeneous k-ary and spurious homogeneous k-ary branches.

The homogeneous and heterogeneous homotopy maps can expressed in general as

$$\bar{F}(x, T, P, \lambda, \{p_i\}_{i=1}^{N_p}) = 0 \tag{3.3}$$

and

$$\bar{F}^o(x, x^I, x^{II}, s, T, P, \lambda, \{p_i\}_{i=1}^{N_p^o}) = 0 \tag{3.4}$$

where $\{p_i\}_{i=1}^{N_p}$ and $\{p_i\}_{i=1}^{N_p^o}$ are the parameters associated with the property models used to compute the VLE and VLLE, respectively. For example, these models in-

clude vapor pressure correlations and activity coefficient equations. The homogeneous map is $n+1$ equations in terms of $n+N_p+3$ variables and parameters. The homogenous branches are computed by specifying N_p+1 values of the variables and parameters, leaving (3.3) $n+1$ equations in terms of $n+2$ variables with which standard continuation methods can be applied to determine $\bar{c}(\xi)$. In chapter 2, the N_p physical property parameters and temperature or pressure are specified to compute the azeotropes. Similarly, the heterogeneous map is $3n+2$ equations in terms of $3n+N_p^o+4$ variables and parameters. The heterogenous branches are computed by specifying N_p^o+1 variables and parameters (N_p^o physical property parameters and temperature or pressure in chapter 2) and using standard continuation methods to compute $\bar{c}^o(\xi)$.

The basic idea described in this section is to append an additional equation, which defines the point of interest, to system (3.3) or (3.4), thereby removing a degree of freedom from the augmented system. This allows an additional parameter that was fixed in order to compute $\bar{c}(\xi)$ or $\bar{c}^o(\xi)$ to be treated as a variable so that standard continuation methods can be applied to the new system of equations to determine how the point of interest varies. In order to use continuation to track $c(\xi) \in f^{-1}(0)$ of an underdetermined system $f : \mathbb{R}^m \longrightarrow \mathbb{R}^{m-1}$, the following conditions must be satisfied in the neighborhood, $\mathcal{D} \subset \mathbb{R}^m$, of the point at which the continuation is to be started:

1. f is continuously differentiable in $\mathcal{D}$,
2. The derivative ∇f of f is locally Lipschitzian on $\mathcal{D}$, and
3. The regularity set of f, $R(f) = \{z \in \mathcal{D} \mid \nabla f(z) \text{ has full rank } m-1\}$ is nonempty.

These criteria must be satisfied when selecting the additional parameter to free at each of the points described below.

Solution Points

A solution point is simply a point on the homotopy branch that satisfies the necessary conditions for homogeneous or heterogeneous azeotropy (i.e., a point at which the homotopy branch crosses $\lambda = 1$). Thus, the additional equation mentioned above is simply

$$\lambda = 1. \tag{3.5}$$

Instead of augmenting (3.3) and (3.4) with equation (3.5), λ is simply replaced by unity, thereby reducing the homotopy maps down to the necessary conditions for azeotropy and heteroazeotropy. This allows an additional variable or parameter to be selected from the set $\{x, T, P, \lambda, \{p_i\}_{i=1}^{N_p}\}$ or $\{x, x^I, x^{II}, s, T, P, \lambda, \{p_i\}_{i=1}^{N_p^o}\}$ (satisfying the continuation criteria above) and treated as a variable. Standard continuation methods can then be applied to the new underdetermined set of equations to examine how the azeotrope and heteroazeotrope conditions vary. Since only the necessary conditions are satisfied at each continuation point, a phase stability test must be used at each step during the continuation to determine when the azeotrope or heteroazeotrope disappears or switches between each other. The need for an expensive phase stability test makes the tracking of solution branches computationally unattractive.

The examination of the phase equilibrium structure using the approach described above is straightforward and, with the exception of the phase stability test, quite efficient. However, an azeotrope or heteroazeotrope must be known a priori in order to start the continuation, that is, we must start off with a point on the homotopy branch that crosses $\lambda = 1$. The following two sections describe how the existence of incipient azeotropes and heteroazeotropes can be predicted by examining the bifurcation and intersection points associated with nonphysical branches, branches that do not cross $\lambda = 1$ or do so at points that do not satisfy the necessary conditions due to a variable bound violation.

Bifurcation Points

A bifurcation point is where a k-ary branch intersects a $(k+1)$-ary branch (both branches are either homogeneous or heterogeneous). Without loss of generality, the following conditions are satisfied on a k-ary homogeneous branch:

$$\begin{aligned} x_j &\neq 0 \quad j = 1, \ldots, k, && (3.6)\\ \alpha_j &= 0 \quad j = 1, \ldots, k, && (3.7)\\ x_j &= 0 \quad j = k+1, \ldots, n, \text{ and} && (3.8)\\ \alpha_j &\neq 0 \quad j = k+1, \ldots, n && (3.9) \end{aligned}$$

where $\alpha_j = 1 - (\lambda K_j + (1-\lambda)P_j^s/P) = 1 - \bar{K}_j$. As shown in chapter 2, a necessary condition for a bifurcation onto a $(k+1)$-ary branch at a point $\tilde{\xi}$ is

$$\alpha_j(\tilde{\xi}) = 0 \quad \text{for some } j = k+1, \ldots, n. \tag{3.10}$$

Thus, this is the equation that is appended to (3.3) to make the system fully determined. Without loss of generality, assume for some $\tilde{\xi}$, $\alpha_{k+1}(\tilde{\xi}) = 0$. At a bifurcation point corresponding to the intersection of a k-ary and a $(k+1)$-ary branch the following set of $k+2$ equations are satisfied (the equations associated with zero mole fractions are not written):

$$\begin{pmatrix} x_1\left(1 - [\lambda K_1(T,P,x) + (1-\lambda)P_1^s(T)/P]\right) \\ \vdots \\ x_k\left(1 - [\lambda K_k(T,P,x) + (1-\lambda)P_k^s(T)/P]\right) \\ \sum_{i=1}^{k} x_i - 1 \\ \alpha_{k+1}(T,P,x,\lambda) \end{pmatrix} = 0. \tag{3.11}$$

Equation (3.11) is $k+2$ equations in terms of $k+N_p+3$ variables and parameters. By selecting $k+3$ variables from the set $\{\{x_i\}_{i=1}^k, T, P, \lambda, \{p_i\}_{i=1}^{N_p}\}$ such that the continuation crieteria are satisfied, (3.11) becomes $k+2$ variables in terms of $k+3$ variables

(only the k nonzero mole fraction elements are considered and these are treated as independent since the summation of mole fractions constraint is handle explicitly). Again, standard continuation methods can be applied to this underdetermined system to examine how the bifurcation point varies.

Of particular interset is under what conditions the bifurcation point occurs at $\lambda = 0$ or $\lambda = 1$. A bifurcation point at $\lambda = 1$ corresponds to the case where the $(k+1)$-ary azeotrope emerges from a k-ary azeotrope. This is illustrated in an example in section 4.2.2 where an acetone-ethanol homogeneous azeotrope emerges from the pure acetone vertex as pressure is increased above approximately 8.6 bar. A bifurcation at $\lambda = 0$ will most commonly occur between a pure component and a binary homogeneous branch and it is a consequence of two components, which form an azeotrope, having the same boiling temperature at a certain pressure. This is the exception briefly mentioned in chapter 2 where more than n branches occur at $\lambda = 0$. This point corresponds to the case where the bifurcation point associated with the physical binary azeotrope switches the pure component branch from which it emerges from. This phenomenon is illustrated in section 4.2.1 for the chloroform-methanol system.

In the heterogeneous case, the necessary condition for a bifurcation from a k-ary branch onto a $(k+1)$-ary branch at a point $\tilde{\xi}$ is

$$\delta_j(\tilde{\xi}) = \left[s(\tilde{\xi}) - 1\right] \bar{\gamma}_j^I(\tilde{\xi}) - \left[s(\tilde{\xi}) - \bar{K}_j^I(\tilde{\xi})\right] \bar{\gamma}_j^{II}(\tilde{\xi}) = 0 \tag{3.12}$$

for some $j = k+1, \ldots, n$, where $\bar{\gamma}_j^I = \lambda\gamma_j^I + (1-\lambda)$, $\bar{\gamma}_j^{II} = \lambda\gamma_j^{II} + (1-\lambda)$, and $\bar{K}_j^I = \lambda K_j^I + (1-\lambda)P_j^s/P$. This is the additional equation appended to (3.4) to remove a degree of freedom. Again, without loss of generality, we can assume that condition (3.12) is satisfied for $j = k+1$ at $\tilde{\xi}$. Similar to the homogeneous case, the

following set of equations are satisfied at the heterogeneous bifurcation point:

$$\begin{pmatrix} x_1 - \bar{K}_1^I(x^I, T, P, \lambda)x_1^I \\ \vdots \\ x_k - \bar{K}_k^I(x^I, T, P, \lambda)x_k^I \\ \bar{\gamma}_1^I(x^I, T, P, \lambda)x_1^I - \bar{\gamma}_1^{II}(x^{II}, T, P, \lambda)x_1^{II} \\ \vdots \\ \bar{\gamma}_k^I(x^I, T, P, \lambda)x_k^I - \bar{\gamma}_k^{II}(x^{II}, T, P, \lambda)x_k^{II} \\ x_1 - sx_1^I - (1-s)x_1^{II} \\ \vdots \\ x_k - sx_k^I - (1-s)x_k^{II} \\ \sum_{i=1}^{k} x_i - 1 \\ \sum_{i=1}^{k} x_i^I - 1 \\ \delta_{k+1}(x, x^I, x^{II}, T, P, s, \lambda) \end{pmatrix} = 0. \tag{3.13}$$

This is a system of $3k+3$ equations in terms of $3k + N_p^o + 4$ unknowns. By selection $3k+4$ variables or parameters from the set $\{x, x^I, x^{II}, T, P, s, \lambda, \{p_i\}_{i=1}^{N_p}\}$, not including the zero mole fractions and subject to the continuation criteria, standard continuation methods can be applied to (3.13) to determine how the heterogeneous bifurcation point varies.

Of particular interest for the heterogeneous case is where the homogeneous azeotrope becomes a heteroazeotrope at the point where the vapor-liquid equilibrium surface enters the liquid-liquid binodal. Thus, the variation of intersection points, described below, is more interesting than the variation of bifurcation points for the heterogeneous case.

Intersection Points

For the case where the bifurcation diagrams described in chapter 2 are constructed at constant pressure, an intersection point is an intersection of a spurious homogeneous

branch and a projection of a heterogeneous branch into (x, T, λ)-space. Alternatively, the intersection point can be viewed as an intersection between a heterogeneous branch and a spurious homogeneous surface in $(x, x^I, x^{II}, s, T, \lambda)$-space. This intersection occurs at a point on the heterogeneous branch where $s = 1$ and thus, this is the additional constraint appended to the heterogeneous homotopy map. As shown in chapter 2, the rank of the Jacobian of the heterogeneous homotopy map, equation (3.4), is deficient by at least one at an intersection point. The case where the rank of this matrix is deficient by exactly one corresponds to the case of a "normal" intersection point while the others are degenerate and require further analysis. However, this condition was not encountered in any of the systems analyzed in chapter 4.

As stated above, the heterogeneous homotopy map is appended with the constraint $s = 1$. However, rather than increasing the size of (3.4), the liquid phase fraction is replaced by unity and the redundant equations are removed. Thus, the set of constraints satisfied at a k-ary intersection point are

$$\begin{pmatrix} x_1(1 - [\lambda K_1(T,P,x) + (1-\lambda)P_1^s(T)/P]) \\ \vdots \\ x_k(1 - [\lambda K_k(T,P,x) + (1-\lambda)P_k^s(T)/P]) \\ \lambda\left[\gamma_1(T,P,x)x_1 - \gamma_1^{II}(T,P,x^{II})x_1^{II}\right] + (1-\lambda)\left[x_1 - x_1^{II}\right] \\ \vdots \\ \lambda\left[\gamma_k(T,P,x)x_k - \gamma_k^{II}(T,P,x^{II})x_k^{II}\right] + (1-\lambda)\left[x_k - x_k^{II}\right] \\ \sum_{i=1}^{k} x_i - 1 \\ \sum_{i=1}^{k} x_i^{II} - 1 \end{pmatrix} = 0. \qquad (3.14)$$

System (3.14) was formed from system (3.4) by removing x^I, s, the $n - k$ zero mole fraction vector elements, and the equations defining the overall liquid composition. The singularity of $\nabla \bar{F}^o$ at the intersection point has been removed by eliminating $x - sx^I - (1-s)x^{II} = 0$ from (3.2). Similar to the other cases above, equation (3.14) is $2k + 2$ equations in terms of the following variables, $x, x^{II} \in \mathbb{R}^k$, T, P,

λ, and $\{p_i\}_{i=1}^{N_p^o}$. Specifying N_p^o of these variables and/or parameters (subject to the continuation criteria) allows standard continuation methods to be applied to examine how the intersection point varies. The point where $\lambda = 1$ on the intersection point curve corresponds to the case where the azeotrope composition on the vapor-liquid equilibrium surface just touches the liquid-liquid binodal.

3.4 Conclusion

The extensions of the heteroazeotrope finding algorithm described in this chapter allows the phase equilibrium structure of a multicomponent mixture to be explored systematically and efficiently. The power in this approach lies in the ability to identify incipient azeotropes and heteroazeotropes, that is, azeotropes and heteroazeotropes that do not exist (in the physical composition space) under current conditions or property model parameter values, but may exist under different conditions or parameter values. This methodology provides insights into both the physical system and the property models used to compute equilibrium. The following chapter contains several example problems illustrating the approach described in this chapter.

Chapter 4

Numerical Examples

This chapter contains numerical examples illustrating the approach described in chapter 2 for the computation of the azeotropes and heteroazeotropes present in a multicomponent mixture as well as examples illustrating the extensions described in chapter 3 for examining changes in phase equilibrium structure.

In the first section, the azeotropes and heteroazeotropes present in several multicomponent mixtures are computed. The second section contains eight examples illustrating the approaches discussed in chapter 3. All numerical calculations were performed using the NRTL equation to model the liquid phase activity coefficients and the Extended Antoine Correlation was used to compute the vapor pressure. Constants for both models were obtained from [5]. In all cases, the vapor phase was treated as an ideal gas.

4.1 Computation of Homogeneous and Heterogeneous Azeotropes

The following heterogeneous systems were examined: 1) benzene, ethanol, and water, 2) ethyl acetate, ethanol, and water, 3) water, acetone, and chloroform, 4) toluene, ethanol, and water, 5) benzene, isopropanol, and water, 6) methanol, benzene, and heptane, 7) benzene, ethanol, water, and heptane, and 8) benzene, ethanol, water,

and cyclohexane. Tables 4.1 and 4.2 contain a summary of the experimental and computed values for the azeotropes and heteroazeotropes at a pressure of 1 atm. Experimental values were obtained from [60]. The literature indicated that the benzene-ethanol-heptane azeotrope does not exist above 0.24 atm, although, the NRTL model parameters employed did predict this ternary azeotrope at 1 atm. The value reported in Table 4.2 is at 0.24 atm. The thick lines in Figures 4-1 through 4-8 correspond to the heterogeneous branches and the dashed regions denote where the liquid phase fraction is outside its physical bounds of zero and unity. The thin lines in these figures correspond to homogeneous branches. Circles in these diagrams correspond to bifurcation points and the diamonds appear at intersection points. All bifurcation diagrams in this section were constructed at a pressure of one bar (the computed values are only slightly different than those in the table computed at one atm).

4.1.1 Benzene-Ethanol-Water System

At one bar, this system exhibits two binary homogeneous azeotropes, one binary heteroazeotrope, and a ternary heteroazeotrope. The binary homogeneous water-ethanol branch bifurcates off the lower boiling ethanol branch (the binary azeotrope is minimum boiling) at $\lambda = 0.820$. Similarly, the minimum boiling binary homogeneous benzene-ethanol branch also bifurcates off the lower boiling ethanol branch at $\lambda = 0.0185$. As shown in chapter 2, this must occur. Notice that this second bifurcation occurs at a relatively small value for λ. By using the exact criteria for a bifurcation on a homogeneous branch, equation (2.21), there is no chance of missing bifurcations close to zero by stepping over them on the first step. The more interesting branches for this mixture bifurcate off the pure benzene branch. Figure 4-1 contains a bifurcation diagram showing the branches bifurcating off this branch.

A spurious homogeneous benzene-water branch bifurcates off the pure benzene branch at $\lambda = 0.0153$. Along this branch, an intersection point is identified at $\lambda = 0.278$ from which the actual binary benzene-water heteroazeotrope is obtained using the heterogeneous homotopy map. Also identified on the spurious binary branch is

Table 4.1: Experimental and computed values for azeotropes and heteroazeotropes at a pressure of 1 atm. Heteroazeotropes are denoted by boldface.

Components	Experimental Composition	Experimental Temp. (K)	Computed Composition	Computed Temp. (K)
Benzene,	(0.5607,0.4393,0)	341.25	(0.5547,0.4453,0)	340.86
Ethanol,	**(0.7003,0,0.2997)**	**342.40**	**(0.7010,0,0.2990)**	**342.50**
Water	(0,0.9037,0.0963)	351.32	(0,0.8953,0.1047)	351.30
	(0.5387,0.2282,0.2331)	**338.01**	**(0.5273,0.2815,0.1912)**	**337.17**
Ethyl Acetate,	(0.5380,0.4620,0)	344.96	(0.5532,0.4467,0)	344.56
Ethanol,	**(0.6885,0,0.3115)**	**343.53**	**(0.6743,0,0.3257)**	**344.96**
Water	(0,0.9037,0.0963)	351.32	(0,0.8953,0.1047)	351.30
	(0.5789,0.1126,0.3085)	343.38	(0.5370,0.1813,0.2817)	343.65
Water,	**(0.1604,0,0.8396)**	**329.25**	**(0.1640,0,0.8360)**	**328.29**
Acetone,	(0,0.3397,0.6603)	337.15	(0,0.3449,0.6551)	337.30
Chloroform	**(0.1626,0.4844,0.3530)**	**333.55**	**(0.1832,0.4031,0.4137)**	**332.79**
Toluene,	(0.1905,0.8095,0)	349.85	(0.1907,0.8093,0)	350.00
Ethanol,	**(0.4439,0,0.5561)**	**357.25**	**(0.4411,0,0.5589)**	**357.65**
Water	(0,0.9037,0.0963)	351.32	(0,0.8953,0.1047)	351.30
	(0.2736,0.3971,0.3293)	**347.55**	**(0.2600,0.4668,0.2732)**	**346.98**
Benzene,	(0.6022,0.3978,0)	344.89	(0.5980,0.4020,0)	344.91
Isopropanol,	**(0.7003,0,0.2997)**	**342.40**	**(0.7010,0,0.2990)**	**342.50**
Water	(0,0.6875,0.3125)	353.25	(0,0.6650,0.3350)	353.55
	(0.5650,0.1861,0.2489)	**339.66**	**(0.5265,0.2345,0.2390)**	**338.86**
Methanol,	(0.6031,0.3969,0)	330.65	(0.6070,0.3930,0)	331.09
Benzene,	(0,0.9945,0.0055)	353.25	Not Predicted at 1 atm	—
Heptane	**(0.7684,0,0.2316)**	**332.25**	**(0.7773,0,0.2227)**	**332.12**
	N/A	N/A	(0.6261,0.2950,0.0789)	330.80

a bifurcation point onto a spurious homogeneous ternary branch at $\lambda = 0.322$. Moving forward in λ from this bifurcation point, a spurious homogeneous ternary azeotrope is obtained which lies outside the physical composition space. Moving in the reverse direction from the bifurcation point, an intersection with a heterogeneous ternary branch is identified at $\lambda = 0.295$ from which the actual ternary heteroazeotrope is obtained. In this example, the ternary heteroazeotrope can also be obtained through a bifurcation on the binary heterogeneous branch (this bifurcation occurs at $\lambda = 0.289$). There are two additional bifurcation points on the spurious ternary branch as shown in Figure 4-2 (this figure includes the pure ethanol branch and the ethanol-benzene

Table 4.2: Experimental and computed values for azeotropes and heteroazeotropes at a pressure of 1 atm. Heteroazeotropes are denoted by boldface.

Components	Experimental Composition	Experimental Temp. (K)	Computed Composition	Computed Temp. (K)
Benzene,	(0.5607,0.4393,0,0)	341.25	(0.5547,0.4453,0,0)	340.86
Ethanol,	**(0.7003,0,0.2997,0)**	**342.40**	**(0.7010,0,0.2990,0)**	**342.50**
Water,	(0,0.9037,0.0963,0)	351.32	(0,0.8953,0.1047,0)	351.30
Heptane	**(0.5387,0.2282,0.2331,0)**	**338.01**	**(0.5273,0.2815,0.1912,0)**	**337.17**
	(0.9945,0,0,0.0055)	353.25	Not Predicted at 1 atm	—
	N/A	N/A	(0,0.3478,0,0.6522)	344.55
($P = 0.24$ atm)	(0.6462,0.3307,0,0.0231)	305.53	(0.6377,0.3464,0,0.0159)	305.78
	N/A	N/A	**(0,0,0.4505,0.5495)**	**352.45**
	N/A	N/A	**(0,0.4606,0.1793,0.3601)**	**342.33**
	(0.4691,0.2383,0.2216,0.0709)	**337.94**	**(0.5204,0.2820,0.1909,0.0067)**	**337.17**
Benzene,	(0.5607,0.4393,0,0)	341.25	(0.5547,0.4453,0,0)	340.86
Ethanol,	**(0.7003,0,0.2997,0)**	**342.40**	**(0.7010,0,0.2990,0)**	**342.50**
Water,	(0,0.9037,0.0963,0)	351.32	(0,0.8953,0.1047,0)	351.30
Cyclohexane	**(0.5387,0.2282,0.2331,0)**	**338.01**	**(0.5273,0.2815,0.1912,0)**	**337.17**
	(0.5372,0,0,0.4628)	350.71	(0.5432,0,0,0.4568)	350.62
	(0,0.5491,0,0.4509)	338.05	(0,0.4390,0,0.5610)	338.00
	(0,0,0.3000,0.7000)	**342.65**	**(0,0,0.2993,0.7007)**	**342.51**
	(0,0.2688,0.1675,0.5638)	**335.75**	**(0,0.3199,0.1509,0.5292)**	**335.43**
	(0.0924,0.4409,0,0.4668)	338.20	(0.1084,0.4224,0,0.4692)	337.82
	(0.1630,0.2237,0.2334,0.3800)	**335.25**	**(0.1906,0.2857,0.1609,0.3628)**	**335.00**

branch that bifurcates off it). The first occurs at the turning point of the spurious ternary branch where there is an intersection with a homogeneous benzene-ethanol branch at $\lambda = 0.178$. This turning point occurs on the boundary of $\mathcal{S}$ and thus, corollary 1 in chapter 2 is not violated. In addition, the point at which the spurious ternary branch crosses $x_B = 0$ ($\lambda = 0.399$) corresponds to an intersection with a homogeneous ethanol-water branch. Higher dimensional branches can typically be obtained through bifurcations on several lower dimensional branches. However, the theory developed in chapter 2 indicates where the bifurcations can be found, limiting the amount of branch tracking required. Finally, since the spurious ternary azeotrope lies outside the physical composition space, the ternary heteroazeotrope can also be obtained by a third, independent mechanism by using the spurious azeotrope homo-

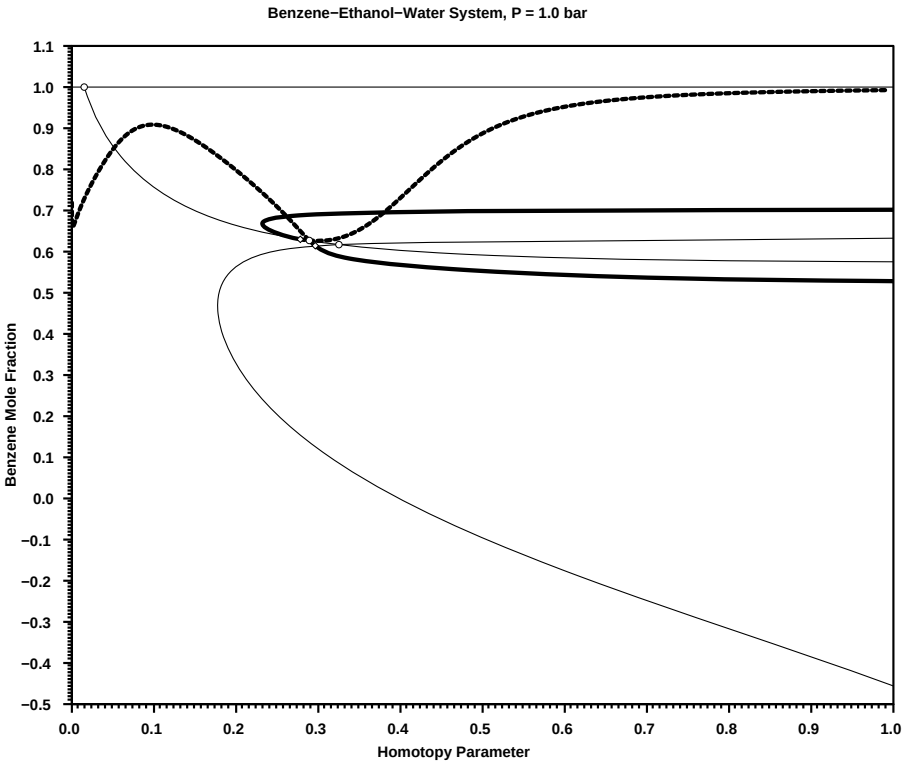

Figure 4-1: Benzene mole fraction versus the homotopy parameter for ternary system: benzene, ethanol, and water. Thin lines are homogeneous curves and thick lines are heterogeneous curves.

topy map described in chapter 2 (this is referred to as mechanism 2 in section 2.5). The homotopy paths connecting the heteroazeotropes to the spurious homogeneous azeotropes using the spurious homotopy map, equation (2.39), is shown in Figure 4-3.

4.1.2 Ethyl Acetate-Ethanol-Water System

At one bar, this system exhibits two homogeneous binary azeotropes, one binary heteroazeotrope, and a homogeneous ternary azeotrope. As in the system above, the homogeneous ethanol-water branch bifurcates off the pure ethanol branch. Figure 4-4 contains a bifurcation diagram showing the branches bifurcating off the pure ethyl acetate branch.

The first bifurcation point off the pure ethyl acetate branch (at $\lambda = 0.0337$) corresponds to the ethyl acetate-ethanol branch from which the homogeneous binary

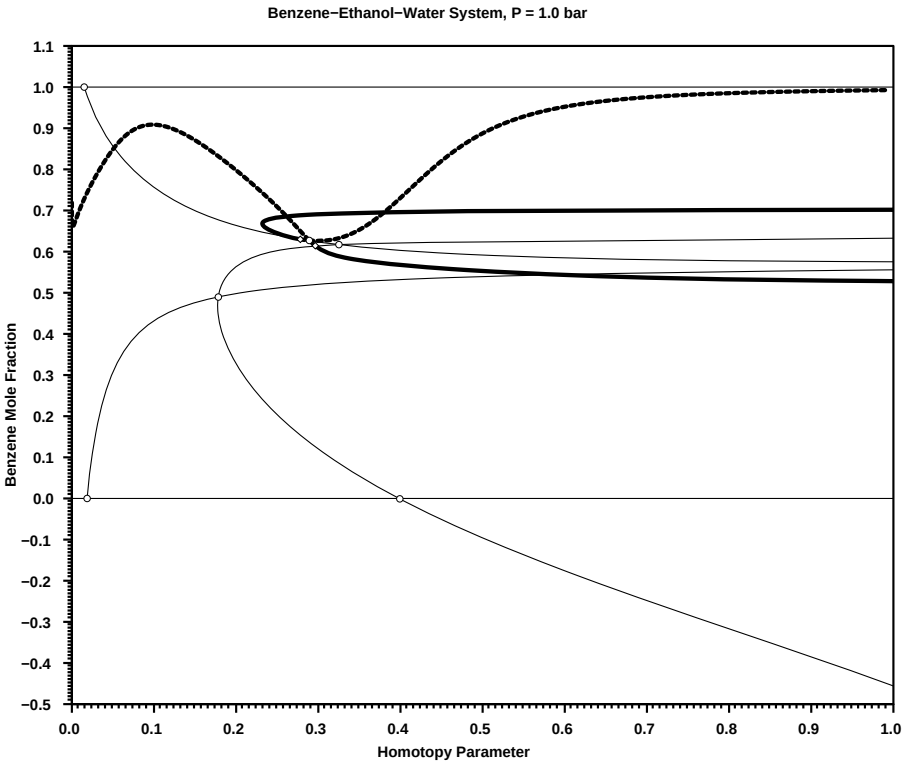

Figure 4-2: Benzene mole fraction versus the homotopy parameter for ternary system: benzene, ethanol, and water. Thin lines are homogeneous curves and thick lines are heterogeneous curves.

azeotrope is obtained. A bifurcation point is identified on this binary branch at $\lambda = 0.544$ which corresponds to a ternary branch from which the homogeneous ternary azeotrope is obtained. A second bifurcation point is identified along the pure ethyl acetate branch at $\lambda = 0.282$ which corresponds to a spurious homogeneous ethyl acetate-water branch. At $\lambda = 0.742$ on this spurious branch an intersection point is identified from which the actual ethyl acetate-water heteroazeotrope is obtained.

Section 4.2.4 also contains an example of this ternary mixture. There it is shown how a ternary heteroazeotrope is readily predicted at a different pressure by moving slightly past $\lambda = 1$ on the ternary homogeneous branch.

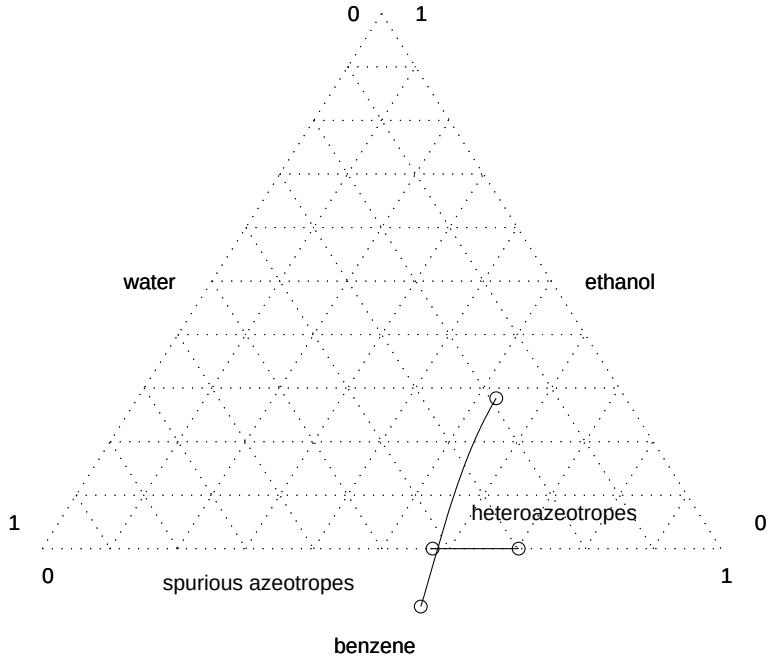

Figure 4-3: Homotopy paths connecting heteroazeotropes to spurious homogeneous azeotropes for the benzene, ethanol, and water system at 1.0 bar.

4.1.3 Water-Acetone-Chloroform System

At one bar, this system exhibits one homogeneous binary azeotrope, one binary heteroazeotrope, and a ternary heteroazeotrope. Figure 4-5 contains a bifurcation diagram showing the branches bifurcating off the pure chloroform branch from which all solutions are obtained.

The first bifurcation point ($\lambda = 0.059$) identified along the pure chloroform branch corresponds to the spurious chloroform-water branch. Along this branch an intersection point is identified at $\lambda = 0.265$ from which the actual binary heteroazeotrope is obtained. In addition, a bifurcation point is identified along the spurious binary branch at $\lambda = 0.20$ which corresponds to a spurious ternary branch. Three intersection points are identified along the spurious ternary branch ($\lambda = 0.67$, 0.73 and 0.96), all of which correspond to the same heterogeneous ternary branch. Two heterogeneous ternary solutions are identified on this branch, one physical and the other nonphysical

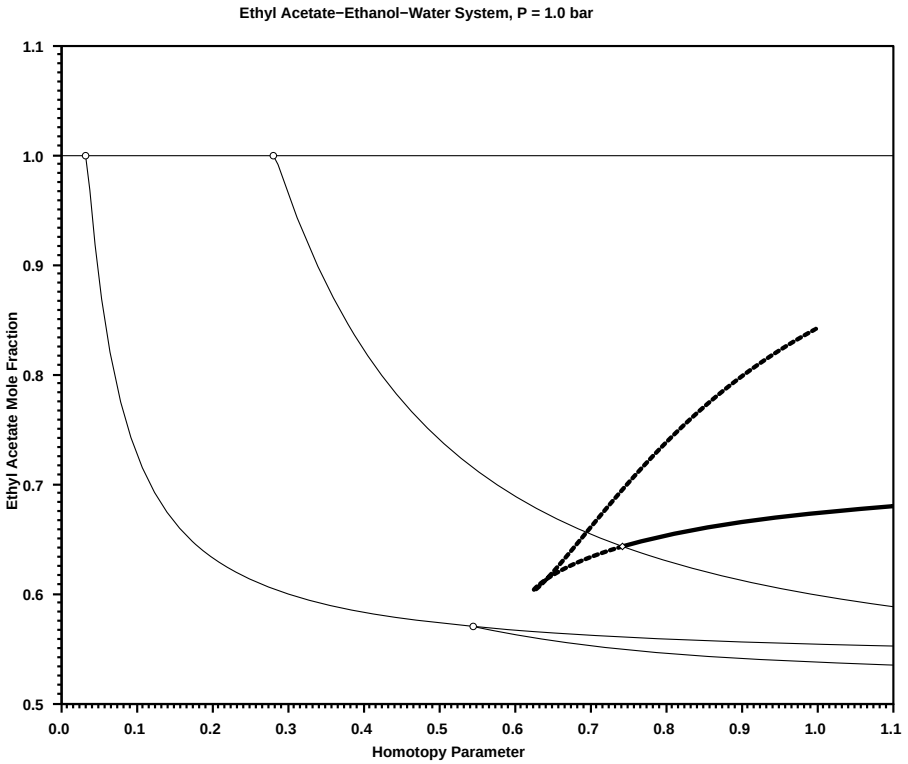

Figure 4-4: Ethyl acetate mole fraction versus the homotopy parameter for ternary system: ethyl acetate, ethanol, and water. Thin lines are homogeneous curves and thick lines are heterogeneous curves.

(the liquid phase fraction lies outside the range zero and unity). Note that in this case, the ternary heterogeneous branch cannot be obtained through a bifurcation on the binary heterogeneous branch. However, this is consistent with the theory in chapter 2 in that the ternary heterogeneous branch is obtainable through an intersection with a ternary spurious branch (under reasonable assumptions, all heteroazeotropes will be obtained either through intersections with spurious homogeneous branches *or* through bifurcations on lower dimensional heterogeneous branches). The second bifurcation point on the pure chloroform branch ($\lambda = 0.278$) leads to an actual homogeneous chloroform-acetone azeotrope.

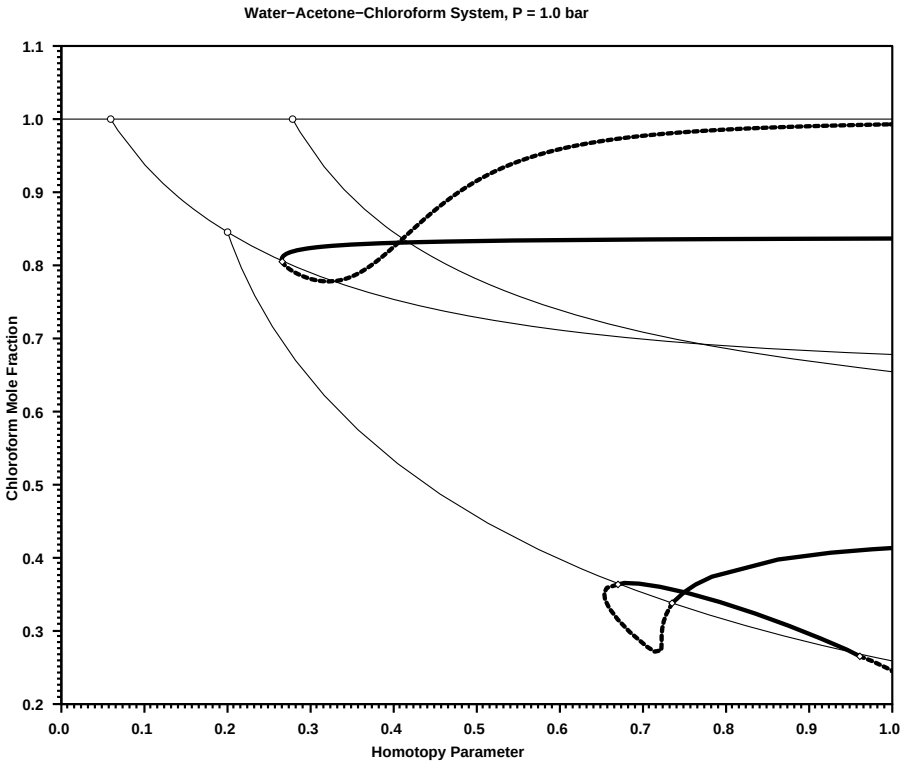

Figure 4-5: Chloroform mole fraction versus the homotopy parameter for ternary system: water, acetone, and chloroform. Thin lines are homogeneous curves and thick lines are heterogeneous curves.

4.1.4 Toluene-Ethanol-Water System

At one bar, this system exhibits two homogeneous binary azeotropes, one binary heteroazeotrope, and one ternary heteroazeotrope. The binary ethanol-water azeotrope is obtained just as it was in the previous systems containing these two components, via a bifurcation on the pure ethanol branch. In addition, the homogeneous toluene-ethanol branch bifurcates off the pure ethanol branch. Figure 4-6 contains a bifurcation diagram showing the branches bifurcating off the pure water branch.

The spurious homogeneous toluene-water branch bifurcates off the pure water branch at $\lambda = 0.0006$. This example again shows the importance of using exact value for the bifurcation λ's. Along this branch an intersection with the binary heterogeneous branch and another bifurcation point are identified, $\lambda = 0.367$ and 0.913, respectively. The actual binary heteroazeotrope and a spurious homogeneous ternary azeotrope are obtained from these two points. A second intersection point

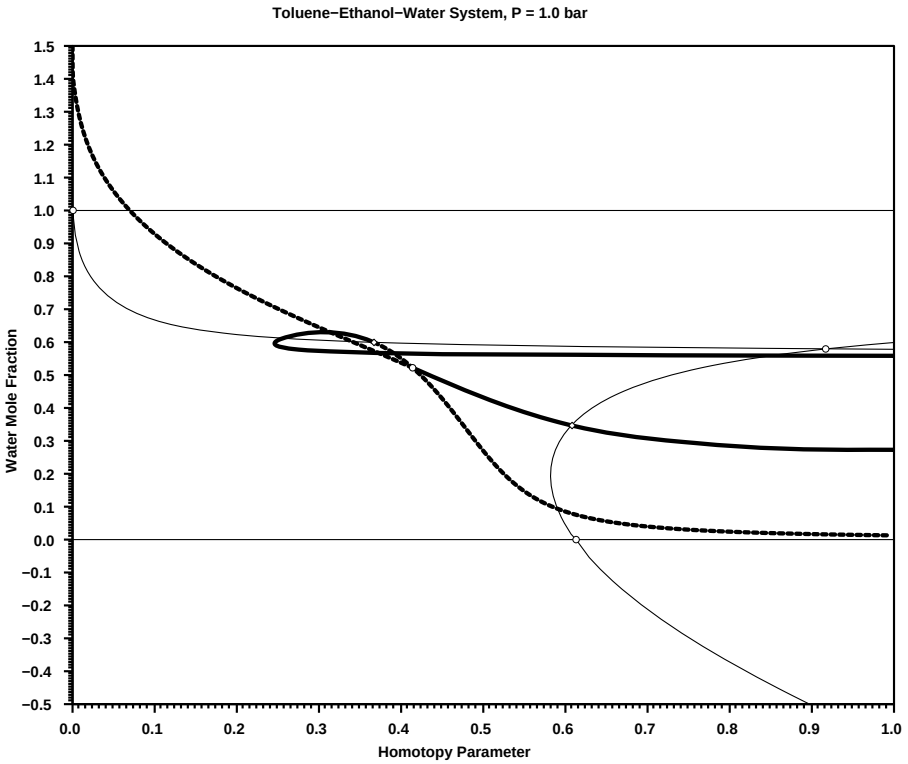

Figure 4-6: Water mole fraction versus the homotopy parameter for ternary system: toluene, ethanol, and water. Thin lines are homogeneous curves and thick lines are heterogeneous curves.

is identified along the spurious ternary branch from which the corresponding ternary heteroazeotrope is obtained. As with the benzene-ethanol-water system, the ternary heteroazeotrope can also be obtained through a bifurcation on the binary heterogeneous branch and through the spurious azeotrope homotopy map (mechanisms 1 and 2). Furthermore, the point at which the spurious ternary branch crosses $x_W = 0$ corresponds to an intersection with the homogeneous toluene-ethanol branch which bifurcates off the pure ethanol branch and leads to the toluene-ethanol azeotrope at $\lambda = 1$. The dramatic difference between this bifurcation diagram and bifurcation diagram of the physically similar mixture, benzene, ethanol, and water, is due to the fact that at a pressure of one bar, $T_B^s < T_W^s < T_T^s$, where T^s denotes the boiling temperature. In the benzene-ethanol-water mixture the spurious homogeneous branches bifurcate off the lower boiling benzene branch, whereas in this example, these bifurcations occur on the pure water branch. Like the benzene-ethanol-water system,

the spurious ternary azeotrope in this mixture lies outside the physical composition space and is thus obtainable with the spurious homotopy map. Figure 4-7 contains the homotopy branches associated with this map.

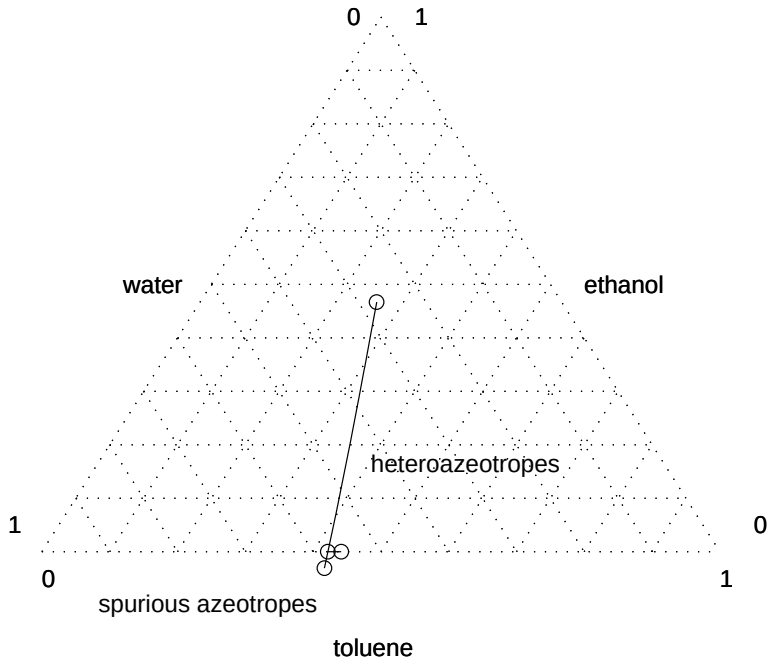

Figure 4-7: Homotopy paths connecting heteroazeotropes to spurious homogeneous azeotropes for the toluene, ethanol, and water system at 1.0 bar.

4.1.5 Benzene-Isopropanol-Water System

At one bar, this system exhibits two binary azeotropes, a binary heteroazeotrope, and a ternary heteroazeotrope. Figure 4-8 contains a bifurcation diagram showing the branches bifurcating off the pure benzene branch.

The first bifurcation point on the pure benzene branch ($\lambda = 0.0153$) corresponds to the spurious benzene-water branch, along which an intersection point is identified at $\lambda = 0.278$ that leads to the actual binary heteroazeotrope. The second bifurcation point on the pure benzene branch ($\lambda = 0.027$) corresponds to a benzene-isopropanol

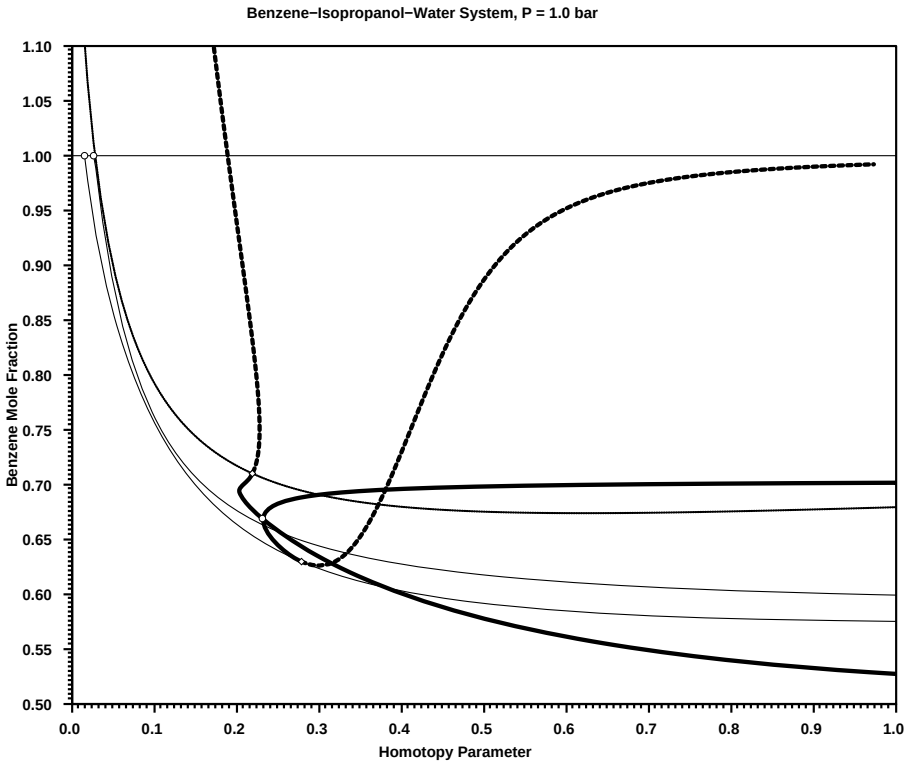

Figure 4-8: Benzene mole fraction versus the homotopy parameter for ternary system: benzene, isopropanol, and water. Thin lines are homogeneous curves and thick lines are heterogeneous curves.

branch, leading to an actual binary azeotrope. The close proximity of these two bifurcation points on the pure benzene branch again illustrates the importance of using the exact criteria for bifurcation points. Clearly, it would be very easy to completely miss these two bifurcation points by jumping over them (causing a cancellation in the sign change of the determinant of the Jacobian) with even a fairly small stepsize. Along the binary heterogeneous branch, a bifurcation point is identified at $\lambda = 0.233$ which corresponds to a ternary heterogeneous branch, from which the actual ternary heteroazeotrope is obtained. Note, however, in this case the spurious ternary branch is *not* obtainable through bifurcations on lower dimensional homogeneous branches. The spurious ternary branch is identified (drawn as a medium thickness line in Figure 4-8) by monitoring the s-component on the ternary heterogeneous branch and this spurious ternary branch lies completely outside the physical composition space, $\mathcal{C}$. As with the water-acetone-chloroform system above, this is consistent with the

theory in chapter 2 in that the ternary heterogeneous branch is obtainable through a bifurcation on a lower dimensional heterogeneous branch. Furthermore, even though the ternary spurious homogeneous azeotrope is not obtainable from the lower dimensional homogeneous branches, the ternary heteroazeotrope is still obtained through mechanism 2 (using the spurious homotopy map). Figure 4-9 contains the homotopy branches associated with this mechanism for obtaining the ternary heteroazeotrope.

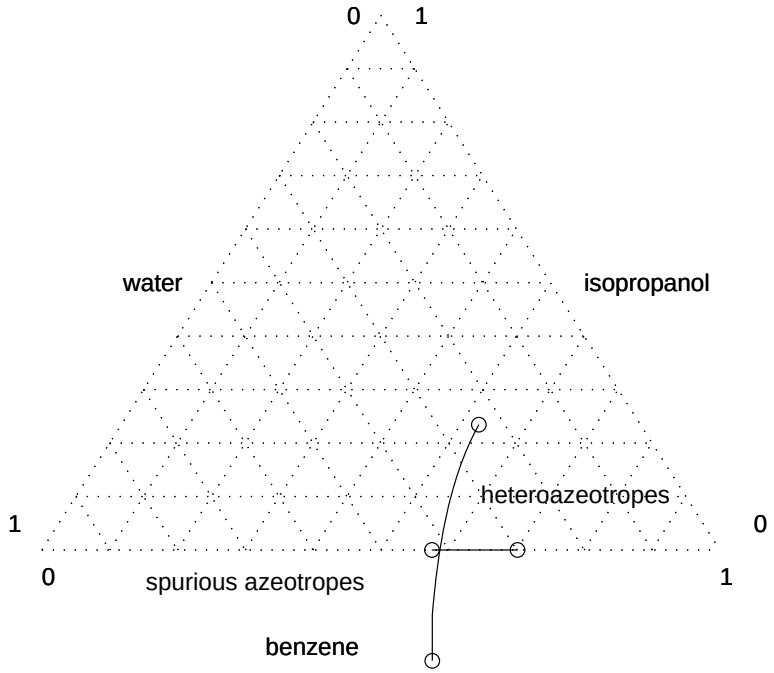

Figure 4-9: Homotopy paths connecting heteroazeotropes to spurious homogeneous azeotropes for the benzene, isopropanol, and water system at 1.0 bar.

4.1.6 Methanol-Benzene-Heptane System

At a pressure of bar, the phase equilibrium model employed in this calculation predicts the methanol-benzene-heptane mixture exhibits a homogeneous methanol-benzene azeotrope, a methanol-heptane heteroazeotrope, and a ternary homogeneous azeotrope. Experimental data was not available for the ternary azeotrope. Experi-

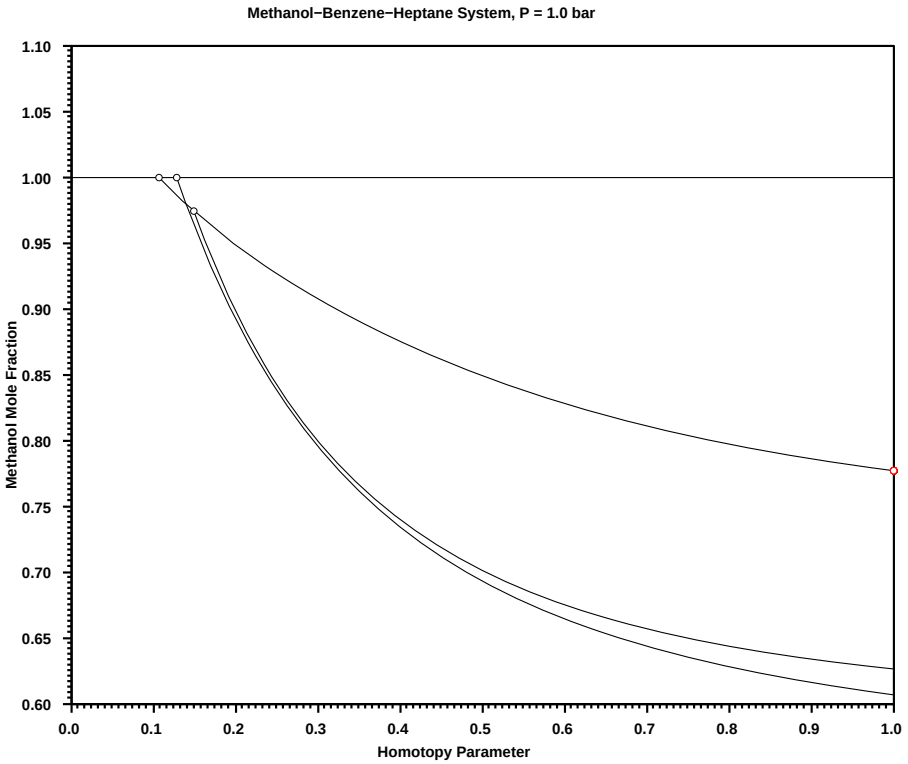

Figure 4-10: Methanol mole fraction versus the homotopy parameter for ternary system: methanol, benzene, and heptane.

mental data did indicate that a homogeneous benzene-heptane azeotrope exists, however, this azeotrope was not predicted by the phase equilibrium model. Section 4.2.8 contains an example where the techniques developed in chapter 3 are applied to this mixture to determine the conditions under which this missing azeotrope is predicted.

Figure 4-10 contains the bifurcation diagram for this mixture. All predicted azeotropes and heteroazeotropes are obtained from the pure methanol branch.

The first bifurcation on the pure methanol branch ($\lambda = 0.106$) leads to a spurious methanol-heptane azeotrope. The intersection point with the binary heterogeneous branch occurs very close to $\lambda = 1$ (identified by the diamond in Figure 4-10). Along this spurious homogeneous branch, a bifurcation point is identified at $\lambda = 0.149$ which leads to the ternary homogeneous azeotrope. The second bifurcation point on the pure methanol branch ($\lambda = 0.128$) corresponds to a branch leading to the methanol-benzene azeotrope.

4.1.7 Benzene-Ethanol-Water-Heptane System

The algorithm for computing homogeneous and heterogeneous azeotropes described in this thesis computes them in a hierarchical manner: ternary azeotropes are computed from binary branches, quaternary azeotropes from ternary branches, and so on. This is not generally a problem because we often desire all homogeneous and heterogeneous azeotropes predicted by the phase equilibrium model. An advantage of this approach is that we may use precomputed information from previously examined systems when analyzing higher dimensional systems. For example, in section 4.1.1, the benzene, ethanol, and water system was examined and all homogeneous and heterogeneous azeotropes were computed. In this section, heptane is added to this mixture. Along the heterogeneous benzene-ethanol-water branch, computed in section 4.1.1, a bifurcation point corresponding to a heterogeneous quaternary branch was identified at $\lambda = 0.920$. This branch led to the stable quaternary heteroazeotrope. Figure 4-11 contains the bifurcation diagram for this mixture with the additional quaternary heterogeneous branch.

The benzene-water, benzene-water-ethanol, and benzene-ethanol-water-heptane heteroazeotropes are obtained from the pure benzene branch. The ethanol-benzene, ethanol-benzene-heptane, ethanol-heptane, and ethanol-water homogeneous azeotropes and a ethanol-heptane-water heteroazeotrope are obtained from the pure ethanol branch. Finally, a heptane-water heteroazeotrope is obtained from the pure heptane branch. As shown in Table 4.2, no exerimental data was found for several of azeotropes and heteroazeotropes in this mixture. These solutions do, however, satisfy the necessary and sufficient conditions for homogeneous and heterogeneous azeotropy.

4.1.8 Benzene-Ethanol-Water-Cyclohexane System

At a pressure of one bar, a mixture of benzene, ethanol, water, and cyclohexane exhibit four homogeneous binary azeotropes, two heterogeneous binary azeotropes, one homogeneous ternary azeotrope, two heterogeneous ternary azeotropes, and a quaternary heteroazeotrope (see Table 4.2). A very rich bifurcation diagram can be

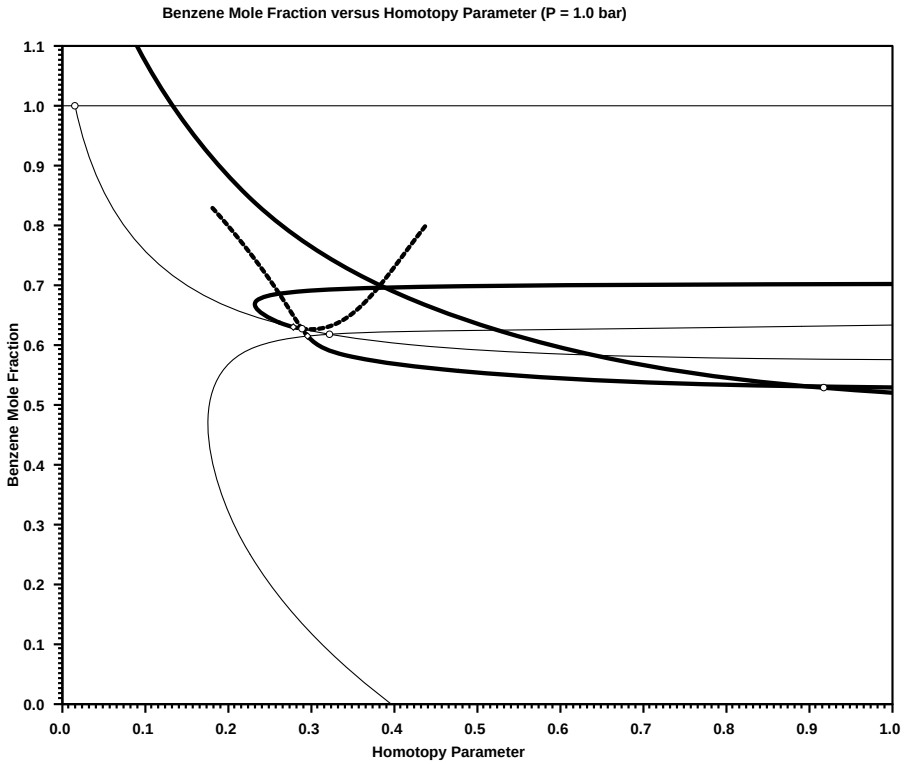

Figure 4-11: Benzene mole fraction versus the homotopy parameter for quaternary system: benzene, ethanol, water, and heptane.

constructed from the pure benzene branch. From this diagram, shown in Figure 4-12, the following heteroazeotropes are obtained: benzene-water, benzene-water-ethanol, benzene-water-cyclohexane, and benzene-water-cyclohexane-ethanol. In addition, a benzene-cyclohexane homogeneous azeotrope is obtained.

Three binary bifurcation points are obtained off the pure ethanol branch leading to homogeneous ethanol-cyclohexane, ethanol-benzene, and ethanol-water azeotropes. A stable ethanol-cylohexane-benzene homogeneous azeotrope and a spurious ethanol-cyclohexane-water azeotrope are obtained from the ethanol-cylcohexane binary branch. An intersection point is identified along this spurious branch leading to the stable ethanol-cyclohexane-water heteroazeotrope. The cyclohexane-water heteroazeotrope is obtained from the spurious cyclohexane-water branch which bifurcates off the pure cyclohexane branch.

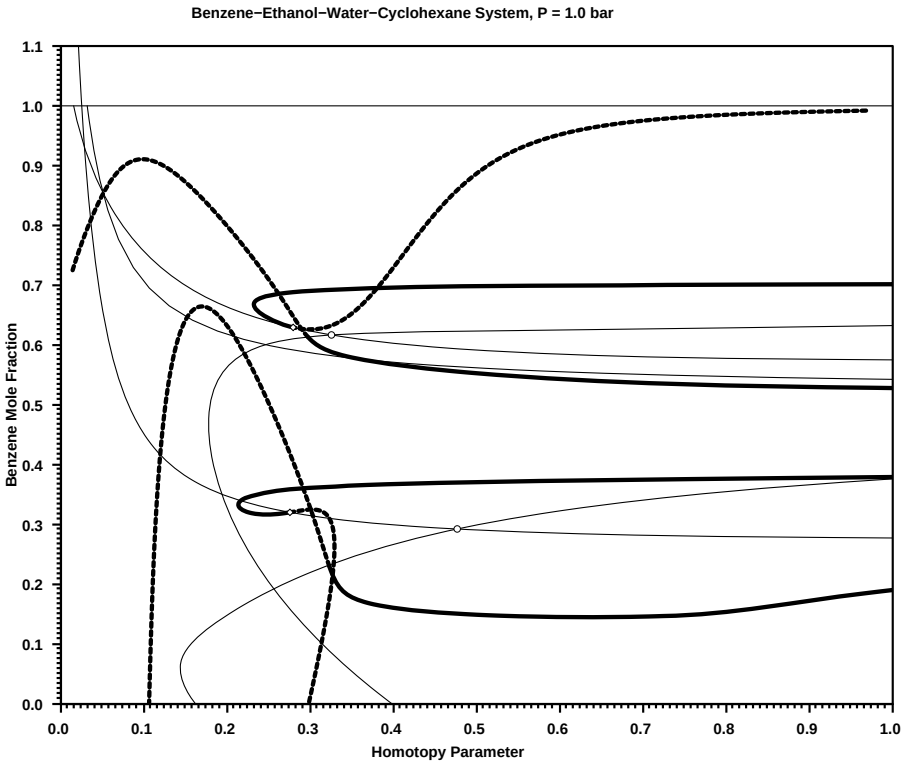

Figure 4-12: Benzene mole fraction versus the homotopy parameter for quaternary system: benzene, ethanol, water, and cyclohexane.

4.2 Changes in Phase Equilibrium Structure

In this section, examples associated with the extensions of the heteroazeotrope finding algorithm discussed in chapter 3 are presented. There is limited experimental data on azeotropes and heteroazeotropes at the pressures computed in this section. However, all solutions obtained satisfy the necessary and sufficient conditions for homogeneous and heterogeneous azeotropy (subject to the limitations of the model and parameters employed).

4.2.1 Chloroform-Methanol System

A mixture of chloroform and methanol will form a homogeneous azeotrope at a pressure of one bar. The homogeneous homotopy branch associated with this minimum boiling binary azeotrope bifurcates off the pure chloroform branch at $\lambda = 0.028$.

However, a chloroform-methanol bifurcation point also appears on the pure methanol branch at a value of $\lambda = -0.073$. This branch does not lead to a binary azeotrope since it is unable to cross $\lambda = 0$ in order to reach $\lambda = 1$. As pressure is increased, the value of λ at the bifurcation point decreases on the pure chloroform branch and increases on the pure methanol branch. At a pressure of 1.8 bar, the bifurcation points on both branches occur at $\lambda = 0$. At higher pressures, the bifurcation point on the chloroform branch occurs at $\lambda < 0$ and the bifurcation point on the methanol branch occurs at $\lambda > 0$. Thus, the branch associated with the binary azeotrope now bifurcates off the pure methanol branch. Figure 4-13 contains the bifurcation diagram for this system at three different pressures.

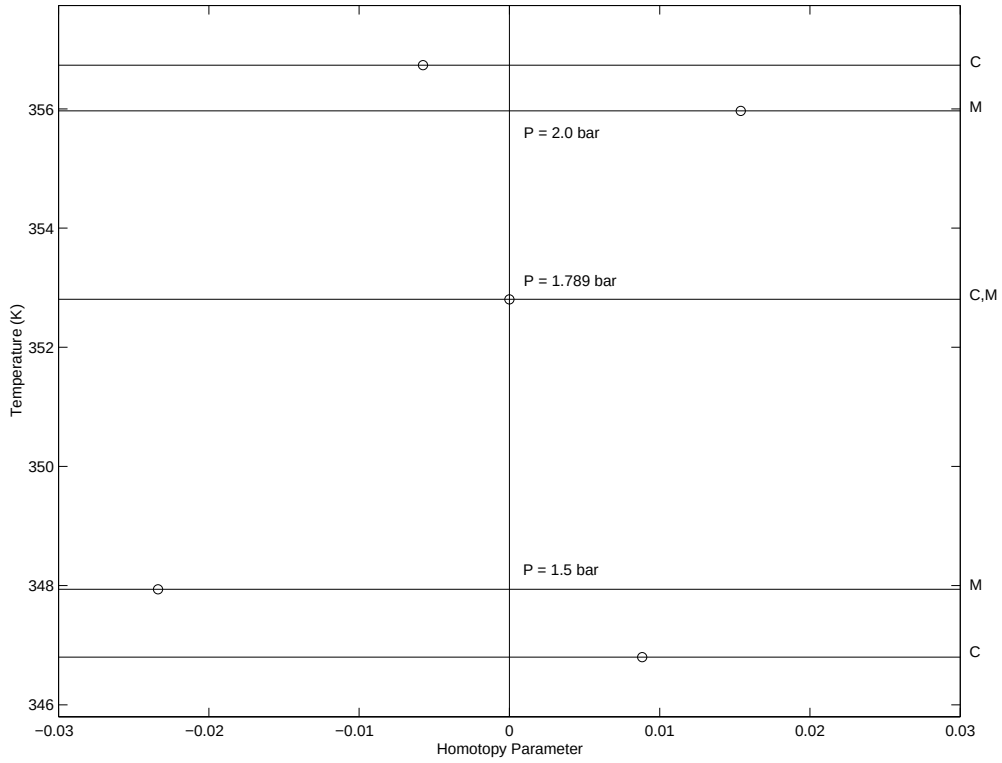

Figure 4-13: Bifurcation diagram for the chloroform-methanol system at pressures of 1.5 bar, 1.8 bar, and 2.0 bar.

This behavior can be explained physically by looking at the boiling temperatures of pure chloroform and methanol as a function of pressure. As shown in chapter 2, the minimum boiling chloroform-methanol azeotrope will be obtained through a bifurcation on the lower boiling component's branch. At a pressure less than 1.8 bar,

chloroform has a lower boiling point than methanol. At 1.8 bar, both chloroform and methanol have computed boiling points of 352.8 K. At pressures greater than 1.8 bar, methanol boils at a lower temperature than chloroform. Figure 4-14 contains the *T-xy* diagram for the chloroform-methanol system.

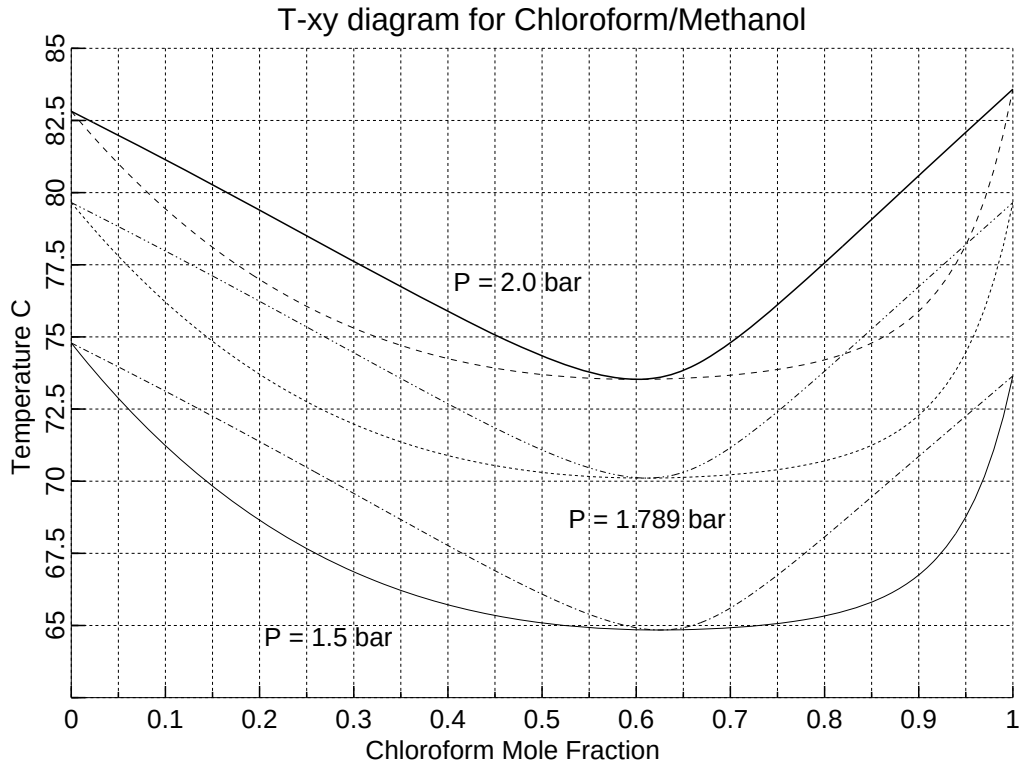

Figure 4-14: *T-xy* diagram for the chloroform-methanol system at 1.5 bar, 1.789 bar, and 2.0 bar.

4.2.2 Acetone-Ethanol System

At a pressure of one bar, a binary acetone-ethanol azeotrope is not predicted. However, if the pure acetone branch is extended beyond $\lambda = 1$, a bifurcation point corresponding to an acetone-ethanol branch is identified at $\lambda = 1.8$. This value of λ at the bifurcation point decreases with increasing pressure and equals unity at a pressure of 8.6 bar. At this point, the acetone-ethanol azeotrope emerges from the pure acetone vertex.

4.2.3 Water-2-Butoxyethanol System

A mixture of water and 2-butoxyethanol forms a homogeneous azeotrope at a pressure of one bar. An intersection with a heterogeneous branch is identified along the binary homogeneous branch at $\lambda = 1.03$. Following this heterogeneous branch to $\lambda = 1$, a solution is found that satisfies the necessary conditions for heteroazeotropy except for the fact that the liquid phase fraction equals 1.01. Figure 4-15 contains a diagram showing how the intersection point varies with pressure. The intersection point crosses $\lambda = 1$ at a pressure of 1.18 bar. At this pressure, the azeotropic point on the vapor-liquid equilibrium surface just touches the liquid-liquid region. The water-2-butoxyethanol mixture exhibits both an upper and a lower critical solution temperature. However, the point at which the heteroazeotrope leaves through the top of the liquid-liquid region was not predicted with this model. A better description of the vapor phase nonideality may rectify this deficiency.

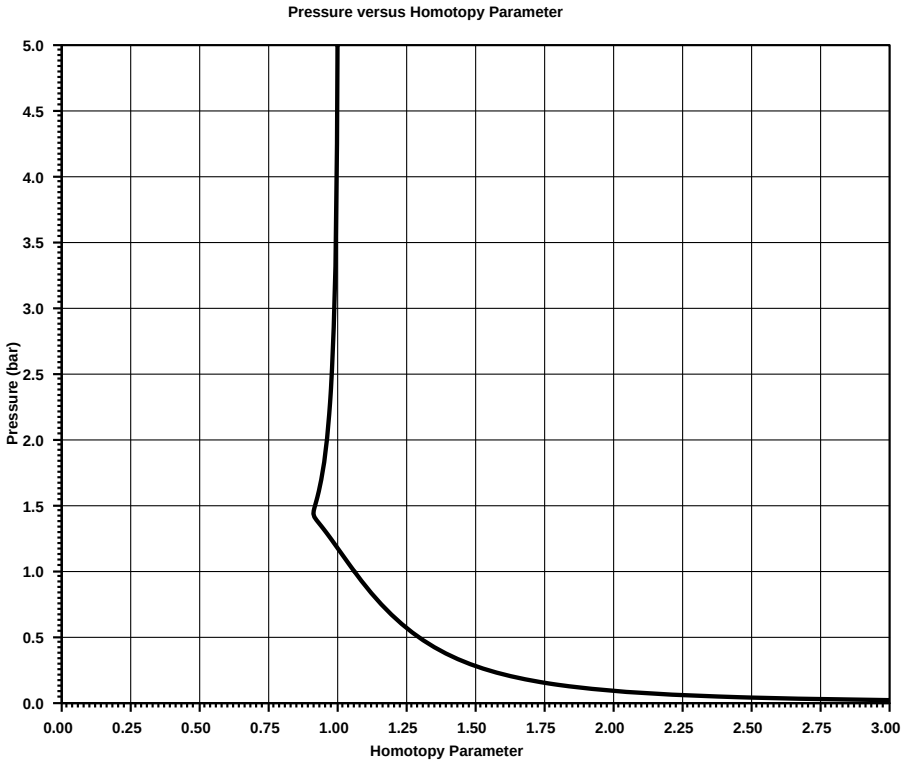

Figure 4-15: Pressure versus the value of λ at the intersection point for the water-2-butoxyethanol system.

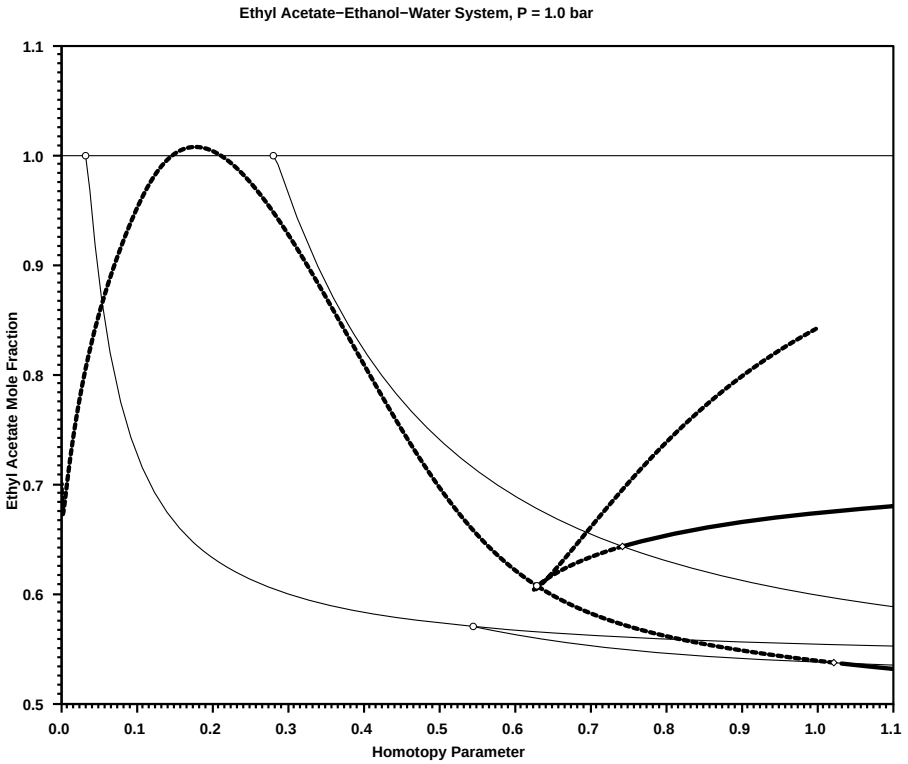

Figure 4-16: Ethyl acetate mole fraction versus the homotopy parameter at 1.0 bar for ternary system: ethyl acetate, ethanol, and water. Thin lines are homogeneous curves and thick lines are heterogeneous curves. The continuation is performed beyond $\lambda = 1$ in this figure.

4.2.4 Ethyl Acetate-Ethanol-Water System

As shown in section 4.1.2, at a pressure of one bar, a mixture of ethyl acetate, ethanol, and water exhibits a homogeneous ethanol-water azeotrope, a heterogeneous ethyl acetate-water azeotrope, and a homogeneous ternary azeotrope. In bifurcation diagram shown in Figure 4-4, the continuation was stopped at $\lambda = 1$. Figure 4-16 contains the same bifurcation diagram except with the continuation performed beyond $\lambda = 1$ on the ternary homogeneous branch.

If an intersection search is performed beyond $\lambda = 1$, an intersection with a ternary heterogeneous branch is located at $\lambda = 1.013$. The corresponding heterogeneous ternary branch crosses $\lambda = 1$ with a value of s equal to 1.008. Figure 4-17 contains a diagram showing how this intersection point varies with pressure.

At a pressure of approximately 0.67 bar, the intersection point occurs at $\lambda = 1$

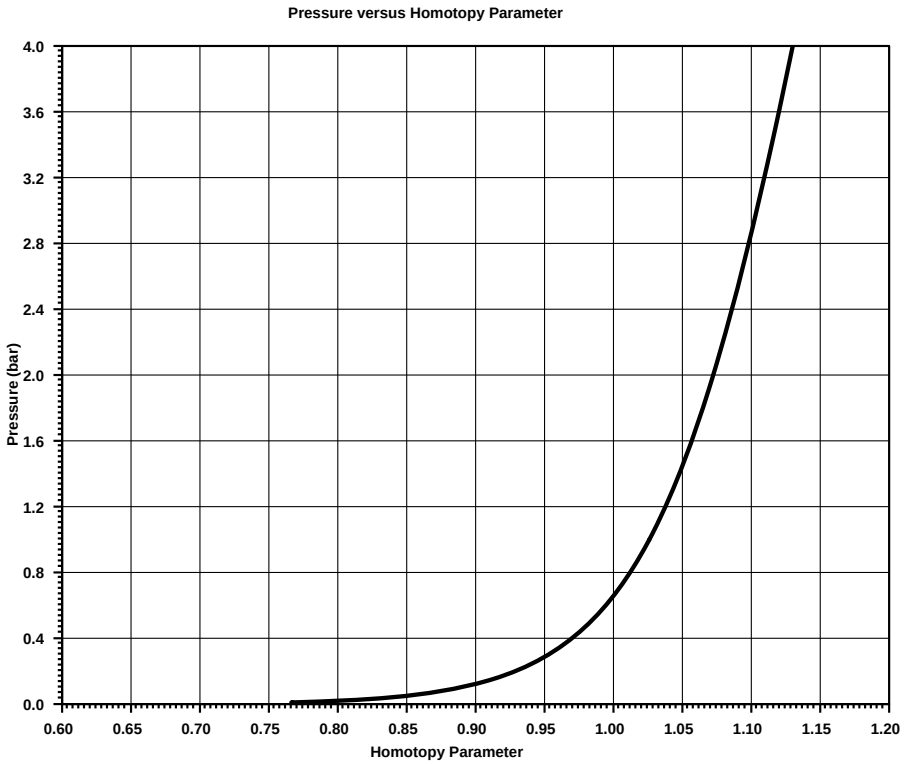

Figure 4-17: Pressure versus the value of λ at the intersection point for the ethyl acetate, ethanol, water system.

corresponding to the pressure where the azeotropic composition on the liquid-vapor equilibrium surface just touches the liquid-liquid binodal (which exhibits an upper critical solution temperature for this system).

4.2.5 Water-1,2-Dichloroethane System

A mixture of water and 1,2-dichloroethane forms a heteroazeotrope at one bar. The intersection point varies as a function of pressure as shown in Figure 4-18. At a pressure of approximately 0.057 bar, the heteroazeotrope leaves the liquid-liquid region and the heteroazeotrope becomes a homogeneous azeotrope.

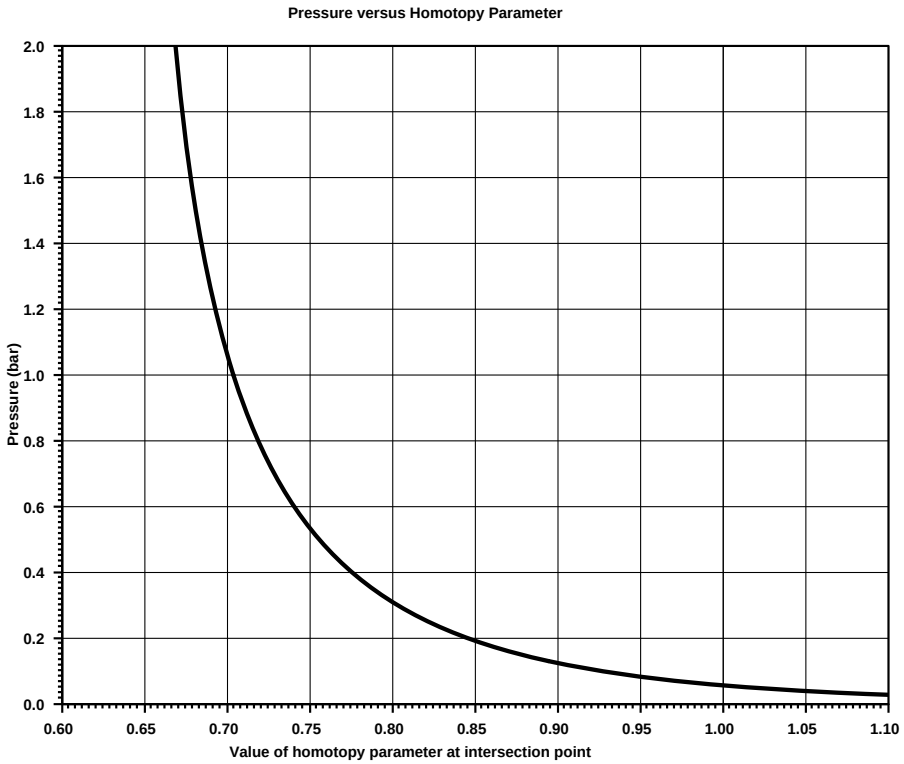

Figure 4-18: Pressure versus the value of λ at the intersection point for the water-1,2-dichloroethane system.

4.2.6 Tetrahydrofuran-Water System

The NRTL activity coefficient model, extended Antoine vapor pressure correlation, and ideal gas model predict that a mixture of tetrahydrofuran (THF) and water, which exhibits both a LCST and an UCST, does not form an azeotrope or heteroazeotrope at pressures less than approximately 0.64 bar, forms a homogeneous azeotrope at pressures between 0.64 bar and 2.3 bar, forms a heteroazeotrope between 2.3 bar and 18.5 bar, and forms a homogeneous azeotrope above 18.5 bar. Figure 4-19 shows how the location of the intersection point on the spurious homogeneous binary branch varies with pressure. Figure 4-20 shows the same diagram with the region around $\lambda = 1$ (the region of interest) enlarged. This system is different from the others in this paper in that there are two disconnected intersection point curves. The curve associated with the intersection points found at pressures less than approximately 2.3 bar lie on a curve that does not cross $\lambda = 1$ except at a non-physical solution at

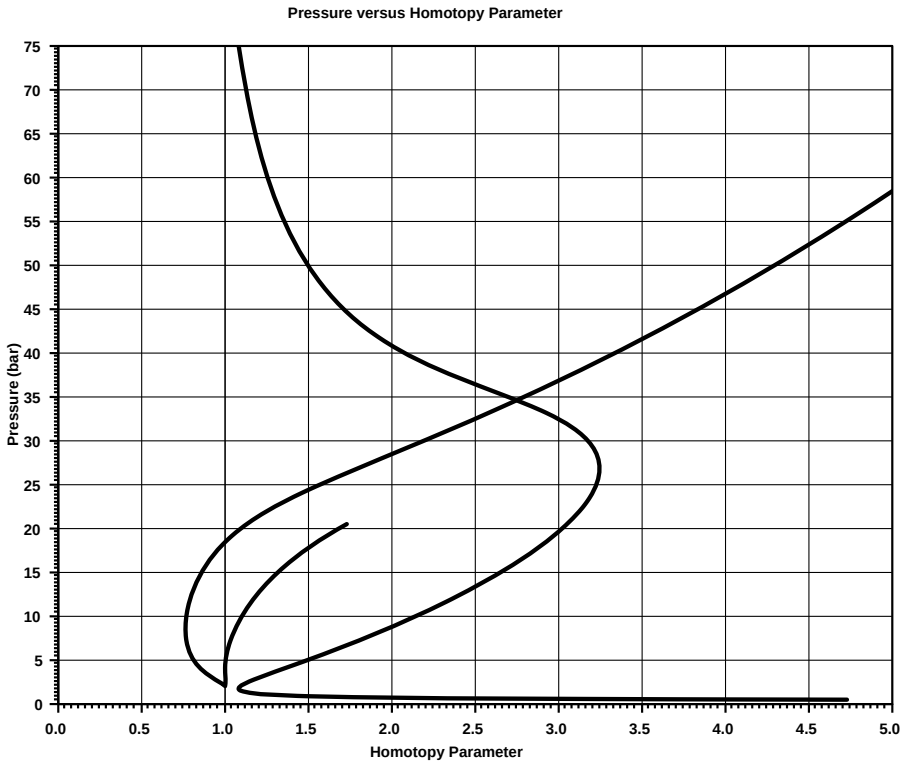

Figure 4-19: Pressure versus the value of λ at the intersection point for the THF-water system.

approximately 89 bar. This curve does not provide much useful information except for suggesting a second curve exists. However, it may be possible that the two intersection point curves are connected via a complex curve that bifurcates off the low pressure curve at the turning point at $\lambda \approx 1.07$. This connection in the complex domain was not explored, however, since the second curve is easily found by performing the intersection point search at a single pressure above 2.3 bar.

4.2.7 Water-Acetone-Chloroform System

As shown in section 4.1.3, at a pressure of one bar, a mixture of water, acetone, and chloroform forms a water-chloroform heteroazeotrope, a homogeneous acetone-chloroform azeotrope, and a ternary heteroazeotrope. Figure 4-5 contains the bifurcation diagram for this system constructed at one bar. The single intersection point on the spurious water-chloroform homogeneous branch does not cross $\lambda = 1$ for any

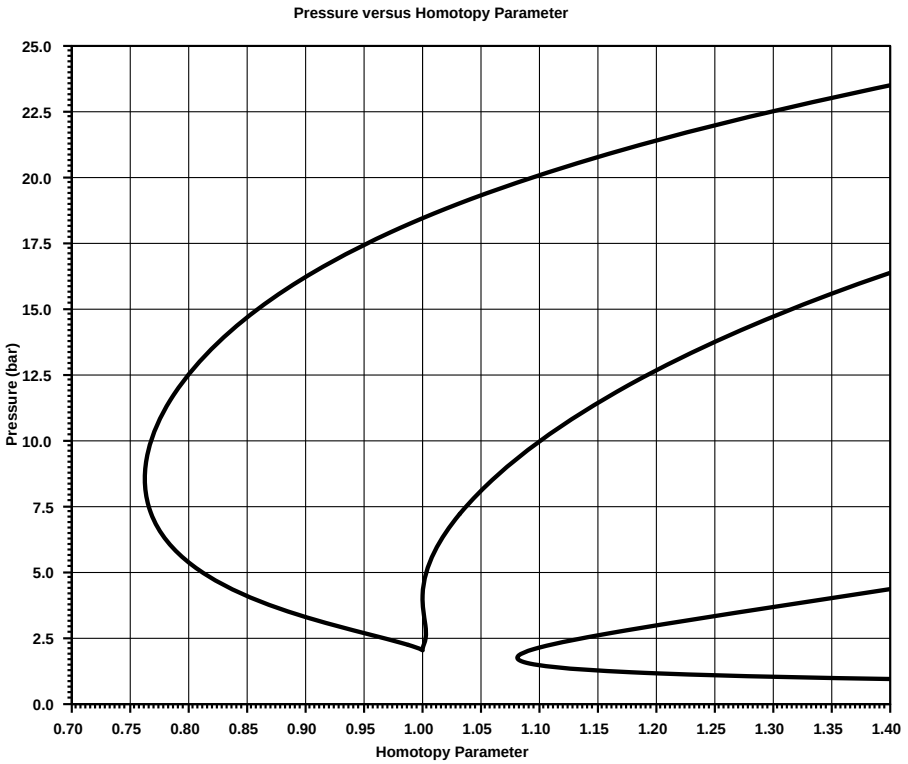

Figure 4-20: Pressure versus the value of λ at the intersection point for the THF-water system. Enlarged region of interest at $\lambda = 1$.

pressure. There are three intersection points on the ternary homogeneous branch. Figure 4-21 shows how these intersection points vary with pressure.

At one bar, the intersection points on the ternary branch at $\lambda = 0.74$ and $\lambda = 0.95$ actually lie on the same intersection point curve. At pressures above approximately 0.65 bar, three intersection points appear on the ternary branch. At a pressure of 0.65 bar, two of the intersection points disappear and as pressure is decreased, the intersection point (originally at $\lambda = 0.67$ at one bar) crosses $\lambda = 1$ at approximately 0.32 bar. At this point, the heteroazeotrope becomes a homogeneous azeotrope.

4.2.8 Methanol-Benzene-Heptane System

The heteroazeotrope finding algorithm was applied to this mixture in section 4.1.6 and it was found that an additional ternary azeotrope was predicted but the benzene-heptane azeotrope was not predicted using the extended Antoine vapor pressure cor-

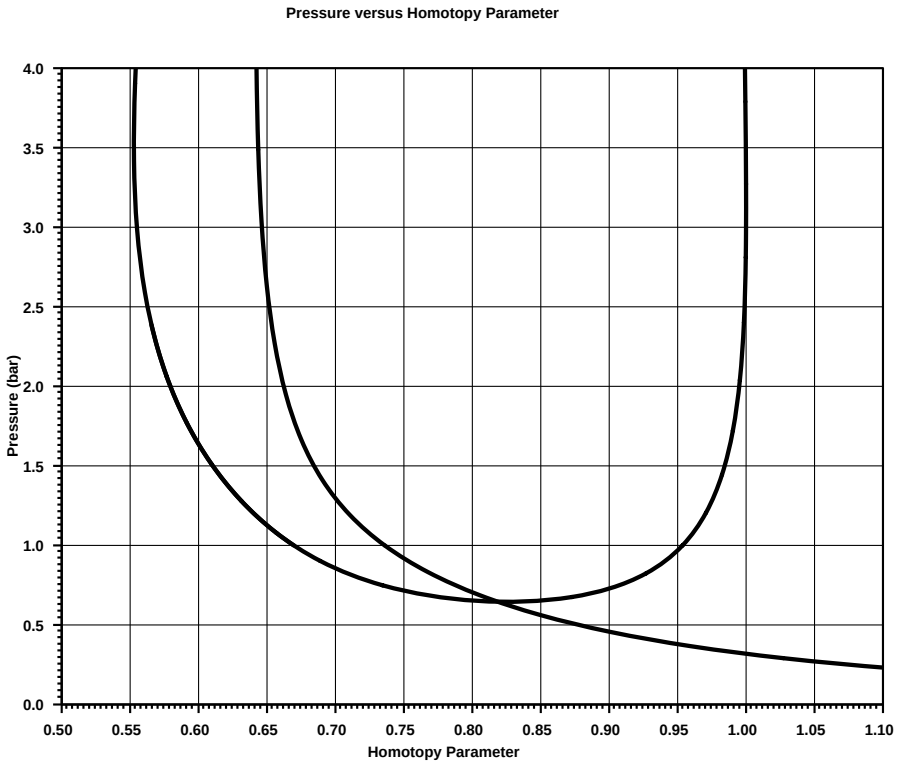

Figure 4-21: Pressure versus the value of λ at the intersection point for the water, acetone, chlorofrom system.

relation and the NRTL activity coefficient model with parameters obtained from [5]. This system is further analyzed here due to this discrepancy.

First, topological consistency will be checked for both the experimental values and the values computed in section 4.1.6. Figure 4-22 contains a diagram showing the relative boiling temperatures of the pure components and azeotropes identified experimentally. Although the benzene-heptane azeotrope has a measured boiling temperature equal to that of pure benzene, it is assumed that this is due to experimental error and that the azeotrope has a slightly lower boiling point (and is thus a minimum boiling azeotrope and not a non-elementary arm-chair fixed-point). According to this figure, the pure components are all stable nodes and have indices equal to +1 ($I_1^+ = 3$ and $I_1^- = 0$), the two homogeneous azeotropes, methanol-benzene and benzene-heptane, are saddle nodes with indices -1 ($I_2^- = 2$), and the binary heteroazeotrope is an unstable node with index +1 ($I_2^+ = 1$). Applying the topological

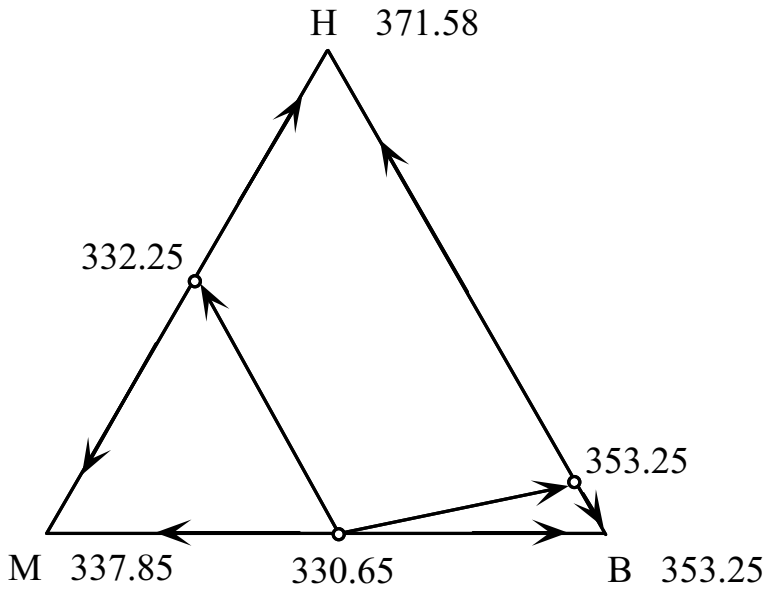

Figure 4-22: Gibbs composition simplex containing measured boiling temperatures (K) of the azeotropes present in the methanol, benzene, and heptane system at 1 atm.

consistency test described in chapter 1, we find

$$\begin{aligned}
\sum_{k=1}^{n} 2^k (I_k^+ - I_k^-) &\stackrel{?}{=} (-1)^{n-1} + 1 \\
2(I_1^+ - I_1^-) + 4(I_2^+ - I_2^-) &\stackrel{?}{=} (-1)^{n-1} + 1 \\
2(3-0) + 4(1-2) &\stackrel{?}{=} (-1)^{n-1} + 1 \\
6 - 4 = 2 &= (-1)^2 + 1 = 2,
\end{aligned}$$

and thus, topological consistency is satisfied for these measured values.

Figure 4-23 contains the Gibbs composition simplex showing the computed boiling temperatures of the pure components and azeotropes predicted in section 4.1.6. In this case, the methanol and heptane vertices are stable nodes with indices +1 ($I_1^+ = 2$), the benzene vertex is a saddle node ($I_1^- = 1$), the methanol-heptane heteroazeotrope and methanol-benzene azeotrope are saddle nodes ($I_2^- = 2$ and $I_2^+ = 0$), and the ternary azeotrope is an unstable node with index +1 ($I_3^+ = 1$ and $I_3^- = 0$). Again,

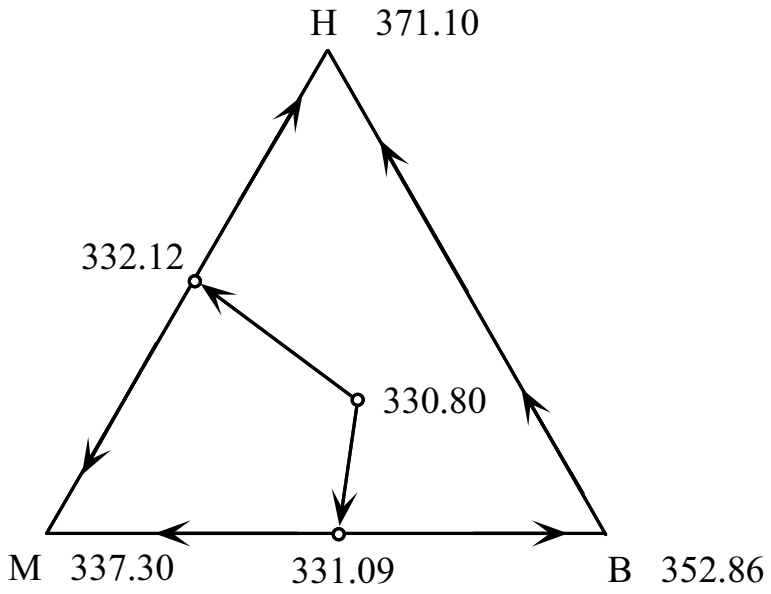

Figure 4-23: Gibbs composition simplex containing computed boiling temperatures (K) of the azeotropes present in the methanol, benzene, and heptane system at 1 atm.

applying the topological consistency test, we find

$$\begin{aligned}
\sum_{k=1}^{n} 2^k (I_k^+ - I_k^-) &\stackrel{?}{=} (-1)^{n-1} + 1 \\
2(I_1^+ - I_1^-) + 4(I_2^+ - I_2^-) + 8(I_3^+ - I_3^-) &\stackrel{?}{=} (-1)^{n-1} + 1 \\
2(2-1) + 4(0-2) + 8(1-0) &\stackrel{?}{=} (-1)^{n-1} + 1 \\
2 - 8 + 8 = 2 &= (-1)^2 + 1 = 2,
\end{aligned}$$

and thus, topological consistency is also satisfied for these computed values.

Unfortunately, the topological consistency test is not much help in this situation. Figure 4-24 contains the Txy-diagram for the benzene-heptane mixture. Notice that it appears as though a binary azeotrope should emerge from the pure benzene vertex (dT/dx_B is nearly tangent at this point). This suggest that under slight pressure variation, the azeotrope should be found. At a pressure of 1 atm, the benzene-heptane bifurcation point on the pure benzene branch occurs at $\lambda = 1.21$ and pressure must be increased to approximately 7.2 atm in order for this bifurcation point to

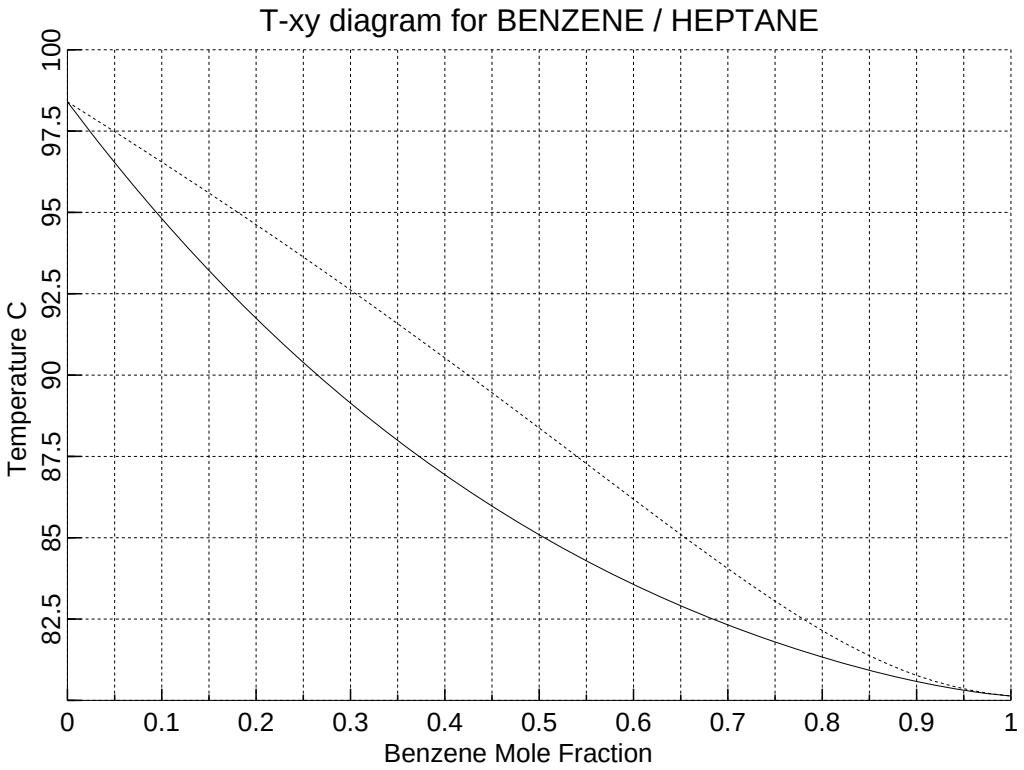

Figure 4-24: Txy-diagram for a mixture of benzene and heptane at a pressure of 1 atm.

move to $\lambda = 1$ (the case where the binary azeotrope emerges from the pure benzene vertex). Given the shape of the *Txy*-diagram and the experimental data indicating the presence of a binary azeotrope, this required pressure change is clearly not reasonable. However, by analyzing the sensitivity of the location of this bifurcation point to the parameters in the Antoine vapor pressure correlation for heptane, it is found that a slight perturbation in the first Antoine correlation parameter, $C_{1,H}$, results in a large change in the location of the bifurcation value for λ. Figure 4-25 contains a diagram showing $C_{1,H}$ versus the bifurcation value of λ (the location of the bifurcation point on the pure benzene branch corresponding to the benzene-heptane branch). Table 4.3 contains the results of the azeotrope finding algorithm after changing the value of $C_{1,H}$ from 87.829 to 87.915.

The indices of the fixed-points in this system are the same as the original computed values except that benzene is now a stable node (thus, $I_1^+ = 3$ and $I_1^- = 0$) and the

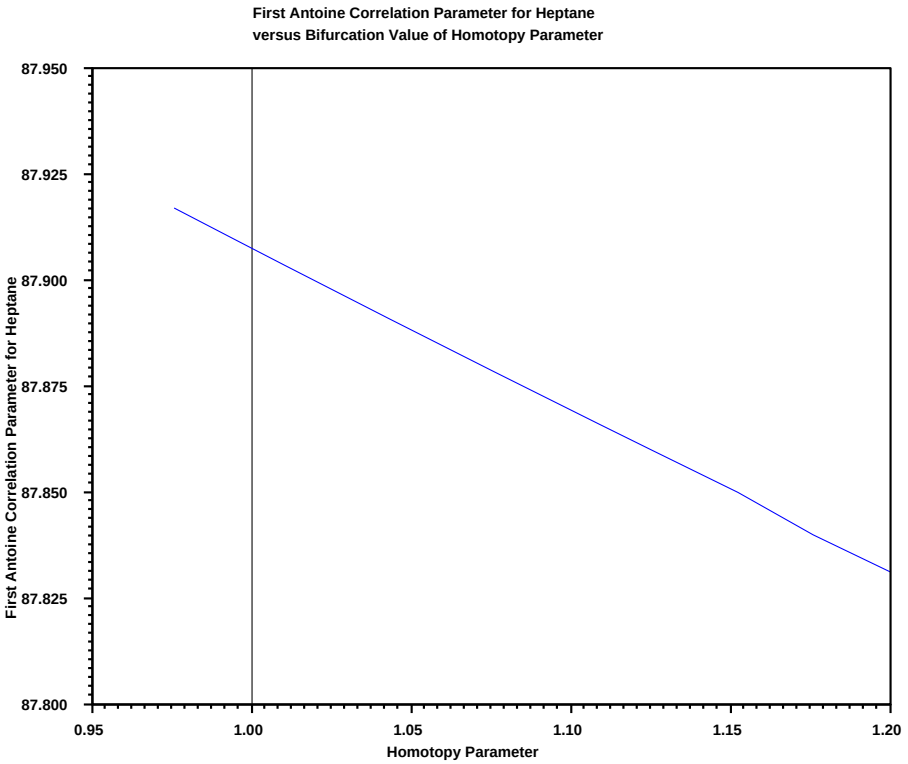

Figure 4-25: Bifurcation value of λ (the location of the bifurcation point on the pure benzene branch corresponding to the benzene-heptane branch) versus the first parameter in the Antoine correlation for heptane at a pressure of 1 atm.

new benzene-heptane azeotrope is a saddle node ($I_2^- = 3$). Thus,

$$\begin{aligned}
\sum_{k=1}^{n} 2^k (I_k^+ - I_k^-) &\stackrel{?}{=} (-1)^{n-1} + 1 \\
2(I_1^+ - I_1^-) + 4(I_2^+ - I_2^-) + 8(I_3^+ - I_3^-) &\stackrel{?}{=} (-1)^{n-1} + 1 \\
2(3-0) + 4(0-3) + 8(1-0) &\stackrel{?}{=} (-1)^{n-1} + 1 \\
6 - 12 + 8 = 2 &= (-1)^2 + 1 = 2,
\end{aligned}$$

and topological consistency is satisfied for the new set of computed values. This small change in the value of $C_{1,H}$ results in the prediction of the benzene-heptane azeotrope without significantly changing the values of the other azeotropes contained in the system. However, the effect of perturbing $C_{1,H}$ changes the computed boiling temperature of heptane from 371.10 K to 368.59 K. A better approach would be to

Table 4.3: Experimental and computed values for azeotropes and heteroazeotropes at a pressure of 1 atm for the methanol, benzene, heptane mixture. First Antoine Correlation parameter for heptane changed to 87.917. Heteroazeotropes are denoted by boldface.

Components	Experimental Composition	Experimental Temp. (K)	Computed Composition	Computed Temp. (K)
Methanol,	(0.6031,0.3969,0)	330.65	(0.6070,0.3930,0)	331.09
Benzene,	(0,0.9945,0.0055)	353.25	(0,0.9948,0.0052)	353.28
Heptane	**(0.7684,0,0.2316)**	**332.25**	**(0.7622,0,0.2378)**	**331.57**
	N/A	N/A	(0.6303,0.2658,0.1039)	330.59

examine the location of the benzene-heptane bifurcation point while adjusting several parameters in such a way that the predicted heptane boiling temperatures remains near its experimental value.

In summary, the ability to predict the bifurcation and intersection points associated with incipient azeotropes and heteroazeotropes and determining the sensitivity of these points to the model parameters is a very powerful tool when estimating physical property parameters. The technique provides a systematic means of exploring the phase equilibrium structure and the capabilities (or limitations) of the phase equilibrium model.

Part II

Computational Differentiation

Chapter 5

Overview of Computational Differentiation

5.1 Introduction

The previous part of this thesis discusses topics important in the analysis of heteroazeotropic systems, in particular, the computation of the azeotropes and heteroazeotropes present in a multicomponent mixture and the efficient calculation of changes in phase equilibrium structure with respect to system and/or property model parameters. This part of the thesis, Computational Differentiation, discusses an important topic in the simulation and optimization of heteroazeotropic systems (or any system for that matter), namely the calculation of numerical derivatives. This section provides some motivation of why numerical derivatives are required in calculations and discusses the advantages of obtaining them accurately and efficiently. This is followed by a discussion of several ways to compute numerical derivatives with the computer, including hand-coding, finite difference approximation, reverse polish notation evaluation, symbolic differentiation, and automatic differentiation. This chapter is concluded with a discussion on equation-oriented process simulation, an applica-

tion where careful attention to the way derivatives are represented and computed can make a substantial impact.

The following chapter discusses the technique of automatic differentiation in more detail. This is followed by a chapter describing a new class of automatic differentiation techniques developed in this thesis. Finally, this section of the thesis is concluded with a chapter containing numerical comparisons between symbolic differentiation and automatic differentiation and several examples illustrating the improvements to automatic differentiation as a result of this thesis.

Approximations to derivatives are required in many numerical procedures. For example, Newton's method for solving a system of nonlinear equations, $f(x) = 0$, requires the Jacobian matrix of the residuals in the recursion formula:

$$x^{k+1} = x^k - \left[\nabla f(x^k)\right]^{-1} f(x^k).$$

Another example is the integration of differential-algebraic equations (DAEs) and stiff ordinary differential equations (ODEs) using a linear multistep method such as those utilizing the backard differentiation formulas (BDF)[16]. Solving a DAE or ODE, $f(\dot{y}, y, t) = 0$, using the BDF method requires the evaluation of the following iteration matrix:

$$\left(\begin{array}{c|c} \dfrac{\partial f}{\partial y} & \alpha \cdot \dfrac{\partial f}{\partial \dot{y}} \end{array} \right).$$

Numerical derivatives are also required in parametric sensitivity calculations, continution methods, and many optimization algorithms. This list can be expanded to fill the remainder of this thesis. The performance of all of these methods is often dramatically improved by using accurate values for the derivatives obtained efficiently.

The topic of this part of the thesis is computational differentiation. By computational differentiation, we mean some process by which derivatives are obtained with a computer. The term numerical derivatives refers to the actual values used to ap-

proximate derivatives, and they are obtained with some computational differentiation method.

When derivatives are obtained with a computer, error may be introduced in two forms: truncation error and roundoff error. If the derivative is obtained by truncating a Taylor series expansion of the original expression, truncation error is introduced due to the fact that the numerical derivative is only an approximation of the exact derivative. On the other hand, roundoff error is the unavoidable consequence of performing the evaluation with the finite precision arithmetic of a computer.

Two approaches for the calculation of derivatives that have been widely used in the past are hand-coding and finite difference approximations. Hand-coding of derivatives is primarily used when the system of equations is relatively small and the derivative expressions are easily derived. In contrast, finite difference approximations are primarily used for large and/or complex systems of equations. The disadvantages of these two approaches, discussed in later sections, has led to the exploration of alternative means for obtaining numerical derivatives. Since the 1950s, researchers have realized that derivatives can be obtained automatically and exactly (to roundoff error) with the computer by applying the simple rules of differentiation (product rule, quotient rule, etc.) to some computer representation of the expression to be differentiated. Three such approaches commonly used are symbolic differentiation, reverse polish notation (RPN) evaluation, and, more recently, automatic differentiation. Symbolic differentiation computes derivatives by applying the rules of differentiation to a tree representation of the equations, generating new expressions (trees) for each partial derivative desired. RPN evaluation simultaneously computes function and derivative values from the *reverse polish notation* form of the expression. In contrast to the two former approaches, which work with equations, automatic differentiation was originally designed for the differentiation of computer programs written in languages such as C and FORTRAN. Derivatives are obtained by applying the chain-rule to the sequence of elementary arithmetic operations contained in a subroutine for the computation of function values. All three approaches above are carried out automatically by the computer and are thus suitable for large problems that would be difficult and

error prone to differentiate by hand. Furthermore, the numerical derivatives obtained in these three approaches, as well as hand-coding, are exact (to roundoff error) and, thus, free of truncation error. The following section discusses these five approaches in more detail.

5.2 Approaches to the Computation of Derivatives

5.2.1 Hand-coded Derivatives

One way to obtain the derivatives of a set of functions is to derive the analytic expressions by hand and then code a subroutine for their computation. This approach has the advantage that the derivatives are exact (to roundoff error) and through careful coding of the expresssions, their evaluation can be quite efficient. However, this approach is tedious, extremely time consuming, and very error prone for complex and/or large-scale equation systems. It is not practical for most problems of interest, yet it is still commonly employed in practice and can be attributed to many of the headaches experienced by members of the simulation community.

5.2.2 Finite-Difference Approximation of Derivatives

A common alternative to hand-coding is numerical approximation of derivatives by finite differences. One example is the simple forward difference approximation:

$$\frac{\partial f_i}{\partial x_j} \approx \frac{f_i(x + h_j e_j) - f_i(x)}{h_j}$$

where e_j is the *j-th* Cartesian basis vector and h_j is the perturbation in variable x_j. The truncation error associated with this approximation is $\mathcal{O}(|h_j|)$ (not including roundoff error). A more accurate approximation is the centered finite difference

approximation:

$$\frac{\partial f_i}{\partial x_j} \approx \frac{f_i(x + h_j e_j) - f_i(x - h_j e_j)}{2h_j}.$$

The advantage of this over the previous approximation is that the trunctation error is $\mathcal{O}(h_j^2)$ (not including roundoff error), however, it costs roughly twice as much to evaluate. Finite difference approximation has the advantage that it is trivial to code and can use existing subroutines for the computation of function values. One problem with this approach lies with the tradeoff between truncation error and roundoff error. For large perturbations, the truncation error is significant. As the perturbation is reduced to minimize truncation error, the effect of roundoff error becomes greater. There is an optimal perturbation for computing the derivative, however, this optimal perturbation is difficult to determine and varies with different functions and variable values and, as a result, it is often selected in an ad hoc, suboptimal manner. Furthermore, even with the optimal perturbation, the number of significant digits in the value of the derivative is much less than that of the original function value. This loss of accuracy is not acceptable in many applications. Another problem with finite differences is that the cost of computing the Jacobian is $n + 1$ times the cost of evaluating the set of functions for the forward finite difference approximation and $2n$ times the cost of evaluating the set of functions for the centered finite difference approximation, where n is the number of variables. This may be prohibitively expensive for large systems of equations. Curtis, Powell, and Reid made the crucial observation that in the case where the Jacobian is sparse and the sparsity pattern is known explicitly, the columns of the Jacobian can be grouped into $n_c \leq n$ structurally orthogonal groups (two columns are structurally orthogonal if they do not both have nonzero entries in the same row) [30]. The cost of the Jacobian evaluation is reduced because variables in the same group may be perturbed simultaneously. As a result, the cost of the Jacobian evaluation is $n_c + 1$ times the cost of evaluating the set of functions for the forward finite difference approximation and $2n_c$ times the cost of evaluating the set of functions for the centered finite difference approximation. This

is referred to as the CPR algorithm and can be viewed as a graph coloring algorithm. Let $f : \mathbb{R}^n \longrightarrow \mathbb{R}^m$ denote a system of nonlinear equations and suppose we want to estimate its Jacobian matrix $\nabla f(x) \in \mathbb{R}^{m \times n}$ using finite differences in the minimum number of evaluations of $f(x)$ (it is assumed that it is more efficient to evaluate $f(x)$ as a whole rather than component by component). This problem can be formulated in general as follows. Determine vectors d_1, d_2, ..., $d_p \in \mathbb{R}^n$ such that Ad_1, Ad_2, ..., Ad_p determines $A \in \mathbb{R}^{m \times n}$ directly with minimum p. Note that for the CPR algorithm, $p = n_c$ and

$$(d_i)_j = \begin{cases} 1 & \text{if variable } x_j \text{ is in group } i \\ 0 & \text{otherwise.} \end{cases} \tag{5.1}$$

The general problem above amounts to partitioning the columns of A into p column groups such that each column belongs to exactly one group. This problem can be related to the graph coloring problem [29] and the chromatic number of the graph is equal to the minimum value for p. Determination of the chromatic number of a graph (and hence, minimum number of evaluations required to estimate $\nabla f(x)$ by finite differences) is NP-complete, however, approximate algorithms such as LFO (largest-first ordering), SLO (smallest-last ordering), and IDO (incidence-degree) ordering can be used to obtain column grouping yielding improved performance over the CPR algorithm [27]. Having computed the vectors d_i, $i = 1, \ldots, p$, the Jacobian can be estimated. Compute $\nabla f(x)d_i$ as follows,

$$\nabla f(x)d_i = f(x + d_i) - f(x) + \mathcal{O}(\|d_i\|), \tag{5.2}$$

for $i = 1, \ldots, p$. By construction, $\nabla f(x)d_1$, $\nabla f(x)d_2$, ..., $\nabla f(x)d_p$ uniquely determine $\nabla f(x)$. Obviously, for large, sparse problems this saving can be dramatic and should be employed whenever possible. Unfortunately, accuracy is not affected by this technique.

5.2.3 RPN Evaluation of Derivatives

The standard *infix* form of an equation corresponds to the usual way of writing a mathematical expression. Using this representation of an equation within the computer has the advantages that it is a very compact form of storage and it is easy for the user to read the expression. For example, the expression

$$(x^2 + y)(y^2 + 4)$$

may be represented within the computer using infix notation by the following sequence of fifteen identifiers: $($, x, $\hat{}$, 2, $+$, y, $)$, $\cdot$, $($, y, $\hat{}$, 2, $+$, 4, $)$. The disadvantage of the infix notation is that it is difficult to identify the operands of the various operators automatically with the computer and evaluation and differentiation of this form of the expression is extremely cumbersome, requiring several passes through the expression. An alternative to the infix representation of an expression is *reverse polish notation* (RPN), familar to those who have used a Hewlett-Packard calculator. Using RPN, the expression above may be stored in computer memory as the following sequence of eleven identifiers: x, 2, $\hat{}$, y, $+$, y, 2, $\hat{}$, 4, $+$, $\cdot$. This representation is compact, yet preserves most of the information about the expression that is present in the tree notation described in the following section. An expression stored in RPN form is evaluated with the use of a *stack*. A stack is a *first in, last out* data structure where elements are *pushed* onto the top of the stack and removed by *popping* them from the top of the stack. Starting from the left of an RPN expression and moving right, constants and variables are pushed onto the stack. When a binary operator is encountered, the two top elements are popped from the stack, and the binary operation is performed on these values. This result is then pushed onto the stack, resulting in a new stack with one less element. If a unary operator or function is encountered, the top element of the stack is popped, the operation is performed, and the result is pushed onto the stack. When the entire expression has been processed, the stack contains a single element, the value of the expression. In order to differentiate an

RPN expression, a second derivative stack is used in parallel with the main stack described above. If a constant is pushed onto the main stack, its derivative, 0, is pushed onto the derivative stack. If a variable is pushed onto the main stack, its derivative, 0 or 1 depending upon which variable we are differentiating with respect to, is pushed onto the derivative stack. When operators are encountered in the RPN expression, the derivatives of these operators, defined by the simple rules of differentiation, are applied. These derivative operators, in general, operate on elements of both the main stack and second stack, and thus, derivatives and functions are evaluated simultaneously. If the n partial derivatives of a vector-valued function $f : \mathbb{R}^n \longrightarrow \mathbb{R}$ are required, then $n + 1$ stacks must be maintained. Derivatives obtained with this method are exact (free from truncation error) and the time complexity of the RPN evaluation is similar to that of interpreted symbolic derivative evaluation (described below). For a detailed discussion of RPN evaluation see [67] and [123].

5.2.4 Symbolic Differentiation

An alternative to hand-coding of derivatives, finite difference approximations, and RPN evaluation is symbolic differentiation. Symbolic differentiation is used in software packages such as REDUCE [58], MACSYMA [85], and Maple [23]. These computer algebra packages have experienced widespread use due to their ability to perform such operations as polynomial factorization, simplification, symbolic integration, and other formula manipulations, in addition to differentiation. They are designed to be general purpose tools capable of handling large-scale problems requiring the automation of algebraic computation. In the symbolic environment, the mathematical expressions can be represented as rooted directed trees. A directed tree is a connected acyclic directed graph (digraph) where removal of one of the edges results in a disconnected digraph. A rooted tree is a tree where one of the vertices is distinguished as the root. The interior vertices of the trees representing expressions are intrinsic functions (sin, exp, etc.) or binary and unary operators (+, -, /, $\cdot$, etc.) with one or two edges pointing to their operands. The leaves of the tree (vertices without any

children) are either literals (e.g., 2.4, 3.5, π) or variables. The precedence relationship of the expression is preserved in the structure of the tree. For example, the equation

$$y = \frac{\sin(x_1)ln(x_1x_2x_3) + \cos(x_2)x_3^2}{\exp(x_1x_2) + 1} \tag{5.3}$$

is shown in tree form in Figure 5-1.

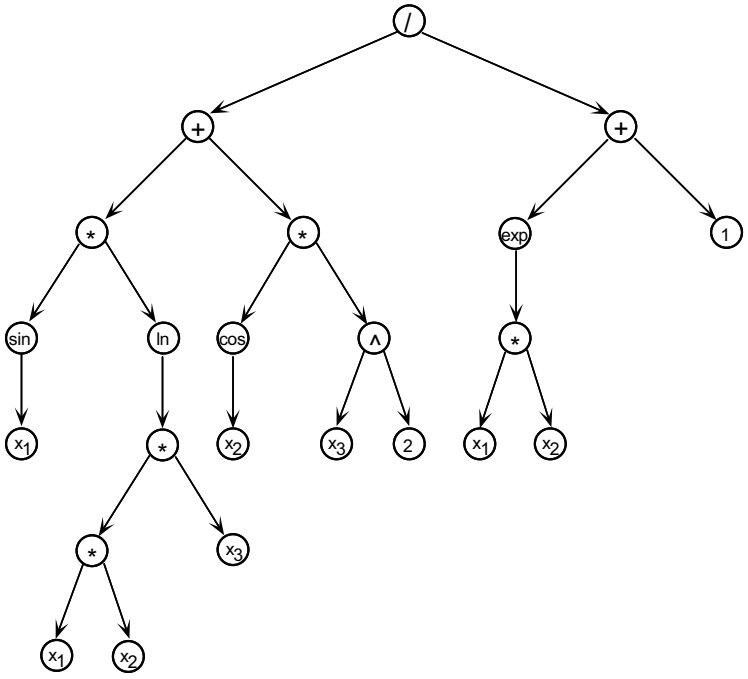

Figure 5-1: Tree form of equation (5.3)

The tree is a clear but wasteful representation of an expression. Notice that there are three distinct vertices representing x_1 and x_2 and two vertices representing x_3. In practice, the actual graph of equation (5.3) would have one vertex allocated for each variable present in the expression and all operands containing these variables would point to the same vertex. Furthermore, notice the product x_1x_2 appears twice in (5.3). This too can be represented by a single subgraph, further reducing the memory required to store the symbolic form of the equation. By exploiting these *common subexpressions*, the expression can be represented as a directed acyclic graph (DAG). The task of identifying and exploiting the common subexpressions is a common task when hand-coding expressions in FORTRAN or C (performed when intermediate vari-

ables are introduced to avoid writing the same subexpression several times). However, the task of identifying the common subexpressions present in an abitrary collection of graphs is a combinatorial problem and accounts for the majority of the cost associated with generating FORTRAN code from expressions generated in Maple. Common subexpressions not only reduce amount of memory required to represent an expression in computer memory, but can also be exploited in the evaluation of an expression, as will be shown below.

The data structure representing DAG vertices in computer memory must have the following fields: a field specifying the vertex type (literal, variable, addition operator, multiplication operator, etc.), pointers to the left and right children of the vertex, and, if the vertex is a literal or a variable, a real-valued field containing the current value. A formula represented in computer memory as a graph using the data structures described above may be evaluated using the recursive algorithm shown in Figure 5-2.

```
REAL EVALUATE( v )
 1  if v = literal or variable then
 2     ▷ Simply return the value of the vertex
 3     return value[v] ;
 4  elseif v = addition operator then
 5     ▷ Return the sum of the operands, the left and right children
 6     return EVALUATE(left[v]) + EVALUATE(right[v]) ;
 7  elseif v = multiplication operator then
 8     ⋮
 9  ▷ Similar code for other binary operators and unary functions
10     ⋮
11  end
```

Figure 5-2: Recursive algorithm for evaluating the graph form of a symbolic expression.

Procedure EVALUATE takes the vertex data structure as an argument, v, and returns the numeric value of the subexpression rooted at the argument. The expression value[v] accesses the value field of v. Expression left[v] and right[v] access the left and right children of v. When EVALUATE is called with the root vertex as an

argument, the return value is the value of the expression represented by the DAG. Procedure EVALUATE makes no attempt to exploit the occurrence of common subexpressions present in an equation graph. This can be performed very easily by adding two additional fields to the vertex data structure, a boolean field indicating whether or not the vertex is a common subexpression and an binary variable flag to indicate whether or not the subgraph rooted at the common subexpression has already been computed for the current set of variable values. A simple preprocessing step based on a depth-first search can be used to find and mark the common subexpression vertices present in the graph (this task is considered in more detail in chapter 7 and is thus, omitted here). An additional global binary integer variable is also employed for this common subexpression exploitation. At the beginning of a calculation (e.g., DAE integration, solution of a system of nonlinear equations, etc.), the global binary variable is set to unity and the flags of all common subexpression vertices present in the graph are set to zero. Each time any of the values of the variables are modified, the global binary variable is changed from zero to unity or unity to zero, depending on its current value. As the residual vector and/or Jacobian matrix are evaluated and a common subexpression is encountered during the interpretive evaluation of a graph [1], the vertex flag is compared to the global binary variable. If the values are equal then the common subexpression has been previously encountered and the precomputed value of the subexpression is returned, otherwise, the subexpression is evaluated, this resulting value is stored in the common subexpression vertex, and the vertex flag is set equal to the global variable, ensuring that each time this common subexpression is encountered later (and the independent variable values have not changed), the value will not be recomputed. As shown in chapter 8, this modification can yield substantial improvements when both the residual vector and Jacobian matrix are evaluated at the same values for the independent variables.

The symbolic expression can be differentiated by applying the simple rules of dif-

[1] Evaluating the symbolic form of an expression held in computer memory is referred to as *interpretive* evaluation. The alternative is to write a subroutine containing the expression which can be evaluated by calling the *compiled* routine.

ferentiation recursively to the DAG. For example, consider the product rule, $d(u \cdot v) = v \cdot du + u \cdot dv$. The differentiation operator is applied recursively to the subexpressions u and v, which may be complicated subgraphs of the graph representing $u \cdot v$. Note that the subexpressions u and v appear both in the original expression $u \cdot v$ and the derivative of this expression. This observation can be exploited in a symbolic environment by not creating new subgraphs for u and v in the derivative expression, but simply sharing the same subgraphs in the original formula. Thus, the original expression and its partial derivatives form an interconnected collection of DAGs.

Once the derivative expresssions have been generated, there are several ways in which numerical values may be obtained. One way is to apply the recursive algorithm described above for evaluating a graph to each derivative expression. This approach will be referred to as the *interpretive* approach for symbolic derivative evaluation. Alternatively, the graph form of the expressions can be used to generate a C or FORTRAN subroutine for derivative evaluation (the graph is converted to infix notation and written to a file). This resulting subroutine may be compiled and linked to code requiring the numerical derivatives. The residual and derivative expressions may be optimized both at the code generation level (common subexpression identification) and during compilation. Once the residual and Jacobian subroutines have been compiled, the evaluation of the residual vector and Jacobian matrix using this approach is much faster than if the evaluation were performed using the interpretive approach. As a result, the compiled approach is prefered when residual vectors and Jacobian matrices are required repeatedly (as is the case with an iterative equation solver or integrator). Most symbolic computation packages provide utilities for generating subroutines from symbolically derived derivative expressions. There are, however, situations where interpretive evaluation is a better choice (see section 5.3 on equation-oriented process simulation).

The advantage of symbolic differentiation is that the derivatives are exact (free from truncation error). If, however, the original expression contains quotients or functions, the resulting expressions for the partial derivatives are often much more complex than the original function. As a result, in a naive implementation of sym-

bolic differentiation (no common subexpression evaluation), the cost of computing a gradient is roughly n times the cost of evaluating the function, where n is the number of variables in the function. Performance is improved by optimizing the DAGs and performing common subexpression evaluation, however, this still often falls short of the performance of automatic differentiation, discussed in the following section.

5.2.5 Automatic Differentiation

Three prerequisites for a differentiation method are given by Iri [63]: it should be fast (low complexity with respect to the computation of what is being differentiated), it should be free from truncation error, and it should be able to be performed automatically. The methods discussed previously are defficient in one or more of these prerequisites: hand-coding of derivatives cannot be performed automatically, finite difference approximations are not exact nor, like symbolic or RPN derivatives, computed very efficiently. Automatic differentiation (AD), a relatively recent technique developed for the automatic computation of derivatives, does in fact have all of these desired properties. A detailed discussion of AD will be deferred to the following chapter, however, some of its advantages will be discussed here. Consider a general nonlinear function, $f : \mathbb{R}^n \longrightarrow \mathbb{R}$. The cost of evaluating the gradient of f using finite differences is $n+1$ times the cost of evaluating f alone (using the simple forward finite difference approximation). The cost is roughly n times the cost of f alone when using RPN evaluation or symbolic differentiation. Using the *reverse mode* of automatic differentiation (described later), the cost of evaluating the gradient is bounded above by three times the cost of evaluating f alone, independent of the number of variables or the functional form of f. Furthermore, it is possible to simultaneously compute f, its gradient, and a relatively tight estimate of the roundoff error generated during the function evaluation at a cost bounded above by five times the cost of evaluating f alone. AD was originally applied to the automated differentiation of computer programs (e.g., C functions or FORTRAN subroutines) containing loops, conditional statements, and other sophisticated programming structures, making AD a very flex-

ible and powerful technique. AD is discussed in detail in the following chapter. This is followed by a chapter describing a new class of automatic differentiation techniques developed in this thesis.

5.3 Equation-Oriented Process Simulators

Equation-oriented process simulation is one application where the careful selection of a computational differentiation method can make a substantial impact and thus, warrants some discussion. Furthermore, many of the differentiation techniques discussed and developed in this thesis have been implemented in the equation-oriented process simulator ABACUSS[2].

5.3.1 Process Flowsheet Simulation

Process flowsheet simulation packages have been around for approximately forty years. These tools can be roughly broken down into two categories, modular simulators and equation-oriented simulators. Prior to the development of these packages, simulation was performed through the use of problem specific routines written by the modeler in a procedural language such as FORTRAN. These subroutines were compiled and linked to numerical routines to solve the particular problem at hand. Although this process offers a great deal of flexibility, it is time consuming, error prone, and requires a great deal of expertise on the part of the modeler. In light of this difficulty, more advanced packages were developed to assist the modeler in process flowsheet simulation. Among the first to be developed were modular simulators. In this framework, the process model flowsheet is assembled by connecting specific modules from a unit operation library. The advantage of selecting unit operations from a library is that the user is freed of the time consuming task of writing code for each distinct

[2]ABACUSS (Advanced Batch And Continuous Unsteady-State Simulator) process modeling software, a derivative work of gPROMS software, ©1992 by Imperial College of Science, Technology, and Medicine.

unit operation appearing in the flowsheet model. Furthermore, these libraries form information repositories, storing not only the model formulation but also custom tailored solution algorithms for converging the unit operation in a numerical calculation. Often, sophisticated graphical user interfaces (GUIs) are provided to assist the user in the construction of the flowsheet. There are two main approaches for performing a steady-state flowsheet simulation, the sequential modular and the simultaneous modular approaches. Historically, the flowsheets were solved using the sequential modular approach. Here, the inputs to a module are specified and the unit operation subroutine computes the outputs (using a robust custom tailored algorithm) which are then inputs to some other unit operation routine downstream in the flowsheet. The disadvantage of this approach is that the solution strategy is strongly tied to the topology of the flowsheet. Furthermore, recycles in the flowsheet present a problem because the inputs to some unit operations are not known a priori. In such cases, the flowsheet must be solved in an iterative fashion by "tearing" recycle streams. That is, initial guesses are provided for variables associated with the torn recycle streams and the flowsheet is solved in a sequential fashion to compute new estimates for these unknown quantities. The process is continued until the difference between successive values for the unknown variables is sufficiently small. A great deal of research has gone into developing solution strategies for improving the efficiency and robustness of recycle streams convergence. The sequential modular approach also performs poorly when some of the inputs are not known a priori but are to be calculated by specifying some of the outputs (the so-called design problem). These "design specs" are solved in an iterative manner, much like the convergence of the recycle streams. The simultaneous modular approach was developed to remedy the shortcoming of the sequential modular approach when applied to flowsheets with complicated recyle streams and design specifications. In the simultaneous modular approach, the torn recycle streams and design specifications are solved simultaneously (although there are still nested iterations within the physical property and unit operation routines). The modular approach to flowsheet simulation has proven invaluable for steady-state simulation and optimization problems. This is due in part to the enormous amount

of research that has gone into developing flowsheet convergence strategies and to the sophisticated, custom tailored solution algorithms within each of the unit operation routines. The approach, however, has been less successful in solving dynamic simulation problems. One reason for the limited success of the modular approach is that the behavior of a dynamic system depends greatly on the physical characteristics (size, geometry, etc.) of the unit operations. Accurate predictions of the dynamic responses requires very detailed dynamic models of the equipment. It is impossible for a modular simulation package to provide a complete library containing every possible unit operation, in sufficient detail, required for a general process flowsheet. This deficiency has been one motivation for the development of equation-oriented process simulators.

In an equation-oriented environment, the user writes the flowsheet model using a high-level equation-based symbolic language. Unlike "low-level" languages such as C or FORTRAN, the languages provided by an equation-oriented simulator allow the user to write models in a very clear and concise manner. Equations are written in an input file much as they would on a piece of paper, using a compact mathematical notation. This flexible, high-level representation allows the model to be completely decoupled from the solution procedure. And thus, a variety of numerical techniques can be applied to the same flowsheet model. Furthermore, the equation-oriented representation can be used as a repository for process knowledge. In the initial stages of process developement, the process model is used as a tool for design. As knowledge of the process grows, the model ideally becomes more sophisticated in order to reflect the increased understanding of the process. The process model can be used for the initial design, start-up, simulations and optimizations during the lifetime of the physical process, and for final decommissioning. The potential impact of an accurate process model over the lifetime of the process greatly justifies the initial cost associated with properly modeling the plant.

Many equation-oriented process simulators allow procedures to be incorporated into the flowsheet model (for example, gPROMS and SPEEDUP allow physical properties to be computed from pre-existing subroutine libraries). Although this allows the very rich collection of legacy models to be exploited, it has the disadvantage that

many details are hidden within black box subroutines. For example, 'hidden discontinuities' within a subroutine can wreak havoc with the error control mechanisms of DAE integrators [21, 57] and will cause completely wrong parametric sensitivities to be computed [47]. Discontinuities should be handled explicitly to ensure efficient, robust, and correct solution of the problem.

Although the equation-oriented simulator is superior to the modular simulator for performing dynamic simulations, the opposite is true with steady-state simulations. Equation-oriented simulators allow a very general representation of the model to be written and, as a result, the solution procedures must also be very general (as opposed to the custom tailored algorithms found in the unit operation modules of a modular simulator). Although, several strategies have been developed to improve the robustness of steady-state simulations and the consistent initialization step of dynamic simulations (including block decomposition and various tearing strategies), these calculations remain very difficult to solve robustly.

Equation-oriented process simulators can be broken down into roughly two categories: complied architecture and interpretive architecture. In the compilied architecture, the equations contained in the input file are converted to a FORTRAN subroutine that is compiled and linked to other numerical routines. In most packages, symbolic derivatives are constructed from the symbolic form of the equations and a FORTRAN subroutine for Jacobian evaluation is also constructed. In an interpretive architecture, the symbolic form of the process model and Jacobian is maintained in computer memory. These data structures are "interpreted" to provide numerical values for the residual vectors and Jacobian matrix, as explained above. Although evaluation of compiled expressions can be as much as an order of magnitude faster than interpreted expressions, the interpretive architecture has proven invaluable in applications such as hybrid discrete/continuous dynamic simulation. Most 'continuous' processes exhibit a significant degree of discrete behaviour. These discrete changes are due to both operating conditions imposed on the process (e.g., digital regulatory control, start-up or shut-down operations, equipment failure, and control system set-point changes) and to the particular representation of physico-chemical changes

in the process model (e.g., flow reversals, phase change, and laminar/turbulent flow transitions). Discrete changes occur at time events and state events. A time event is a particular time where an event is known to happen (e.g., fill tank for thirty seconds). A state event, also referred to as an implicit discontinuity, is an event whose time of occurrence is a function of the state of the system (e.g., fill tank until level reaches one meter). At each event, the functional form of the process model changes. When performing a hybrid discrete/continuous simulation, events are detected (usually by monitoring a discontinuity function which crosses zero at a particular event), the functional form is changed, the system is reinitialized, and the integration is continued with the new functional form of the model until the next event is encountered. In most cases, the sequence of functional forms the model assumes during the course of a simulation is not known a priori (due to the occurrence of implicit discontinuities). Furthermore, the number of different functional forms of the model is exponential in the number of implicit discontinuities present. These properties of the hybrid discrete/continuous simulation problem make the compiled architecture less practical for most problems. It is possible to create general subroutines that imbed all functional forms of the system of equations associated with **IF** and **CASE** equation, however, for a general schedule (a set of discrete actions imposed on a continuous model), the resulting subroutine will be very large and complex. Furthermore, the implementation of such a routine that embeds all possible functional forms of the model would be difficult and error prone. Although, the interpretive architecture is attractive due to the capability to rapidly switch between functional forms at a discontinuity, it also introduces some complications that were not as important in the compiled architecture. As stated above, interpretive evaluation of equation and derivative graphs is much more costly than compiled evaluation. During normal calcuations with compiled code, the majority of the cost is often associated with the LU factorization of some iteration matrix (this is why modern BDF integration codes try to use a previously factorized Jacobian matrix for as many steps as possible). In an interpretive architecture (and, indeed, even the compiled architecture), the residual and Jacobian evaluations can account for a substantial fraction of the overall calculation cost (this

is particularly true for parametric sensitivity calculations as shown in chapter 8). Furthermore, substantially more memory is required to represent an equation system and Jacobian matrix in computer memory than as a compiled subroutine, limiting the size of systems that can be handled. The remainder of this part of the thesis discusses improvements that can be made in both the way the equation system and Jacobian are represented and evaluated in an interpretive architecture. Modifications tailored to this environment have yieled substantial performance in many calculations performed within this type of process simulators.

5.4 Conclusion

Several computational differentiation approaches have been discussed in this chapter. It is concluded that AD offers several advantages over other, more traditional, methods for computing numerical derivatives. Also discussed is the topic of equation-oriented process simulation. This topic is illustrated because it is an application where the careful selection of a differentiation method can have a substantial impact both in the size of the problems that may be handled and the performance of the calculations performed. The next chapter of this thesis discusses the topic of AD in more detail. This is followed by a chapter describing the improvements to AD developed in this thesis. This part of the thesis is then concluded with a chapter containing numerical examples illustrating the improvement.

Chapter 6

Automatic Differentiation

As discussed in the previous section, automatic differentiation (AD) represents a class of very powerful analytical differentiation techniques. In this chapter, AD is discussed in detail. First, the idea of representing a computation as a sequence of elementary operations is described and the concept of a computational graph is introduced. Second, a description of the forward and reverse modes of automatic differentiation (both scalar and several vector versions) is presented. This is followed by a description of how the reverse mode of AD may be used to obtain relatively tight estimates of the roundoff error associated with the computation of a function. This chapter is concluded with a section containing a brief history, literature review, and a list of several implementations of AD available. The following chapter discusses the contributions made to this field by this work.

6.1 Computational Graph and Accumulation

This section introduces the idea of representing a system of equations as a sequence of elementary operations and introduces the concept of a *computational graph* (CG). In addition, a small example is presented motivating the use of AD over other computational differentiation techniques.

Most systems of equations of interest can be expressed as a sequence of N elementary operations and assignments to elementary variables:

$$\begin{aligned} \textbf{for } j &= 1 \textbf{ to } N \\ v_j &= v_j(v_i)_{i \in \mathcal{A}_j} \\ \textbf{end} & \end{aligned}$$

where

$$\mathcal{A}_j \subset \{1, 2, \ldots, j-1\} \tag{6.1}$$

is the set of indices of the arguments of elementary operation j. Typically, the elementary operation is a binary or unary operator ($+$, $-$, $/$, $\cdot$) or a unary function (sin, cos, ln, etc.), in which case $\mathcal{A}_j$ consists of one or two elements. The analysis presented in this thesis, however, applies to elementary operations which are functions of any arbitrary number of previously computed elementary variables. The elementary operations may be executed in any order provided that all arguments of any particular operation have been previously computed. The order of operations corresponds to different *indexing* of the elementary variables. Other sources in the literature impose the following ordering on the elementary variables: $v_j \equiv x_j$ for $j = 1, \ldots, n$ are the independent variables, $v_j \equiv z_j$, for $j = n+1, \ldots, n+p$ are the intermediate variables, and $v_j \equiv y_j$ for $j = n+p+1, \ldots, n+p+m$ are the dependent variables of the system of equations $f : \mathbb{R}^n \longrightarrow \mathbb{R}^m$ represented as a sequence of elementary operations. The intermediate variables simply allow the evaluation of $y = f(x)$ to be broken down into a sequence of elementary operations. For most systems of equations of interest, p is much greater that $n + m$. This discussion does not impose a restriction on the elementary variable indexing since different orderings can be extracted from a graph representation of a system of equations more efficiently; all that is required is that $i < j$, $\forall i \in \mathcal{A}_j$. Thus, we denote the set of indices of

independent variables as $\mathcal{I} \subset \{1, 2, \ldots, n+p+m\}$, the set of indices of dependent variables as $\mathcal{D} \subset \{1, 2, \ldots, n+p+m\}$, and the set indices of intermediate variables as $\mathcal{E} \subset \{1, 2, \ldots, n+p+m\}$. These index sets have the following properties: $\mathcal{I} \cap \mathcal{D} = \mathcal{I} \cap \mathcal{E} = \mathcal{D} \cap \mathcal{E} = \emptyset$, $\mathcal{I} \cup \mathcal{D} \cup \mathcal{E} = \{1, 2, \ldots, n+p+m\}$, $|\mathcal{I}| = n$, $|\mathcal{D}| = m$, $|\mathcal{E}| = p$, where $|\cdot|$ denotes the cardinality of the set, and $N = n + m + p$. These properties may require the insertion of elementary assignments into the elementary operation list. Representing a computation as a sequence of elementary operations is a routine task for anyone who programs in languages such as C or FORTRAN, where complex expressions are regularly broken down into several computational steps by the introduction of intermediate variables. This is done to not only make the coding of the complex expression easier to understand (and, therefore, debug) but also to avoid repeated computation of the same quantity.

Equation (5.3),

$$y = \frac{\sin(x_1)\ln(x_1x_2x_3) + \cos(x_2)x_3^2}{\exp(x_1x_2) + 1},$$

(shown as a tree in Figure 5-1) can be computed by the following sequence of elementary operations:

$$\begin{aligned}
v_1 &= x_1 \\
v_2 &= x_2 \\
v_3 &= x_3 \\
v_4 &= v_1v_2 \\
v_5 &= v_4v_3 \\
v_6 &= 2 \\
v_7 &= \sin(v_1) \\
v_8 &= \ln(v_5) \\
v_9 &= \cos(v_2)
\end{aligned}$$

$$
\begin{aligned}
v_{10} &= v_3^{v_6} \\
v_{11} &= v_7 v_8 \\
v_{12} &= v_9 v_{10} \\
v_{13} &= \exp(v_4) \\
v_{14} &= 1 \\
v_{15} &= v_{13} + v_{14} \\
v_{16} &= v_{11} + v_{12} \\
v_{17} &= v_{16} / v_{15} \\
y &= v_{17}
\end{aligned}
$$

The relationship between a sequence of elementary operations and the DAG representation of a systems of equations is very close: each interior vertex in the DAG corresponds to an elementary operation. The remainder of this section discusses how the partial derivatives of an expression represented as a sequence of elementary operations may be obtained.

Let $\mathcal{V} = \mathcal{I} \cup \mathcal{D} \cup \mathcal{E}$ denote the complete index set. For each v_j, $j \in (\mathcal{V} - \mathcal{I})$, we define the *elementary partial derivative* as:

$$
d_{j,i} \equiv \frac{\partial v_j}{\partial v_i} \qquad i \in \mathcal{A}_j. \tag{6.2}
$$

For the majority of elementary operations encountered, these partial derivatives are well-defined and continuous in some neighborhood of the elementary function's arguments. Some obvious exceptions are logarithms, square root operations, and inverse trigonometric functions with arguments outside their respective domains. Key assumptions are made on the cost of evaluating an elementary operation and its associated elementary partial derivatives in the following section and used for the complexity analysis throughout the remainder of this part of the thesis.

The partial derivative of y with respect to x_1 can be computed from the represen-

tation of $f(x)$ shown above and the associated elementary partial derivatives using the chain rule:

$$\begin{aligned}\frac{\partial y}{\partial x_1} &= \frac{\partial v_{17}}{\partial v_{16}}\frac{\partial v_{16}}{\partial v_{11}}\frac{\partial v_{11}}{\partial v_7}\frac{\partial v_7}{\partial v_1} + \frac{\partial v_{17}}{\partial v_{16}}\frac{\partial v_{16}}{\partial v_{11}}\frac{\partial v_{11}}{\partial v_8}\frac{\partial v_8}{\partial v_5}\frac{\partial v_5}{\partial v_4}\frac{\partial v_4}{\partial v_1} + \frac{\partial v_{17}}{\partial v_{15}}\frac{\partial v_{15}}{\partial v_{13}}\frac{\partial v_{13}}{\partial v_4}\frac{\partial v_4}{\partial v_1} \\ &= d_{17,16}d_{16,11}d_{11,7}d_{7,1} + d_{17,16}d_{16,11}d_{11,8}d_{8,5}d_{5,4}d_{4,1} + d_{17,15}d_{15,13}d_{13,4}d_{4,1}.\end{aligned}$$

The elementary partial derivatives are computed using the simple rules of differentiation such as the product rule and the quotient rule as well as the rules for differentiating the standard functions such as sin, cos, and ln. For example, consider $v_{17} = v_{16}/v_{15}$ shown in the elementary operation form of equation (5.3). The elementary partial derivatives are: $\partial v_{17}/\partial v_{16} = 1/v_{15}$ and $\partial v_{17}/\partial v_{15} = -v_{17}/v_{15}$. The DAG form of an equation where the edges have been labeled with the elementary partial derivatives will be referred to as a *computational graph* (CG). Figure 6-1 contains the CG of equation (5.3).

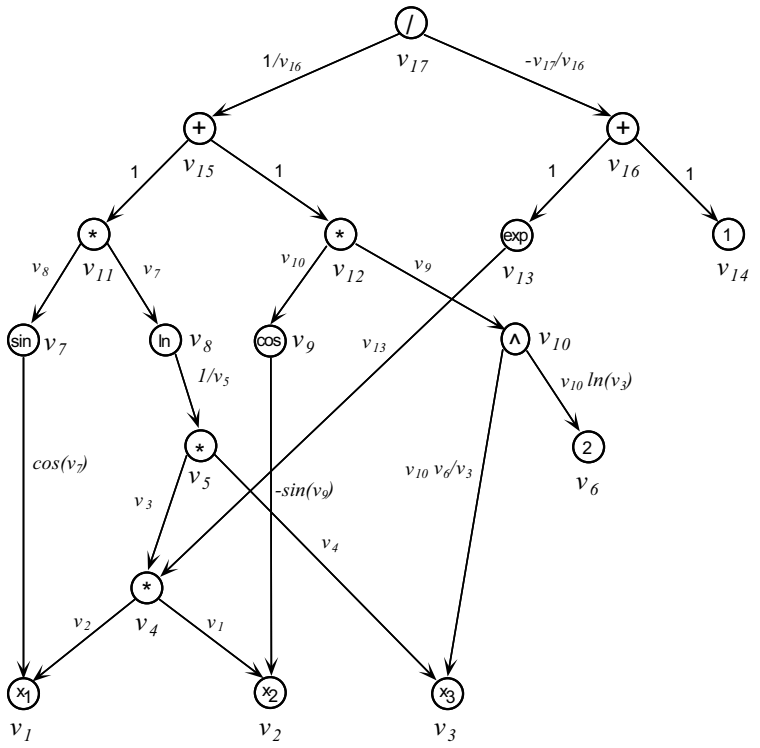

Figure 6-1: Computational graph form of equation (5.3)

In general, the partial derivative of $y = f(x)$, where f is represented as a compu-

tational graph, is given by

$$\frac{\partial y}{\partial x_i} = \sum_{P \in \mathcal{P}(x_i, f)} \prod_{e \in P} (\text{elementary partial derivative attached to } e)\,, \tag{6.3}$$

where $\mathcal{P}(x_i, f)$ is the set of all paths connecting the root vertex of f to the independent variable vertex x_i in the computational graph and the elementary partial derivative at edge e, connecting v_i and v_j, is $d_{i,j}$. Equation (6.3) is simply the chain rule. Notice that the CG of and equation contains all information required to evaluate and expression and all of its partial derivatives.

The discussion above explains how a general system of nonlinear equations can be represented as a sequence of elementary operations and how the partial derivatives of these equations can be extracted from the computational graph (the basis for all of the AD methods discussed in this thesis). At first it may seem counter-intuitive that automatic differentiation should perform any better than other analytical derivative evaluation techniques, such as symbolic differentiation. After all, the same quantities are being computed and the form of the partial derivative expressions are given by the elementary rules of differentiation which apply to all techniques. This section is concluded with a small example comparing symbolic differentiation to one variant of automatic differentiation, the *scalar sweep reverse mode*. The details of this technique will be discussed in detail in the following section.

Consider the simple nonlinear equation

$$z = xy \cdot \cos(xy). \tag{6.4}$$

An optimized graph of this equation is shown in Figure 6-2. Applying the rules of differentiation to graph shown in Figure 6-2 results in the following expressions:

$$\partial z/\partial x = \cos(xy) \cdot y + (xy) \cdot (-\sin(xy) \cdot y) \tag{6.5}$$

$$\partial z/\partial y = \cos(xy) \cdot x + (xy) \cdot (-\sin(xy) \cdot x) \tag{6.6}$$

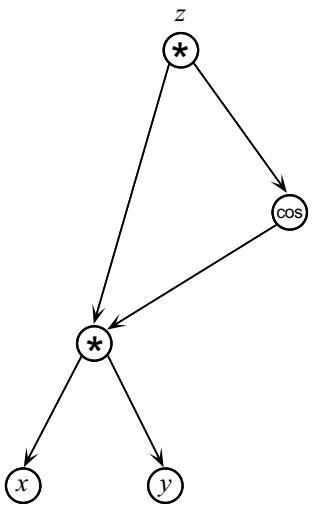

Figure 6-2: Directed acyclic graph (DAG) of equation (6.4).

The DAG of the original expression and partial derivatives are shown in Figure 6-3. Note that the term $-\sin(xy)$ appears twice in the partial derivative expressions. It is possible to include rules in the symbolic differentiation code that recognize this possibilty and only generate one expression for $-\sin(xy)$, however, handling these special cases can dramatically increase the cost associated with the symbolic differentiation. Furthermore, the identification and removal of these redundant expressions from an arbitrary graph a posteriori is a combinatorial process.

Suppose the evaluation sequence for computing z, $\partial z/\partial x$, and $\partial z/\partial y$ were extracted from the graph shown in Figure 6-3. Based on this graph, the minimum number of operations required to compute these quantities is

$$\begin{aligned}
\alpha_1 &= xy \\
\alpha_2 &= \cos(\alpha_1) \\
z &= \alpha_1\alpha_2 \\
\partial z/\partial x &= \alpha_2 y + \alpha_1(-\sin(\alpha_1)y) \\
\partial z/\partial y &= \alpha_2 x + \alpha_1(-\sin(\alpha_1)x),
\end{aligned}$$

requiring a total of eight multiplies, two additions, two unary minuses, one cos, and

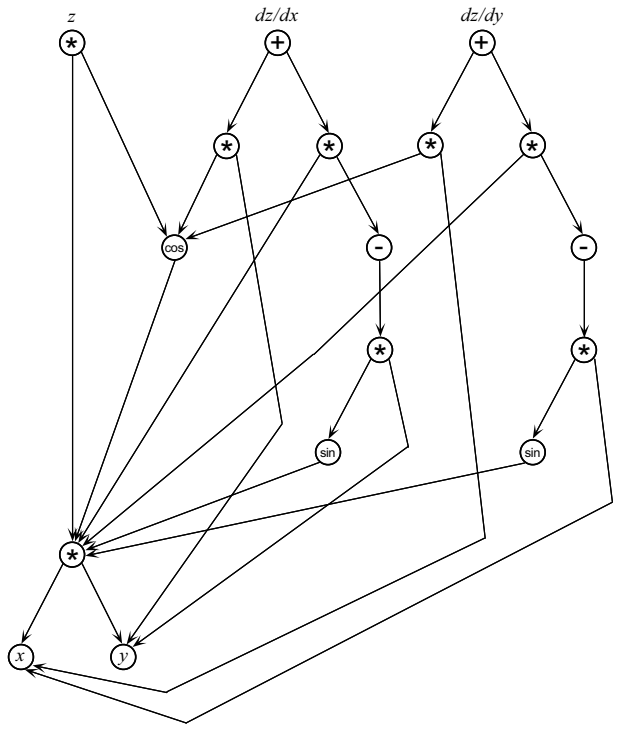

Figure 6-3: Directed acyclic graph (DAG) of equation (6.4) and partial derivatives (6.5) and (6.6).

two sin. Even if the redundant expressions were identified and handled properly, the cost would be decreased by one sin and one unary minus evaluation. By inspection, we can see that the partial derivative expressions can be further simplified to reduce the cost of the evaluation, however, to perform this automatically with the computer would require *symbolic transformation* and common subexpression identification, which, as mentioned above, is quite expensive. Optimizations such as these account for a substantial fraction of the time associated with compiling large residual routines with the optimization flags turned on. Similar problems occur when generating FORTRAN code from within symbolic manipulation codes such as Maple, where for reasonably small problems (less than 100 equations), computer memory (and user patience) is exhausted before the code is generated.

Figure 6-4 contains the computational graph of equation (6.4). Applying one variant of the chain-rule (equation (6.3)) results in the following sequence of operations

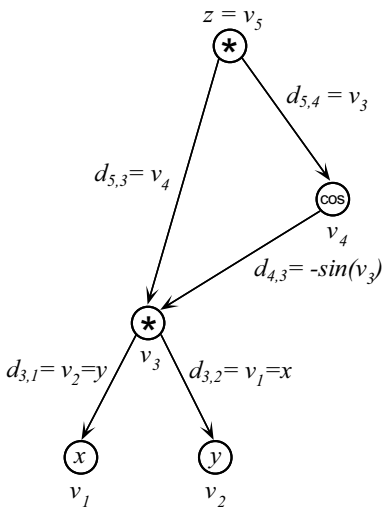

Figure 6-4: Computational graph (CG) of equation (6.4).

for computing z, $\partial z/\partial x$, and $\partial z/\partial y$:

$$\begin{aligned} v_3 &= xy \\ v_4 &= \cos(v_3) \\ z &= v_3 v_4 \\ d_{4,3} &= -\sin(v_3) \\ \alpha &= d_{5,3} + d_{4,3} d_{5,4} \\ \partial z/\partial x &= \alpha d_{3,1} \\ \partial z/\partial y &= \alpha d_{3,2} \end{aligned}$$

requiring a total of five multiplies, one addition, one unary minus, one cos, and one sin. Even in this trivial example, the way in which the partial derivatives are computed can have a significant impact on the cost of the evaluation.

6.1.1 Memory Savings

Another observation that can be made in the previous example problem is that the memory required to represent an expression and its partial derivatives is much less

when stored as a CG rather than a family of DAGs (compare Figure 6-4 to Figure 6-3). The number of vertices required to represent the original equation and partial derivatives as a CG is five for the original expression and two additional vertices for the non-trivial elementary partial derivative, $d_{4,3} = -\sin(v_3)$. The number of vertices required for the DAG representation is five for the original expression and an additional twelve for the partial derivatives (or nine if the term $-\sin(xy)$ is represented once). This difference is significant and is especially important from within an interpretive environment where the graph data structures are persistent throughout the course of a calculation. This dramatic saving is due to the fact that much of the information required to compute the partial derivative is stored implicitly in the CG, that is, we know a priori that we must multiply elementary partial derivatives along paths and sum these products over all paths connecting a root vertex to a particular independent variable vertex; on the other hand, this information must be stored explicitly in the DAG representation. Figure 6-5 contains an example of the elementary partial derivatives associated with some basic binary operators. In the case of the

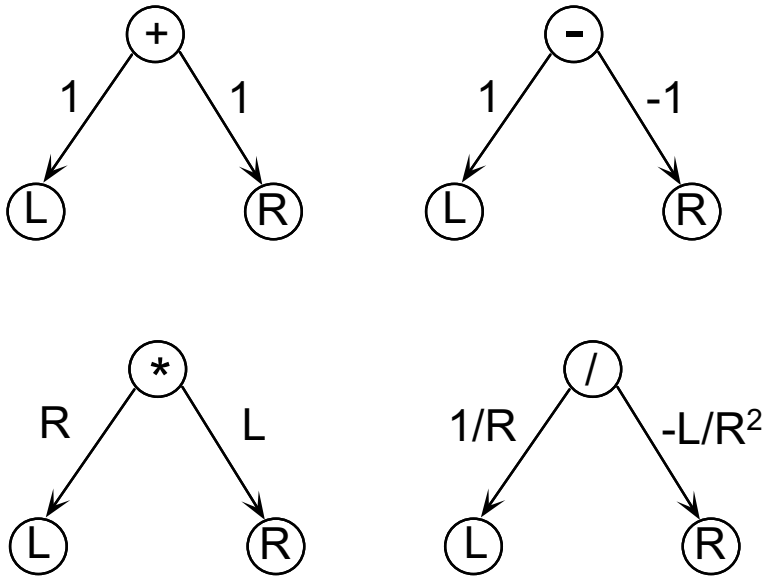

Figure 6-5: Elementary partial derivatives associated with binary operators: $+$, $-$, $\times$, and $/$.

addition and subtraction operator, only one unity vertex and one minus unity vertex needs to be allocated for the entire system of equations. No additional vertices need

to be allocated for the multiplication operator and only three additional vertices need to be allocated for the division operator. The substantial memory savings provided by the CG representation are further illustrated in chapter 8 of this thesis.

6.2 Forward and Reverse Modes

Obtaining partial derivatives using the computational graph and equation (6.3) is referred to as *accumulation* and is simply the application of the chain-rule. Various modes of AD can be developed by performing this accumulation in different orders. The two basic modes are the forward and reverse sweeps. First the scalar variants of the forward and reverse modes will be discussed, followed by a discussion of several vector sweep versions.

A key assumption is now made on the evaluation of an elementary operation and its associated elementary partial derivatives. This assumption will provide the basis for several complexity bounds derived later.

Assumption 1 *Elementary operations and their elementary partial derivatives are evaluated such that*

$$1 \leq \frac{cost\{g_j = g_j + \gamma_j \cdot \nabla v_j(v_i)_{i \in \mathcal{A}_j}\}}{cost\{v_j(v_i)_{i \in \mathcal{A}_j}\}} \leq 3, \tag{6.7}$$

where γ_j is an arbitrary scalar, g_j is an arbitrary vector of dimension $|\mathcal{A}_j|$, and ∇ denotes the vector of partial derivatives with respect to the $|\mathcal{A}_j|$ arguments of v_j.

In other words, the calculation of the vector of elementary partial derivatives of an operation multiplied by an arbitrary scalar and added to an arbitrary vector is no more expensive than three times the cost of evaluating the elementary operation alone [52]. As stated in [52], this bound is very reasonable for single-processor machines even taking into account memory accesses; it is rather conservative for most arithmetic binary operators and elementary functions and tight for multiplication (with the assumption that addition is half the cost of multiplication). The situation is better

for a multiprocessor. For example, the dot product elementary operation will require at least $\log_2 |\mathcal{A}_j|$ cycles, whereas its gradient incrementation is a SAXPY which can be computed in $\mathcal{O}(1)$ time. Furthermore, several independent elementary operations can be parallelized over multiple processors. However, the discussion in this text will be limited to single-processor machines and thus provide an upper bound on the accumulation costs.

6.2.1 Scalar Accumulation

Consider the system of equations $y = f(x)$, $f : \mathbb{R}^n \longrightarrow \mathbb{R}^m$, and assume this system of equations is represented as a sequence of elementary operations. Let $J(x)$, $J : \mathbb{R}^n \longrightarrow \mathbb{R}^{m \times n}$, denote the Jacobian matrix of this system of equations. Suppose the independent variables, $x \in \mathbb{R}^n$, are differentiable with respect to some parameter and this differentiation is denoted by x'. By the chain-rule, we have the recurrence relationship,

$$v'_j = \sum_{i \in \mathcal{A}_j} d_{j,i} v'_i \in \mathbb{R}. \tag{6.8}$$

By Assumption 1, the cost of computing v'_j for all $j \in \mathcal{V}$ is bounded by

$$1 \leq \frac{cost\{y' = J(x)x'\}}{cost\{f(x)\}} \leq 3, \tag{6.9}$$

where $cost\{f(x)\} \equiv \sum_{j \in \mathcal{V}} cost\{v_j(v_i)_{i \in \mathcal{A}_j}\}$. The evaluation of $J(x)x'$ in this manner is referred to as a *forward sweep*. The term "forward" refers to the fact that the Jacobian is constructed, or *accumulated*, from the elementary partial derivatives by moving forward through the list of elementary operations from vertex 1 to vertex $N = |\mathcal{V}|$[1]. The full Jacobian matrix can be obtained through n forward sweeps with x' initialized to each of the n Cartesian basis vectors in $\mathbb{R}^n$. As a result, the cost of evaluating the

[1]This is due to the fact that the summation in (6.8) is for $i \in \mathcal{A}_j$ and $i < j$ for all i.

Jacobian is bounded above by $3n$ times the cost of evaluating $f(x)$ alone, a result much worse than computing the Jacobian matrix through finite difference approximation. (It should be noted that the methods for computing Jacobian matrices in this chapter are free of truncation error.)

In the forward sweep, or forward mode, of automatic differentiation, we begin at the independent variable vertices and work up to dependent variables vertices. Alternatively, we can define a new set,

$$\mathcal{S}_i = \{i < j \leq n + m + p \mid i \in \mathcal{A}_j\} \tag{6.10}$$

(in the graph representation j is in $\mathcal{S}_i$ if vertex v_j is a parent of vertex v_i) and define a new recurrence relationship,

$$\hat{v}_i = \sum_{j \in S_i} d_{j,i} \hat{v}_j \in \mathbb{R}, \tag{6.11}$$

where $\hat{v}_i \equiv \frac{\partial}{\partial v_i} \hat{y}^T f(x) \in \mathbb{R}$ represents the *sensitivity* of the system of equations with respect to elementary variable v_i with a fixed adjoint weight vector $\hat{y} \in \mathbb{R}^m$. As above, the cost of computing the $\hat{v}_i$'s for all $i \in \mathcal{V}$ is bounded above by

$$1 \leq \frac{cost\{\hat{x} = \hat{y}^T J(x)\}}{cost\{f(x)\}} \leq 3. \tag{6.12}$$

The Jacobian can be evaluated by performing m *reverse sweeps* with the $\hat{y}$'s initialized to each of the m Cartesian basis vectors in $\mathbb{R}^m$. In this case, the list of operations is traversed in the opposite order of the forward sweep, hence, reverse sweep, or reverse mode. The cost of evaluating the Jacobian in this manner is bounded above by $3m$ times the cost of evaluating $f(x)$ alone. Again, this is unacceptable in most cases. Note however that the gradient of a single equation in the system, say $f_j(x)$, can be evaluated in a single reverse sweep with $\hat{y}$ initialized to e_j. The reverse mode of automatic differentiation *is* optimal for gradient evaluations. Furthermore, it is

possible to compute a relatively tight bound on the roundoff error associated with a function evaluation in addition to its gradient at a small multiple of the cost of evaluating the function alone [64] (see section 6.3).

The memory required to accumulate the Jacobian matrix using the forward mode is the same as the memory required to evaluate the set of residuals. However, the memory required to accumulate the Jacobian matrix using the reverse mode is $\mathcal{O}(|\mathcal{V}|)$ larger than that of the residual evaluation. This is due to the storage of adjoint quantities during Jacobian accumulation.

The inefficiencies of the scalar sweeps above are due to the fact that many of the v'_j and $\hat{v}_j$, $j \in \mathcal{V}$, are zero (especially early in the calculation) and, thus, there are many needless multiplications and additions of zero performed. The next section discusses how these unnecessary multiplications and additions can be eliminated by performing *vector sweeps*.

6.2.2 Vector Accumulation

The recurrence relationships (6.8) and (6.11) can be written in vector form as

$$\nabla v_j = \sum_{i \in \mathcal{A}_j} d_{j,i} \nabla v_i \in \mathbb{R}^n \tag{6.13}$$

and

$$\bar{v}_i = \sum_{j \in S_i} d_{j,i} \bar{v}_j \in \mathbb{R}^m, \tag{6.14}$$

respectively, where $\bar{v}_i = (\partial y_1/\partial v_i, \partial y_2/\partial v_i, \ldots, \partial y_m/\partial v_i)$. By initializing ∇v_j, $j \in \mathcal{I}$, or $\bar{v}_i$, $j \in \mathcal{D}$, to the appropriate Cartesian basis vectors in $\mathbb{R}^n$ or $\mathbb{R}^m$ and initializing all other ∇v_j or $\bar{v}_i$ to zero, the entire Jacobian can be evaluated in a single forward or reverse sweep. The additional memory required for the vector sweeps is $\mathcal{O}(n\,|\mathcal{V}|)$ for the forward mode and $\mathcal{O}(m\,|\mathcal{V}|)$ for the reverse mode. Furthermore, the ratio of the

cost of evaluating the Jacobian to the cost of evaluating $f(x)$ using the vector sweeps is still bounded above by $3n$ for the forward mode and $3m$ for the reverse mode, however, the vector sweep versions will out perform the corresponding scalar sweep versions due to the fact that the vector version passes through the elementary operation list once, eliminating duplicate computations. That is, if the entire Jacobian is constructed with one pass through the elementary operation list (as opposed to constructing the Jacobian a column at a time with the forward mode or a row at a time with the reverse mode) there will be no duplicate passes through common sections of the operations list (i.e., sequences of elementary operations that are common to more than one dependent or independent variable). As with the scalar sweeps, the inefficiency of the vector sweep forward and reverse modes is due to multiplication and addition of zero entries in the ∇v_j's and the $\bar{v}_i$'s.

Sparse Vector Accumulation

One obvious way to eliminate the unnecessary operations is to work with sparse vector data structures for ∇v_j and $\bar{v}_i$ [13]. By only performing the necessary operations in the vector recurrence relations (6.13) and (6.14), we find that the cost of evaluating the Jacobian using the sparse vector sweep version of the forward mode is bounded above by $3\hat{n}$ times the cost of evaluating $f(x)$ alone and the cost of evaluating the Jacobian using the sparse vector sweep version of the reverse mode is bounded above by $3\hat{m}$ times the cost of evaluating $f(x)$ alone, where $\hat{n}$ and $\hat{m}$ are the maximum number of nonzero entries in any row and in any column of the Jacobian matrix, respectively. One disadvantage of this approach, however, is the overhead associated with the need for indirect addressing for the sparse vector representation of ∇v_j and $\bar{v}_i$.

To illustrate the Jacobian evaluation techniques described above consider the fol-

lowing system of nonlinear equations:

$$f_1 = b_1 x_2 + x_1 + x_1 x_2 \tag{6.15}$$

$$f_2 = \sin(x_1) - x_1 x_2 \frac{x_2}{x_3} \tag{6.16}$$

$$f_3 = x_1 x_2 \frac{x_2}{x_3} - (x_1 + b_2) \tag{6.17}$$

$$f_4 = b_3 x_3 + \frac{x_2}{x_3} + b_4. \tag{6.18}$$

This system of equations is shown as a DAG in Figure 6-6. Each vertex in the equation

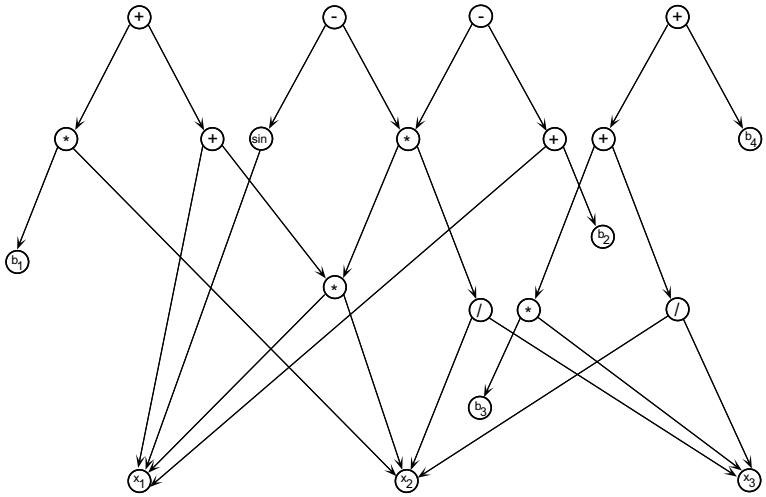

Figure 6-6: Directed acylic graph of system of equations (6.15)- (6.18).

graph corresponds to an elementary operation with the left and right children as the arguments. Given this representation of a system of equations along with a list of the dependent variable vertices, the algorithm shown in Figure 6-7, based on the *depth-first search*, can be used to label the vertices of the graph. In this algorithm, the left and right children of vertex v are denoted by left[v] and right[v], respectively. In addition, it is assumed that each vertex in the graph has an integer field, denoted by index[v], that holds the index of the vertex.

The complexity of IndexVertices is $\mathcal{O}(|\mathcal{V}|)$ (the complexity of depth-first search is proportional to the number of vertices and edges in an arbitrary graph, however, in an equation DAG, the number of edges is roughly twice the number of vertices,

```
IndexVertices(v,CurrentIndex)
 1  ▷ Recursively set children labels.
 2  if (left[v] ≠ NULL) and (index[left[v]] = 0) then
 3     IndexVertices(left[v],CurrentIndex)
 4  end
 5  if (right[v] ≠ NULL) and (index[right[v]] = 0) then
 6     IndexVertices(right[v],CurrentIndex)
 7  end
 8  ▷ Set label of argument.
 9  if v ≠ constant then
10     index[v] ← CurrentIndex
11     CurrentIndex ← CurrentIndex + 1
12  end
```

Figure 6-7: Algorithm for enumerating vertices based on depth-first search.

$|\mathcal{V}|$. Prior to calling procedure IndexVertices, the index field of all vertices in the graph are initialized to zero and the variable `CurrentIndex` is initialized to unity. Procedure IndexVertices is then called with each dependent variable vertex as an argument. Figure 6-8 shows the graph of equations (6.15)-(6.18) with vertices labeled with the indices obtained from procedure IndexVertices and edges labeled with the elementary partial derivatives. The index sets for the graph in Figure 6-8 are:

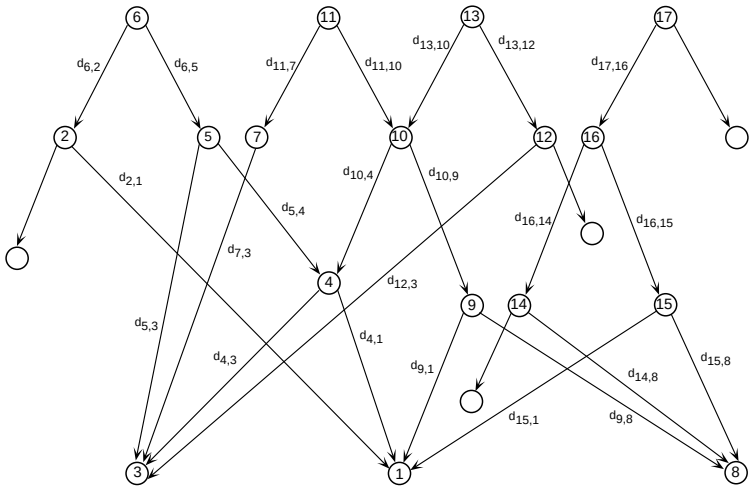

Figure 6-8: Computational graph of system of equations (6.15)-(6.18).

$\mathcal{I} = \{1, 3, 8\}$, $\mathcal{D} = \{6, 11, 13, 17\}$, and $\mathcal{E} = \{2, 4, 5, 7, 9, 10, 12, 14, 15, 16\}$. The vertices

corresponding to the constants b_1, b_2, b_3, and b_4 are not labeled due to the fact that they do not participate in the Jacobian evaluation other than for the calculation of elementary partial derivatives.

Once the elementary partial derivatives have been computed, the information contained in the computational graph is all that is required for computing the Jacobian matrix. In the case of the vector forward mode, the recurrence relationship (6.13) is executed in the order of increasing vertex indices. The execution order for the forward mode is shown in Figure 6-9. In the case of the vector reverse mode, the recurrence relationship (6.14) is executed in the order of decreasing vertex indices. The execution order for the reverse mode is shown in Figure 6-10. Note that for this example, $\hat{n} = n$ and $\hat{m} = m$.

$$
\begin{aligned}
\nabla v_1 &= e_2 = (0,1,0) \\
\nabla v_2 &= d_{2,1}\nabla v_1 = (0, d_{2,1}\nabla v_1^2, 0) \\
\nabla v_3 &= e_1 = (1,0,0) \\
\nabla v_4 &= d_{4,3}\nabla v_3 + d_{4,1}\nabla v_1 = (d_{4,3}\nabla v_3^1, d_{4,1}\nabla v_1^2, 0) \\
\nabla v_5 &= d_{5,3}\nabla v_3 + d_{5,4}\nabla v_4 = (d_{5,3}\nabla v_3^1 + d_{5,4}\nabla v_4^1, d_{5,4}\nabla v_4^2, 0) \\
\nabla v_6 &= d_{6,2}\nabla v_2 + d_{6,5}\nabla v_5 = (d_{6,5}\nabla v_5^1, d_{6,2}\nabla v_2^2 + d_{6,5}\nabla v_5^2, 0) \\
\nabla v_7 &= d_{7,3}\nabla v_3 = (d_{7,3}\nabla v_3^1, 0, 0) \\
\nabla v_8 &= e_3 = (0,0,1) \\
\nabla v_9 &= d_{9,1}\nabla v_1 + d_{9,8}\nabla v_8 = (0, d_{9,1}\nabla v_1^2, d_{9,8}\nabla v_8^3) \\
\nabla v_{10} &= d_{10,4}\nabla v_4 + d_{10,9}\nabla v_9 = (d_{10,4}\nabla v_4^1, d_{10,4}\nabla v_4^2 + d_{10,9}\nabla v_9^2, d_{10,9}\nabla v_9^3) \\
\nabla v_{11} &= d_{11,7}\nabla v_7 + d_{11,10}\nabla v_{10} \\
&= (d_{11,7}\nabla v_7^1 + d_{11,10}\nabla v_{10}^1, d_{11,10}\nabla v_{10}^2, d_{11,10}\nabla v_{11}^3) \\
\nabla v_{12} &= d_{12,3}\nabla v_3 = (d_{12,3}\nabla v_3^1, 0, 0) \\
\nabla v_{13} &= d_{13,10}\nabla v_{10} + d_{13,12}\nabla v_{12} \\
&= (d_{13,10}\nabla v_{10}^1 + d_{13,12}\nabla v_{12}^1, d_{13,10}\nabla v_{10}^2, d_{13,10}\nabla v_{10}^3) \\
\nabla v_{14} &= d_{14,8}\nabla v_8 = (0, 0, d_{14,8}\nabla v_8^3) \\
\nabla v_{15} &= d_{15,1}\nabla v_1 + d_{15,8}\nabla v_8 = (0, d_{15,1}\nabla v_1^2, d_{15,8}\nabla v_8^3) \\
\nabla v_{16} &= d_{16,14}\nabla v_{14} + d_{16,15}\nabla v_{15} = (0, d_{16,15}\nabla v_{15}^2, d_{16,14}\nabla v_{14}^3 + d_{16,15}\nabla v_{15}^3) \\
\nabla v_{17} &= d_{17,16}\nabla v_{16} = (0, d_{17,16}\nabla v_{16}^2, d_{17,16}\nabla v_{16}^3)
\end{aligned}
$$

Figure 6-9: Vector sweep forward mode evaluation of Jacobian of equations (6.15)-(6.18). The vector components are denoted by $\nabla v_j = (\nabla v_j^1, \ldots, \nabla v_j^n)$.

$$
\begin{aligned}
\hat{v}_{17} &= e_4 = (0,0,0,1) \\
\hat{v}_{16} &= d_{17,16}\hat{v}_{17} = (0,0,0,d_{17,16}\hat{v}_{17}^4) \\
\hat{v}_{15} &= d_{16,15}\hat{v}_{16} = (0,0,0,d_{16,15}\hat{v}_{16}^4) \\
\hat{v}_{14} &= d_{16,14}\hat{v}_{16} = (0,0,0,d_{16,14}\hat{v}_{16}^4) \\
\hat{v}_{13} &= e_3 = (0,0,1,0) \\
\hat{v}_{12} &= d_{13,12}\hat{v}_{13} = (0,0,d_{13,12}\hat{v}_{13}^3,0) \\
\hat{v}_{11} &= e_2 = (0,1,0,0) \\
\hat{v}_{10} &= d_{11,10}\hat{v}_{11} + d_{12,10}\hat{v}_{13} = (0,d_{11,10}\hat{v}_{11}^2, d_{13,10}\hat{v}_{13}^3, 0) \\
\hat{v}_{9} &= d_{10,9}\hat{v}_{10} = (0,d_{10,9}\hat{v}_{10}^2, d_{10,9}\hat{v}_{10}^3, 0) \\
\hat{v}_{8} &= d_{9,8}\hat{v}_{9} + d_{14,8}\hat{v}_{14} = (0,d_{9,8}\hat{v}_{9}^2, d_{9,8}\hat{v}_{9}^3, d_{14,8}\hat{v}_{14}^4 + d_{15,8}\hat{v}_{15}^4) \\
\hat{v}_{7} &= d_{11,7}\hat{v}_{11} = (0,d_{11,7}\hat{v}_{11}^2, 0, 0) \\
\hat{v}_{6} &= e_1 = (1,0,0,0) \\
\hat{v}_{5} &= d_{6,5}\hat{v}_{6} = (d_{6,5}\hat{v}_{6}^1, 0, 0, 0) \\
\hat{v}_{4} &= d_{5,4}\hat{v}_{5} + d_{10,4}\hat{v}_{10} = (d_{5,4}\hat{v}_{5}^1, d_{10,4}\hat{v}_{10}^2, d_{10,4}\hat{v}_{10}^3, 0) \\
\hat{v}_{3} &= d_{5,3}\hat{v}_{5} + d_{7,3}\hat{v}_{7} + d_{4,3}\hat{v}_{4} + d_{12,3}\hat{v}_{12} \\
&= (d_{5,3}\hat{v}_{5}^1 + d_{4,3}\hat{v}_{4}^1, d_{7,3}\hat{v}_{7}^2 + d_{4,3}\hat{v}_{4}^2, d_{4,3}\hat{v}_{4}^3 + d_{12,3}\hat{v}_{12}^3, 0) \\
\hat{v}_{2} &= d_{6,2}\hat{v}_{6} = (d_{6,2}\hat{v}_{6}^1, 0, 0, 0) \\
\hat{v}_{1} &= d_{2,1}\hat{v}_{2} + d_{4,1}\hat{v}_{4} + d_{9,1}\hat{v}_{9} + d_{15,1}\hat{v}_{15} \\
&= (d_{2,1}\hat{v}_{2}^1 + d_{4,1}\hat{v}_{4}^1, d_{v,1}\hat{v}_{4}^2 + d_{9,1}\hat{v}_{9}^2, d_{v,1}\hat{v}_{4}^3 + d_{9,1}\hat{v}_{9}^3, d_{15,1}\hat{v}_{15}^4)
\end{aligned}
$$

Figure 6-10: Vector sweep reverse mode evaluation of Jacobian of equations (6.15)-(6.18). The vector components are denoted by $\bar{v}_i = (\bar{v}_i^1, \ldots, \bar{v}_i^n)$.

The operation count for the sparse vector forward mode is 33 multiplications and 6 additions (not including the cost of evaluating the elementary partial derivatives). The operation count for the sparse vector reverse mode is 31 multiplications and 7 additions. Of course, it is possible to eliminate ten trivial multiplications by unity in the forward mode and seven trivial multiplications in the reverse mode, however, this selective skipping is difficult within this framework. Furthermore, these trivial operations become neglibible as the problem size is increased. This small example is not to compare performance of the various modes of AD since for small dense problems the forward and reverse sweep implementations, as well as the other approaches described in the following chapter, require roughly the same number of operations. If sparse vector operations were not performed, the operation count would be 69

multiplications and 27 additions for the forward mode and 88 multiplications and 36 additions for the reverse mode.

Compressed Vector Accumulation

The indirect addressing required by the sparse vector sweep versions of the forward and reverse mode not only incurs additional overhead but also dramatically reduces the possibility of parallelization and vectorization. An alternative to the sparse implementation is based on grouping the columns and rows into structurally orthogonal sets. This is illustrated for the forward mode of AD in [6]. Two columns of a matrix are structurally orthogonal if they do not have nonzero entries in the same row. Similarly, two rows of a matrix are structurally orthogonal if they do not have nonzero entries in the same column. As discussed in the previous chapter, this technique is also used to improve the performance of finite diffence approximation of Jacobian matrices when the occurrence information is known explicitly. Suppose there are $\bar{n}$ structurally orthogonal columns in $J(x) \in \mathbb{R}^{m \times n}$ and $\hat{n} \leq \bar{n} \leq n$. The vector ∇v_j in the recurrence relationship (6.13) can be compressed into a $\bar{n}$ component vector. The Jacobian matrix can then be computed using recurrence relationship (6.13) with ∇v_j, $j \in \mathcal{I}$, initialized to e_k, where k is the structurally orthogonal group variable j is a member of, and all other ∇v_j initialized to zero. Upon completion of the sweep, ∇v_j, $j \in \mathcal{D}$, contain the partial derivatives of equation j with respect to variables in each of the $\bar{n}$ groups and the Jacobian is compressed into an $m \times \bar{n}$ dimension matrix. The Jacobian can be uncompressed, using the occurrence information and column group information, at a small fraction of the cost of evaluating the Jacobian. Similarly, suppose there are $\bar{m}$ structurally orthogonal rows in $J(x)$ with $\hat{m} \leq \bar{m} \leq m$. The sensitivity vector, $\bar{v}_i$, can be compressed into $\bar{m}$ components and the Jacobian matrix can be evaluated using the recurrence relationship (6.14) with $\bar{v}_i$, $i \in \mathcal{D}$, initialized to e_k (where row i is in group k) and all other $\bar{v}_i$ initialized to zero. Upon completion of a reverse sweep, $\bar{v}_i$, $i \in \mathcal{I}$, contain partial derivatives of equations in each of the $\bar{m}$ groups with respect to independent variable i and the Jacobian is compressed into an

$\bar{m} \times n$ dimension matrix. Again, the cost of uncompressing the Jacobian is minimal. This version of the vector sweep forward and reverse mode computes the Jacobian matrix at cost bounded above by $3\bar{n}$ (for the forward mode) and $3\bar{m}$ (for the reverse mode) times the cost of evaluating $f(x)$ alone. The additional memory over the scalar sweep versions is $O(\bar{n}\,|\mathcal{V}|)$ for the forward mode and $O(\bar{m}\,|\mathcal{V}|)$ for the reverse mode. Both the time and space complexity of this version of the vector sweep forward and reverse modes is larger than the sparse vector forward and reverse modes, however, the efficiency of the two methods depends greatly on the particular problem at hand and the computing environment where the computations are being performed.

6.3 Rounding Error Estimation

In addition to favorable time complexity for gradient calculations, the reverse mode of automatic differentiation has the ability to provide relatively tight estimates on the roundoff error associated with the computation of a function [64]. Let Δv_j denote the roundoff error associated with the computation of the *j-th* elementary operation of a scalar-valued function f. Let $\tilde{v}_j$ and $\tilde{v}_i$, $i \in \mathcal{A}_j$, denote the computed values of v_j and v_i. The roundoff error is estimated as follows:

$$\begin{aligned} \Delta v_j &= \tilde{v}_j(\tilde{v}_i)_{i\in\mathcal{A}_j} - v_j(v_i)_{i\in\mathcal{A}_j} \\ &= v_j(\tilde{v}_i)_{i\in\mathcal{A}_j} - v_j(v_i)_{i\in\mathcal{A}_j} + \tilde{v}_j(\tilde{v}_i)_{i\in\mathcal{A}_j} - v_j(\tilde{v}_i)_{i\in\mathcal{A}_j} \end{aligned} \tag{6.19}$$

If the roundoff error is sufficiently small, the first two terms of equation (6.19) can be approximated by a first-order Taylor series:

$$v_j(\tilde{v}_i)_{i\in\mathcal{A}_j} - v_j(v_i)_{i\in\mathcal{A}_j} \approx \sum_{i\in\mathcal{A}_j} \frac{\partial v_j}{\partial v_i} \Delta v_i \tag{6.20}$$

Thus,

$$\Delta v_j = \sum_{i \in \mathcal{A}_j} \frac{\partial v_j}{\partial v_i} \Delta v_i + \delta v_j, \tag{6.21}$$

where first term is the error *propagated* to vertex j from previous computations and $\delta v_j = \tilde{v}_j(\tilde{v}_i)_{i \in \mathcal{A}_j} - v_j(\tilde{v}_i)_{i \in \mathcal{A}_j}$ is the error *generated* at vertex j. By applying (6.21) all the way up to the root vertex, the total accumulated roundoff error in the evaluation of f is given by

$$\Delta f = \sum_{j=1}^{n+p} \frac{\partial f}{\partial v_j} \cdot \delta v_j \tag{6.22}$$

Assuming no error in the input data, the absolute roundoff error is bounded by

$$|\Delta f| \leq \epsilon \sum_{j=n+1}^{n+p} \left| \frac{\partial f}{\partial v_j} \right| \cdot |v_j| = \epsilon \sum_{j=n+1}^{n+p} |\hat{v}_j| \cdot |v_j| \tag{6.23}$$

where ϵ is the machine precision and $|\delta v_j| \leq \epsilon \, |v_j|$. If the absolute error in the input data is known, the total absolute error associated in the computation of f is bounded by

$$|\Delta f| \leq \epsilon \sum_{j=n+1}^{n+p} |\hat{v}_j| \cdot |v_j| + \sum_{j=1}^{n} |\hat{v}_j| \cdot |\Delta x_j| \, . \tag{6.24}$$

Using the reverse mode, the cost of computing a function, its gradient, and an estimate of the roundoff error is bounded above by six times the cost of computing the function alone. These roundoff error estimates can be used, for example, in the stopping criteria for iterative solution methods.

6.4 Literature Review

The literature associated with automatic differentiation is quite extensive. The origins of the forward mode of AD can be traced back to the late 1950s and early 1960s in the work of Beda [11] and Wengert [118]. The book by Rall [92] gives an excellent description of the work done in AD between the 1950s and 1980. Speelpenning was the first to observe that the reverse mode of AD can out perform the forward mode in many cases [104]. Resulting from a thesis originally aimed at optimizing the code generated by the forward mode by common subexpression sharing, he realized that optimal *gradient* code can be obtain directly from the graph without optimization. The choice between the forward and reverse modes for Jacobian evaluations, however, is not apparent and depends on the particular problem at hand. An excellent discussion of AD is presented by Griewank in [51].

Automatic differentiation has been aimed primarily at the differentiation of *programs* (containing conditional statements, loops, etc.). Two distinct approaches are used for automatic differentiation: pre-compilation and operator overloading. In the precompliler approach, the user provides a subroutine in a language such as C or FORTRAN for the calculation of a set of equations. The precompiler breaks down the arithmetic operations in this subroutine into a set of elementary arithmetic operations and computes the elementary partial derivatives for these operations. With this information, a new subroutine is generated which computes the residuals and the Jacobian. This new subroutine can be compiled and linked to other programs. Some automatic differentiation procompilers currently available are: JAKEF [59], GRESS [61], PADRE2 [68], and ADIFOR [14]. The second approach makes use of the *operator overloading* capability of many modern programming languages such as C++, Ada, and PASCAL-SC. Using operator overloading, the programmer can define new data types and redefine arithmetic operators and intrinsic functions taking these new data types as arguments so that the compiler generates the additional instructions required for the Jacobian evaluation. Some automatic differentiation packages employing operator overloading are: ADOL-C [50], BC1 [26], and GC1 [28].

Chapter 7

A New Class of Automatic Differentiation Methods

In this chapter, a new class of differentiation methods, known as 'subgraph reduction', developed in this thesis is presented. By virtue of considering differentiation methods from within an interpretive equation-oriented simulator, we have taken a "graph centered" approach, that is, the equation graph has been the basis for the algorithms developed rather than considering the elementary operation sequence and using the graph to illustrate the ideas. This graph centered approach has led to a novel method for AD that is particularly advantageous for use within an interpretive equation-oriented simulator. The basic approach is described in the first part of this chapter. This is followed by temporal and spatial complexity analysis. Finally, each variant of the approach is described in detail in the implementation section. Both interpreted and compiled versions of the algorithm are described. Numerical comparisons illustrating the advantages of this new approach are presented in the following chapter.

7.1 Subgraph Reduction Approach

Consider the system of equations:

$$y_1 = sin(3^{x_1 x_2}) \tag{7.1}$$

$$y_2 = 2 + 3^{x_1 x_2} - \prod_{j=1}^{4} x_j \tag{7.2}$$

$$y_3 = 3^{x_1 x_2} - \prod_{j=1}^{4} x_j + x_2 + x_3 x_4 \tag{7.3}$$

$$y_4 = 3^{x_1 x_2} - x_2^{x_4}. \tag{7.4}$$

The DAG of this system is shown in Figure 7-1. To reiterate, the interior vertices of

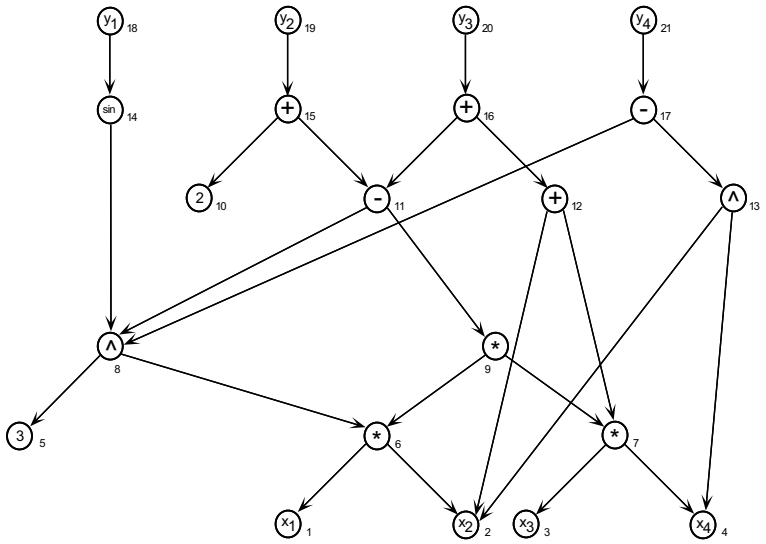

Figure 7-1: DAG representation of equations (7.1)-(7.4).

the graph correspond to the elementary operations with the children corresponding to the arguments. The edges in this graph point from operator to operand. Vertices without any children correspond to independent variables and constants. Vertices without any parents correspond to the dependent variables. With the edges labeled with the elementary partial derivatives, the partial derivatives of the dependent variables with respect to the independent variables can be easily extracted from the graph using some variant of automatic differentiation described in the previous chap-

ter. A partial derivative is simply the sum over all paths connecting the dependent variable vertex to the independent variable vertex of the products of the elementary partial derivatives along these paths. The order in which these sums and products are performed is what basically distinguishes between the various modes of automatic differentiation.

Note that the DAG shown in Figure 7-1 contains several subgraphs that are shared by several equations: the subgraph rooted at vertex 11 is shared by equations (7.2) and (7.3). The subgraph rooted at vertex 8 is shared by equations (7.1) and (7.4) and the subgraph rooted at vertex 11. The subgraph rooted at vertex 6 is shared by the subgraphs rooted at vertices 8 and 9. Finally, the subgraph rooted at vertex 7 is shared by the subgraphs rooted at vertices 9 and 12. As before, the subgraphs rooted at vertices 6, 7, 8, and 11 will be referred to as common subexpressions of equation system (7.1)-(7.4). Independent variable vertices v_2 and v_4 also have multiple incident edges, however, they are not considered common subexpressions in this discussion.

If the scalar sweep version of the forward and reverse modes are used to compute the Jacobian, the accumulation is performed column-by-column or row-by-row. Common subexpressions present a problem in this case due to the fact that there will be repeated calculations as the Jacobian is assembled. Re-evaluation of common subexpressions during Jacobian evaluation can be avoided by assembling the entire Jacobian matrix simultaneously using the sparse or compressed vector versions of the forward and reverse mode.

This chapter describes a new approach for Jacobian accumulation that obviates the need for performing vector accumulations by replacing certain sections of the graph with simplified graphs. This allows the Jacobian to be accumulated efficiently through a series of scalar accumulations. Like the automatic differentiation techniques described in the previous chapter there are two basic variants of this subgraph simplification or reduction approach, the forward and reverse modes. These two basic modes are discussed in the following section. This is then followed by a discussion of a hybrid approach where different variants are applied to different regions of the graph.

This approach requires a graph representation of a system of equations. This requirement does not present a problem since most systems of equations of interest, even when represented as a general FORTRAN subroutine, can be readily and automatically converted to an equation graph. This will be discussed in more detail in the implementation section of this chapter.

7.1.1 Reverse Mode

Like the other versions of the reverse mode discussed in the previous chapter, this version accumulates the elementary partial derivatives by moving through the elementary operation list from operation N down to operation 1 or, equivalently, from the dependent variable vertices at the top of the CG down to the independent variable vertices at the bottom. The definition of a common subexpression is different for the reverse and forward modes of the subgraph reduction approach.

Definition 1 *In the reverse mode, a common subexpression is defined as a vertex in a subgraph reachable from more than one dependent variable vertex.*

Unfortunately, this definition overcounts the number of common subexpressions present in a graph (each vertex contained in a subgraph rooted at a common subexpression vertex will be considered a common subexpression). A more precise definition is needed for the common subexpression vertices that are of interest in this chapter. First, define the following *vertex dependent variable index set*:

$$\mathcal{O}_d(v) = \{j \in \mathcal{D} \quad | \quad \text{vertex } v \text{ is reachable from dependent variable vertex } v_j\} \tag{7.5}$$

where $\mathcal{D}$ is the set of indices of the dependent variables. For example, for the graph shown in Figure 7-1, $\mathcal{O}_d(v_{12}) = \{20\}$, $\mathcal{O}_d(v_9) = \{19, 20\}$, and $\mathcal{O}_d(v_6) = \{18, 19, 20, 21\}$. Next, define the *vertex dependent variable and common subexpres-*

sion index set:

$$\overline{\mathcal{O}}_d(v) = \{j \in \mathcal{V} \quad | \quad \text{vertex } v \text{ is reachable from dependent variable or common subexpression vertex } v_j\} \tag{7.6}$$

By constructing $\overline{\mathcal{O}}_d(\cdot)$ for each vertex in the graph in a specific order (described below), this set can be used to define the common subexpression vertices:

Definition 2 *Vertex v_k is a common subexpression if $\overline{\mathcal{O}}_d(v_i) \neq \overline{\mathcal{O}}_d(v_j)$ for at least one pair of $i, j \in \mathcal{S}_k$.*

Recall that in section 6.1, an elementary operation in the elementary operation list form of an equation system corresponds to an interior vertex in the DAG representation of the system. Furthermore, the indexing of the elementary operations was defined such that the indices of all arguments of an elementary operation are less than the index of the elementary operation itself, that is, $i < j \ \ \forall i \in \mathcal{A}_j$ (this ensures all elementary variables in an argument list have been precomputed). If the vertices in the corresponding equation graph are indexed in a similar manner (in this case, the index of a vertex v is greater than the indices of all vertices contained in the subgraph rooted at v), then this ordering along with definition 2 above can be used to identify the common subexpressions. The vertices are visited in the order of decreasing value of index. At each vertex v_j, $\overline{\mathcal{O}}_d(v_j)$ is constructed from its parents index sets as follows,

$$\overline{\mathcal{O}}_d(v_j) = \cup_{i \in \mathcal{S}_j} \overline{\mathcal{O}}_d(v_i)\,. \tag{7.7}$$

If during the construction of the index set the condition in definition 2 is satisfied, v_j is flagged as a common subexpression. By visiting the vertices in the order of decreasing value of index, the definition of a common subexpression above is well-posed. Figures 7-2, 7-3, and 7-4 contain example graphs where the dependent variable and common

subexpression vertex index set has been constructed for each vertex in the graph. The common subexpression vertices are highlighted in the graph.

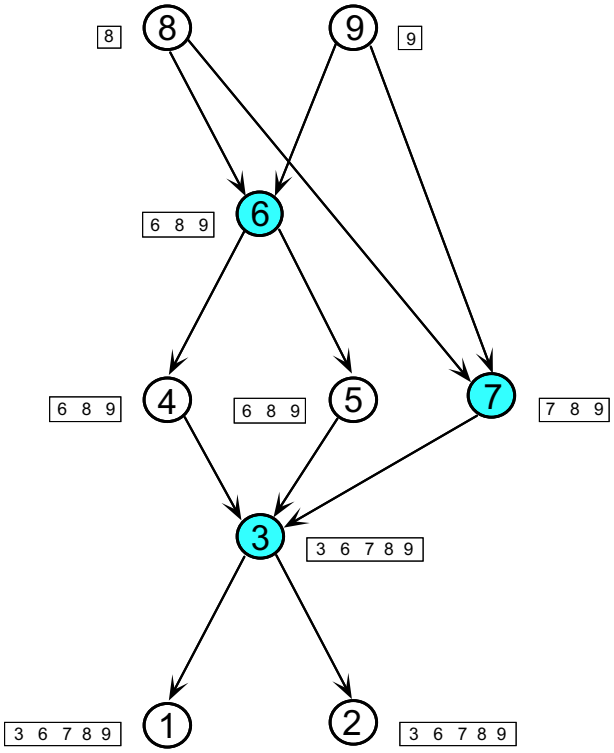

Figure 7-2: Common subexpression identification. Example 1.

Definition 2 is used in the example above where vertices 6, 7, 8, and 11 in the graph in Figure 7-1 are identified as common subexpressions. If the scalar sweep version of the reverse mode were applied to this graph, the accumulation below these vertices would be performed several times as the Jacobian was accumulated row-by-row. The reverse mode of the subgraph reduction approach avoids these redundant calculations by replacing the subgraphs rooted at common subexpression vertices with simplified graphs containing single edges pointing from the common subexpression vertex to each of the independent variables contained in the subgraph. The "not-so-elementary" partial derivatives attached to each of these new edges is equal to the partial derivative of the common subexpression with respect to the independent variable the edge points to. Figure 7-5 contains an equation graph before and after this subgraph reduction. The common subexpression vertices in this graph are identified by squares. (Notice that although vertex 9 in Figure 7-5 has two incident edges, it is

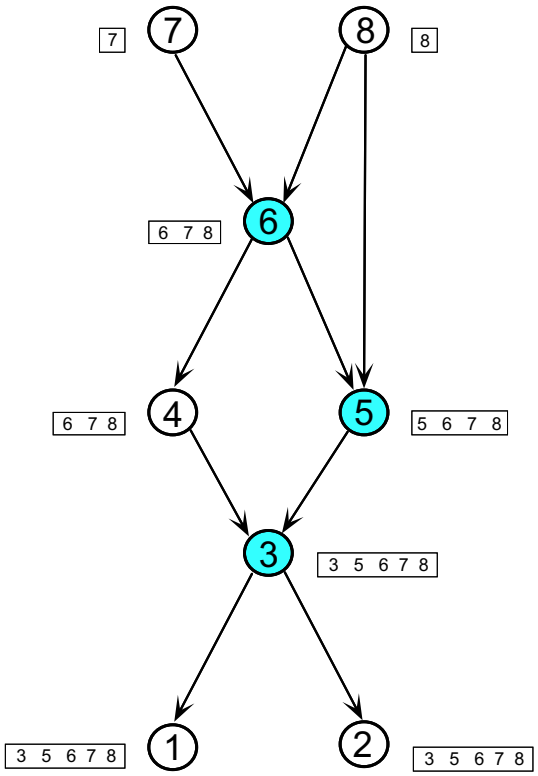

Figure 7-3: Common subexpression identification. Example 2.

not considered a common subexpression due to the fact that it is reachable from only a single dependent variable vertex.) The order in which the graph reduction problem is performed is extremely important. In the graph contained in Figure 7-5, vertex 8 is a common subexpression that is reachable from common subexpression vertex 13. Similarly, vertex 5 is a common subexpression vertex that is reachable from both common subexpressions 8 and 13. This observation defines a ranking for the common subexpressions.

Definition 3 *Given two common subexpression vertices* v_i *and* v_j, $rank[v_i] > rank[v_j]$ *if* v_i *is reachable from* v_j.

For the graph in Figure 7-5, $\text{rank}[v_5] > \text{rank}[v_8] > \text{rank}[v_{13}]$. By simplifying the subgraphs rooted at the common subexpressions in the order of *decreasing* rank, redundant calculations are avoided by using precomputed information. Once all of the subgraphs rooted at common subexpressions have been simplified in a hierarchical

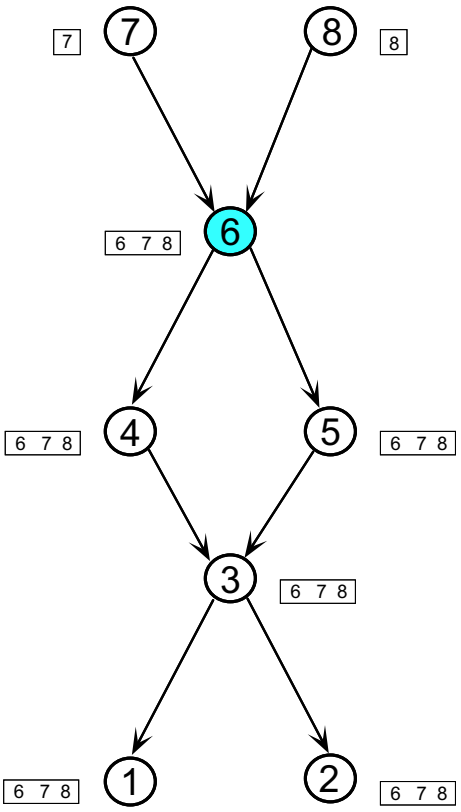

Figure 7-4: Common subexpression identification. Example 3.

manner, the Jacobian can be computed efficiently through a series of scalar reverse sweep accumulations. The details of this process are discussed in the implementation section later in this chapter.

7.1.2 Special Handling of Linear Equations

As described in chapter 5, the cost of computing the gradient of a function using the reverse mode of automatic differentiation is bounded from above by three times the cost of computing the function alone. In contrast, the cost of computing a gradient using symbolic differentiation is roughly n times the cost of computing the function where n is the number of independent variables[1]. Although the reverse mode of automatic differentiation out performs symbolic differentiation in many cases, the sit-

[1]This becomes a relatively loose upper bound when common subexpression evaluation is employed.

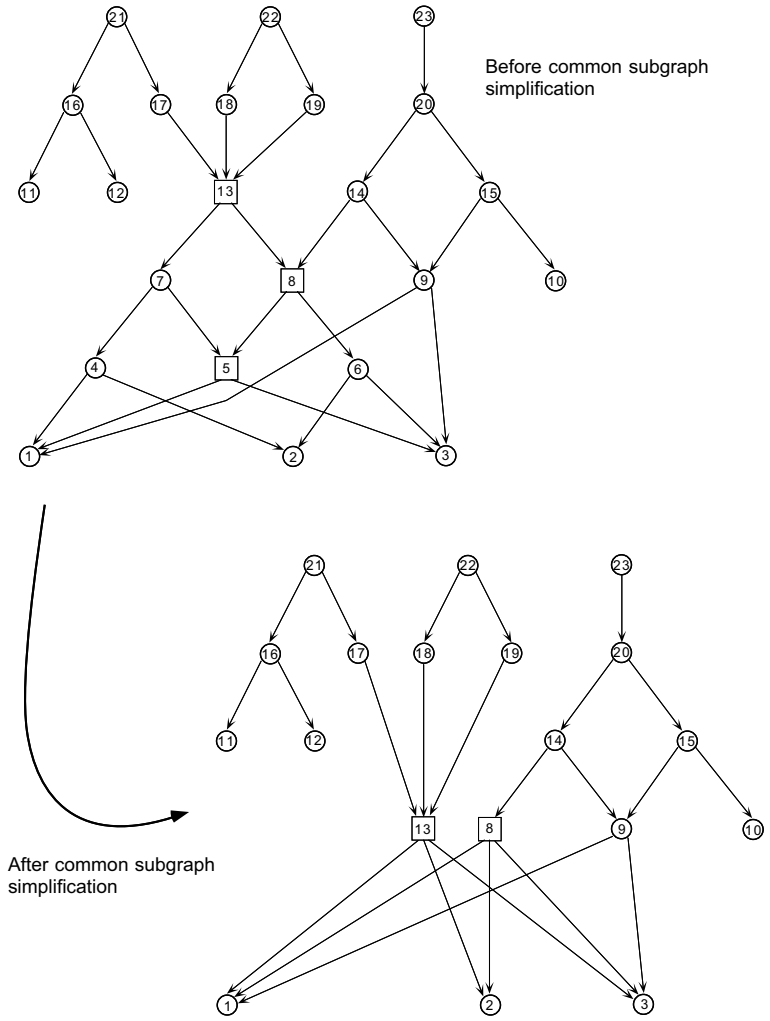

Figure 7-5: Equation graph before and after subgraph simplification.

uation is different for linear equations: the cost of evaluating a linear equation and its gradient using symbolic differentiation is essentially the same as the cost of evaluating the linear equation alone (the symbolic derivatives are simply constants), whereas, the same bound as above holds for the reverse mode. This observation is particularly important in process simulation where many of the equations encountered are linear (conservation equations, summation of mole fraction constraints, etc.). Linear equations can be handled in a similar manner as common subexpression vertices in the approach described above. Given the graph of a system of equations, the first step is to identify the linear equations and linear subgraphs of nonlinear equations. This can be performed simultaneously with the common subexpression processing

with very little additional cost (see implementation section). Once these equations are identified, their constant gradients can be computed and saved. This is identical to reducing the subgraphs rooted at common subexpression vertices. When a linear equation or subgraph is encountered during a Jacobian evaluation, its gradient is simply assigned rather than computed. As shown in the implementation section, this special handling of linear equations can be readily incorporated (particularly in the interpretive implementation) into the framework discussed in this chapter.

7.1.3 Forward Mode

The forward mode version of the subgraph reduction approach, like the other forward mode versions of automatic differentiation discussed in this thesis, accumulates the elementary partial derivatives by moving through the elementary operation list from operation 1 to operation N or, equivalently, from the independent variable vertices at the bottom of the CG up to the dependent variable vertices at the top. As mentioned above, the common subexpression is defined differently for the forward mode of the subgraph reduction approach.

Definition 4 *A common subexpression for the forward mode version of the subgraph reduction approach is defined as an interior vertex that has at least two operands that are functions of independent variables.*

To make this definition more precise, define the following *vertex independent variable index set*:

$$\mathcal{O}_i(v) = \{j \in \mathcal{I} \quad | \quad \text{independent variable vertex } v_j \text{ is contained in the graph rooted at vertex } v\} \tag{7.8}$$

where $\mathcal{I}$ is the set of indices of the independent variables. For example, for the graph shown in Figure 7-1, $\mathcal{O}_i(v_{12}) = \{2,3,4\}$, $\mathcal{O}_i(v_9) = \{1,2,3,4\}$, and $\mathcal{O}_d(v_6) =$

$\{1, 2\}$. Vertex v_k is a common subexpression vertex if there exists at least two $i, j \in \mathcal{A}_k$, such that $\mathcal{O}_i(v_i) \neq \emptyset$ and $\mathcal{O}_i(v_j) \neq \emptyset$. Obviously, this definition will result in many more vertices defined as common subexpressions than the reverse mode of the subgraph reduction approach. Because of the different order in which the vertices are encountered during the accumulation for the forward and reverse modes, the definition of common subexpressions in each case is not symmetric. This difference will become apparent in the implementation section in this chapter.

7.1.4 Hybrid Approaches

As shown in section 7.3, the temporal complexity (computational cost) of the reverse and forward modes of the subgraph reduction approach are related to the number of common subexpressions in the graph. The fact that these common subexpressions are defined differently for each mode, indicates that one implementation may perform better than the other depending on the structure of the graph of the system of equations of interest (see Figure 8-1). Alternatively, it may be advantageous to apply both modes to different regions of the graph in order to achieve improved performance over exclusive application of either the forward or reverse mode.

One simple algorithm is to examine the common subexpressions in the graph with large independent and dependent variable index sets. The number of operations required for both the forward and reverse modes can be readily extracted from the graph and the difference in the number of operations and memory requirements can be used as a basis to determine which mode is better suited for this region of the graph.

7.1.5 Markowitz Criteria Approach

A second "graph-oriented" approach has recently been presented in the literature [52]. In this work, the authors view the Jacobian accumulation as a *vertex elimination* problem and solve it using a greedy heuristic. The results of this paper are

summarized below. Figure 7-6 contains an example where two vertices are successively eliminated from a graph. The Jacobian accumulation process can be viewed

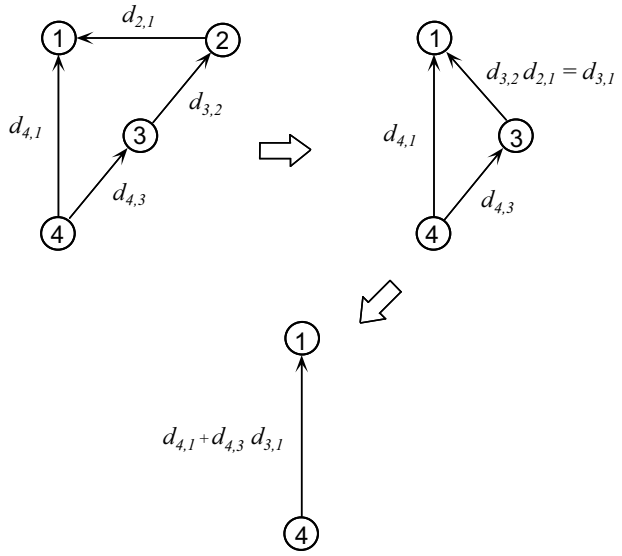

Figure 7-6: Vertex elimination problem. Vertices v_2 and v_3 are successively eliminated from the graph.

as a vertex elimination process where all intermediate vertices in the graph are eliminated, resulting in a bipartite graph with the dependent variable vertices on the top and the independent variable vertices at the bottom.

Consider the following recurrence relation (originally shown in equation (6.8)),

$$v'_j = \sum_{i \in \mathcal{A}_j} d_{j,i} v'_i. \tag{7.9}$$

For the Jacobian accumulation problem, removing vertex v_k from the CG is equivalent to eliminating it from the reucurrence above. For all $j \in \mathcal{S}_k$, substitute

$$\begin{aligned} v'_j &= d_{j,k} \cdot v'_k + \sum_{k \neq i \in \mathcal{A}_j} d_{j,i} v'_i \\ &= \sum_{i \in \mathcal{A}_k} d_{j,k} d_{k,i} \cdot v'_i + \sum_{k \neq i \in \mathcal{A}_j} d_{j,i} v'_i \\ &= \sum_{i \in \tilde{\mathcal{A}}_j} \tilde{d}_{j,i} \cdot v'_i, \end{aligned}$$

where $\tilde{\mathcal{A}}_j = \mathcal{A}_k \cup \mathcal{A}_j - \{k\}$, and

$$\tilde{d}_{j,i} = d_{j,i} + d_{j,k} \cdot d_{k,i}$$

for all index pairs $j \in \mathcal{S}_i$ and $i \in \mathcal{A}_k$. The number of multiplications performed at vertex k during the elimination process is equal to the Markowitz count,

$$\text{mark}(k) = |\mathcal{A}_k| \cdot |\mathcal{S}_k| \,. \tag{7.10}$$

Furthermore, one addition is performed each time a new edge is constructed (see Figure 7-6 where an additional edge connecting v_4 to v_1 is introduced). Hence, the cost of computing a vertex when viewed as a vertex elimination problem is related to the Markowitz count as follows,

$$\text{cost}(J(x)) \propto \sum_{i \in \mathcal{E}} \text{mark}(i). \tag{7.11}$$

Of course, mark(i) corresponds to the Markowitz count when vertex i is eliminated which, of course, may change during the course of the elimination. The ideal elimination order is the one that minimizes (7.11), however, the solution to this problem is conjectured to be NP-hard. The authors chose a greedy heuristic to solve this problem approximately.

This is a very elegant and insightful way of viewing the Jacobian accumulation process. The problem with this approach, as stated by the authors, is that the approach requires significantly more memory than either the forward or reverse vector sweep versions of automatic differentiations. Furthermore, the additional cost associated with determining the elimination ordering is not included, which, as the authors suggest, is substantial. Both of these problems make this approach appear less attractive, particularly when the accumulation sequence may change several times over the course of a calculation (as is the case with hybrid discrete/continuous simulation).

7.2 Implementation

The details of the algorithms described in this chapter are discussed here. As mentioned above, this approach requires a graph representation of the system of equations of interest. If the approach is applied within some symbolic programming environment (e.g., Maple, SPEEDUP, and ABACUSS) then this graph is already available, having been constructed from the infix notation form of the equations provided by the user directly from the keyboard (Maple) or through an input file (SPEEDUP and ABACUSS). If the system of equations is in the form of a C or FORTRAN subroutine, the graph can be readily constructed by moving sequentially through the statement list. Arbitrarily complex DAGs can be constructed through the use of intermediate variables that take the form of common subexpressions within the graph.

Compiled and interpreted implementations will be described. In the compiled mode, the equation graph is an intermediate data structure from which the accumulation sequence is extracted and written to a file that can be compiled and linked to other routines to provide residual and derivative values. Once this new routine has been constructed, the memory associated with the computational graph can be recovered. In the interpreted mode, the equation graph is persistent and used throughout the course of the particular calculation being performed. This difference results in slightly modified algorithms, tailored to each of the two architectures.

The algorithms described below take an equation graph "as is", meaning that common subexpression vertices are already present in the graph and there is no attempt to identify others. The search for common subexpressions is a combinatorial problem and has been the focus of much research in the past [49]. The identification and exploitation of common subexpressions can offer significant speed and memory improvements for many problems, however, this problem can be handled as a preprocessing step to the differentiation algorithm.

Let $\mathbf{G} = (\mathbf{V},\mathbf{E})$ denote the computational graph of the equation system $f : \mathbb{R}^n \longrightarrow \mathbb{R}^m$ where $\mathbf{V}$ denotes the set of vertices and $\mathbf{E}$ denotes the set of edges. As stated in the previous chapter, the graph representation and the elementary operation represen-

tation of a system of equations are closely related: each vertex in the computational graph represents an elementary operation (this requires the introduction of elementary assignments for each independent and dependent variable). See the Computational Graph and Accumulation section in chapter 6 for a description of the elementary operation list representation. Using the notation in that section, there are a total of $|\mathcal{V}|$ vertices ($n = |I|$ independent variable vertices and $m = |\mathcal{D}|$ dependent variable vertices) in $\mathbf{G}$. For this discussion, assume that the system of equations can be broken down into elementary operations corresponding to binary and unary operators (e.g., $+$, $-$, $/$, etc.) and unary functions (e.g., sin, cos, exp, etc.). In this case, the number of edges in $\mathbf{G}$ is bounded above by twice the number of vertices. This observation is relevant when analyzing the complexity of the steps involved in this algorithm. The direction of the edges in $\mathbf{G}$ point from operator to operand, however, the vertex data structures also contain a parent list (that is, a list of all vertices that point to them). Let $\mathcal{P}(v_k) = \{v_i\}_{i \in \mathcal{S}_k}$ denote the parent list of vertex v_k where the set $\mathcal{S}_k$ is defined by (6.10).

7.2.1 Compiled Implementation

In this case, computational graph $\mathbf{G}$ is a temporary data structure used to generate code for Jacobian and residual evaluation. The code generation is broken down into two distinct phases: a) graph preprocessing and b) extraction of the accumulation sequence (the actual code generation step). Obviously, the case will be different for both the forward and reverse mode.

Reverse Mode

The preprocessing step results in a list of common subexpressions, sorted by rank, that is used during the subsequent code generation step. For this phase of the algorithm, it is assumed that the data structures representing the graph vertices contain an integer field used for indexing and a boolean flag used to indicate whether or not the vertex is a common subexpression. The preprocessing for the reverse mode can be summarized

in the following steps.

1. Initialize the index field of all vertices in the graph to zero,

2. Index vertices in $\mathbf{G}$ by calling IndexVertices (see Figure 6-7) from each dependent variable v_j, $j \in \mathcal{D}$, and store vertices in a list, $\mathcal{L}$, sorted by *decreasing* value of the index,

3. Determine dependent variable and common subexpression vertex index set, $\overline{\mathcal{O}}_d(v_k)$, for each $k \in \mathcal{L}$ (in decreasing order of index), and

4. Identify all common subexpressions and store in a list sorted by *decreasing* value of rank.

To illustrate the algorithm, the computational graph of the system of equations,

$$f_1 = b_1 x_2 + x_1 + x_1 x_2 \tag{7.12}$$

$$f_2 = sin(x_1) - x_1 x_2 \frac{x_2}{x_3} \tag{7.13}$$

$$f_3 = x_1 x_2 \frac{x_2}{x_3} - (x_1 + b_2) \tag{7.14}$$

$$f_4 = b_3 x_3 + \frac{x_2}{x_3} + b_4, \tag{7.15}$$

is shown in Figure 7-7. This is the same system examined in the previous chapter (shown as a DAG in Figure 6-6). Initialization of the index field of all vertices in the graph can be performed using a depth-first search and has complexity $\mathcal{O}(|\mathcal{V}|)$. The indexing of the variables is performed using the depth-first search based algorithm, IndexVertices, shown in Figure 6-7 in chapter 6. This initial indexing is used to sort the vertices for later processing and to rank the common subexpressions identified later. The common subexpression rank is related to the index by the following relation,

$$\text{rank}\,[v_i] > \text{rank}\,[v_j] \quad \text{if} \quad i < j.$$

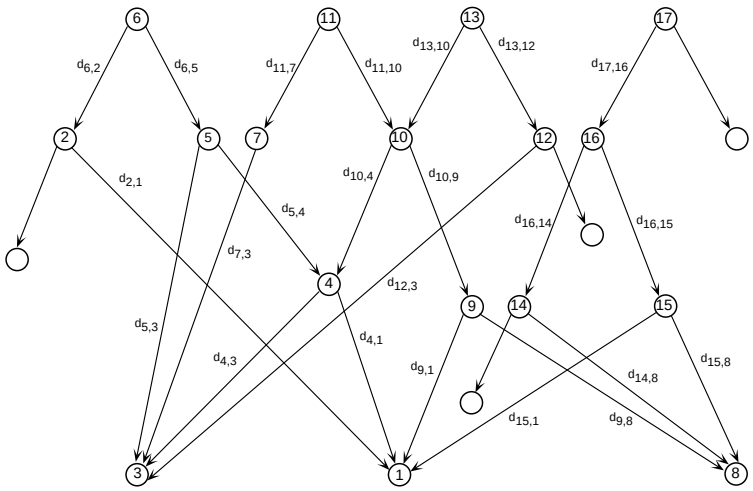

Figure 7-7: Computational graph of system of equations (7.12)-(7.15).

This is due to the fact that IndexVertices numbers vertices below vertex v before numbering vertex v itself. The CG shown in Figure 7-7 has vertices indexed in this manner. The storing of vertices in the sorted list can be accomplished by adding an additional vertex pointer field to the vertex data structure and can be performed simultaneously while indexing. (Actually, creating the list while performing the indexing results in an ordering opposite that required by the reverse mode.) The complexity of this step (including the reversing of the list) is $\mathcal{O}(|\mathcal{V}|)$. The next step is to construct the dependent variable and common subexpression index set for each vertex in $\mathbf{V}$. As mentioned earlier, the dependent variable and common subexpression index set of a vertex is related to the dependent variable and common subexpression index sets of its parents by

$$\overline{\mathcal{O}}_d\,(v) = \cup_{u \in \mathcal{P}(v)} \overline{\mathcal{O}}_d\,(u)\,.$$

Let $\mathcal{L}(\mathbf{G})$ denote the *sorted* list of vertices generated in the first step of the preprocessing phase of the algorithm (for the example above, the first entry of this list is v_{17}, the second is v_{16}, and so on down to v_1). First, the dependent variable and common subexpression index sets of the dependent variables are constructed:

for each $k \in \mathcal{D}$ **do**

$\overline{\mathcal{O}}_d(v_k) = \{k\}$;

end

Obviously, the complexity of this step is $\mathcal{O}(|\mathcal{D}|)$. The dependent variable and common subexpression index set is then constructed for each vertex in the graph by moving through the vertex list in decreasing order of index:

for each $v_k \in \mathcal{L}(\mathbf{G})$ **do**

$\overline{\mathcal{O}}_d(v_k) = \cup_{u \in \mathcal{P}(v_k)} \overline{\mathcal{O}}_d(u)$;

if v_k is a common subexpression (as defined in definition 2) **then**

$\overline{\mathcal{O}}_d(v_k) = \overline{\mathcal{O}}_d(v_k) \cup \{k\}$;

Flag v_k as a common subexpression ;

Insert v_k into common subexpression list ;

end

end

Common subexpressions can be easily identified while the dependent variable and common subexpression index sets are constructed: a common subexpression is a vertex that has at least two parents with distinct dependent variable and common subexpression index sets (the order in which the vertices are encountered ensures this definition is not circular). In addition, by moving through the vertex list in reverse order, common subexpressions are inserted into the front of a second list as they are identified. This common subexpression list is already sorted in the order of decreasing value of common subexpression rank by virtue of the order in which the vertices are examined. A flag is set in each common subexpression vertex so that they can be easily identified during subsequent processing. A very pessimistic upper bound for the common subexpression identification is $\mathcal{O}(\kappa^2 |\mathcal{V}|)$ where κ is equal to the number of common subexpressions and dependent variables contained in the graph, however, for non-pathological or sparse problems, this step is extemely fast even for very large systems (requiring a fraction of a second for systems containing tens of thousands of equations).

After the preprocessing step, we have a sorted list of common subexpression vertices. Let $\mathcal{CS}$ denote this list. For the graph shown in Figure 7-7,

$$\mathcal{CS} = \{v_4, v_{10}\}.$$

All memory required to hold the dependent variable and common subexpression index sets can be recovered at this point. The basic idea of the code generation step is to create lists of vertices in an order that they will be encountered during the accumulation phase. The algorithm for generating these lists, ReverseAccumulationList, is shown in Figure 7-8. This algorithm assumes the vertex data structure has a boolean field named encountered and accessed in the usual manner (e.g., encountered[v]).

```
ReverseAccumulationList(ℒ,v)
 1  ▷ If vertex has already been encountered, return.
 2  if encountered[v] = true then
 3      return ;
 4  else
 5      encountered[v] ← true ;
 6  end
 7  if v = independent variable or common subexpression then
 8      ▷ Insert this vertex to the front of the list.
 9      InsertVertex(ℒ,v) ;
10      return
11  end
12  ▷ Recursively visit left and right children and flag paths.
13  ActivePath(v,left[v]) ;
14  ReverseAccumulationList(ℒ,left[v]) ;
15  ActivePath(v,right[v]) ;
16  ReverseAccumulationList(ℒ,right[v]) ;
17  ▷ Insert this vertex to the front of list.
18  InsertVertex(ℒ,v) ;
19  end
```

Figure 7-8: Algorithm for generating a list of vertices in the order required by the reverse mode.

Procedure ReverseAccumulationList constructs a list of vertices in a *reverse* order, that is, the order they will be encountered during a reverse scalar sweep. The procedure InsertVertex simply inserts a vertex to the front of list $\mathcal{L}$. The procedure ActivePath simply flags the edge between vertex v and its children as being active (a vertex field is assumed to be available for this). This flag is used during the subsequent code generation phase. The complexity of ReverseAccumulationList is proportional to the number of vertices contained in the subgraph rooted at the argument vertex. Let $\mathcal{L}(v_k)$ denote the list constructed by calling ReverseAccumulationList with $\mathcal{L}$ initialized to $\emptyset$ and $v = v_k$ (and the encountered field in all vertices in the subgraph rooted at v_k set to `false`). The following lists are generated from vertices v_4, v_{10}, v_{17}, v_{13}, v_{11}, and v_6:

$$\begin{aligned}
\mathcal{L}(v_4) &= \{v_4, v_1, v_3\}, \\
\mathcal{L}(v_{10}) &= \{v_{10}, v_9, v_8, v_1, v_4\}, \\
\mathcal{L}(v_{17}) &= \{v_{17}, v_{16}, v_{15}, v_1, v_{14}, v_8\}, \\
\mathcal{L}(v_{13}) &= \{v_{13}, v_{12}, v_3, v_{10}\}, \\
\mathcal{L}(v_{11}) &= \{v_{11}, v_{10}, v_7, v_3\}, \text{ and} \\
\mathcal{L}(v_6) &= \{v_6, v_5, v_4, v_3, v_2, v_1\}.
\end{aligned}$$

These lists contain the vertices in the order in which the accumulation would be performed using the reverse mode to compute the gradient of the subexpression rooted at each of the argument vertices. For example, the following sequence of operations

can be generated given the list $\mathcal{L}(v_{17})$ above:

$$\begin{aligned}
\hat{v}_{17} &= 1 \\
\hat{v}_{16} &= d_{17,16} \cdot \hat{v}_{17} \\
\hat{v}_{15} &= d_{16,15} \cdot \hat{v}_{16} \\
\hat{v}_{1} &= d_{15,1} \cdot \hat{v}_{15} \\
\partial f_4 / \partial x_2 &= \hat{v}_{1} \\
\hat{v}_{14} &= d_{16,14} \cdot \hat{v}_{16} \\
\hat{v}_{8} &= d_{14,8} \cdot \hat{v}_{14} + d_{15,8} \cdot \hat{v}_{15} \\
\partial f_4 / \partial x_3 &= \hat{v}_{8}
\end{aligned}$$

The subgraph rooted at vertex v_{17} does not contain common subexpressions and thus the gradient may be efficiently computed through a scalar sweep. The exploitation of the common subexpressions is illustrated later in this section where the entire Jacobian of equations (7.12)-(7.15) is constructed.

Pseudocode for computing the accumulation sequence is shown in Figure 7-9. The arguments of procedure ReverseAccumulation are a list of vertices, $\mathcal{L}$, generated as described above, a gradient vector, ∇f, a constant vector, g, and a scalar, α. Here it is assumed that the vertex data structures have the following fields: a real variable, adjoint, and two vertex pointers for the left and right elementary partial derivatives. The procedure GetNextVertex simply removes the next vertex in the list and returns it. Function d(u,v) simply returns the value of the elementary partial derivative attached to the edge connecting vertices u and v. The set $\mathcal{P}'(v)$ denotes the subset of parents of v that are *active* and is defined as

$$\mathcal{P}'(v) = \{u \in \mathcal{P}(v) \mid \text{Path between } u \text{ and } v \text{ is active.}\}$$

```
ReverseAccumulation(L,∇f,g,α)
 1  ▷ Copy the constant vector g into the gradient vector ∇f.
 2  ∇f ← g ;
 3  ▷ Initialize the dependent variable vertex adjoint to the constant α
 4  v ← GetNextVertex(L) ;
 5  adjoint[v] ← α ;
 6  ▷ Move through the list until empty.
 7  while L ≠ ∅ do
 8      v ← GetNextVertex(L) ;
 9      adjoint[v] ← 0 ;
10      ▷ Loop over the active parents.
11      for each u ∈ P'(v) do
12              adjoint[v] ← adjoint[v] + d(u,v) · adjoint[u] ;
13      end
14      if v = independent variable x_j then
15              (∇f)_j ← (∇f)_j + adjoint[v] ;
16      else if v = common subexpression then
17              ∇f ← ∇f + adjoint[v] · ∇v ;
18      end
19  end
```

Figure 7-9: Algorithm for reverse accumulation from vertex list.

where the path is flagged as active during the list construction. The paths are flagged as active during the vertex list construction due to the fact that during a particular gradient evaluation, when a common subexpression is encountered, not every parent is involved, only those associated with the graph of the vertex whose gradient is desired. For example, when evaluating the gradient of vertex v_{13} in Figure 7-7 and vertex v_{10} is encountered, only the edge connecting v_{13} and v_{10} is active while the edge connecting v_{11} to v_{10} is not (the opposite holds when evaluating the gradient of vertex v_{11}).

On line 17 of ReverseAccumulation, it is assumed that the gradients of all common subexpressions contained in the graph rooted at the argument vertex have been precomputed (i.e., the graph has been previously reduced). The order in which ReverseAccumulation is called during a Jacobian evaluation (described below) ensures this will be the case. If the list, $\mathcal{L}$, was constructed from the dependent

variable vertex corresponding to the graph of $f : \mathbb{R}^n \longrightarrow \mathbb{R}$, then the following quantity will be computed upon completion of ReverseAccumulation:

$$g + \alpha \nabla f.$$

Notice that all operations in ReverseAccumulation are scalar operations except for the SAXPY operation on line 17. This can be handled in one of several ways. First, ∇f can be represented as dense n vector, in which case the complexity at this step would be $\mathcal{O}(n)$. However, for large, sparse problems, there will be many unnecessary calculations performed. Alternatively, the independent variable index sets for all dependent variable and common subexpression vertices can be precomputed ($\mathcal{O}(\hat{n}\,|\mathcal{V}|)$ complexity assuming only unary and binary operators and functions are contained in the graph) and sparse vector operations can be performed. Although this was cited as a deficiency of the sparse vector implementations of the forward and reverse modes, it should be noted that in the subgraph reduction approach, this indirect addressing is *only* performed at the common subexpression vertices and independent variables, not at every vertex contained in the subgraph rooted at a common subexpression as it would in the sparse vector implementations. Finally, if the approach were implemented in a vector computer, it may be advantageous to use the compressed vector ideas described in the previous chapter. In this case, ∇f would be a dense $\bar{n}$ dimensional vector. As with the vector implementations of the forward and reverse modes, the optimal representation of ∇f depends both on the problem at hand and the computer architecture in which the computations are being performed.

Code for the gradient evaluation can readily be generated using a modified version of ReverseAccumulation; instead of computing the adjoints, lines of code corresponding to the operations being performed are written to a file. Given the procedures described above, the code generation phase of the algorithm can be summarized as follows.

Initial Phase — Residual and Elementary Partial Derivative Evaluation

The first step is to extract the elementary operation list from the graph and write the computational sequence to a file. As a result, residuals are computed simultaneously with the Jacobian. Second, the expressions for the elementary partial derivatives are written to the file (these calculations use some of the elementary variables computed during the residual evaluation).

Subgraph Reduction Phase — Common Subexpression Gradient Evaluation

It is assumed that the encountered field of all vertices in the computational graph of the system of equations of interest have been initialized to `false` and the active fields of all edges have been initialized to `non-active`. The complexity of this step is proportional to the number of vertices and edges in the graph ($\mathcal{O}(|\mathcal{V}|)$ if only unary and binary operators and functions are present).

For each common subexpression in the list $\mathcal{CS}$ (in the order of decreasing rank) perform the following steps:

1. Call procedure ReverseAccumulationList to generate a list of vertices $\mathcal{L}$,
2. Call the modified ReverseAccumulation procedure to generate code for computing the gradient of the common subexpression (in this case, g is set to zero and α is set to unity), and
3. For all vertices in $\mathcal{L}$, reset the encountered field to `false` and the active field of its edges to `non-active` (obviously this step is proportional to the number of vertices in $\mathcal{L}$ plus the number of edges associated with these vertices).

The precomputed elementary partial derivatives are used in this accumulation sequence. The steps above correspond to reducing the subgraphs rooted at the common subexpression vertices. Performing these steps in the order of decreasing common

subexpression rank ensures that the gradient of every common subexpression encountered in ReverseAccumulation will have had code for its evaluation previously written.

Jacobian Accumulation Phase — Dependent Variable Gradient Evaluation

Once the graph has been reduced in the step above, the full Jacobian can be constructed evaluating the gradient of each dependent variable vertex.

For each dependent variable vertex, v_k, $k \in \mathcal{D}$, perform the following steps:

1. Call procedure ReverseAccumulationList to generate a list of vertices $\mathcal{L}$,
2. Call the modified ReverseAccumulation procedure to generate code for computing the equation gradient (Jacobian row). If $g \neq 0$ and $\alpha \neq 1$ then the quantity computed will be α times the current row of the Jacobian plus g,
3. For all vertices contained in $\mathcal{L}$, reset the encountered field to `false` and the active field of its edges to `non-active`.

The gradient of each dependent variable is independent of the gradients of the other dependent variables and thus, the order in which the dependent variables are examined does not matter.

Applying this approach to equations (7.12)-(7.15) results in the accumulation sequence shown in Figures 7-10, 7-11, and 7-12. The computation of the residuals and elementary partial derivatives have been omitted in this example. Furthermore, some of the trivial initializations (e.g., line 9 of ReverseAccumulation) have been omitted.

The operation count required to compute the Jacobian is 29 multiplications and 7 additions (not including the evaluation of the elementary partial derivatives). This is roughly the same as the sparse vector forward and reverse described in the previous section (however, in this framework, it is much easier to avoid trivial multiplications by unity). This is not a surprise since it is expected that the sparse and compressed vector techniques and the subgraph reduction approach will perform similarly for

Common subexpression v_4

$$\begin{aligned} \partial v_4/\partial x_1 &= d_{4,3} \\ \partial v_4/\partial x_2 &= d_{4,1} \end{aligned}$$

Common subexpression v_{10}

$$\begin{aligned} \hat{v}_9 &= d_{10,9} \\ \hat{v}_1 &= d_{9,1} \cdot \hat{v}_9 \\ \partial v_{10}/\partial x_2 &= \hat{v}_1 \\ \hat{v}_8 &= d_{9,8} \cdot \hat{v}_9 \\ \partial v_{10}/\partial x_3 &= \hat{v}_8 \\ \hat{v}_4 &= d_{10,4} \\ \partial v_{10}/\partial x_1 &= \hat{v}_4 \cdot \partial v_4/\partial x_1 \\ \partial v_{10}/\partial x_2 &= \partial v_{10}/\partial x_2 + \hat{v}_4 \cdot \partial v_4/\partial x_2 \end{aligned}$$

Figure 7-10: Reverse mode subgraph reduction of computational graph shown in Figure 7-7.

small, dense problems without complicated subgraphs rooted at common subexpression vertices. As stated in the previous chapter, this example problem is not used to compare the performance of the various approaches. Examples are contained in the following chapter that illustrate the improvement offered by the subgraph reduction approach.

Forward Mode

The preprocessing step for the forward mode of the subgraph reduction approach also results in a list of common subexpressions, sorted by rank, that is used during the subsequent code generation phase. The preprocessing can be summarized in the following steps.

1. For each vertex in the graph, initialize the index field to zero, the encountered field to `false`, and the active fields of all edges to `non-active`,

2. Index vertices in **G** by calling IndexVertices (see Figure 6-7) from each dependent variable v_j, $j \in \mathcal{D}$, and store vertices in a list sorted by *increasing* value

Gradient of equation f_1

$$\begin{aligned}
\hat{v}_6 &= 1 \\
\hat{v}_5 &= d_{6,5} \cdot \hat{v}_6 \\
\hat{v}_4 &= d_{5,4} \cdot \hat{v}_5 \\
\partial f_1/\partial x_1 &= \hat{v}_4 \cdot \partial v_4/\partial x_1 \\
\partial f_1/\partial x_2 &= \hat{v}_4 \cdot \partial v_4/\partial x_2 \\
\hat{v}_3 &= d_{5,3} \cdot \hat{v}_5 \\
\partial f_1/\partial x_1 &= \partial f_1/\partial x_1 + \hat{v}_3 \\
\hat{v}_2 &= d_{6,2} \cdot \hat{v}_6 \\
\hat{v}_1 &= d_{2,1} \cdot \hat{v}_2 \\
\partial f_1/\partial x_2 &= \partial f_1/\partial x_2 + \hat{v}_1
\end{aligned}$$

Gradient of equation f_2

$$\begin{aligned}
\hat{v}_{11} &= 1 \\
\hat{v}_{10} &= d_{11,10} \cdot \hat{v}_{11} \\
\partial f_2/\partial x_1 &= \hat{v}_{10} \cdot \partial v_{10}/\partial x_1 \\
\partial f_2/\partial x_2 &= \hat{v}_{10} \cdot \partial v_{10}/\partial x_2 \\
\partial f_2/\partial x_3 &= \hat{v}_{10} \cdot \partial v_{10}/\partial x_3 \\
\hat{v}_7 &= d_{11,7} \cdot \hat{v}_{11} \\
\hat{v}_3 &= d_{7,3} \cdot \hat{v}_3 \\
\partial f_2/\partial x_3 &= \partial f_2/\partial x_3 + \hat{v}_3
\end{aligned}$$

Figure 7-11: Reverse mode subgraph reduction approach applied to computational graph shown in Figure 7-7 after subgraph reduction (continued in next figure).

of the index (this is achieved by appending the vertices to the end of a list),

3. Determine the independent variable index set, $\mathcal{O}_i(v_k)$, for each $k \in \mathcal{V}$, and

4. Identify all common subexpressions and store in a list sorted by *decreasing* value of rank.

These steps are described in detail below.

As with the reverse mode, the cost of initializing the fields of all vertices in the graph is $\mathcal{O}(|\mathcal{V}|)$. Furthermore, the vertex indexing and list construction can be performed simultaneously in $\mathcal{O}(|\mathcal{V}|)$ time by appending the vertices to the end of the list

Gradient of equation f_3

$$\begin{aligned}
\hat{v}_{13} &= 1 \\
\hat{v}_{12} &= d_{13,12} \cdot \hat{v}_{13} \\
\hat{v}_{3} &= d_{12,3} \cdot \hat{v}_{12} \\
\partial f_3/\partial x_1 &= \hat{v}_3 \\
\hat{v}_{10} &= d_{13,10} \cdot \hat{v}_{13} \\
\partial f_3/\partial x_1 &= \partial f_3/\partial x_1 + \hat{v}_{10} \cdot \partial v_{10}/\partial x_1 \\
\partial f_3/\partial x_2 &= \hat{v}_{10} \cdot \partial v_{10}/\partial x_2 \\
\partial f_3/\partial x_3 &= \hat{v}_{10} \cdot \partial v_{10}/\partial x_3
\end{aligned}$$

Gradient of equation f_4

$$\begin{aligned}
\hat{v}_{17} &= 1 \\
\hat{v}_{16} &= d_{17,16} \cdot \hat{v}_{17} \\
\hat{v}_{15} &= d_{16,15} \cdot \hat{v}_{16} \\
\hat{v}_{1} &= d_{15,1} \cdot \hat{v}_{15} \\
\partial f_4/\partial x_2 &= \hat{v}_1 \\
\hat{v}_{14} &= d_{16,14} \cdot \hat{v}_{16} \\
\hat{v}_{8} &= d_{15,8} \cdot \hat{v}_{15} + d_{14,8} \cdot \hat{v}_{14} \\
\partial f_4/\partial x_3 &= \hat{v}_8
\end{aligned}$$

Figure 7-12: Reverse mode subgraph reduction approach applied to computational graph shown in Figure 7-7 after subgraph reduction (continued).

as they are indexed. For the forward mode, rank and index are related as follows,

$$\text{rank}\left[v_i\right] > \text{rank}\left[v_j\right] \quad \text{if} \quad i > j.$$

The next step is to construct the independent variable index set for each vertex in $\mathbf{V}$. The independent variable index set of a vertex is related to the independent variable index sets of its children by

$$\mathcal{O}_i\left(v_k\right) = \cup_{j \in \mathcal{A}_k} \mathcal{O}_i\left(v_j\right).$$

Let $\mathcal{L}(\mathbf{G})$ denote the *sorted* list of vertices generated in the first step of the preprocessing phase of the algorithm (for the graph shown in Figure 7-7, the first entry of this list is v_1, the second is v_2, and so on up to v_{17}). First, the index sets of the independent variables are constructed:

for each $k \in \mathcal{I}$ **do**
    $\mathcal{O}_i(v_k) = \{k\}$;
end

The independent variable index set is then constructed for each vertex in the graph by moving through the vertex list in increasing order of index (this ensures that the definition of common subexpression for the forward mode is not circular):

for each $v_k \in \mathcal{L}(\mathbf{G})$ **do**
    $\mathcal{O}_i(v_k) = \cup_{j \in \mathcal{A}_k} \mathcal{O}_i(v_j)$;
    if v_k is a common subexpression (as defined in definition 4) **then**
        Flag v_k as a common subexpression ;
        Append v_k to the end of the common subexpression list ;
    end
end

The complexity of this step is $\mathcal{O}(\hat{n}^2 |\mathcal{V}|)$ where $\hat{n}$ is the maximum number of nonzeros in any row of the Jacobian matrix. The factor of $\hat{n}^2$ is due to the fact that, in an arbitrary graph, each vertex may have $\hat{n}$ children each of which have index sets of cardinality $\hat{n}$. However, the complexity of this step will be $\mathcal{O}(\hat{n} |\mathcal{V}|)$ if only unary and binary operators and functions are present in the graph. As with the reverse mode implementation, this bound is pessimistic. Once again, common subexpressions can be easily identified while the independent variable index sets are constructed with little additional cost: a common subexpression is a vertex that has at least two children vertices with nonempty independent variable index sets. By moving through the vertex list in forward order, common subexpressions are inserted into the front of a second list as they are identified. A flag is set in each common subexpression vertex so that they can be easily identified during subsequent processing. The independent

variable index sets associated with the dependent variables contains the Jacobian occurrence information. Let $\mathcal{CS}$ denote the common subexpression list. For the example shown in Figure 7-7, the common subexpression list will be

$$\mathcal{CS} = \{v_{16}, v_{15}, v_{10}, v_9, v_5, v_4\}.$$

As with the reverse mode, the preprocessing results in a sorted list of common subexpression vertices which is used in the subsequent code generation phase. The memory required to hold the independent variable index sets can again be recovered. Like the reverse mode, the accumulation sequence will be generated from lists extracted from the graph in the proper order. For the forward mode, the algorithm for generating these lists, ForwardAccumulationList, is shown in Figure 7-13. The

```
ForwardAccumulationList(L,v,v^o)
 1  ▷ If vertex has already been encountered, return.
 2  if encountered[v] = true then
 3     return ;
 4  else
 5     encountered[v] ← false ;
 6  end
 7  ▷ If vertex is not original vertex, append to end of list.
 8  if v ≠ v^o then
 9     AppendVertex(L,v) ;
10  end
11  ▷ Return if we've reached the end of a chain.
12  if v = dependent variable or
       ( v = common subexpression and v ≠ v^o) then
13     return ;
14  end
15  ▷ Recursively visit all parents of vertex v.
16  for each u ∈ P(v) do
17     ActivePath(u,v) ;
18     ForwardAccumulationList(L,u,v^o) ;
19  end
```

Figure 7-13: Algorithm for generating a list of vertices for the forward mode.

arguments of ForwardAccumulationList are a vertex list, $\mathcal{L}$, the current vertex to be processed, v, and the original vertex from which ForwardAccumulationList was called, v^o. Procedure ActivePath simply flags the edge connecting u and v as being *active* (a field in the vertex data structure is assumed to be available for this). This flag is used during the subsequent code generation phase. Once the list has been constructed using ForwardAccumulationList, the vertices in the list must be sorted in the order of increasing value of index. This step can be performed in $\mathcal{O}(l \log_2 l)$ time, where l is the number of entries in the list. Let $\mathcal{L}(v_k)$ denote the list generated by calling ForwardAccumulationList with v_k as an argument. For the example shown above, the lists generated from the common subexpressions would be (after sorting),

$$\begin{aligned}
\mathcal{L}(v_{16}) &= \{v_{17}\}, \\
\mathcal{L}(v_{15}) &= \{v_{16}\}, \\
\mathcal{L}(v_{10}) &= \{v_{11}, v_{13}\}, \\
\mathcal{L}(v_9) &= \{v_{10}\}, \\
\mathcal{L}(v_5) &= \{v_6\}, \text{ and} \\
\mathcal{L}(v_4) &= \{v_5, v_{10}\}.
\end{aligned} \tag{7.16}$$

Applying the procedure to each of the independent variable vertices results in the following lists,

$$\begin{aligned}
x_1 \,:\, \mathcal{L}(v_3) &= \{v_4, v_5, v_7, v_{11}, v_{12}, v_{13}\}, \\
x_2 \,:\, \mathcal{L}(v_1) &= \{v_2, v_4, v_6, v_9, v_{15}\}, \text{ and} \\
x_3 \,:\, \mathcal{L}(v_8) &= \{v_9, v_{14}, v_{16}, v_{15}\}.
\end{aligned} \tag{7.17}$$

Pseudocode for computing the forward accumulation sequence is shown in Figure 7-14. The arguments to ForwardAccumlation are a list of vertices, $\mathcal{L}$, generated

```
ForwardAccumulation(L, f̃, v)
1  ▷ Loop through vertex list.
2  v' ← 1 ;
3  while L ≠ ∅ do
4     u ← GetNextVertex(L) ;
5     ▷ Sum over all active paths.
6     u' ← Σ_{w∈A'_u} d(u,w)w' ;
7     if (u = common subexpression and u ≠ v) then
8        ▷ Incorporate precomputed sensitivity vector.
9        f̃ ← f̃ + u'ũ ;
10    else if u = dependent variable f_i then
11       f̃_i ← f̃_i + u' ;
12    end
13 end
```

Figure 7-14: Algorithm for forward accumulation from vertex list.

as described above, a sensitivity vector, $\tilde{f}$, and the vertex corresponding to the sensitivity vector (i.e., $\tilde{f} = \partial f/\partial v$ and v is either a common subexpression or independent variable vertex). The summation term involves only elements associated with paths from v to the current vertex (these are flagged by procedure ActivePath during the list construction). Upon completion of this procedure, $\tilde{f}$ contains the following information:

$$\tilde{f} = \langle \frac{\partial f_1}{\partial v}, \ldots, \frac{\partial f_m}{\partial v} \rangle.$$

As with the reverse mode, $\tilde{f}$ can be represented as a dense m vector, a compressed $\bar{m}$ vector, or a sparse vector depending on the problem and computer environment at hand. If the sparse or compressed vector approach is applied, then the independent variable index set of each dependent variable and common subexpression must be determined a priori.

Code for the accumulation can be readily generated using a modified form of ForwardAccumulation, where, like the reverse mode, lines of code are written to a file rather than actually performing the computation. The code generation phase is similar to the reverse mode and consists of the following phases.

Initial Phase — Residual and Elementary Partial Derivative Evaluation

As with the reverse mode, the first step is to extract the elementary operation list from the graph and write the computational sequence for residual and elementary partial derivative evaluation to a file.

Subgraph Reduction Phase — Common Subexpression Sensitivity Evaluation

It is assumed that the encountered field of all vertices in the computational graph of the system of equations of interest has been initialized to `false` and the active fields of all edges have been initialized to `non-active`. The complexity of this step is proportional to the number of vertices and edges in the graph ($\mathcal{O}(|\mathcal{V}|)$ if only unary and binary operators and functions are present).

For each common subexpression in the list $\mathcal{CS}$ (in the order of decreasing rank) perform the following steps:

1. Call procedure ForwardAccumulationList to generate a list of vertices $\mathcal{L}$,
2. Call the modified ForwardAccumulation procedure to generate code for computing the sensitivity of the dependent variables with respect to the current common subexpression, and
3. For all vertices contained in $\mathcal{L}$, reset the encountered field to `false` and the active field of its edges to `non-active` (like the reverse mode, this step is proportional to the number of vertices in $\mathcal{L}$ plus the number of edges associated with these vertices).

The precomputed elementary partial derivatives are used in this accumulation sequence. The steps above correspond to reducing the subgraphs above the common subexpression vertices. Performing these steps in the order of decreasing common subexpression rank ensures that the sensitivity vector of every common subexpression encountered in ForwardAccumulation will have had code for its evaluation previously written.

Jacobian Accumulation Phase — Column Evaluation

Once the graph has been reduced in the step above, the full Jacobian can be constructed by evaluating the sensitivity of dependent variable vertices with respect to each independent variable vertex (i.e., constructing the Jacobian column-by-column).

For each independent variable vertex, v_k, $k \in \mathcal{I}$, perform the following steps:

1. Call procedure FowardAccumulationList to generate a list of vertices $\mathcal{L}$,
2. Call the modified ForwardAccumulation procedure to generate code for computing the Jacobian column,
3. For each vertex in $\mathcal{L}$, reset the encountered field to `false` and the active field of its edges to `non-active`.

Like the reverse mode, the order in which the independent variables are processed does not matter.

Applying this approach to equations (7.12)-(7.15) results in the accumulation sequence shown in Figures 7-15, 7-16, and 7-17 contain the accumulation sequence that can be extracted from the CG shown in Figure 7-7.

The Jacobian is accumulated column-by-column with 23 multiplications and 7 additions. In the forward mode of the subgraph reduction approach, trivial multiplications by unity are easily eliminated which accounts for the lower operation count than the other AD approaches examined with this example problem.

Common subexpression v_{16}

$$\partial f_4/\partial v_{16} = d_{17,16}$$

Common subexpression v_{15}

$$\partial f_4/\partial v_{15} = d_{16,15} \cdot \partial f_4/\partial v_{16}$$

Common subexpression v_{10}

$$\begin{aligned} \partial f_2/\partial v_{10} &= d_{11,10} \\ \partial f_3/\partial v_{10} &= d_{13,10} \end{aligned}$$

Common subexpression v_9

$$\begin{aligned} \partial f_2/\partial v_9 &= d_{10,9} \cdot \partial f_2/\partial v_{10} \\ \partial f_3/\partial v_9 &= d_{10,9} \cdot \partial f_3/\partial v_{10} \end{aligned}$$

Common subexpression v_5

$$\partial f_1/\partial v_5 = d_{6,5}$$

Common subexpression v_4

$$\begin{aligned} \partial f_1/\partial v_4 &= d_{5,4} \cdot \partial f_1/\partial v_5 \\ \partial f_2/\partial v_4 &= d_{10,4} \cdot \partial f_2/\partial v_{10} \\ \partial f_3/\partial v_4 &= d_{10,4} \cdot \partial f_3/\partial v_{10} \end{aligned}$$

Figure 7-15: Forward mode subgraph reduction of computational graph shown in Figure 7-7.

7.2.2 Interpreted Implementation

In the previous section, algorithms are presented describing how code can be generated for Jacobian evaluation by extracting the accumulation sequence from the graph. These same algorithms can be applied within an interpretive architecture as well. However, a slightly different implementation of the reverse mode of the subgraph reduction approach has be found more attractive in an interpretive architecture, particularly when applied to the hybrid discrete/continuous simulation problem.

First, suppose there is a symbolic form of the computational graph of a system of equations of interest available. Procedure ReverseMode shown in Figure 7-18 can

First column of Jacobian matrix.

$$\begin{aligned}
\partial f_1/\partial x_1 &= d_{5,3} \cdot \partial f_1/\partial v_5 \\
\partial f_1/\partial x_1 &= \partial f_1/\partial x_1 + d_{4,3} \cdot \partial f_1/\partial v_4 \\
\partial f_2/\partial x_1 &= d_{4,3} \cdot \partial f_2/\partial v_4 \\
\partial f_3/\partial x_1 &= d_{4,3} \cdot \partial f_3/\partial v_4 \\
v_7' &= d_{7,3} \\
v_{11}' &= d_{11,7} \cdot v_7' \\
\partial f_2/\partial x_1 &= \partial f_2/\partial x_1 + v_{11}' \\
v_{12}' &= d_{12,3} \\
v_{13}' &= d_{13,12} \cdot v_{12}' \\
\partial f_3/\partial x_1 &= \partial f_1/\partial x_1 + v_{13}'
\end{aligned}$$

Second column of Jacobian matrix.

$$\begin{aligned}
v_2' &= d_{2,1} \\
\partial f_1/\partial x_2 &= d_{4,1} \cdot \partial f_1/\partial v_4 \\
\partial f_2/\partial x_2 &= d_{4,1} \cdot \partial f_2/\partial v_4 \\
\partial f_3/\partial x_2 &= d_{4,1} \cdot \partial f_3/\partial v_4 \\
v_6' &= d_{6,2} \cdot v_2' \\
\partial f_1/\partial x_2 &= \partial f_1/\partial x_2 + v_6' \\
\partial f_2/\partial x_2 &= \partial f_2/\partial x_2 + d_{9,1} \cdot \partial f_2/\partial v_9 \\
\partial f_3/\partial x_2 &= \partial f_3/\partial x_2 + d_{9,1} \cdot \partial f_3/\partial v_9 \\
\partial f_4/\partial x_2 &= d_{15,1} \cdot \partial f_4/\partial v_{15}
\end{aligned}$$

Figure 7-16: Forward mode subgraph reduction approach applied to the CG shown in Figure 7-7 after subgraph reduction (continued in next figure).

be used to compute the Jacobian of the system of equations row-by-row by calling this procedure from each of the dependent variable vertices. If ReverseMode is called with v initialized to the root vertex of the graph representing $f : \mathbb{R}^n \longrightarrow \mathbb{R}$, $\hat{v}$ initialized to the scalar α, and ∇f initialized to the constant vector g, then the following quantity is computed,

$$g + \alpha \cdot \nabla f.$$

Third column of Jacobian matrix.

$$\begin{aligned}
\partial f_2/\partial x_3 &= d_{9,8} \cdot \partial f_2/\partial v_9 \\
\partial f_3/\partial x_3 &= d_{9,8} \cdot \partial f_3/\partial v_9 \\
v'_{14} &= d_{14,8} \\
\partial f_4/\partial x_3 &= d_{15,8} \cdot \partial f_4/\partial v_{15} \\
v'_{16} &= d_{16,14} \cdot v'_{14} \\
\partial f_4/\partial x_3 &= \partial f_4/\partial x_3 + v'_{16} \cdot \partial f_4/\partial v_{16}
\end{aligned}$$

Figure 7-17: Forward mode subgraph reduction approach applied to the CG shown in Figure 7-7 after subgraph reduction (continued).

Procedure ReverseMode is simply a recursive implementation of the chain-rule:

$$\frac{\partial y}{\partial x_i} = \sum_{P \in \mathcal{P}(x_i, f)} \prod_{e \in P} (\text{elementary partial derivative attached to } e)\,, \tag{7.18}$$

where $\mathcal{P}(x_i, f)$ is the set of all paths connecting the root vertex and independent variable vertex x_i in the computational graph and the elementary partial derivative at edge e, connecting v_i and v_j, is $d_{i,j}$. The products in this formula are performed on lines 7 and 8 of ReverseMode. Here, the run-time stack is employed to hold these quantities as they are being computed. The summation part of the chain-rule formula above is performed on line 4. This recursive implementation has been found to be very efficient when applied in an interpretive architecture. Furthermore, the normal scalar sweep version of the reverse mode requires $\mathcal{O}(|\mathcal{V}|)$ memory to store the vertex adjoints, whereas in this implementation, by employing the run-time stack of the computer, the memory requirements are reduced to $\mathcal{O}(\log_2(|\mathcal{V}|))$ (the depth of the graph) on average[2]. The disadvantage of this approach, however, is that the accumulation below subgraphs rooted at common subexpression vertices will be performed several times. This drawback can be remedied by applying the techniques discussed in this chapter. First, it is assumed that the vertex data structures have an integer rank field and two integer pointers into a real workspace array (this real workspace is used

[2]Of course it is possible to create pathological problems which are simply long chains of unary operations, in which case, the depth of the graph is $\mathcal{O}(|\mathcal{V}|)$.

```
ReverseMode( v, v̂, ∇f )
 1  if v = independent variable x_j then
 2     ▷ Accumulate elementary partial derivative product
 3     ▷ in appropriate position of gradient vector.
 4     ∇f[j] ← ∇f[j] + v̂
 5  elseif v = binary operator then
 6     ▷ Compute adjoint quanities for left and right child vertices.
 7     v̂_left ← v̂ · ∂v/∂left[v]
 8     v̂_right ← v̂ · ∂v/∂right[v]
 9     ▷ Pass adjoints down to child vertices.
10     ReverseMode(left[v],v̂_left,∇f)
11     ReverseMode(right[v],v̂_right,∇f)
12  elseif v = unary operator or intrinsic function then
13     ⋮
14     ▷ Code similar to above.
15     ⋮
16  end
```

Figure 7-18: Recursive version of scalar sweep reverse mode of automatic differentiation.

to hold the sparse gradients of the dependent variables and common subexpressions). As in the compiled implementation of the reverse mode, the Jacobian evaluation is broken down into two steps, a preprocessing phase and an accumulation phase. The preprocessing phase, described below, simply identifies and ranks common subexpressions and allocates the workspace required to hold their sparse gradients. A modified version of ReverseMode is then used to accumulate the equation gradients.

The preprocessing phase can be summarized in the following steps.

1. Initialize the rank field of all vertices in the graph to zero,

2. Call procedure Dfs-Rank (see Figure 7-19) from each dependent variable vertex, v_k, $k \in \mathcal{D}$, to set the vertex rank fields,

3. Identify all common subexpressions and store in a list sorted by decreasing value of rank (see below),

4. For each common subexpression vertex and each dependent variable vertex, determine the independent variable index set (the dimensionality of the index sets is the amount of memory that must be allocated in the real workspace), and

5. Allocate the sparse gradient workspace by setting the workspace pointers of each common subexpression and dependent variable vertex.

```
DFS-RANK(v)
1  if v = independent variable or constant then
2     return ;
3  end
4  rank[v] ← rank[v] + 1 ;
5  DFS-RANK(left[v]) ;
6  DFS-RANK(right[v]) ;
7  end
```

Figure 7-19: Depth-first search (DFS) algorithm for setting the rank field of the graph vertices for subsequent common subexpression identification and ranking.

The children of a vertex are encountered at least as many times as their parents in procedure DFS-RANK. A common subexpression vertex, in this case, is defined as a vertex v such that

$$\text{rank}[v] > \text{rank}[u] \qquad \text{for at least one } u \in \mathcal{P}(v).$$

Furthermore, the value of the rank field gives the relative ordering of the common subexpressions present in the graph. Since a child of a vertex is encountered at least as many times during DFS-RANK as its parents, if common subexpression u is contained in the graph of common subexpression v then $\text{rank}[u] > \text{rank}[v]$, which is precisely the definition of ranking for the reverse mode.

The definition of common subexpression and algorithm for identifying them is different for the compiled version of the reverse mode than the interpretive version

described here. This difference, which results in more common subexpressions for the interpretive version, is due to the way the vertices are encountered during the DFS-based ReverseMode (and the modified version of this routine for the subgraph reduction approach, ModifiedReverseMode, described below). This discrepancy between the definitions of common subexpressions for the compiled and interpretive versions is discussed later in this section.

Let $\mathcal{CS}$ denote the sorted list of common subexpression vertices. The next step is to determine the independent variable index sets of the common subexpressions in $\mathcal{CS}$ and the dependent variable vertices, v_k, $k \in \mathcal{D}$. This occurrence information is used to determine how much space must be allocated in the real workspace array used to hold sparse gradients and required when performing the sparse vector operations performed in the subsequent accumulation step. The complexity of constructing these index sets is $\mathcal{O}(\hat{n}^2 \, |\mathcal{V}|)$ for the same reasoning described in the forward mode compiled implementation. As before, the complexity will be $\mathcal{O}(\hat{n} \, |\mathcal{V}|)$ if only unary and binary operators are present in the graph. The independent variable index sets should be constructed in the following order: common subexpressions in the order of decreasing rank and then dependent variable vertices (this ensures the same part of the graph will only be encountered once during the construction of the index sets). Upon completion of the preprocessing step, we have a sorted list of common subexpression vertices, workspace allocated for the storage of the common subexpression and dependent variable vertex gradients, and the independent variable index sets (occurrence information) required to perform the sparse vector operations. The workspace used to compute the dependent variable gradients should be the actual array used to hold the Jacobian matrix (thereby eliminating the need for copying the computed Jacobian and also reducing the amount of space required for its computation). Furthermore, typical sparse LU decomposition routines (e.g., MA48 [39]) require more real workspace than the amount needed to hold the sparse Jacobian matrix. This additional workspace can be used to store the common subexpression gradients during the Jacobian evaluation and thus, very little additional memory is required for this approach. The modified version of ReverseMode is shown in Figure 7-20. This

is identical to the previous version except for the fact that it is assumed that the common subexpression gradients have been precomputed (i.e., their subgraphs have been reduced) and can be incorporated into the equation gradient via the chain-rule (line 4 of ModifiedReverseMode).

```
ModifiedReverseMode( v, v̂, ∇f)
 1  if v = common subexpression vertex then
 2     ▷ Incorporate gradient of common subexpression vertex, ∇v,
 3     ▷ into ∇f by means of the chain rule.
 4     ∇f ← ∇f + v̂ · ∇v
 5  elseif v = independent variable x_j then
 6     ▷ Accumulate elementary partial derivative product
 7     ▷ in appropriate position of gradient vector.
 8     ∇f[j] ← ∇f[j] + v̂
 9  elseif v = binary operator then
10     ▷ Compute adjoint quantities for left and right child vertices.
11     v̂_left ← v̂ · ∂v/∂left[v]
12     v̂_right ← v̂ · ∂v/∂right[v]
13     ▷ Pass adjoints down to child vertices.
14     ModifiedReverseMode(left[v],v̂_left,∇f)
15     ModifiedReverseMode(right[v],v̂_right,∇f)
16  elseif v = unary operator or intrinsic function then
17     ⋮
18     ▷ Code similar to above.
19     ⋮
20  end
```

Figure 7-20: Recursive version of scalar sweep reverse mode of automatic differentiation with common subexpression vertex elimination.

The Jacobian accumulation step can be summarized as follows.

1. Compute common subexpression gradients in the order of decreasing value of rank by calling ModifiedReverseMode from each vertex in $\mathcal{CS}$ and

2. Compute the dependent variable gradients (the Jacobian rows) by calling ModifiedReverseMode from each dependent variable vertex, v_k, $k \in \mathcal{D}$.

Performing the gradient computations in the order described above ensures that each time a common subexpression is encountered in ModifiedReverseMode, its gradient will have been precomputed.

As stated above, the definition of common subexpression is different for the compiled and interpretive versions of the reverse mode of the subgraph reduction approach. This is due to the fact that by using the DFS-based ModifiedReverseMode and by storing the vertex adjoints in the run-time stack, the accumulation along the edges between a dependent variable vertex and a common subexpression vertex contained in its graph is independent for each path connecting these two vertices (the same holds for common subexpression vertices and higher ranking common subexpression vertices contained in the same subgraph). To illustrate this, consider the fragment of a computational graph shown in Figure 7-21. Suppose that vertices

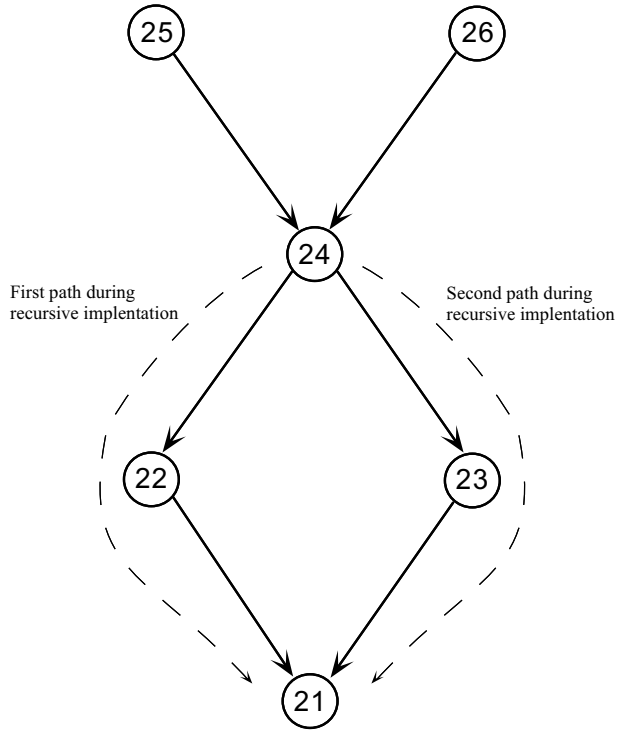

Figure 7-21: Fragment of a computational graph. Vertices v_{25} and v_{26} correspond to dependent variables.

v_{25} and v_{26} correspond to dependent variables and that a complicated graph lies below v_{21}. For both the compiled and interpretive versions of the reverse mode, v_{24} will be

considered a common subexpression. Now suppose we are computing the gradient of common subexpression v_{24} (i.e., reducing this portion of the graph). The following operations would be carried out in the compiled implementation,

$$\begin{aligned}
\hat{v}_{24} &= 1 \\
\hat{v}_{23} &= d_{24,23} \cdot \hat{v}_{24} \\
\hat{v}_{22} &= d_{24,22} \cdot \hat{v}_{24} \\
\hat{v}_{21} &= d_{22,21} \cdot \hat{v}_{22} + d_{23,21} \cdot \hat{v}_{23} \\
&\vdots
\end{aligned}$$

Vertex v_{21} is encountered once and the accumulation below this vertex is carried out in the usual manner. Now consider the evaluation of the gradient of vertex v_{24} using ModifiedReverseMode. The accumulation (products of the elementary partial derivatives) would be carried out along the left path to v_{21} from v_{24} and below then the accumulation would be carried out along the right path to v_{21} from v_{24} and below a second time. Clearly, this is just the redundancy we are trying to avoid. In the interpretive version of the reverse mode of the subgraph reduction approach, vertex v_{21} will also be considered a common subexpression and its gradient will have been precomputed. The disadvantage, however, is that the chain-rule operation will be performed twice at this vertex when evaluating the gradient of common subexpression v_{24}. This can be avoided through the use of lists (in which case, the operations for both the compiled and interpretive implementations of this approach will be the same), however, the additional advantages described below often warrants these occasional redundant operations.

The advantages of this interpretive version of the reverse mode are as follows. First, the special handling of the common subexpressions allows the efficient recursive implementation of the reverse mode to be applied without fear of the accumulation being performed below the common subexpressions multiple times. Second, there is substantial memory savings by storing sparse vectors only at the common subexpres-

sion vertices[3] and the memory required to store the vertex adjoints is $\mathcal{O}(\log_2(|\mathcal{V}|))$. Furthermore, if memory is limited, only a portion of the common subexpressions can be handled, resulting in less optimal, yet still improved, performance. Simple graph analysis can be used to determine which common subexpression should be discarded. In addition, linear equations can be handled very efficiently and in a consistent manner in this approach. Linear equations and linear subgraphs of nonlinear equations can be identified during the common subexpression search at little additional cost. The linear subgraphs are determined in a recursive manner (through a call to a modified DFS-RANK); a subgraph rooted at v is linear if v is an addition operator and both of its children are linear, or v is a multiplication operator and one of its children is a constant and the other is linear (or constant), or v is a function and its argument is a constant, etc. Sparse gradients can be allocated for these linear equations that can be computed once a priori and used for every subsequent Jacobian evaluation. This has been found to yield significant performance improvements for typical process models containing several linear equations as shown in chapter 8. Lastly, but particularly important for Jacobian evaluations required during a hybrid discrete/continuous simulation, is the preprocessing step is very efficient (taking a fraction of a second for problems containing several tens of thousands of equations). In a hybrid discrete/continuous simulation, the functional form of the process model may change many times over the course of a calculation. Efficient preprocessing can have a substantial impact on the overall calculation times in this situation. The disadvantage of this approach (as opposed to the compiled implementation above), however, is that the chain-rule operation will be performed as many times as the common subexpression is encountered (as described above). This leads to some inefficiency if the common subexpression is encountered multiple times during a single gradient calculation (the operations are not redundant if the common subexpression is encountered multiple times during different gradient evaluations). However, in most cases, this inefficiency is more than compensated for by the other advantages of this approach.

[3]In fact, very little (if any) additional memory is required if the workspace used in the LU factorization routines is employed to hold these sparse vectors.

7.3 Analysis of the Subgraph Reduction Approach

In this section, the memory requirements and operation count of the new approaches are analyzed. Let cost$(\cdot)$ and space$(\cdot)$ denote the number of operations required and the amount of memory required, respectively, when computing some quantity. These are different than the asymptotical bounds shown elsewhere in this thesis (denoted by $\mathcal{O}(\cdot)$) in the they represent the exact cost (arithmetic operations, indirect addressing, memory accesses, etc.) and space. The temporal and spatial complexity analysis described below does not take into account the space required to store the computational graph or the cost associated with evaluating the elementary partial derivatives.

7.3.1 Reverse Mode

In the case of the sparse vector implementation of the reverse mode, the following bounds hold [52],

$$\text{space}(\nabla f(x)) \leq \hat{m} \cdot \text{space}(f(x)) \tag{7.19}$$

$$\text{cost}(\nabla f(x)) \leq 3\hat{m} \cdot \text{cost}(f(x)). \tag{7.20}$$

where space$(f(x))$ is $\mathcal{O}(|\mathcal{V}|)$. As stated in chapter 6, both of these bounds are somewhat pessimistic, especially in the case of systems of equations with large, sparse Jacobian matrices.

In the case of the reverse mode implementation of the subgraph reduction approach, the amount of space required to evaluate the Jacobian matrix is

$$\text{space}(\nabla f(x)) \leq \text{space}(f(x)) + \sum_{j=1}^{N_{cs}} \hat{n}_j \hat{s} \tag{7.21}$$

where N_{cs} is the number of common subexpressions in the graph, $\hat{s}$ is the space required to hold a single partial derivative, and $\hat{n}_j$ is the number of independent

variables contained in common subexpression j, the cardinality of the independent variable index set, $\mathcal{O}_i(v_j)$. The summation term accounts for the sparse gradients stored at the common subexpression vertices. If dense vectors are used then $\hat{n}_j$ is replaced by n and if compressed vectors are used then $\hat{n}_j$ is replaced by $\bar{n}$, the number of structurally orthogonal columns in the Jacobian matrix. In the case of the interpretive version, the term space$(f(x))$ (the space required to hold the vertex adjoints) is replaced by a term proportional to the height of the computational graph. In order to derive the operation count bound, consider the three equations represented by the equation graph shown in Figure 7-5 (after subgraph reduction). Equation f_1 is rooted at vertex 21, equation f_2 is rooted at vertex 22, and equation f_3 is rooted at vertex 23. Suppose these equations are evaluated such that common subexpressions are not recomputed. When f_1 is evaluated, values of the subexpressions rooted at vertices 8 and 13 are stored. When f_2 is evaluated, the previously computed value stored at vertex 13 is used. Similarly, the value stored at vertex 8 is used when evaluating f_3. Figure 7-22 shows which vertices are encountered during each equation evaluation.

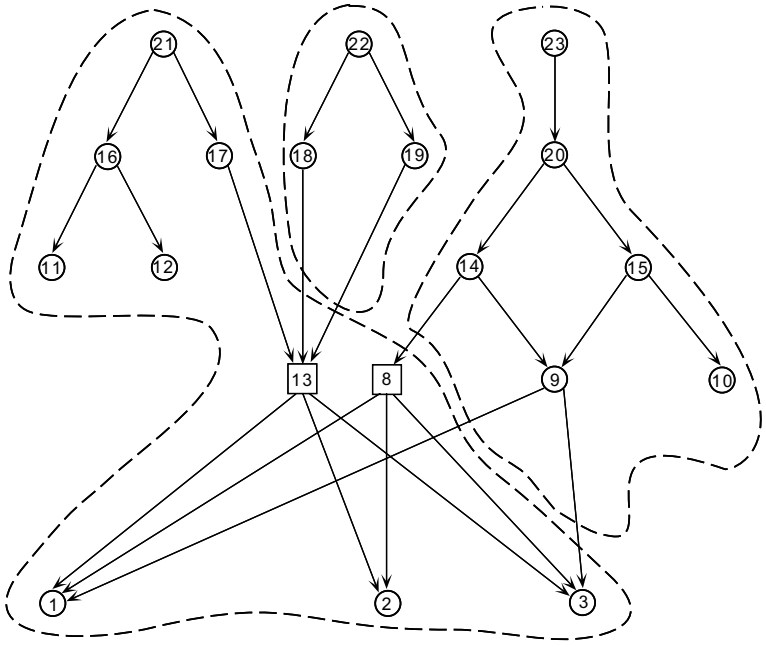

Figure 7-22: Vertices encountered during residual and Jacobian evaluation.

Using the new approach, the cost of evaluating the gradient of f_1, ∇f_1, is bounded

above by 3 times the cost of evaluating f_1 (this includes setting up the new edge values during the subgraph simplification) plus the cost of the chain-rule operation at common subexpression vertex 13 (when a common subexpression is encountered, the precomputed gradient is incorporated into the Jacobian row via the chain-rule – see line 17 of the algorithm for reverse accumulation in the subgraph reduction approach, Figure 7-9). Denote the cost of the chain-rule operation at vertex 13 by C_{13}^{cr}. This cost depends on the particular variation of the reverse mode of the subgraph reduction approach applied (e.g., sparse, compressed, or dense vector representation of the gradients) and the computer environment in which the computations are being performed (e.g., a serial or a vector computer). The cost of evaluating ∇f_2 is bounded above by 3 times the cost of evaluating f_2 (exploiting common subexpressions in the evaluation) plus the cost of the chain-rule operation at vertex 13. Finally, the cost of evaluating ∇f_3 is bounded above by 3 times the cost of evaluating f_3 plus the cost of the chain-rule operation at vertex 8, C_8^{cr}. The cost of evaluating the entire Jacobian is

$$\begin{aligned} \text{cost}(\nabla f(x)) &= \text{cost}(\nabla f_1) + \text{cost}(\nabla f_2) + \text{cost}(\nabla f_3) \qquad (7.22) \\ &\leq 3 \cdot \text{cost}(f_1) + C_{13}^{cr} + 3 \cdot \text{cost}(f_2) + C_{13}^{cr} + 3 \cdot \text{cost}(f_3) + C_8^{cr} \\ &\leq 3 \cdot \text{cost}(f) + \sum_{j=1}^{N_{cs}} \hat{m}_j C_j^{cr} \end{aligned}$$

where $\hat{m}_j$ is the number of equations that contain common subexpression j, that is, the cardiality of the dependent variable index set, $\mathcal{O}_d(v_j)$. If sparse vector operations are performed at the common subexpression vertices, the cost of the chain-ruling at a common subexpression vertex is equal to the number of independent variables contained in the common subexpression multiplied by the cost of the operation $a =$

$a + b * c$ (denote this cost by $\hat{c}$)[4]. Thus, the cost of a Jacobian evaluation is

$$\text{cost}(\nabla f(x)) \leq 3 \cdot \text{cost}(f(x)) + \sum_{j=1}^{N_{cs}} \hat{m}_j \hat{n}_j \hat{c}. \tag{7.23}$$

If the Jacobian evaluation is performed on a vector computer, it may be advantageous to represent the common subexpression gradients as compressed $\bar{n}$ dimensional vectors, in which case, an $\mathcal{O}(1)$ SAXPY call can be employed.

7.3.2 Forward Mode

In the case of the sparse vector implementation of the forward mode, the following bounds hold [52],

$$\text{space}(\nabla f(x)) \leq \hat{n} \cdot \text{space}(f(x)) \tag{7.24}$$

$$\text{cost}(\nabla f(x)) \leq 3\hat{n} \cdot \text{cost}(f(x)). \tag{7.25}$$

Like the sparse reverse mode, these bounds are somewhat pessimistic.

In the case of the forward mode implementation of the new approach, the amount of space required to evaluate the Jacobian matrix is:

$$\text{space}(\nabla f(x)) \leq \text{space}(f(x)) + \sum_{j=1}^{N_{cs}} (\hat{m}_j - 1)\hat{s} \tag{7.26}$$

where N_{cs} is the number of common subexpressions (as defined for the reverse mode) in the graph and $\hat{m}_j$ is the number of dependent variables reachable from common subexpression j (i.e., the cardinality of the dependent variable vertex set, $\mathcal{O}_d(v_j)$). The first term above, space($f(x)$), accounts for the intermediate variable stored at the graph vertices (the w' in Figure 7-14)) and the second term accounts for the

[4]This cost should also include the cost of the indirect addressing of the sparse vectors and all memory accesses required.

additional memory required to hold the common subexpression adjoints. The cost of a Jacobian evaluation is given by:

$$\text{cost}(\nabla f(x)) \leq 3 \cdot \text{cost}(f(x)) + \sum_{j=1}^{N_{cs}^*} \hat{m}_j \hat{n}_j \hat{c} \tag{7.27}$$

where N_{cs}^* is the number of common subexpression vertices defined for the forward mode.

In the remainder of this section, the operation count and the spatial complexity of the sparse vector reverse mode and the reverse mode version of the new approach will be compared. Similar arguments hold for the comparison between the sparse forward mode and the forward mode of the new approach. Suppose an equation graph contains a common subexpression vertex v shared by $\hat{m}_v$ equations. In addition, suppose there are no other common subexpression vertices in the subgraph rooted at v and v is not contained in the subgraph of another common subexpression vertex (v will be referred to as an *isolated* common subexpression). Let $\mathbf{G}(v)$ denote the subgraph rooted at v. The amount of memory used in $\mathbf{G}(v)$ for the sparse vector implementation is $N_v \hat{m}_v \hat{s}$, where N_v is the total number of vertices in $\mathbf{G}(v)$ and $\hat{s}$ is defined above. The amount of memory used in the new approach is $(N_v - 1 + \hat{n}_v)\hat{s}$, where $\hat{n}_v$ is the number of independent variables contained in $\mathbf{G}(v)$. The quantity $N_v - 1$ accounts for the storage of adjoints (this memory can be recovered once the subgraph has been reduced) and $\hat{n}_v$ accounts for the sparse gradient stored at v. Since there are at least $\hat{n}_v$ vertices in $\mathbf{G}(v)$ (the independent variables), $N_v > \hat{n}_v$. In addition, $\hat{m}_v \geq 2$ since v is a common subexpression. Thus, $N_v \hat{m}_v \hat{s} > (N_v - 1 + \hat{n}_v)\hat{s}$. Now suppose an additional common subexpression vertex, u, is added to $\mathbf{G}(v)$. The amount of memory required by the sparse vector reverse mode increases by $N_u \delta\hat{m}_u \hat{s}$ where N_u is the number of vertices in $\mathbf{G}(u)$ and $\delta\hat{m}_u = |\mathcal{O}_d(u) - \mathcal{O}_d(v)|$ is number of additional equations u is common to. The additional memory required by the new approach is $(\hat{n}_u - 1)\hat{s}$ (or zero if $\mathcal{O}_d(u) = \mathcal{O}_d(v)$). Again, $N_u > \hat{n}_v$ and, thus, for $\delta\hat{m}_u > 0$, less additional memory is required for the new approach. However, since $\delta\hat{m}_u$ may

equal zero, it is possible to construct simple computational graphs that require less memory for Jacobian evaluation with the sparse vector reverse mode than with the new approach. However, as the common subexpressions become larger and more complex (where memory usage becomes an issue), the memory savings of the new approach become dramatic. In order to compare the computational performance of the sparse vector reverse mode and the new approach, consider a computational graph with a single isolated common subexpression vertex v which contains $\hat{n}_v$ independent variables and is shared by $\hat{m}_v$ equations. The cost of accumulating the elementary partial derivatives above v is the same for both approaches since this step is simply a series of scalar multiplications and additions. The cost below v is bounded above by $3\hat{m}_v \cdot \text{cost}(v)$ for the sparse vector reverse mode and bounded above by $3 \cdot \text{cost}(v)$ for the new approach. There is an additional cost at v for the new approach due to the chain-rule operation. This cost is $\hat{m}_v\hat{n}_v\hat{c}$, where $\hat{c}$ is the same as defined above.[5] The difference between the two approaches is

$$\begin{aligned} \Delta C &= C_1 - C_2 \\ &= 3\hat{m}_v K \cdot \text{cost}(v) - (\hat{m}_v\hat{n}_v\hat{c} + 3K \cdot \text{cost}(v)) \end{aligned}$$

where C_1 is the cost of accumulating the Jacobian using the sparse vector reverse mode, C_2 is the cost using the new approach, and $0 < K \leq 1$ is some constant that changes the inequality in the bound for the cost below v to an equality. What is important is that the cost below v for the sparse vector reverse mode is precisely $\hat{m}_v$ times greater than the new approach. The cost of the new approach is *less* when

$$\Delta C > 0$$

[5]This is an overestimate since the actual number of additions depends on the occurrence information of the equations containing the common subexpression.

or

$$\text{cost}(v) > \frac{\hat{m}_v \hat{n}_v \hat{c}}{3K(\hat{m}_v - 1)}.$$

For large complicated subexpressions, cost(v) will be large and K will be close to unity. Thus, we can expect the new approach to perform substantially better when the common subexpressions are large and complex (see Figure 8-1). The two approaches perform similarly for simple common subexpressions since both cost(v) and $\hat{n}_v$ are small. Extending this comparison to the forward mode version of the subgraph reduction approach and the sparse vector sweep version of the forward mode is straightforward. By analogy, the forward mode of the subgraph reduction approach will perform better if the subgraphs *above* common subexpression vertices are large and complicated. This analysis provides a metric for deciding how to reduce various parts of the computational graph in a hybrid mode for the subgraph reduction approach; the cost (time and space) can be determined for both the forward and reverse mode at each common subexpression vertex to decide how each portion of the graph should be reduced.

7.4 Conclusions

A new class of automatic differentiation techniques are developed in this thesis. This approach can be applied to both the forward and reverse modes, resulting in sometimes dramatically lower operation counts and memory requirements than other approaches for computational differentiation. This approach requires a graph of the system of equations of interest, which can be readily and efficiently generated from most representations of the system of equations. The algorithms required to generate the accumulation sequence for the Jacobian evaluation are efficient both in terms of temporal and spatial complexity, allowing very large, complex systems of equations to be considered. Furthermore, a simple and efficient graph analysis can be used to

determine which version of our new approach is best suited to a given problem or if a hybrid algorithm may yield beneficial results. Of particular importance is the efficient interpretive implementation of the reverse mode of the subgraph reduction approach. As shown in the following chapter, this modification dramatically reduces the memeory requirements and improves the computational efficiency associated with Jacobian evaluation when applied within an interpretive architecture equation-oriented simulator.

Chapter 8

Numerical Examples

8.1 Comparison of Automatic Differentiation and Symbolic Differentiation

The various approaches for residual and Jacobian evaluation described in chapters 5, 6, and 7 were implemented in the software package ABACUSS[1]. ABACUSS is an equation-oriented process simulator capable of efficient symbolic manipulation of large-scale systems of nonlinear equations. A particularly important feature of ABACUSS is the ability to simulate combined discrete/continuous processes. When simulating such processes, it is convenient to use an interpretive architecture which allows for rapid switching between different functional forms of the process model that occur at discontinuities in the problem. If compiled code were used for residual and Jacobian evaluations, a new residual and Jacobian subroutine would have to be generated each time the system of equations changed at a discontinuity. Alternatively, subroutines for all possible systems of equations could be generated a priori

[1] ABACUSS (Advanced Batch And Continuous Unsteady-State Simulator) process modeling software, a derivative work of gPROMS software, ©1992 by Imperial College of Science, Technology, and Medicine.

resulting in much faster simulation execution. The disadvantage of this approach is the number of possible systems of equations grows exponentially with the number of discontinuities in the problem, making this approach infeasible for all but the most trivial examples. Thus, these various residual/Jacobian evaluation approaches are performed in an interpretive manner. It is important to emphasize, however, that the complexity bounds described in chapters 6 and 7 consider only the number of arithmetic operations and memory accesses required to compute the system of equations and partial derivatives. They do not, however, take into account the various overheads associated with carrying out the calculations in an interpretive environment.

Comparisons of five approaches for residual and Jacobian evaluation are shown below. In approach 1, the Jacobian is computed by symbolic differentiation and the residual and partial derivative expressions are evaluated without taking advantage of common subexpressions. Approach 2 also evaluates the symbolic expressions for the residuals and partial derivatives, however, common subexpressions are exploited in the evaluation. Approach 3 computes the residuals the same way as approach 2, however, the Jacobian matrix is evaluated using the recursive reverse mode of automatic differentiation shown in Figure 7-18. The Jacobian is constructed row by row from each dependent variable node. Approach 4 uses the recursive implementation of the subgraph reduction approach of automatic differentiation (see Figure 7-20) with special handling of linear equations. Finally, approach 5 uses the sparse vector implementation of the reverse mode. All numerical calculations shown below were performed on a Hewlett-Packard 9000/735 workstation. Table 8.1 contains a summary of the various approaches used.

The various approaches described above are used to evaluate the residuals and Jacobian matrix of thirteen example problems. Table 8.2 contains the dimensions of these systems of equations and the memory required to store the symbolic system of equations and Jacobian. The second through sixth columns of this table contain the dimensions of the systems of equations: n is the number of variables, m is the total number of equations, m_{linear} is the number of equations that are linear, nz is the number of entries in the Jacobian matrix that are not identically zero, and n_{cs} is the

Table 8.1: Description of computational differentiation methods tested.

Approach	Residual Evaluation Method	Jacobian Evaluation Method
1	Interpreted graph, no exploitation of common subexpressions.	Symbolic expressions, no exploitation of common subexpressions.
2	Interpreted graph, with exploitation of common subexpressions.	Symbolic expressions, with exploitation of common subexpressions.
3	Interpreted graph, with exploitation of common subexpressions.	Basic recursive scalar sweep reverse mode.
4	Interpreted graph, with exploitation of common subexpressions.	Recursive reverse mode of subgraph reduction approach with handling of linear equations.
5	Interpreted graph, with exploitation of common subexpressions.	Sparse vector sweep reverse mode.

number of common subexpressions present in the system of equations. The remaining two columns contain the number of vertices in the graph representing the system of equations and the partial derivative information. The first two approaches are based on symbolically generated derivatives and, thus, require the same number of vertices to represent the system of equations and Jacobian. Similarly, the remaining three approaches are all based on the computational graph representation of a system of equations and require the same number of vertices.

Table 8.3 shows the performance of the various approaches. Columns 2 through 6 contain the time, in microseconds, for a residual and Jacobian evaluation. Table 8.4 contains the ratio of the time for a Jacobian evaluation to the time for a residual evaluation.

The first four systems of equations in the Tables 8.2 and 8.3 are contrived highly nonlinear equations illustrating the performance capable with the reverse mode of automatic differentiation applied in a symbolic environment. The equations are shown in Appendix F. The first problem consists of a single highly nonlinear equation without any common subexpressions. In this problem, there are no linear equations or

Table 8.2: Dimensions of the systems of equations examined and memory allocated for each of the various graphs.

	System dimensions					Total number of vertices	
System	n	m	m_{linear}	nz	n_{cs}	Symbolic expressions	Computational graph
Pathological 1	3	1	0	3	0	87	37
Pathological 2	20	10	0	110	2	4400	242
Pathological 3	154	150	0	12,950	4	55,000	8,900
Pathological 4	602	100	0	600	3	18,000	4,800
UNIQUAC	171	160	10	1,280	5	9,700	3,400
NRTL	341	330	100	1,910	2	12,000	5,300
Bubble Point	171	171	12	1,271	5	10,000	3,600
Flash Vessel	494	494	43	3,245	14	24,000	9,200
Helmholtz	12	7	6	48	0	404	217
Valve	40	40	25	111	0	356	204
Weir	6	6	4	11	0	62	41
PSA	3,614	3,614	810	18,616	18	233,000	81,000
Batch Column	2,435	2,435	1,269	11,060	833	114,000	20,000

common subexpressions which is why approach 4 takes same amount of time as approach 3. The performance of the subgraph reduction reverse mode (approach 4) is particularly dramatic in problems 2 and 3 which contain several highly nonlinear equations with complicated common subexpressions. The overhead of operating in an interpretive environment is particularly evidenced by the poor performance of the sparse vector reverse mode (approach 5) for this problem. The remaining nine systems of equations are common chemical engineering examples. *UNIQUAC* and *NRTL* are activity coefficient models which compute activity coefficients for a ten component system. *Bubble Point* computes the bubble point temperature of a ten component system using the UNIQUAC model for the liquid phase activity coefficients. *Flash Vessel* models the operation of a flash vessel for a ten component system. Liquid phase nonideality is modelled with the UNIQUAC model. *Helmholtz* computes the Helmholtz free energy of a five component system. *Valve* is a set of equations describing the flow through a buffer tank that has a linear inlet valve and an equal-percentage outlet valve. *Weir* models the filling of a tank with a weir. *PSA* is a partial differential equation model of a pressure swing adsorption system containing two columns.

Table 8.3: Timing for residual and Jacobian evaluation.

	Timings in microseconds				
System	1	2	3	4	5
Pathological 1	196	102	81	81	121
Pathological 2	52,150	7,805	3,750	1,130	4,710
Pathological 3	238,692	130,769	92,769	68,615	156,615
Pathological 4	45,150	26,700	17,150	16,650	24,050
UNIQUAC	53,500	31,167	20,167	20,000	32,833
NRTL	33,040	22,400	12,000	11,680	19,200
Bubble Point	32,417	22,500	12,583	12,167	19,750
Flash Vessel	85,250	58,250	36,500	35,750	56,000
Helmholtz	902	454	462	436	752
Valve	735	615	684	496	786
Weir	70	62	60	55	80
PSA	442,000	336,000	268,000	264,000	378,000
Batch Column	434,000	270,000	188,000	180,000	202,000

Backward finite differences are used for the spatial discretization. Finally, *Batch Column* is an index 2 DAE (differential-algebraic equation) formulation of a five tray batch distillation column separating five components. Liquid phase activity coefficients are computed using the UNIQUAC model. Before the residuals and Jacobian matrix were evaluated, the index was reduced to 1 by differentiating a subset of the algebraic equations. This index reduction results in new equations that share common subexpressions with the equations in the original index 2 formulation. In this problem, the common subexpressions are, however, relatively simple and exploiting them does not significantly reduce the cost of the Jacobian evaluation as shown by the difference between the timings for approaches 3 and 4. In all cases, the subgraph reduction reverse mode of automatic differentiation with special handling of linear equations and common subexpressions performs significantly better than the other evaluation methods. Since typical chemical engineering problems contain a significant number of linear equations (mass and energy balances, summation of mole fractions, etc.), it is important to handle these equations properly. This is particularly evident in the *Valve* and *Weir* example problems. In these cases, approaches 3 and 5 do not perform as well as approach 2. The linear equation and common subexpression

Table 8.4: Ratio of time for Jacobian evaluation to time for a residual evaluation.

	cost$\{\nabla f\}$/cost$\{f\}$				
System	1	2	3	4	5
Pathological 1	5.64	2.38	1.61	1.61	3.00
Pathological 2	20.37	24.59	11.30	2.71	14.70
Pathological 3	5.48	6.20	4.09	2.76	7.48
Pathological 4	5.14	3.41	1.83	1.73	2.98
UNIQUAC	7.64	3.16	1.75	1.67	3.48
NRTL	7.38	4.96	2.19	2.04	4.11
Bubble Point	5.95	4.51	2.08	1.98	3.84
Flash Vessel	5.32	3.66	1.92	1.80	3.39
Helmholtz	3.90	1.43	1.46	1.32	3.00
Valve	1.87	1.57	1.86	1.07	2.29
Weir	1.54	1.50	1.40	1.00	2.20
PSA	3.33	2.43	1.75	1.69	2.86
Batch Column	5.29	2.46	1.41	1.37	1.66

analysis is very inexpensive and can be performed easily in a symbolic environment. Finally, note that the ratio of the time for a Jacobian evaluation to the time for a residual evaluation, shown in Table 8.4, are for most cases well below the upper bound given for the approaches tested. This is typical for sparse problems. The ratios for approaches 3 and 4 are the similar to those expected from compiled code. The other ratios in this table are higher than that possible in compiled code due to overhead of working in an interpretive environment.

The comparisons shown above illustrate the superior performance of automatic differentiation compared to the other evaluation methods performed on symbolic expressions. The typical chemical engineer, however, is often more concerned about calculations other than simple residual and Jacobian evaluations. The next example compares the approaches above when they are applied to evaluate residuals and Jacobians during the solution of a DAE and its parametric sensitivity equations. Maly and Petzold [73] have developed a new exact algorithm for dynamic sensitivity calculations that avoids the need to factor the Jacobian matrix defining the sensitivity equations at each integration step. This advance offers significant speed improvements over existing methods [69], which require a Jacobian factorization at each step

to calculate accurate sensitivities [65] (the original method of [69] effectively solves a perturbed sensitivity system, the solution of which can not be guaranteed to lie within some distance of the true sensitivity system). Even though the method of Maly and Petzold does not require a Jacobian factorization at each step, analytic evaluation of the sensitivity equations (i.e., their residuals) will require one or more Jacobian *evaluations* per step (although this number is unrelated to the number of parameters for which sensitivities are required). Hence, while this new method is more efficient, we have observed that for realistically sized problems (100-60,000 equations), the repeated Jacobian evaluations at each step make up a significant fraction of the overall computational costs. This example thus shows the significant improvements in overall solution time that the methods introduced in chapters 6 and 7 can yield.

The problem examined is the pressurization/blowdown operation of a two column pressure swing adsorption system (PSA in Table 8.2). The calculations are performed using ABACUSS and DSL48S [42, 43], which is a modified form of DASSL [16] with the sparse linear algebra solver MA48 [39] embedded. The first example shown in Table 8.5 determines the sensitivity to the temperature in one column. The second example determines the sensitivity to the temperatures in each of the columns. Each of these examples determine the sensitivities during a thirty second operation. The second through fifth columns of Table 8.5 contain the number of integration steps, residual evaluations, Jacobian factorizations, and sensitivity residual evaluations, respectively, required for the calculation. Finally, the remaining four columns contain the time, in seconds, required for the sensitivity calculation. In this example, only approaches 1, 2, 3, and 4 are used to compute the residuals and Jacobian.

Table 8.5: Sensitivity calculation results.

	Sensitivity calculation statistics				Timings in seconds			
Number of parameters	Steps	Resid. Eval.	Jacobian Fact.	Sensitivity Resid. Eval.	1	2	3	4
1	195	453	35	906	293.26	222.14	193.18	189.26
2	190	441	32	1,323	306.21	234.52	207.44	201.74

As expected, approach 4 performs significantly better than the other evaluation

methods. This example shows the importance of evaluating residuals and Jacobians efficiently. A Jacobian evaluation is required for each Jacobian factorization and each sensitivity residual evaluation. When these evaluations are performed by interpreting symbolic expressions, the cost of this evaluation may be significant compared to the cost of factorization, which is typically considered to dominate the cost of a numerical calculation. However, the empirically observed complexity of factorizations is nearly linear, rather than the much more pessimistic asymptotic analysis.

8.2 Comparison of Subgraph Reduction and Sparse Vector Approaches

Several example problems are compared to illustrate the performance of the various approaches described in this thesis. The first problem, PSA, is a model of a pressure swing adsorption system also examined in the previous section. Backward finite difference spatial discretization of this problem converts the PDE into a set of differential/algebraic equations (DAEs). This problem was tested for several refinements of the discretization. UNIQUAC is an activity coefficient model which computes the activity coefficients for a fifty component system. The number of equations is much greater than fifty due to the large number of intermediate variables required for the calculation. The last problem, BATCHCOLUMN, is an index-2 model of a batch distillation column. An equivalent index-1 model is generated automatically through a series of differentiations, increasing the number of common subexpressions present in the problem. Tables 8.7 and 8.8 contain a summary of the problem dimensions, memory requirements, and operation counts for four approaches (summarized in Table 8.6: the reverse mode of our new approach (subgraph reduction reverse mode or SRRM), the sparse vector reverse mode (SRM), the forward mode of our new approach (subgraph reduction forward mode or SRFM), and the sparse vector forward mode (SFM).

Table 8.6: Summary of approaches tested.

Approach	Description
SRRM	Subgraph reduction, reverse mode
SRM	Sparse vector sweep reverse mode
SRFM	Subgraph reduction, forward mode
SFM	Sparse vector sweep forward mode

Table 8.7 contains the dimensions of the various systems of equations examined and the amount of memory required to hold the vertex gradients and adjoints. In this table, n is the number of variables, m is the number of equations, nz is the number of entries in the Jacobian matrix that are not identically zero, c_{cs} is the number of common subexpressions (as defined for the reverse mode) in the equation graph, and the remaining four columns contain the total number of entries required to hold the vertex gradients and adjoints. Table 8.8 contains the number of operations required to accumulate the entire Jacobian using the various approaches. The numbers given in this table are the number of multiplications and additions shown as an ordered pair: (multiplies,adds). These numbers do not include the cost of evaluating the elementary partial derivatives since this cost is the same for each approach. The total time for performing the graph analysis steps described in chapter 7 was less than 0.30 seconds on an HP 735 workstation. Linear equations were not exploited (by precomputing their constant gradients a priori) in any of these examples.

Table 8.7: Dimensions of the systems of equations examined and memory allocated for each of the various graphs.

	System dimensions				Total number of entries			
System	n	m	nz	n_{cs}	SRRM	SRM	SRFM	SFM
PSA, N=100	3,018	3,018	15,607	20	35,081	49,829	48,839	85,255
PSA, N=200	6,018	6,018	31,207	20	68,461	99,629	97,639	170,455
PSA, N=500	15,018	15,018	78,007	20	175,081	249,029	244,039	426,055
PSA, N=700	21,018	21,018	109,207	20	245,081	348,629	341,639	596,455
UNIQUAC	2,851	2,800	30,400	5	37,397	86,499	67,340	442,670
BATCHCOLUMN	13,837	13,837	32,863	2,197	83,193	94,231	90,966	138,568

As shown in Tables 8.7 and 8.8, our new approaches require less memory and perform fewer operations than the other approaches. In the case of the PSA problem, the operations count of the SRFM approach grows closer to that of the SRRM

Table 8.8: Number of operations during Jacobian accumulation (multiplies,adds).

System	SRRM	SRM	SRFM	SFM
PSA, N=100	(53472,5802)	(58626,5802)	(47621,5998)	(88048,5802)
PSA, N=200	(96867,11602)	(107221,11602)	(95221,11998)	(149311,14022)
PSA, N=500	(242067,29002)	(268021,29002)	(238021,29998)	(373111,35022)
PSA, N=700	(338867,40602)	(375221,40602)	(333221,41998)	(522311,49022)
UNIQUAC	(64682,5450)	(83599,5450)	(83599,5450)	(424168,5450)
BATCHCOLUMN	(65913,2817)	(70400,5214)	(61624,4194)	(95747,4353)

approach. This is due to the fact that the common subexpression size does not grow dramatically with the number of edges pointing to the common subexpression. Since the cost of the SRFM approach increases with the number of edges pointing to the common subexpression, the savings become less dramatic. Although the operation counts for each of the approaches described above are somewhat similar, measuring only arithmetic operations does not take into account a distinct advantage of the subgraph reduction approaches, namely, limiting the sparse vector operations to only the common subexpression vertices. The remainder of this section presents a small example where the operation counts between the various approaches are dramatically different.

As described in the previous section, the new version of the reverse mode should perform better than the others for problems with large common subexpressions. The next example is the simple system of equations:

$$x_j + (\sum_{i=1}^{n} x_i)x_j \quad j = 1, \ldots, n. \tag{8.1}$$

The subgraph of the common subexpression vertex in this system, $\sum_{i=1}^{n} x_i$, grows with the size of the problem. The four approaches are applied to this problem for different values of n. Figure 8-1 contains a plot of the number of multiplies required during the Jacobian accumulation (not counting elementary partial derivative evaluation) versus the dimension of the system.

As shown in Figure 8-1, the reverse mode version of the new approach performs significantly better than the other approaches as the size of the common subexpression increases.

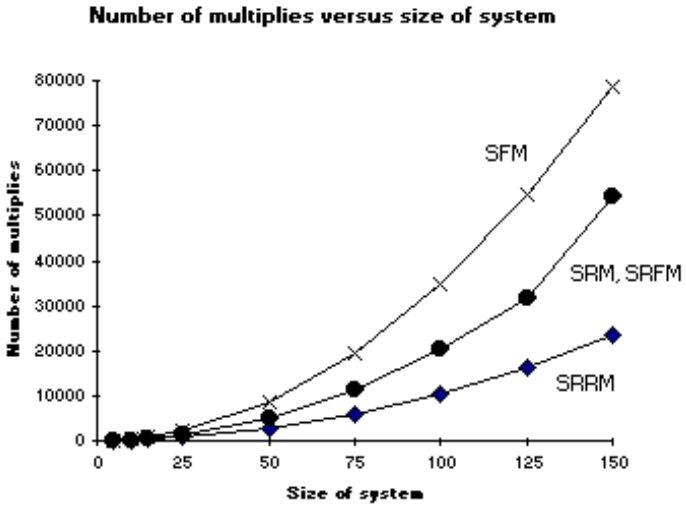

Figure 8-1: Number of multiplies versus system size for system (8.1).

Part III

A Homotopy Approach for Nonconvex Nonlinear Optimization

Chapter 9

A Homotopy Approach for Nonconvex Nonlinear Optimization

The rapid progress being made in the speed and memory of computer resources is allowing mathematical models describing chemical processes to become increasingly complex. The advent of such models requires that more sophisticated algorithms be employed in order to obtain a solution. The poor performance of the locally convergent algorithms used in current process simulators when applied to these complex problems warrants, in many cases, the use of the more powerful, globally convergent algorithms. Highly nonlinear mathematical models also pose a particular a problem to current optimization methods. Experience has shown that state-of-the-art optimization algorithms, such as SQP [53, 91] and the MINOS/Augmented package [82], have difficulty converging highly nonlinear problems. The problem is further complicated when nonlinear inequality constraints define nonconvex feasible regions. In addition to convergence problems, multiple local optima are generally a result of highly nonlinear objective functions and constraints. The approach described in this part of the thesis is motivated by the following facts: (1) increasingly complex models are being used to describe chemical process unit operations more accurately, (2) the improved disturbance rejection capabilities of modern control strategies is making it possible to

operate a chemical process closer to (and possibly within) complex solution regimes, and (3) there is now a widespread use of equation-oriented process simulators in the chemical process industries. These simulators generally have the symbolic form of the set of equations describing a flowsheet model explicitly available to manipulate. This opens the door to new approaches to flowsheet optimization not possible when current optimization algorithms were being developed. This chapter discusses an alternative approach to nonlinear optimization. In this approach, the necessary conditions for local optimality of a nonlinear optimization problem are reformulated as a system of nonlinear equations and solved using the 'globally convergent' homotopy continuation method. Theoretically, this approach has the following benefits: (1) homotopy continuation methods can locate solutions where other locally convergent methods fail, (2) there is a possibility of locating multiple points satisfying the necessary condition for local optimality, and (3) the system of equations remains sparse. Unfortunately, several numerical problems were encountered when this method was applied to the large problems for which it was intended. This chapter describes the approach mentioned above for nonlinear optimization, presents some small example problems, and concludes with a discussion as to why the method failed for the large-scale systems of interest.

9.1 Introduction

Over the past several decades, process modeling technologies have been proven an invaluable tool in the development and optimization of chemical processes. Originally, modular simulators dominated this technology, however, equation-oriented simulators are increasingly being used due to their flexibility in defining a model and their superior ability over modular simulators to solve optimization and dynamic problems (see chapter 5). A recent trend seen in equation-oriented simulators is the use of an interpretive architecture [9] [89]. Process simulators based on an interpretive architecture maintains the symbolic form of the process model in computer memory

where it is available to manipulate throughout the solution sequence. This feature allows for the exploration of combined symbolic and numeric algorithms, algorithms which combine numerical aspects with symbolic manipulation for constructing the problem and/or enhancing numerical calculations.

As chemical companies and process contractors strive to remain competitive, the flowsheet models employed while designing a process or studying an existing one are becoming increasingly detailed, accurate, and complex. The more accurately a simulation reflects the actual process, less pilot plant information is required and better cost estimates may be given to the customer. The impact of this is that models are becoming increasingly difficult to solve even as computer architecture continues to improve. In addition, even when a more accurate model is employed, nonlinear programs are constrained to avoid complex solution regimes, possibly preventing operation at "better" steady-states. At one time, this was acceptable because the PID controllers used were generally not capable of controlling within these possibly unstable regimes. However, as control strategies become more advanced (for example the model predictive controller) the improved disturbance rejection may make operation within such regimes realizable. The complex solution spaces of these models are characterized by hysteresis, multiple steady-state solutions (stable and unstable), and possibly periodic or chaotic behavior. The classic example of multiple steady-state solutions in process models is the exothermic reaction. Consider the simple exothermic series reaction,

$$A \longrightarrow B \longrightarrow C, \tag{9.1}$$

carried out in a nonadiabatic CSTR. Farr and Aris [41] identified up to seven steady-state solutions and twenty-three solution diagrams. In addition to exothermic reactions, autocatalytic, enzyme, and aerobic fermentation reactions typically exhibit multiple solutions and hysteresis under some operating conditions. However, reactors are not the only unit operation that exhibit this complex behavior. Intricate separation schemes are also known to have multiple steady-state solutions. For example,

interlinked distillation systems have been shown to separate ternary mixtures more efficiently than multiple, separate distillation columns [107, 110]. By applying homotopy continuation, Chavez *et al.*determined multiple steady-state solutions for three different interlinked column configurations [24]. As described in chapter 1, heterogeneous azeotropic distillation columns also exhibit multiple steady-states. In the work of Kovach and Seider [62] and Widagdo *et al.*[120], multiple solutions were found as a second liquid phase was added to the trays. In addition, it was necessary to employ homotopy continuation to converge the model near limit points.

Supercritical extractors are another example of a unit operation exhibiting a complex solution space. The fluid in this process is near its critical point and thus small variations in operating conditions may cause abrupt phase change. In addition, multiple solutions may exist when operating within the two phase region. Cygnarowicz and Seider studied the extraction of acetone from water with supercritical CO_2 [32]. For high solvent/feed ratios at a given pressure (greater than the critical pressure of CO_2) two solutions were obtained, corresponding to local and global minima of the utility cost. Nonlinearities arise not only in unit operations, but also in flowsheet integrated to achieve greater thermodynamic efficiency [99].

Homotopy continuation techniques have been used to determine solutions to systems of equations where Newton or quasi-Newton methods fail, either due to poor starting guesses or singularities. Although homotopy continuation methods can be traced back to 1934 in the work of Lahaye, they have only been applied to chemical engineering problems within the last decade. While homotopy continuation techniques are generally more computationally expensive than the locally convergent Newton or quasi-Newton methods, they nevertheless have been successfully applied to relatively large problems. In the calculations involving the interlinked distillation systems described above [24], three-hundred and fifty equations were solved (fifty percent of the equations were nonlinear and contained transcendental terms). Book [56] used homotopy continuation to solve the Williams-Otto plant model, containing five units, nine streams, and six species. The nonlinear partition of this model involved 85 equations. Book also used homotopy continuation to simulate a distillation column model, where

the nonlinear portion consisted of nine-hundred and seventeen equations. Homotopy continuation has also been applied to parametric optimization. Vasudevan *et al.*used artificial parameter homotopy continuation to connect a known KKT point of a nonlinear program with one set of parameters to an unknown KKT point of the same problem but with a second set of parameters [114]. The parameters of interest in the objective function and constraints are replaced by:

$$c_i = \lambda b_i + (1 - \lambda) a_i \qquad i = 1, \ldots, n_p$$

where n_p is the number of parameters being considered, a_i is the value of parameter i in the known problem, and b_i is the value of parameter i in the desired problem. The KKT necessary conditions are converted to a system of nonlinear equations using the same procedure as described in section 9.3. Homotopy continuation is applied to this system of equations, allowing λ to vary from zero (the original problem) to unity (the desired problem).

Equation-oriented process simulation technology has steadily improved over the last several years. As these simulators grow in popularity, they are being used to model increasingly complex processes such as those described above. Optimization problems on flowsheets containing these complex unit operations generally contain highly nonlinear constraints and may contain multiple local optima. One example is the minimization of the utility costs for a supercritial extraction process [32]. Successive quadratic programming (SQP) was applied to a model of this flowsheet from several different initial guesses and two minima were found. The multiple optima were due to the retrograde effect of the supercritical fluid. In addition to the occurrence of multiple optima, supercritical extractors operate near the critical point of a substance. Applying Newton's method near the critical point results in many convergence failures.

The topic of this part of the thesis is nonlinear optimization. The following section contains a description of some current optimization techniques. This is followed by an alternative strategy based on homotopy continuation. This approach will be shown

to be related to SQP. This part of the thesis will be concluded with some examples of this approach as well as some pitfalls encountered when applying it to large-scale systems.

9.2 General Strategies for Nonlinear Optimization

Consider the general nonlinear programming problem:

$$\begin{aligned} \text{(NLP)} \qquad & \text{minimize} \qquad f(x) && && (9.2) \\ & \text{subject to} \qquad h(x) && = \; 0 && (9.3) \\ & \qquad\qquad\quad\; g(x) && \geq \; 0 && (9.4) \end{aligned}$$

where $x \in \mathbb{R}^n$, $f : \mathbb{R}^n \longrightarrow \mathbb{R}$, $h : \mathbb{R}^n \longrightarrow \mathbb{R}^m$ $(m < n)$, and $g : \mathbb{R}^n \longrightarrow \mathbb{R}^p$. Here f is the objective function and h and g are the equality and inequality constraints, respectively. For a typical process flowsheet optimization problem, h includes the MESH (Material balance, Equilibrium, Summation of mole/mass fractions, and Heat balance) equations, design constraints, and any other equations required to model the flowsheet, and g represents the allowable limits for the operation of the process. A typical flowsheet optimization problem is large and sparse. Unlike the sparse banded matrices obtained from the discretization of a differential operator, the sparsity pattern of a flowsheet model does not show any structure. The objective function is usually economic and the constraints are generally highly nonlinear. In addition, the constraints generally contain several non-smooth functions. Two types of non-smooth functions occur in flowsheet models: those with non-smoothness due to actual physical phenomena and those in which the non-smoothness is due to modeling abstractions. Some examples of the former type are: the heat of vaporization of a compound at the critical temperature and phase and flow transitions. An example of the second type is physical property models. In general, physical property models contain a certain amount of empiricism and are only valid within a certain range of state variables;

thus, the functions are usually piecewise continuous. The inequality constraints keep the process within an acceptable operating range. For example, pressures must be bounded away from the design pressure of vessels, compositions must be bound away from flammable or explosive regions, and product composition must satisfy purity specifications.

Subject to a suitable constraint qualification, the first-order necessary condition for a candidate point to be a local optimum of NLP are the Karush-Kuhn-Tucker (KKT) conditions. For the nonlinear programming problem above, the KKT conditions are:

$$\nabla f(x^*) - \nabla h(x^*)^T \nu^* - \nabla g(x^*)^T u^* = 0 \tag{9.5}$$

$$h(x^*) = 0 \tag{9.6}$$

$$(u^*)^T g(x^*) = 0 \tag{9.7}$$

$$g(x^*) \geq 0 \tag{9.8}$$

$$u^* \geq 0 \tag{9.9}$$

where u and ν are the KKT multipliers. Conditions (9.7) through (9.9) are the *complementarity conditions*.

The first optimization strategy considered is the *feasible-path* approach. Currently, a popular feasible-path algorithm is the *generalized reduced gradient method* [1]. The generalized reduced gradient method belongs to a class of nonlinear optimization techniques known as Newton-type approaches. In these approaches, gradient information is used to generate a sequence of descent directions such that the objective function is lowered at each iteration. The algorithm is terminated (hopefully at a minimum) when the search direction vector has a sufficiently small magnitude. In the feasible-path strategy, the equality and inequality constraints are satisfied at each iteration. Thus, in the case of flowsheet optimization, the entire flowsheet model must be converged at every iteration. The inefficiency of this requirement has led to the next strategy considered, the *infeasible-path approach*. The previous approach

required the equality and inequality constraints to be satisfied at each iteration of the optimization algorithm. In the infeasible-path approach, the equality constraints are converged simultaneously with the optimization problem. It is still important to satisfy the inequality constraints at each iteration since they may be formulated so that the problem avoids singularities that may be present. Two popular infeasible-path strategies are the SQP (Successive Quadratic Programming) [53, 91] and MINOS/Augmented package [82]. These methods are discussed below.

9.2.1 Successive Quadratic Programming

SQP has been successfully applied to many optimization problems. In this approach, a series of quadratic optimization subproblems are solved to obtain search (or descent) directions. Consider the nonlinear programming problem above. The QP (Quadratic Program) subproblems are of the form:

$$\begin{aligned}
\text{(QP)} \qquad \text{minimize}_d \qquad & f(x^k) + \nabla f(x^k)^T d + \frac{1}{2} d^T \nabla_{xx} \mathcal{L}(x^k, u^k, \nu^k) d && (9.10)\\
\text{subject to} \qquad & h(x^k) + \nabla h(x^k) d = 0 && (9.11)\\
& g(x^k) + \nabla g(x^k) d \geq 0 && (9.12)
\end{aligned}$$

where $\nabla_{xx} \mathcal{L}(x^k, u^k, \nu^k)$ is the Hessian of the Lagrangian. The Lagrangian is given by:

$$\mathcal{L}(x, u, \nu) = f(x) - \nu^T h(x) - u^T g(x). \tag{9.13}$$

Using the Hessian of the Lagrangian in the objective function gives some additional information about the curvature of the constraints that the normal second-order Taylor series expansion of $f(x)$ does not provide. In most implementations of SQP, a positive definite approximation of the Hessian of the Lagrangian is employed. This approximation, updated during the iterative process using, for example, the BFGS (Broyden, Fletcher, Goldfarb, and Shanno) update formula, approaches the actual

Hessian of the Lagrangian as the algorithm converges to a solution. Using this approximation has the following advantages: (1) the positive definite matrix ensures the QP subproblem is well-posed and (2) the approximation formula does not required second-order partial derivatives. In the case of a modular simulator, these partial derivatives would, in general, be evaluated using finite differences. This requirement would be prohibitively expensive.

The search direction obtained by solving this QP subproblem is the same as the direction obtained by applying Newton's method to the KKT necessary conditions (considering only the active inequality constraints and their multipliers). Similar to Newton's method for solving systems of nonlinear equations, this method will not converge unless the algorithm is initialized sufficiently close to the solution. To achieve global convergence, a merit function or trust region strategy may be employed. One example of a merit function is the l_1-*penalty function*:

$$\psi(x;\mu) = f(x) + \mu \left[\sum_{i=1}^{p} \max\{0, g_i(x)\} + \sum_{i=1}^{m} |h_i(x)| \right]$$

where the parameter μ is set to a value greater than the largest (absolute value) KKT multiplier. At each iteration, a stepsize, α, is chosen such that $\psi(x + \alpha d; \mu)$ is minimized. The SQP algorithm has been shown to require relatively few function evaluations compared to other techniques, making it ideal for modular simulators, where function evaluations are expensive. Let x^* denote a solution of the original nonlinear program. If x^* is a regular KKT point, (x^*, u^*, ν^*) satisfies the second-order sufficiency conditions (the projection of the Hessian of the Lagrangian onto the null-space of the active constraints is positive definite), and the algorithm is initialized sufficiently close to the solution, superlinear convergence is obtained [10].

One problem associated with the SQP method is that the QP subproblem may not have a feasible region, and hence no solution, if the linearized constraints are inconsistent at any particular iteration. For example, linearizing the constraint $x_1^2 + x_2^2 = 1$ at the origin results in the inconsistent requirement $-1 = 0$. In addition, nonconvex

constraints not satisfied at the current iterate may also cause this to happen. This problem can be avoided by relaxing the constraints, thereby creating an artificial feasible region, allowing the algorithm to continue [91, 112, 31]. The QP subproblem is reformulated as:

$$
\begin{aligned}
\text{(QP)} \qquad \text{minimize}_d \quad & f(x^k) + \nabla f(x^k)^T d + \frac{1}{2} d^T \nabla_{xx} \mathcal{L}(x^k, u^k, \nu^k) d + \\
& M(\xi + \xi^2/2) && (9.14) \\
\text{subject to} \quad & h(x^k)(1-\xi) + \nabla h(x^k) d = 0 && (9.15) \\
& g(x^k)(1-\xi) + \nabla g(x^k) d \geq 0 && (9.16) \\
& \xi \geq 0 && (9.17)
\end{aligned}
$$

where M is a large positive number. If $\xi = 0$, the original QP subproblem is feasible. For $0 < \xi \leq 1$, the search direction will cause the constraints to be violated and it will be necessary to perform some correction for the violated constraints after the step is taken in the SQP iteration. Finally, if $\xi = 1$ and $d = 0$, it will be necessary to reinitialize the SQP algorithm. This relaxation technique has no rigorous mathematical justification and is subject to failure for some problems. As the size of the nonlinear program increases, the QP subproblem becomes computationally very expensive. Three approaches for improving the efficiency of the QP are given below.

In situations where obtaining an exact Hessian of the Lagrangian matrix is expensive, the approximation described above is generally employed. However, there are two major problems associated with the BFGS update, B_k: (1) it may become ill-conditioned and (2) it is dense (even though $\nabla_{xx}\mathcal{L}$ may be sparse). The first problem is corrected by scaling the variables when necessary. Biegler and Cuthrell (1985) determined when scaling was necessary by monitoring the condition number of B_k. The second problem, which limits the size of problems to less than about 1000 variables, may be mitigated by using decomposition techniques [71, 113]. The decomposition approach is based on the observation that in many flowsheet optimization problems, the number of degrees of freedom $(n - m)$ is relatively small compared to

n. Decomposition strategies exploit this situation by solving a smaller quadratic program in the null-space of the linearized active constraints. A generic reduced-space SQP algorithm has essential four phases: (1) an initialization phase to determine an initial feasible point and to partition the variables into m dependent variables and $n - m$ independent variables, (2) calculate the search direction for the dependent variables using linear algebra, (3) compute the search direction for the independent variables from the smaller QP subproblem, and (4) perform a line search. Steps (2), (3), and (4) are performed iteratively. Step (1) may be applied during the course of the iterative process if a QP subproblem becomes infeasible. If sparse linear algebra techniques are used in step (2), this decomposition strategy extends the size of the nonlinear program that can be solved to several thousand as long as the number of degrees of freedom remains relatively small (less than a few hundred).

There arise many situations where the number of degrees of freedom is not small. This occurs in, for example, continuous optimal control problems, multiperiod design, and data reconciliation. If differential equations are present in the constraints, they are generally reduced to a set of algebraic equations by using finite difference equations [20], the method of weighted residuals [72], or orthogonal collocation [33]. This process results in a large number of unknowns and, thus, increases the degrees of freedom of the nonlinear program. In addition, even if the number of degrees of freedom is small relative to the size of the system, the actual number may be quite large. If this is the case, using the dense BFGS update is too computationally expensive. An alternative technique to large-scale SQP is described by Betts [12]. In this case, the large, sparse Hessian of the Lagrangian is used instead of the dense BFGS update. To ensure the QP subproblem is well-posed, the Hessian of the Lagrangian is modified as follows:

$$\nabla_{xx}\bar{\mathcal{L}}(x,u,\nu) = \nabla_{xx}\mathcal{L}(x,u,\nu) + \tau(|\sigma|+1)I$$

where $\nabla_{xx}\mathcal{L}(x,u,\nu)$ is the exact Hessian of the Lagrangian, σ is the Gerschgorin bound for the most negative eigenvalue of $\nabla_{xx}\mathcal{L}(x,u,\nu)$, and $\tau \in [0,1]$ is the Levenberg parameter. A trust region strategy is used to ensure $\tau \to 0$ as the algo-

rithm converges to a solution. In this approach, the QP subproblem is solved using a technique based on the Schur-complement method of Gill *et al.*[48]. The Schur-complement method makes it possible to infer whether or not the projected Hessian is positive definite. If necessary, the Levenberg parameter is adjusted to ensure the matrix remains positive definite during the iteration process.

A third approach for large-scale SQP is described by Sargent *et al.*[97]. In addition to some minor modifications made to several aspects of the SQP algorithm, an interior-point algorithm is employed in the QP subproblem. In this approach, the exact Hessian of the Lagrangian is employed to take advantage of the sparsity of the system. The interior point algorithm requires the projection of the Hessian of the Lagrangian onto the null space of active constraints to be non-negative definite. To ensure this, the symmetric factorization of $\nabla_{xx}\mathcal{L} = LDL^T$ is modified as follows:

$$(D)_{ii} = \max\{0, (D)_{ii}\} \qquad i = 1, \ldots, n.$$

9.2.2 MINOS

A second state-of-the-art optimization algorithm is the MINOS/Augmented package [82]. MINOS has been applied successfully to large, sparse problems that are mostly linear. In general, more function evaluations are required than with SQP, making MINOS less apt to be used with modular simulators and other situations where function and derivative evaluations are expensive. This algorithm is, however, easily adapted to equation-oriented simulators and has been implemented in ASCEND [89] and SPEEDUP [83]. Like SQP, MINOS is an infeasible path, Newton-type approach. The inequality constraints are converted to equalities by including slack variables with the appropriate sign (determined by their bounds). At each iteration, the constraints are linearized and a complete optimization is performed with the augmented Lagrangian as the objective function (in contrast to the quadratic approximation used in SQP). By projecting the gradient of the objective function onto the null-space of the linearized constraints, unconstrained optimization techniques may be employed

(as long as the variables are monitored so they do not leave their bounds).

In general, the MINOS/Augmented package performs well for large, sparse, mostly linear problems. Since a full optimization is performed in the space of linearized constraints at each iteration, function and gradient evaluations are extensive, making this algorithm best suited for situations where these calculations are very efficient (e.g., equation-oriented simulators).

Currently, SQP [53] is considered to be one of the most efficient methods for solving general nonlinear programming problems. However, SQP has trouble converging highly nonlinear constraints and it is designed to find a single optimal point. In addition, SQP is only applicable when there are fewer than a few hundred degrees of freedom. Ideally, we would like to have an optimization algorithm that can be applied to problems that SQP has difficulty converging, has the ability to find multiple local optima, and is capable of solving large problems with many degrees of freedom. One such approach, discussed in this chapter, is to convert the Karush-Kuhn-Tucker conditions, the first-order necessary conditions for a local optima, into an equivalent set of nonlinear equations and solve these equations using the globally convergent homotopy continuation method. This approach was first proposed by Sun and Seider [108]. In the research proposed here, nonparametric optimization problems will be considered. The KKT necessary conditions will be converted to a system of nonlinear equations using Mangasarian's theorem which guarantees complementary slackness. The artificial parameter differential arclength homotopy continuation method will be applied to this system of equations. This method is capable of determining solutions not obtainable with Newton-type methods. In addition, it is possible to track out multiple (possibly all) KKT points from a single starting point. Finally this problem remains sparse throughout the calculation. Since the KKT necessary conditions are used, some post-processing will be necessary to determine which type of stationary points were obtained.

9.3 Homotopy Approach for Nonlinear Optimization

The problem with solving the necessary conditions for optimality of a nonlinear programming problem (the KKT conditions, (9.5)-(9.9)) directly using some iterative method is that they are a mixed system of nonlinear equations and inequalities. Solutions to equations (9.5)-(9.7), if found, may or may not be KKT points depending on the sign of the inequality residuals and associated KKT multipliers. The following theorem due to Mangasarian allows the complementarity conditions to be converted into an equivalent system of nonlinear equations

Theorem 1 *(Mangasarian)*

Let θ be any strictly increasing function from $\mathbb{R}$ into $\mathbb{R}$ ($a > b \Leftrightarrow \theta(a) > \theta(b)$), and let $\theta(0) = 0$. Then z solves the complementarity problem

$$\begin{aligned} z &\geq 0 \\ g(z) &\geq 0 \\ z^T g(z) &= 0 \end{aligned}$$

for $z \in \mathbb{R}^m$ and $g : \mathbb{R}^m \longrightarrow \mathbb{R}^m$ if and only if

$$\begin{gathered} \theta(|g_1 - z_1|) - \theta(g_1) - \theta(z_1) = 0 \\ \vdots \\ \theta(|g_m - z_m|) - \theta(g_m) - \theta(z_m) = 0. \end{gathered}$$

With this theorem, the KKT conditions may be represented by the following set of nonlinear equations:

$$\begin{aligned} \nabla f(x) - \nabla h(x)^T \nu - \nabla g(x)^T u &= 0 && (9.18) \\ h(x) &= 0 && (9.19) \\ \theta(|g_i(x) - u_i|) - \theta(g_i(x)) - \theta(u_i) &= 0 \qquad i = 1, \ldots, m && (9.20) \end{aligned}$$

Every solution satisfying equations (9.18)-(9.20) is a KKT point of the original NLP. In this work, these equations are solved using homotopy continuation

9.3.1 Homotopy Continuation

Homotopy continuation has been used in the past to solve systems of equations when a good initial guess is not known a priori or when the equations contain many singularities. Suppose $F(x) = 0$ is the set of equations we are interested in solving. One popular homotopy is the *convex linear homotopy* given by:

$$H(x, \lambda) = \lambda F(x) + (1 - \lambda) G(x) \tag{9.21}$$

where λ is the homotopy parameter and $G(x)$ is set of equations which has a known solution $G(x^0) = 0$. The idea behind homotopy continuation is to begin at $\lambda = 0$ and $x = x^0$, where $H(x^0, 0) = 0$, and track the homotopy path given by $H(x, \lambda) = 0$ to $\lambda = 1$ and $x = x^*$, a solution of $F(x) = 0$. Chapter 2 describes a similar application of homotopy continuation, however, in chapter 2, the systems of equations at $\lambda = 0$ and $\lambda = 1$ had physical significance; the system at $\lambda = 0$ corresponded to an ideal representation of phase equilibrium, the system at $\lambda = 1$ corresponded to a nonideal representation, and λ corresponded to a deformation from ideality. The physical homotopies in chapter 2 gave rise to bifurcations in the homotopy branches that were exploited in the computation of azeotropes and heteroazeotropes. As will be

shown below, these bifurcation will typically not be encountered for the homotopies described in this chapter.

One homotopy, which is robust for solving general sets of nonlinear equations, is the Newton homotopy:

$$H(x, \lambda) = F(x) - (1 - \lambda)F(x^0) \tag{9.22}$$

In this case, $G(x) = F(x) - F(x^0)$. Another popular homotopy is the fixed-point homotopy:

$$H(x, \lambda) = \lambda F(x) + (1 - \lambda)(x - x^0) \tag{9.23}$$

Here, $G(x) = x - x^0$.

We are interested in under what conditions a smooth, non-bifurcating path, $c(\xi) \in H^{-1}(0)$, exists and connects $H(x^0, 0)$ and $H(x^*, 1)$, where x^* is a solution to $F(x^*) = 0$, the system of equation whose solution we desire. The question of whether or not a smooth path exists is answered by the following theorem presented in Allgower and Georg [4].

Theorem 2 *Let zero be a regular value of $H(x, \lambda)$, i.e., $\left(\nabla H \mid \frac{\partial H}{\partial \lambda}\right)$ has maximal rank n $\forall$ $(x, \lambda) \in H^{-1}(0) = \{(x, \lambda) \mid H(x, \lambda) = 0, x \in \mathbb{R}^n, \lambda \in \mathbb{R}\}$. Then the curve, c, defined by $H(x, \lambda) = 0$, satisfies one of the following two conditions:*

1. *The curve c is diffeomorphic to a circle (i.e. there is a period $T > 0$ such that $c(s_1) = c(s_2)$ if and only if $s_1 - s_2$ is an integer multiple of T)*

2. *The curve c is diffeomorphic to the real line (i.e., c is injective and c has no accumulation points for $s \longrightarrow \pm\infty$)*

Thus, if 0 is a regular value of $H(x, \lambda)$, the curves will be smooth and non-bifurcating. According to the *parameterized Sard's theorem* [25], the set of x^0, where $H(x^0, 0) = 0$, such that 0 is not a regular value of $H(x, \lambda)$ are in a set of measure zero. A set of measure zero in $\mathbb{R}^n$ is a subset of $\mathbb{R}^n$ that can be contained in the union of neighborhoods, in $\mathbb{R}^n$, with total volume smaller than any positive number. The probability of selecting, at random, an item from a set of measure zero is zero. Thus, as long as x^0 is chosen at random, independent of the structure of the problem, the homotopy path will be smooth and non-bifurcating with probability 1.

The second question, when does the homotopy path cross $\lambda = 1$, holds when:

1. the Jacobian matrix, $\left(\nabla_x H \;\middle|\; \frac{\partial H}{\partial \lambda}\right)$, has maximal rank n on the set $S = \{(x, \lambda) \mid H(x, \lambda) = 0, x \in \mathbb{R}^n, 0 \leq \lambda < 1\}$,
2. $H(x, 0) = 0$ has a unique solution x^0,
3. the set S is bounded.

The first item above will be true, with probablity 1, if x^0 is chosen at random. The second item will be true if $G(x)$ has a single solution, x^0. This is always the case for the fixed-point homotopy, equation (9.23), however, the Newton homotopy, equation (9.22), may have multiple solutions at $\lambda = 0$, corresponding to roots of the equation $F(x) = F(x^0)$. A theorem by Smale [103] provides conditions when the Newton homotopy crosses $\lambda = 1$:

Theorem 3 *(Smale)*

Let $D \subset \mathbb{R}^n$ and $F : \mathbb{R}^n \longrightarrow \mathbb{R}^n$ satisfy the assumptions below:

1. *F is a $C^\infty -$ map,*
2. *$D \subset \mathbb{R}^n$ is open and bounded and ∂D, the boundary of D, is a connected C^∞ manifold of $\mathbb{R}^n$,*
3. *0 is a regular value of F,*

4. $F(x) \neq 0 \ \ \forall\, x \in \partial D$,

5. *the Jacobian* $\nabla F(x)$ *is nonsingular* $\forall\, x \in \partial D$, *and*

6. *the Newton direction* $-\nabla F(x)^{-1}F(x)$ *is not tangent to* ∂D *at* $x \in \partial D$.

Let $x^0 \in \partial D$ *be chosen such that* 0 *is a regular value of the map* $H(x, \lambda) = F(x) - (1 - \lambda)F(x^0)$ *(recall that by Sard's theorem, if* x^0 *is chosen at random,* 0 *will be a regular value of* $H(x, \lambda)$ *with probability 1). Let* C_{x^0} *be the connected component of* $\{(x, \lambda) \mid H(x, \lambda) = 0, x \in \mathbb{R}^n, \lambda \in \mathbb{R}\}$. *Finally, let* $s \in \mathbb{R} \mapsto (x(s), \lambda(s))$ *be a parameterization of* C_{x^0} *such that*

1. $x(0) = x^0$ *and* $\lambda(0) = 0$,

2. $\dot{x}(0)$ *points into* D, *where* $\dot{x} \equiv dx/ds$.

Then there is a parameter $s_0 > 0$ *such that*

1. $x(s) \in D$ *for* $0 < s < s_0$,

2. $x(s_0) \in \partial D$,

3. $\lambda(s_0) > 1$.

Consequently, the curve C_{x^0} *passes through* $D \times \{1\}$ *in an odd number of points* $(x^*, 1) \in D \times \{1\}$ *with* $F(x^*) = 0$.

Thus, provided some conditions, the Newton homotopy will locate at least one, and possibly more, solutions of $F(x) = 0$.

The theorems above establish conditions for the existence of a smooth, non-bifurcating homotopy path connecting $\lambda = 0$ and $\lambda = 1$. The following discussion describes how this path is tracked numerically[1].

The homotopy path described above is tracked using a calculation known as differential arclength continuation. This algorithm was introduced by Klopfenstein [66]. First, the variables (x, λ) are parameterized with respect to arclength, s,

$$H(x(s), \lambda(s)) = 0. \tag{9.24}$$

This equation is then differentiated with respect to arclength to obtain the following set of equations,

$$\begin{pmatrix} \nabla_x H(x,\lambda) & \partial H/\partial\lambda \end{pmatrix} \begin{pmatrix} \frac{dx}{ds} \\ \frac{d\lambda}{ds} \end{pmatrix} = 0 \tag{9.25}$$

(dx/ds and $d\lambda/ds$ are the tangents on the path at the current point). Arclength is defined by

$$\left(\frac{dx}{ds}\right)^T \left(\frac{dx}{ds}\right) + \left(\frac{d\lambda}{ds}\right)^2 = 1. \tag{9.26}$$

The last two equations above are combined to form the following fully-determined system,

$$\begin{pmatrix} \nabla_x H(x,\lambda) & \partial H/\partial\lambda \\ (dx/ds)^T & d\lambda/ds \end{pmatrix} \begin{pmatrix} \frac{dx}{ds} \\ \frac{d\lambda}{ds} \end{pmatrix} = \begin{pmatrix} 0 \\ 1 \end{pmatrix}. \tag{9.27}$$

This system is solved, using the tangent at the previous point within the matrix, for

[1]The task of numerically tracking a path is described in chapter 2 using a locally parameterized continuation process. A different technique is employed in this chapter and an automatic scaling algorithm is developed that better handles the numerical problems encountered in this work. Scaling was not a problem encountered during the continuation with the homotopies of chapter 2.

the tangent at the current point. The next point on the curve is given by

$$x^{k+1} = x^k + h^k \left(\frac{dx}{ds}\right)^k \tag{9.28}$$

$$\lambda^{k+1} = \lambda^k + h^k \left(\frac{d\lambda}{ds}\right)^k \tag{9.29}$$

where h^k is the current stepsize. This predicted point is brought closer to the path using Newton's method with corrector steps orthogonal to the previous tangent,

$$\begin{pmatrix} \nabla_x H(x,\lambda) & \frac{\partial H}{\partial \lambda} \\ \left(\frac{dx}{ds}\right)^T & \frac{d\lambda}{ds} \end{pmatrix} \begin{pmatrix} \delta x \\ \delta \lambda \end{pmatrix} = - \begin{pmatrix} H \\ 0 \end{pmatrix} \tag{9.30}$$

$$x_{j+1} = x_j + \delta x_j \tag{9.31}$$

$$\lambda_{j+1} = \lambda_j + \delta \lambda_j \tag{9.32}$$

The subscript in equations (9.31) and (9.32) refer to the *j-th* corrector iteration. By parameterizing with respect to arclength, a monotonically increasing parameter of the curve, it is possible to trace the path through turning points.

The discussion above contains conditions under which the homotopy path exists and how it can be tracked numerically. According to theorem 1, if zero is a regular value of $H(x,\lambda)$ then the path will be connected. Even if the path is connected, it may be connected at $\pm\infty$. Lin [70] provides conditions for when the homotopy path is connected at $\pm\infty$:

1. If $\lambda \longrightarrow 0$ as $x_i \longrightarrow \pm\infty$ and $x_i/F_i \longrightarrow 0$ then the homotopy path is connected at $-x_i$ and $\lambda = 0$.

2. If $x \longrightarrow \tilde{x}$ as $\lambda \longrightarrow \pm\infty$. The homotopy path is connected at $x = \tilde{x}$ and $\lambda = \mp\infty$.

These conditions determine how the path can be connected at the opposite infinity.

In order to facilitate the detection of when a variable approaches an asymptote,

Seader *et al.* [98] developed two mapping functions which map $(-\infty, \infty) \longrightarrow (-1, 1)$:

$$\tilde{y}_i = \frac{2y_i}{1 + y_i^2} \tag{9.33}$$

and

$$\tilde{y}_i = \frac{y_i}{\sqrt{1 - y_i^2}} \tag{9.34}$$

By taking the predictor step in the mapped space, it is easy to tell when a branch must be switched using the criteria (1) and (2) above.

9.3.2 Summary of Homotopy Approach

The approach described in this chapter is simple: use symbolic transformation techniques to transform an NLP into an equivalent system of nonlinear equations (using Mangasarian's theorem to handle the complementarity condition) and solve the resulting system for multiple solutions using either a Newton homotopy or a fixed point homotopy with variable mappings.

9.4 Comparison with SQP

SQP has been applied successfully to a wide variety of nonlinear optimization problems. A relationship between the approach described in this chapter and SQP is shown below.

First, consider the Newton homotopy given in equation (9.22). Differentiating

with respect to s,

$$\nabla F(x)\frac{dx}{ds} + F(x^0)\frac{d\lambda}{ds} = 0. \tag{9.35}$$

Combining this equation with equation (9.22) set to zero, we obtain,

$$\frac{dx}{ds} = -\frac{d\lambda/ds}{1-\lambda}\nabla F(x)^{-1}F(x). \tag{9.36}$$

Now consider the global Newton method. The global Newton method can be interpreted as the integration of the autonomous ODE system,

$$\frac{dx}{dt} = -\nabla F(x)^{-1}F(x). \tag{9.37}$$

A damped Newton method, given by the iteration formula,

$$x^{k+1} = x^k - \alpha^k \nabla F(x^k)^{-1}F(x^k), \tag{9.38}$$

can be obtained by a first-order explicit integration of equation (9.37) with a stepsize, α^k, chosen such that $\|F(x^{k+1})\| \leq \|F(x^k)\|$. The direction given by equation (9.37) is the same as the direction given in equation (9.36). Allgower and Georg [4] show that unlike the global Newton method, calculations remain stable near singular points with the Newton homotopy.

Now consider the nonlinear programming problem given in equations (9.2)-(9.4).

The KKT conditions may be expressed as:

$$\nabla f(x) - \nabla h(x)^T \nu - \nabla g_A(x)^T u_A = \nabla_x \mathcal{L}(x, \nu, u_A) = 0 \tag{9.39}$$

$$h(x) = 0 \tag{9.40}$$

$$g_A(x) = 0 \tag{9.41}$$

$$(u_A > 0) \tag{9.42}$$

where $g_A(x)$ denotes the set of active inequality constraints. This system of nonlinear equations may be solved using a damped Newton method:

$$\begin{pmatrix} \nabla_{xx}\mathcal{L}(x,\nu,u) & -\nabla h(x)^T & -\nabla g_A(x)^T \\ \nabla h(x) & 0 & 0 \\ \nabla g_A(x) & 0 & 0 \end{pmatrix} \begin{pmatrix} \delta x \\ \delta \nu \\ \delta u \end{pmatrix} = - \begin{pmatrix} \nabla_x \mathcal{L} \\ h(x) \\ g_A(x) \end{pmatrix} \tag{9.43}$$

$$\tag{9.44}$$

$$x_{j+1} = x_j + \alpha_j \delta x_j \tag{9.45}$$

$$\nu_{j+1} = \nu_j + \alpha_j \delta \nu_j \tag{9.46}$$

$$u_{j+1} = u_j + \alpha_j \delta u_j \tag{9.47}$$

As described earlier, SQP converges to an optimum by obtaining search directions from the solution of a QP subproblem formed by taking quadratic approximations of the objective function and linearization of the constraints. The current point is updated by moving in the direction obtained in the QP by a stepsize determined by the minimization of some merit function. It can be shown that the search directions,

$(\delta x, \delta \nu, \delta u)$, obtained by solving system (9.43) above are exactly the same as the search directions obtained in the QP subproblem of the SQP method, thus, SQP can be interpreted as applying a damped Newton method to the KKT conditions of the original problem.

A relationship between the damped Newton method and homotopy continuation is shown above. Since SQP may interpreted as applying a damped Newton method to the KKT conditions of the original problem, there is a relationship between SQP and the homotopy continuation approach for solving NLPs. There are, however, several differences. Stepsize for the SQP method is chosen by minimizing a scalar merit function or a trust region strategy, whereas stepsize for the homotopy continuation approach is determined by the curvature of the path, by the number of corrector iterations required to bring the predicted point back to the path, or by some other stepsize algorithm. Most implementations of the SQP method use a positive-definite approximation of the Hessian of the Lagrangian, $\nabla_{xx}\mathcal{L}(x, \nu, u)$, to ensure that the search directions move the iterates to a beneficial KKT point (i.e., KKT point corresponding to a local minimum), whereas in the homotopy continuation approach the analytical Hessian of the Lagrangian used and solutions may or may not correspond to local minima. The quadratic program subproblem of the SQP method determines the active constraints, whereas the homotopy continuation approach uses Mangasarian's theorem to ensure the complementary slackness condition is satisfied at the solutions. The SQP method will converge to a single KKT point, the homotopy approach may potentially compute several KKT points. The damped and global Newton methods, and thus, the SQP method are unstable in complex solution spaces. This is not a

problem for the homotopy approach. Finally, the approximation of the Hessian of the Lagrangian used in the SQP method (usually, the Broyden, Fletcher, Goldfarb, and Shanno (BFGS) update) is dense regardless of the sparsity of the analytical Hessian which seriously limits the size of the NLP that can be solved.

9.5 Implementation

The approach described in this chapter applies homotopy continuation to a set of nonlinear equations equivalent to the KKT conditions. These equations, (9.5)-(9.9), are a mixture of model equations and partial derivatives. Requiring the user to derive these equations by hand would be impractical and extremely error prone. This is where symbolic manipulation technology plays a pivotal role, making the method practical for large systems of equations.

This algorithm has been implemented in the process simulator ABACUSS[2][9]. As described in chapter 5, ABACUSS is a large-scale, equation-oriented process simulator based on an interpretive architecture. The user writes an optimization problem in the form of equations (9.2)-(9.4) in an input file. These equations are stored in computer memory as a directed acyclic graph when the input file is translated. In addition, partial derivatives are calculated and also stored within the equation graph (see part 2 of this thesis). Flowsheet models are generally very large (10,000-100,000 equations is typical), however, the Jacobian of these equation is very sparse. Thus,

[2]ABACUSS (Advanced Batch And Continuous Unsteady-State Simulator) process modeling software, a derivative work of gPROMS software, ©1992 by Imperial College of Science, Technology, and Medicine.

only partial derivatives that are not identically zero are computed and sparse linear algebra routines are employed in the continuation calculations [39]. The homotopy continuation algorithm used to find the KKT points requires subroutines that return the current values of equations (9.18)-(9.20) and the Jacobian of these equations:

$$\begin{pmatrix} \nabla_{xx}\mathcal{L}(x,\nu,u) & -\nabla h(x)^T & -\nabla g(x)^T \\ \nabla h(x) & 0 & 0 \\ \nabla_x \Theta(x,u) & 0 & \nabla_u \Theta(x,u) \end{pmatrix} \tag{9.48}$$

where $\mathcal{L}(x,\nu,u)$ is the Lagrangian defined previously and

$$\Theta_i(x,u) = \theta(|g_i(x) - u_i|) - \theta(g_i(x)) - \theta(u_i) \tag{9.49}$$

are the Mangasarian constraints. The actual equations (9.18)-(9.20) are, however, not constructed symbolically. When a residual or Jacobian evaluation is required, the values are computed by combining the various terms in the equation.

9.6 Algorithm Improvements

A major advantage of the equation-oriented simulator is that it allows the user to write models independently of the numerical algorithms that will be applied to them. This flexibility has the disadvantage that the equations may not be in a form that is ideal for the solution algorithm. A common example of this is variable scaling. A typical model of a chemical process will have variables representing energy and concentrations

and these variables often differ by several orders of magnitude. Even if the variables are initially the same order of magnitude, they may change significantly along the homotopy path. This observation presents a problem to the arclength continuation algorithm described above. Let $y = (x, \lambda)$. The next point on the curve is predicted by:

$$y^{k+1} = y^k + h^k \left(\frac{dy}{ds}\right)^k \tag{9.50}$$

where $(dy/ds)^k$ is obtained by solving the linear system (9.27) at the current point y^k. The arclength constraint,

$$\left(\frac{dy}{ds}\right)^T \left(\frac{dy}{ds}\right) = 1 \tag{9.51}$$

implies that $|dy_i/ds| \leq 1 \quad \forall i$. Thus,

$$\left|y_i^{k+1} - y_i^k\right| = \left|h^k\right| \left|\left(\frac{dy_i}{ds}\right)^k\right| \leq \left|h^k\right| \quad \forall i \tag{9.52}$$

Now suppose that $y_i^{k+1} \sim 10^5$ and $y_j^{k+1} \sim 10^{-5}$ (y_i^{k+1} may represent the internal energy inside a reactor and y_j^{k+1} may represent the concentration of some chemical species in that reactor). This causes difficulty in the stepsize calculation algorithm. If the stepsize is chosen such that the large magnitude variables are tracked efficiently, i.e.,

$$\left|y_i^{k+1} - y_i^k\right| \sim 10^5$$

the stepsize will be very large, resulting in far more corrector iterations since the variables with smaller magnitude will be further from the path after the predictor step. A more serious problem is that the solution may jump to another branch on the path. In addition, most stepsize algorithms require the user to specify an upper bound for the calculated stepsize. If the variables have large differences in magnitude anywhere along the path, it is very difficult for the users to specify a priori a reasonable upper bound for the stepsize. In order to improve the stepsize selection, the variables are scaled to be order one quantities for the predictor step. Let y denote the original variables and Y denote the scaled variables. Select scaling factors such that

$$Y_j = \beta_j y_j \sim O(1) \qquad \forall j.$$

The arclength criteria must be satisfied in this new space, thus,

$$\frac{dY_j}{d\tilde{s}} = \frac{dY_j}{dy_j}\frac{dy_j}{ds}\frac{ds}{d\tilde{s}} \tag{9.53}$$

$$= \beta_j \frac{dy_j}{ds}\frac{ds}{d\tilde{s}} \tag{9.54}$$

where $\tilde{s}$ denotes the arclength in the new space. To obtain $ds/d\tilde{s}$,

$$\left(\frac{dY}{d\tilde{s}}\right)^T \left(\frac{dY}{d\tilde{s}}\right) = \sum_i \beta_i^2 \left(\frac{dy_i}{ds}\right)^2 \left(\frac{ds}{d\tilde{s}}\right)^2 = 1 \tag{9.55}$$

$$\frac{ds}{d\tilde{s}} = \frac{1}{\sqrt{\sum_i \beta_i^2 \left(dy_i/ds\right)^2}} \tag{9.56}$$

Thus,

$$\frac{dY_j}{d\tilde{s}} = \frac{\beta_j}{\sqrt{\sum_i \beta_i^2 \left(dy_i/ds\right)^2}} \frac{dy_j}{ds} \tag{9.57}$$

The tangent in the original space is obtained as described previously. The stepsize is computed in this new space.

This procedure was tested on the following problem:

$$3.221\left(\frac{x}{1000}\right)^3 + 1.652y^2 = 4.875 \tag{9.58}$$

$$2.050\left(\frac{x}{1000}\right)^2 + 0.700y^2 = 2.730 \tag{9.59}$$

From the initial point, $(x^0, y^0) = (500, 1)$, all four solutions were obtained on a single path. A plot of y versus λ is shown in figure 9-1.

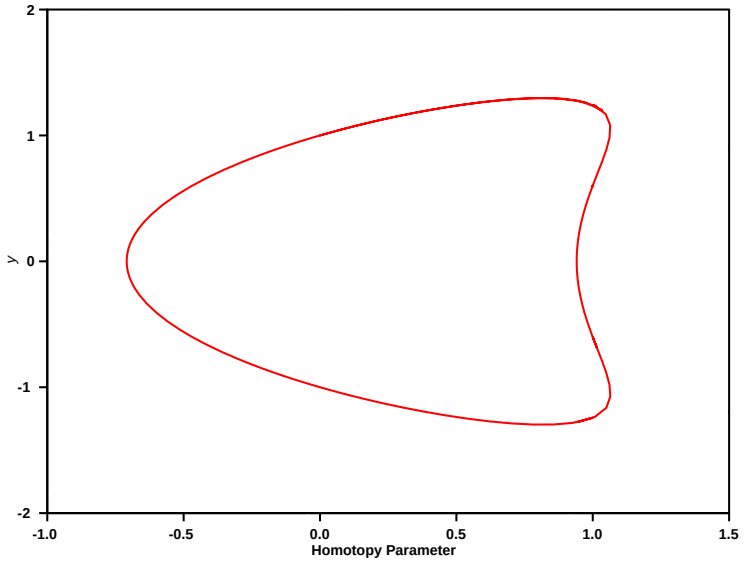

Figure 9-1: y versus λ

Table 9.1 contains the number of steps, from the starting point, taken to reach

each solution and the total number of factorizations of the Jacobian matrix required for both the scaled and unscaled cases.

Table 9.1: Comparison of continuation with and without variable scaling for equations (9.58) and (9.59)

	with scaling		without scaling	
Solution	steps	fact.	steps	fact.
1	15	41	16	35
2	19	62	24	62
3	60	149	44	314
4	65	181	53	339
1	192	466	121	831

In the case where the variables were scaled, approximately two corrector iterations were required per step, indicating that the predicted point was not too far from the path. In the case where the variables were not scaled, approximately six corrector iterations were required per step, thus, the variables were much further from the path. Although, it did not happen in this case, when the predicted variables are far from the path, there is a much greater chance of unwanted branch switching. An additional advantage of the variable scaling is that an upper bound for the step size of 0.1 was used for a great variety of problems, whereas in the case where the variables were not scaled, several attempts were made for each problem to find a suitable upper bound for the stepsize.

9.7 Nonlinear Optimization Example

A simple nonlinear programming problem exhibiting multiple local optima is the minimization of the Himmelblau function [94]. The two cases examined, both uncon-

strained and constrained, are shown below.

$$\min_x \quad (x_1^2 + x_2 - 11)^2 + (x_1 + x_2^2 - 7)^2 \tag{9.60}$$

and

$$\min_x \quad (x_1^2 + x_2 - 11)^2 + (x_1 + x_2^2 - 7)^2 \tag{9.61}$$

$$\text{s.t.} \qquad x_1 - x_2 \geq 0 \tag{9.62}$$

Using a Newton homotopy, all nine solutions of the unconstrained case were obtained from the starting point $x^0 = (-5, -5)$. Of the nine fixed-points, four are local minima, one is a local maxima, and four are saddle-points. To determine the type of fixed-point, the eigenvalues of the Hessian of the objective function were examined. A summary of the calculation is shown in table 9.2.

Table 9.2: Summary of results for problem (9.60)

x_1	x_2	type	steps	fact.
3.000	2.000	minimum	19	50
3.385	0.074	saddle	34	95
3.584	-1.848	minimum	73	189
-0.128	-1.954	saddle	115	296
-0.271	-0.923	maximum	124	327
0.087	2.884	saddle	211	527
-2.805	3.131	minimum	253	626
-3.073	-0.081	saddle	292	721
-3.779	-3.283	minimum	312	784

Using the same homotopy, all seven solutions of the constrained case were obtained from the starting point $x^0 = (6, 5)$ and $u = 10$. A plot of x_1 versus λ is shown in figure

9-2 and a plot of x_2 versus λ is shown in figure 9-3. Of the seven fixed-points, three are local minima, two are local maxima, and two are saddle-points. To determine the type of fixed-point, the eigenvalues of the Hessian of the Lagrangian were examined. A summary of the calculation is shown in table 9.3.

Table 9.3: Summary of results for problem (9.61-9.62)

x_1	x_2	u	type	steps	fact.
3.000	2.000	0.000	minimum	48	112
3.385	0.074	0.000	saddle	94	229
3.584	-1.848	0.000	minimum	141	333
-0.128	-1.954	0.000	saddle	206	488
-0.271	-0.923	0.000	maximum	260	641
-0.500	-0.500	8.000	maximum	469	1115
-3.541	-3.541	32.331	minimum	483	1167

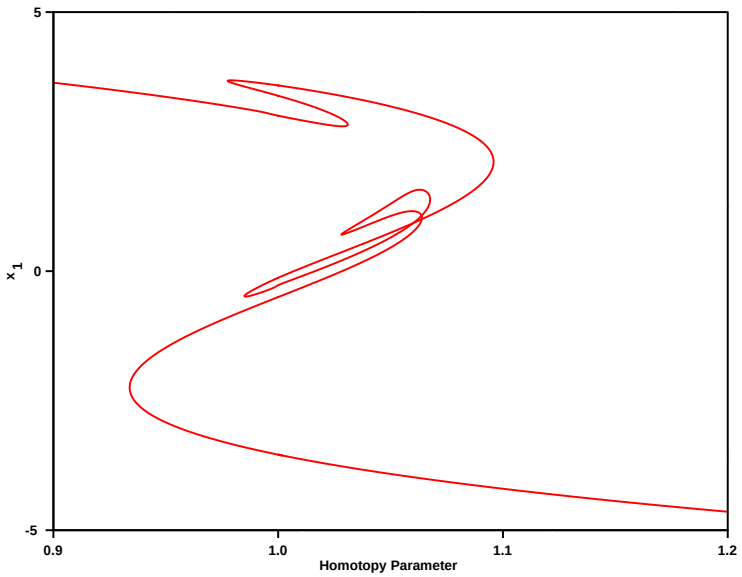

Figure 9-2: x_1 versus λ

The function used for the Mangasarian equations (9.20) was $\theta(z) = z^3$, which is the simplest function that is C^2 and satisfies the requirements given in Magasarian's theorem. This is the same function used by Vasudevan [114] in the parametric optimization of an optimal-fuel orbital problem.

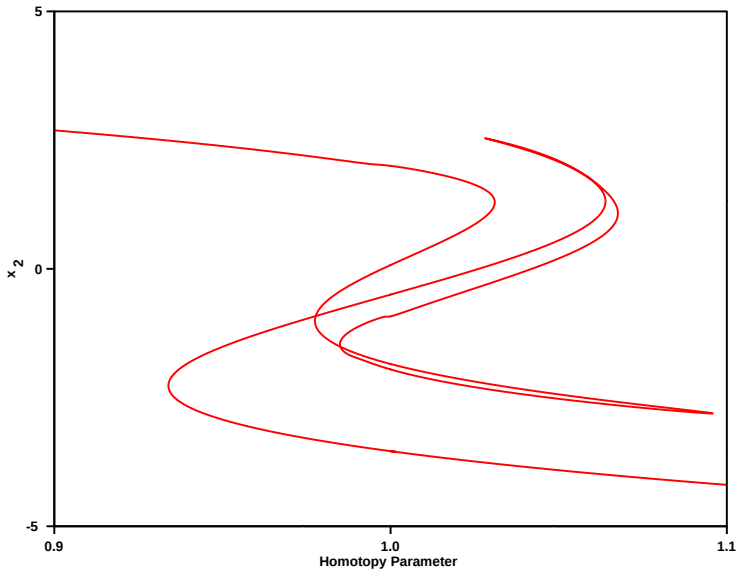

Figure 9-3: x_2 versus λ

9.8 Problems with Approach

Although the homotopy continuation approach looks promising in theory, there are several problems associated with its implementation. One problem with the fixed-point homotopy with mappings (equations (9.33) and (9.34)) is that when the variables approach infinity, a few elements of the Jacobian matrix become very large and the matrix becomes highly ill-conditioned. This leads to inaccurate predictors and correctors which cause the path to be lost. The reasons causing the ill-conditioning are not remedied by row and column scaling and there are no large-scale, sparse QR factorization routines that would make the factorization more stable. In the case of the Newton homotopy, equation (9.22), the homotopy path often goes off to infinity before reaching $\lambda = 1$. For some problems, several attempts had to be made to find a suitable starting point but with limited success. Another problem with this approach is that variable bounds have to be included as inequality constraints. Unlike the SQP

method which handles bound violations by truncating the stepsize, including the additional constraints drastically increases the dimension of the system for problems with many variable bounds. Finally, all fixed-points of equations (9.18)-(9.20) are obtained. This requires post-processing to determine whether or not the solution is a local minimum.

In short, the approach described in this chapter was originally intended for large-scale, highly nonlinear optimization problems with the hope of obtaining several (possibly all) KKT points. The problems described above, including the lack of theoretical guarantees all KKT points will be obtained, were the impetus for deciding to change directions in the research. The two main techniques used in this phase of the research were symbolic transformation/differentiation and homotopy continuation. These approaches were applied with far greater success in the techniques developed in the first two parts of this thesis.

9.9 Conclusions

The approach described in this chapter has the potential to be an effective alternative to SQP for nonlinear optimization. First, unlike the damped Newton method which is applied in SQP, homotopy continuation calculations are stable near singular points. Second, homotopy continuation is capable of systematically generating multiple solutions, whereas the SQP method is designed to find a single local optimal point. Finally, the SQP method is able to handle problems where the number of degrees of freedom are less than a few hundred. In the case of the homotopy continuation

method, the system of equations solved is sparse, thereby allowing large problems to be solved when sparse linear algebra routines are employed. Although this approach seems promising, there have been several problems associated with its implementation as mentioned above. First, the theorems given in section 9.1.3 simply provide conditions for when the homotopy path connecting $\lambda = 0$ and $\lambda = 1$ exists. This path depends on the initial point x^0 and in many cases, several attempts have to be made to find a suitable starting point. Using a fixed-point homotopy with mappings to recover from an unbounded branch by switching to the opposite infinity has problems due to the ill-conditioning of the Jacobian matrix when the variables become very large. Finally, the fact that the variable bounds must be included as inequality constraints will, in some cases, make the problem very large.

Chapter 10

Conclusions

Several topics relevant to the design and simulation of heteroazeotropic systems have been developed in this thesis. Specifically, an algorithm for computing the homogeneous and heterogeneous azeotropes present in a multicomponent mixture and an efficient class of techniques for analytical derivative evaluation have been developed.

A necessary task when analyzing and designing azeotropic and heteroazeotropic systems is the a priori determination of all homogeneous and heterogeneous azeotropes present in a multicomponent mixture. The technique described in chapter 2, under reasonable assumptions, will compute all homogeneous and heterogeneous azeotropes predicted by the phase equilibrium model employed. The technique is independent of both the representation of the nonideality of the mixture and the topology of the liquid-liquid region. Furthermore, the approach can be extendend to handle any number of liquid and/or solid phases in equilibrium (provided there is a suitable model available that can compute the chemical potential of a solid in a solid solution). The approach can also be extended (as described in chapter 3) to systematically and ef-

ficiently explore the phase equilibrium structure of a multicomponent mixture under system and/or property parameter variation, including the capability of detecting incipient homogeneous and heterogeneous azeotropes (i.e., azeotropes that do not exist under current conditions or property parameter values but may exist under different conditions) and the determination of the bifurcation values where they appear, disappear, or switch between each other. This capability of exploring the phase equilibrium structure is a very powerful tool for the modeler, assisting in the specification of the conditions under which to perform a simulation and the interpretation of the simulation results, and for the experimentalist while fitting property model parameters, providing a systematic and efficient means of exploring of the capabilities and limitations of the phase equilibrium model.

The second major topic of this research is relevent to the simulation of heteroazeotropic systems. A new class of automatic differentiation techniques, known as the subgraph reduction approach, have been developed that both increase the speed in which analytical derivatives are computed and reduce the amount of space required for storing and computing the Jacobian matrix of a system of equations. Furthermore, an efficient interpretive version of the reverse mode of the subgraph reduction approach has been developed that dramatically improves the efficiency (increased speed and reduced memory requirements) of derivative evaluation within an interpretive simulator architecture compared to existing approaches. Due to the efficient preprocessing step of this interpretive implementation, this approach is ideally suited for hybrid discrete/continuous simulation.

10.1 Future Directions and Recommendations

Chapter 2 of this thesis provides a general framework for computing univariate thermodynamic states. The examples contained in this thesis consisted of homogeneous azeotropes and heteroazeotropes of systems with two immiscible liquid phase in equilibrium. In principle, the framework can be extended to more than two liquid phases in equilibirium, systems exhibiting eutectics, systems exhibiting heterogeneous reactive azeotropes, etc. The examination of eutectics is currently limited due to the lack of a suitable model that is capable of computing the chemical potential of a solid in a continuous solid solution. Provided such a model is available, the homotopy map would correspond to a deformation from the ideal solubility of the solid components to that predicted by the nonideal chemical potential model.

The extensions to the heteroazeotrope finding algorithm described in chapter 3 can be used while estimating property model parameters and determining the sensitivity of the phase equilibrium model with respect to these parameters. This is briefly explained in section 4.2.8. The technique described in chapter 3 can be used while parameters are being fit to the experimental data and if azeotropes and heteroazeotropes are not predicted by the model under the current set of parameters, the location of the bifurcation or intersection point corresponding to the azeotrope or heteroazeotrope that is not predicted (inside the physical composition space) can be used to systematically adjust the parameters.

The ability to determine how an azeotrope or heteroazeotrope varies with system parameters is very useful when designing separation systems. The techniques

developed in this thesis not only allows the modeler to determine how the azeotropic states vary, but also provides efficient means for predicting when an azeotrope or heteroazeotrope appears by monitoring the location of bifurcation and intersection points associated with nonphysical branches. This technique obviates the need for performing a computationally expensive phase stability test many times during the design study which would be otherwise necessary in order to determine when the new azeotropes or heteroazeotropes appear.

Although the area of heteroazeotropic simulation and analysis has been studied extensively in the past, there are several areas in need of improvement. Currently, in order to correctly construct heterogeneous residue curve maps and perform heteroazeotropic dynamic simulation, a phase stability test must be performed at each step during the integration (and on every tray of a heteroazeotropic column). To guarantee correctness of the residue curves and column profiles, a computationally expensive phase stability test such as that of McDonald or Stadtherr must be applied. The cost of such calculations is a strong motivation to develop novel, more efficient techniques. An alternative approach may be to construct convex regions containing the heterogeneous regions in state space. This can be performed as a preprocessing step using the techniques developed in chapter 3 of this thesis. The location of the trajectory relative to these regions can be used to determine whether or not a more expensive phase stability test should be employed to accurately determine the point at which one or more additional phases appear or disappear. For a certain class of systems it may be possible to decide when it is necessary to perform a phase stability test by monitoring the bifurcation and intersection points associated with the

heteroazeotrope finding algorithm of nonphysical branches.

Another area of intersest is heteroazeotropic batch distillation. Heteroazeotropic distillation is typcially carried out as a continuous process. Processes in the pharmaceutical industry, however, are largely batch in nature. Furthermore, in order to exploit solvent recovery techniques, it may be necessary to combine various streams in a process (or multiple processes) that split into multiple liquid phases. The ability to perform heterogeneous batch distillation would greatly increase the separations possible.

The automatic differentiation techniques developed in this thesis have been successfully applied within an interpretive architecture. Theoretical analysis has been performed to bound the operation count and memory requirements required for compiled implementations of the subgraph reduction approach, however, there are actually several more considerations than simply these bounds. For example, a general residual subroutine containing the system of equations of interest typically contains programming structures such as **IF** equations, **DO** loops, embedded subroutine/function calls, etc., which complicate the matter of generating an equation graph required by the subgraph reduction approach. **IF** equations can be represented in the graph as through the use of conditional vertices, with children correponding to the true and false conditions of the logical expression. Through the use of these conditional vertices, conditional accumulation sequences can be generated for the Jacobian evaluation. **DO** loops can be handled in a couple of ways. First, the loops can be "unrolled", creating a sequence of normal assignments from which the graph can be constructed. Alternatively, the sparse vector implementations of the forward or re-

verse modes can be applied to the subset of equations contained within the loop. This would allow the compiler to perform loop optimizations (a very sophisticated class of techniques) on this portion of the code. The best alternative would depend on the structure of the code contained within the loop and the optimizations possible when unrolling the loop. Embedded subroutines and functions can be handled in a recursive manner: performing the automatic differentiation algorithm to the embedded subroutines and functions, replacing these subroutine and function calls with calls to subroutines that also evaluate the derivatives of the code contained in the original subroutine, and incorporation of these derivatives into the entire Jacobian via the chain-rule. In short, the rich structure of a general subroutine offers several complications and opportunities when applying the techniques of automatic differentiation to generate derivative code.

In addition, the hybrid techniques briefly mentioned in chapter 7 can be further developed to reduce the operation count or memory requirement (or both). In addition, the graph can be analyzed to determine an accumulation sequence that is suitable on a parallel computer.

Another interesting application of automatic differentiation is to combine the accumulation of derivatives from the computational graph with interval arithmetic. Interval Jacobians and Hessians are used in many calculations (e.g., interval Newton/generalized bisection techniques for solving nonlinear systems of equations, global nonlinear optimization, etc.). Furthermore, the order in which the interval operations are performed can dramatically effect the "tightness" of the interval extension (recall that, as described in chapter 2, the interval extension of a function overapproximates

the image set of the function). By analyzing the computational graph, it may be possible to extract accumulation sequences that minimize the width of the interval extension of a Jacobian or Hessian.

Appendix A

Derivation of the Homogeneous and Heterogeneous Residue Curve Maps

A.1 Homogeneous Residue Curve Maps

Suppose we have a quantity L of liquid with composition x contained in a vessel. Next, suppose we boil off a quantity V with composition y. The change in the quantity and composition of the liquid is thus, $L - \Delta L$ and $x + \Delta x$. Applying a species mass balance,

$$xL = (x + \Delta x)(L - \Delta L) + yV. \tag{A.1}$$

Since, by an overall mass balance, $V = \Delta L$,

$$\begin{aligned} xL &= (x + \Delta x)(L - \Delta L) + y\Delta L \\ &= xL - x\Delta L + \Delta x L - \Delta x \Delta L + y\Delta L. \end{aligned} \tag{A.2}$$

Rearranging,

$$L\Delta x = x\Delta L - y\Delta L - \Delta x \Delta L. \tag{A.3}$$

Dividing by $\Delta L/L$,

$$\frac{\Delta x}{\Delta L/L} = x - y - \Delta x. \tag{A.4}$$

Taking the limit as $\Delta L/L \to 0$ (and thus, $\Delta x \to 0$),

$$\lim_{\Delta L/L \to 0} \frac{\Delta x}{\Delta L/L} = \frac{dx}{d\xi} = x - y, \tag{A.5}$$

where $d\xi = dL/L$. The quantity ξ can be interpreted as a dimensionless 'warped time' required to evaporate the contents in the vessel. It can also be interpreted as a dimensionless height in a packed column operating at total reflux [38]. Note that the direction of increasing ξ on the residue curves defined by (A.5) corresponds to the direction of increasing temperature along the curve (implicitly defined by the residue curve map).

A.2 Heterogeneous Residue Curve Maps

Suppose we have a mixture of n_L immiscible liquid phases in equilibrium in a vessel. The amount contained in each liquid phase is $L^{(i)}$ with composition $x^{(i)}$, $i = 1, \ldots, n_L$. Suppose we boil off a quantity V with composition y, which is in equilibrium with the n_L liquid phases. Applying a species mass balance,

$$\sum_{i=1}^{n_L} x^{(i)} L^{(i)} = \sum_{i=1}^{n_L} (x^{(i)} + \Delta x^{(i)})(L^{(i)} - \Delta L^{(i)}) + yV. \tag{A.6}$$

The overall mass balance indicates $V = \sum_{i=1}^{n_L} \Delta L^{(i)}$, thus,

$$\sum_{i=1}^{n_L} x^{(i)} L^{(i)} = \sum_{i=1}^{n_L} (x^{(i)} + \Delta x^{(i)})(L^{(i)} - \Delta L^{(i)}) + y \sum_{i=1}^{n_L} \Delta L^{(i)}. \tag{A.7}$$

Expanding the equation above and cancelling terms, we have,

$$0 = -\sum_{i=1}^{n_L} (x^{(i)} \Delta L^{(i)}) + \sum_{i=1}^{n_L} (\Delta x^{(i)} L^{(i)}) - \sum_{i=1}^{n_L} (\Delta x^{(i)} \Delta L^{(i)}) + y \sum_{i=1}^{n_L} \Delta L^{(i)}. \tag{A.8}$$

Now, define the overall liquid composition as

$$x^o L^o = \sum_{i=1}^{n_L} x^{(i)} L^{(i)} \tag{A.9}$$

where $L^o = \sum_{i=1}^{n_L} L^{(i)}$ is the total number of moles of liquid originally present in the vessel. Substituting the overall composition and quantity into (A.8),

$$\Delta x^o L^o = x^o \Delta L^o - y \Delta L^o - \Delta x^o \Delta L^o. \tag{A.10}$$

As above, divide by $\Delta L^o/L^o$ and take the limit as $\Delta L^o/L^o \to 0$ (noting the $\Delta x^o \to 0$),

$$\lim_{\Delta L^o/L^o \to 0} \frac{\Delta x^o}{\Delta L^o/L^o} = \frac{dx^o}{d\xi} = x^o - y. \tag{A.11}$$

The dimensionless 'warped time', ξ, has the same interpretation as in the homogeneous case above.

Appendix B

Proof of Lemma 1

The necessary conditions are trivial. Assume that we are on a k-ary branch and there is a bifurcation point, corresponding to an intersection between the k-ary branch and a $(k+1)$-ary branch, at $\tilde{\xi}$. Without loss of generality, assume that the necessary conditions for a bifurcation point are satisfied for $j = k+1$. By definition, on a k-ary branch the following conditions always hold

(A1.1) $x_j \neq 0$ for all $j = 1, \ldots, k$,

(A1.2) $\alpha_j = 0$ for all $j = 1, \ldots, k$, and

(A1.3) $x_j = 0$ for all $j = k+1, \ldots, n$.

Similarly, along a $(k+1)$-ary branch the following hold

(A1.4) $x_j \neq 0$ for all $j = 1, \ldots, k+1$,

(A1.5) $\alpha_j = 0$ for all $j = 1, \ldots, k+1$, and

(A1.6) $x_j = 0$ for all $j = k+2, \ldots, n$.

Let $\bar{c}_{(i)}(\xi)$ denote an i-ary branch. At the intersection of a k-ary and a $(k+1)$-ary branch:

$$\bar{c}_{(k)}(\tilde{\xi}) = \bar{c}_{(k+1)}(\tilde{\xi}) = (x(\tilde{\xi}), T(\tilde{\xi}), \lambda(\tilde{\xi})). \tag{B.1}$$

Since $\alpha_j = \alpha_j(\bar{c}(\xi))$, on the k-ary branch, (A1.5) and (B.1) imply

$$\alpha_k(\bar{c}_{(k)}(\tilde{\xi})) = \alpha_{k+1}(\bar{c}_{(k+1)}(\tilde{\xi})) = 0.$$

Furthermore, on the $(k+1)$-ary branch, (A1.3) and (B.1) imply

$$x_{k+1}(\tilde{\xi}) = 0.$$

As above, without loss of generality, assume that the necessary and sufficient conditions are satisfied for $j = k+1$. Necessary and sufficient conditions for a bifurcation from a $(k+1)$-ary branch onto a k-ary branch at $\tilde{\xi}$ are

(A1.7) $x_{k+1}(\tilde{\xi}) = 0$,

(A1.8) $dx_{k+1}/d\xi\big|_{\xi=\tilde{\xi}} \neq 0$,

(A1.9) rank $\nabla \bar{F}(\tilde{\xi}) = N - 1$ where $N = n + 1$, and

(A1.10) $\partial \bar{F}/\partial x_{k+1}\big|_{\xi=\tilde{\xi}} \in \mathcal{R}(\nabla_{[k+1]} \bar{F}(\tilde{\xi}))$ where $\nabla_{[k+1]}$ denotes partial derivatives with respect to all variables except x_{k+1}.

To prove sufficiency we must simply show that at there are two distinct tangents at $\bar{c}(\tilde{\xi})$ where $dx_{k+1}/d\xi = 0$ on one tangent and $dx_{k+1}/d\xi \neq 0$ on the other (this second tangent is known at the bifurcation point when the $(k+1)$-ary branch is being tracked). Let $\mathcal{P}_{(i)} \in \mathbb{R}^{(N-1)\times N}$ denote the projection matrix that removes the *i-th* entry of the vector it multiplies ($\mathcal{P}_{(i)}$ is formed by simply removing the *i-th* row of the identity matrix in $\mathbb{R}^{N\times N}$). Let $\bar{G} = \mathcal{P}_{(k+1)}\bar{F}$. The Jacobian matrix of $\bar{G}$ is equal to the Jacobian matrix of $\bar{F}$ with the $(k+1)$-*th* row removed. Since at $\xi = \tilde{\xi}$ the $(k+1)$-*th* row of $\nabla\bar{F}$ is zero,

$$\text{rank } \nabla\bar{G}(\tilde{\xi}) = \text{rank } \nabla\bar{F}(\tilde{\xi}) = N - 1.$$

Furthermore, condition (A1.10) implies

$$\text{rank } \nabla_{[k+1]}\bar{G}(\tilde{\xi}) = N - 1.$$

Thus, $\nabla_{[k+1]}\bar{G}(\tilde{\xi}) \in \mathbb{R}^{(N-1)\times N}$ has the maximal rank of $N-1$. Now, at $\xi = \tilde{\xi}$,

$$\bar{F}(\tilde{\xi}) = 0.$$

Differentiating with respect to ξ,

$$\nabla\bar{F}(\tilde{\xi})\frac{d\bar{c}}{d\xi} = 0$$

and

$$\nabla \bar{G}(\tilde{\xi}) \frac{d\bar{c}}{d\xi} = \nabla_{[k+1]} \bar{G}(\tilde{\xi}) \frac{d\bar{c}_{[k+1]}}{d\xi} + \frac{\partial \bar{G}}{\partial x_{k+1}}\Big|_{\xi=\tilde{\xi}} \frac{dx_{k+1}}{d\xi} = 0 \tag{B.2}$$

where $\bar{c}_{[k+1]}$ denotes the N dimensional vector formed by removing the $(k+1)$-th element from $\bar{c}$ (i.e., $\bar{c}_{[k+1]} = \mathcal{P}_{(k+1)}\bar{c}$). Taking ξ to be arclength along the curve, the following constraint is also imposed

$$\left(\frac{d\bar{c}}{d\xi}\right)^T \left(\frac{d\bar{c}}{d\xi}\right) = \left(\frac{d\bar{c}_{[k+1]}}{d\xi}\right)^T \left(\frac{d\bar{c}_{[k+1]}}{d\xi}\right) + \left(\frac{dx_{k+1}}{d\xi}\right)^2 = 1. \tag{B.3}$$

Setting $dx_{k+1}/d\xi = 0$, we see that (B.2) and (B.3) become

$$\begin{aligned} \nabla_{[k+1]} \bar{G}(\tilde{\xi}) \frac{d\bar{c}_{[k+1]}}{d\xi} &= 0 \\ \left(\frac{d\bar{c}_{[k+1]}}{d\xi}\right)^T \left(\frac{d\bar{c}_{[k+1]}}{d\xi}\right) &= 1 \end{aligned}$$

and uniquely define $d\bar{c}_{[k+1]}/d\xi$. Condition (A1.9) implies that $\alpha_j(\tilde{\xi}) \neq 0$ for all $j = k+2, \ldots, n$ and thus all variables $x_{k+2}, \ldots, x_n$ remain fixed at zero on both branches that intersect at $\tilde{\xi}$.

Appendix C

Proof of Lemma 2

As in Appendix 1, assume that we are currently on a k-ary branch where the following holds:

(A2.1) $x_j \neq 0$ for all $j = 1, \ldots, k$,

(A2.2) $x_j^I \neq 0$ for all $j = 1, \ldots, k$,

(A2.3) $x_j^{II} \neq 0$ for all $j = 1, \ldots, k$, and

(A2.4) $x_j = x_j^I = x_j^{II} = 0$ for all $j = k+1, \ldots, n$.

The necessary condition for a bifurcation from a k-ary heterogeneous branch to a $(k-1)$-ary heterogeneous branch at a point $\bar{c}^o(\tilde{\xi})$ is

$$x_j(\tilde{\xi}) = 0 \text{ for some } j \in \{1, \ldots, k\}. \tag{C.1}$$

Without loss of generality, assume $x_k(\tilde{\xi}) = 0$ on the k-ary branch. The necessary condition follows from the fact that at the bifurcation point,

$$\bar{c}^o(\tilde{\xi})_{(k-1)} = \bar{c}^o(\tilde{\xi})_{(k)} = (x(\tilde{\xi}), x^I(\tilde{\xi}), x^{II}(\tilde{\xi}), T(\tilde{\xi}), s(\tilde{\xi}), \lambda(\tilde{\xi})) \tag{C.2}$$

where the subscript denotes the dimensionality (the number of nonzero mole fraction elements) of the branch.

The necessary condition for a bifurcation from a $(k-1)$-ary branch to a k-ary branch at a point $\bar{c}^o(\tilde{\xi})$,

$$\left[s(\tilde{\xi}) - 1\right] \bar{\gamma}_j^I(\tilde{\xi}) - \left[s(\tilde{\xi}) - \bar{K}_j^I(\tilde{\xi})\right] \bar{\gamma}_j^{II}(\tilde{\xi}) = 0 \text{ for some } j \in \{k, \ldots, n\}, \tag{C.3}$$

follows from (C.2) and the fact that the condition above holds on the k-ary branch when $x_j(\xi)$, $k \leq j \leq n$, crosses zero at the bifurcation point.

Proof of the necessary and sufficient conditions is similar to the homogeneous case. As mentioned above, assume that the intersection between the $(k-1)$-ary and k-ary heterogeneous branches occur where $x_k = 0$. Necessary and sufficient conditions for a transcritical bifurcation point at $\tilde{\xi}$ are:

(A2.5) $x_k(\tilde{\xi}) = 0$,

(A2.6) $dx_k/d\xi\big|_{\xi=\tilde{\xi}} \neq 0$,

(A2.7) rank $\nabla \bar{F}^o(\tilde{\xi}) = N^o - 1$ where $N^o = 3n + 2$, and

(A2.8) $\partial \bar{F}^o/\partial x_k\big|_{\xi=\tilde{\xi}} \in \mathcal{R}(\nabla_{[k]} \bar{F}^o(\tilde{\xi}))$ where $\nabla_{[k]}$ denotes partial derivatives with re-

spect to all variables except x_k.

Again, to prove sufficiency, we must show that two tangents exist at $\tilde{\xi}$, one with $dx_k/d\xi = dx_k^I/d\xi = dx_k^{II}/d\xi = 0$ and the other where these tangent components are nonzero (this second tangent is known at the bifurcation point). Condition (A2.7) implies the dimensionality of the null-space of $\nabla \bar{F}^o(\xi)$ is two. Furthermore, as shown in section 3.1.1, rows k, $k+n$, and $k+2n$ are linearly dependent. Therefore, one of these rows can be removed without changing the rank of the matrix. Let $\bar{G}^o = \mathcal{P}_{(k)}\bar{F}^o$ where $\mathcal{P}_{(k)}$ is defined in Appendix 1. Thus,

$$\text{rank } \nabla \bar{G}^o(\tilde{\xi}) = \text{rank } \nabla \bar{F}^o(\tilde{\xi}) = N^o - 1 \tag{C.4}$$

and $\nabla \bar{G}^o(\tilde{\xi}) \in \mathbb{R}^{(N^o-1)\times(N^o+1)}$ has maximal rank. As before, let $\nabla_{[k]}\bar{G}^o$ denote the Jacobian matrix with respect to all variables except x_k. Condition (A2.8) implies

$$\text{rank } \nabla_{[k]} \bar{G}^o(\tilde{\xi}) = N^o - 1. \tag{C.5}$$

At $\bar{c}^o(\tilde{\xi})$, the following hold:

$$\bar{F}^o(\tilde{\xi}) = 0 \tag{C.6}$$

$$\mathcal{P}_{(k)}\bar{F}^o(\tilde{\xi}) = \bar{G}^o(\tilde{\xi}) = 0. \tag{C.7}$$

Differentiating,

$$\nabla \bar{G}^o(\tilde{\xi}) \frac{d\bar{c}^o}{d\xi} = \nabla_{[k]} \bar{G}^o(\tilde{\xi}) \frac{d\bar{c}^o_{[k]}}{d\xi} + \frac{\partial \bar{G}^o}{\partial x_k}\Big|_{\xi=\tilde{\xi}} \frac{dx_k}{d\xi} = 0. \tag{C.8}$$

Similar to the homogeneous case, equation (C.8) along with the arclength constraint:

$$\left(\frac{d\bar{c}^o_{[k]}}{d\xi}\right)^T \left(\frac{d\bar{c}^o_{[k]}}{d\xi}\right) + \left(\frac{dx_k}{d\xi}\right)^2 = 1 \tag{C.9}$$

and assumption (A2.6) imply there exists two tangent vectors at $\tilde{\xi}$, one with $dx_k/d\xi = 0$ and the other with $dx_k/d\xi \neq 0$. On $\bar{c}^o(\xi)$, the following hold:

$$x_k = \bar{K}^I_k x^I_k \tag{C.10}$$

$$x_k = s x^I_k + (1-s) x^{II}_k \tag{C.11}$$

These imply that both $dx^I_k/d\xi$ and $dx^{II}/d\xi$ have zero and non-zero values at $\tilde{\xi}$.

Appendix D

Proof all binary heteroazeotropes will be computed with the homotopy method

In this appendix, a proof is presented that guarantees all binary heteroazeotropes will be computed using the homotopy method described in this paper. This proof is necessary since if a k-ary ($k > 2$) heteroazeotrope were obtained through a series of bifurcations on lower dimensional heterogeneous branches, we must have at least a binary heterogeneous branch initially. This is not a problem for the homogeneous case since the k-ary azeotropes are obtained through a series of bifurcations originating from a pure component branch which can always be constructed.

Figure D-1 contain a schematic of a xy-diagram for a mixture exhibiting a heteroazetrope. By applying the same analysis as in section 2.1 for the homogeneous

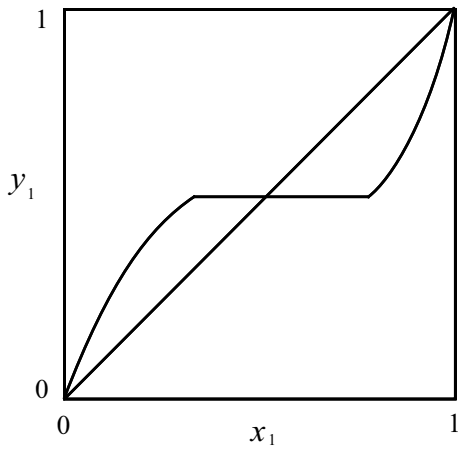

Figure D-1: Schematic of an xy-diagram for a binary mixture exhibiting a heteroazeotrope.

azeotropes, we see that there will always be a bifurcation point on the pure component branch of the lower boiling species in the binary mixture. Furthermore, continuity of the K-value implies that there will always be at least one spurious homogeneous azeotrope corresponding to the heteroazeotrope. It is reasonable to assume that

$$\bar{H}_i^E < \Delta H_i^{vap}, \qquad i = 1, 2, \tag{D.1}$$

and that the following quantities,

$$\left(\frac{\partial \ln \gamma_1}{\partial x_1}\right)_{x_2,T,P} \text{ and } \left(\frac{\partial \ln \gamma_1}{\partial x_2}\right)_{x_1,T,P} \tag{D.2}$$

do not simultaneously vanish at any point in $\mathcal{S}$. Thus, according to theorem 1, the binary homotopy branch passing through the bifurcation point will be diffeomorphic to the real line in the interior of $\mathcal{S}$ and will leave $\mathcal{S}$ through $\lambda = 1$ at the spurious binary azeotrope.

If a binary heteroazeotrope exists then the heterogeneous branch passing through the heteroazeotrope will enter $\mathcal{S}^o$. (The heterogeneous branch will not be tangent at the side of $\mathcal{S}^o$ where $\lambda = 1$ since the heteroazeotrope is necessarily an unstable node[74].) Furthermore, if zero is a regular value of the the binary heterogeneous map in the interior of $\mathcal{S}^o$ then the heterogeneous branch will leave through the side of $\mathcal{S}^o$ defined by $s = 0$ or 1 (provided multiple heteroazeotropy does not occur in $\mathcal{S}^o$) since there is no solution at $x_i = 0$, $i = 1, 2$. Thus, the heterogeneous branch will pass through the spurious binary branch which is reachable from a pure component branch and the binary heteroazeotrope will be computable through the homotopy method.

Appendix E

Rank of Jacobian of Heterogeneous Homotopy Map

This appendix provides a discussion of the rank of the Jacobian of the heterogeneous homotopy map. As described in Chapter 2, zero being a regular value of this homotopy map, along with a couple of other reasonable assumptions on the homotopy path, will guarantee that the homotopy method will compute all heteroazeotropes predicted by the phase equilibrium model of interest.

The Jacobian of the heterogeneous homotopy map is shown in equation (2.57). The first $3n$ columns are partial derivatives with respect to x, x^I, and x^{II}, respectively. The remaining three columns are partial derivatives with respect to T, s, and λ, respectively (assuming pressure is held fixed). The heterogeneous branches are far more complex than their homogeneous counterparts. This complexity is also reflected

in the heterogeneous Jacobian,

$$\nabla \bar{F}^o(x,x^I,x^{II},T,s,\lambda)= \begin{pmatrix} I & \partial \bar{F}_1^o/\partial x^I & 0 & \partial \bar{F}_1^o/\partial T & 0 & \partial \bar{F}_1^o/\partial \lambda \\ 0 & \partial \bar{F}_2^o/\partial x^I & \partial \bar{F}_2^o/\partial x^{II} & \partial \bar{F}_2^o/\partial T & 0 & \partial \bar{F}_2^o/\partial \lambda \\ I & -sI & (s-1)I & 0 & x^{II}-x^I & 0 \\ e^T & 0 & 0 & 0 & 0 & 0 \\ 0 & e^T & 0 & 0 & 0 & 0 \end{pmatrix} \tag{E.1}$$

(see Chapter 2 for a definition of the various terms). In this appendix, the first $3n$ columns of the matrix will be analyzed. First, define

$$\mathcal{A} = \begin{pmatrix} I & \partial \bar{F}_1^o/\partial x^I & 0 \\ 0 & \partial \bar{F}_2^o/\partial x^I & \partial \bar{F}_2^o/\partial x^{II} \\ I & -sI & (s-1)I \end{pmatrix} \tag{E.2}$$

and

$$\bar{\mathcal{A}} = \begin{pmatrix} & \mathcal{A} & \\ e^T & 0 & 0 \\ 0 & e^T & 0 \end{pmatrix}. \tag{E.3}$$

The matrix $\mathcal{A}$ can be further decomposed to

$$\mathcal{A} = \begin{pmatrix} I & -\mathrm{diag}\{\Delta_i\} - \lambda \mathrm{diag}\{x_i^I K_i^I\} G^I & 0 \\ 0 & -\mathrm{diag}\{\bar{\gamma}_i^I\} - \lambda \mathrm{diag}\{x_i^I \gamma_i^I\} G^I & -\mathrm{diag}\{\bar{\gamma}_i^{II}\} - \lambda \mathrm{diag}\{x_i^{II} \gamma_i^{II}\} G^{II} \\ I & -sI & (s-1)I \end{pmatrix} \tag{E.4}$$

$$= \begin{pmatrix} I & -\mathrm{diag}\{\Delta_i\} & 0 \\ 0 & -\mathrm{diag}\{\bar{\gamma}_i^I\} & -\mathrm{diag}\{\bar{\gamma}_i^{II}\} \\ I & -sI & (s-1)I \end{pmatrix} - \lambda \begin{pmatrix} 0 & \mathrm{diag}\{x_i^I K_i^I\} G^I & 0 \\ 0 & \mathrm{diag}\{x_i^I \gamma_i^I\} G^I & \mathrm{diag}\{x_i^{II} \gamma_i^{II}\} G^{II} \\ 0 & 0 & 0 \end{pmatrix} \tag{E.5}$$

$$= \Gamma - \lambda \mathcal{B}. \tag{E.6}$$

By inspection, we see that $\mathcal{N}(\Gamma)$ is spanned by

$$z = \begin{pmatrix} x \\ x^I \\ x^{II} \end{pmatrix}.$$

Furthermore, if $\mathrm{rank}\left[G^I\right] = \mathrm{rank}\left[G^{II}\right] = n - 1$ then $\mathcal{N}(\mathcal{A})$ is also spanned by z. If $\bar{\mathcal{A}}$ is multiplied by z, we have

$$\begin{aligned} \mathcal{A}z &= 0 \\ e^T x &\neq 0 \\ e^T x^I &\neq 0 \end{aligned}$$

when x and x^I lie within the physical composition space. Thus, the first $3n$ columns

of the Jacobian of the heterogeneous homotopy map are independent in the interior of $\mathcal{S}^o$.

Appendix F

Example Problems from Chapter 8

The following equations were developed to illustrate the significant increase in performance of the reverse mode of automatic differentiation applied in a symbolic interpretive environment.

Problem 1

$$\frac{sin(cos(x+y+z)\pi)+1}{1+x^2y^2ze^{xyz}} = 0 \tag{A.1}$$

Problem 2

$$\begin{aligned} y_i &= (\sum_{j=1}^{n}(x_j + ln(x_2\sum_{k=1}^{n}x_k)))(x_i - ln(x_1x_3)) \cdot \\ &\quad (\sum_{j=1}^{n}(x_j + e^{x_3}))x_i \quad \forall i = 1, ..., n \end{aligned} \tag{A.2}$$

where x, $y \in R^n$.

Problem 3

$$x_i(\alpha + \beta - \sum_{j=1}^{n} y_j) = (\sum_{j=1}^{n} y_j z_j) ln(z_i + \alpha + \beta^2) \quad \forall i = 1, ..., n \tag{A.3}$$

$$\alpha = \sum_{j=1}^{n} (\beta + y_j x_j) \tag{A.4}$$

$$\beta = (\sum_{j=1}^{n} (\beta^2 + \gamma z_j)) y_i + i x_i^i \tag{A.5}$$

where α, β, $\gamma \in R$ and x, y, $z \in R^n$.

Problem 4

$$x_i = (\beta \frac{\alpha_1}{0.38} P^{1/0.38} y_i^{1/0.38-1}) z_i / \omega_i - (\beta \alpha_1 P^{1/0.38} y_i^{1/0.38}) \cdot \tag{A.6}$$
$$(\frac{\alpha_1}{0.38} P^{1/0.38} y_i^{1/0.38-1} z_i + \alpha_3 q_i P + \alpha_2 w_i P) / \omega_i^2 + \phi_i \Delta P \quad \forall i = 1, ..., n$$

where α_1, α_2, α_3, β, P, $\Delta P \in R$ and w, x, y, z, ω, $\phi \in R^n$.

Bibliography

[1] J. Abadie and J. Carentier. Generalizations of the wolfe reduced gradient method to the case of nonlinear constraints. In R. Fletcher, editor, *Optimization*, London, 1969. Academic Press.

[2] C. S. Adjiman, S. Dallwig, C. A. Floudas, and A. Neumaier. Global optimization method, αBB, for general twice-differentiable constrained NLPs–I. Theoretical Advances. *Computers and Chemical Engineering*, 22:1137–1158, 1998.

[3] Berit. S. Ahmad and Paul I. Barton. Solvent recovery targeting for pollution prevention in pharmaceutical and specialty chemical manufacturing. *AIChE Symposium Series*, 90(303):59, 1994.

[4] E. L. Allgower and K. Georg. *Numerical Continuation Methods: An Introduction*. Springer–Verlag, Berlin, 1990.

[5] Aspen Technology Inc., Cambridge, MA 02139, USA. *ASPEN PLUS Reference Manual Release 9*, 1995.

[6] B. M. Averick, J. J. Moré, C. H. Bischof, A. Carle, and A Griewank. Computing large sparse jacobian matrices using automatic differentiation. *SIAM J. Sci. Stat. Comput.*, 15:285–294, 1994.

[7] N. Baden and M. L. Michelsen. Computer methods for steady-state simulation of distillation columns. *Institution of Chemical Engineering Symposium Series*, 104:435, 1987.

[8] L. E. Baker, A. C. Pierce, and K. D. Luks. Gibbs energy analysis of phase equilibirum. *Society of Petroleum Engineers Journal*, 22:731–742, 1982.

[9] Paul I. Barton. *The Modeling and Simulation of Combined Discrete/Continuous Processes*. PhD thesis, University of London, London, U.K., May 1992.

[10] M. S. Bazaraa, H. D. Sheralli, and C. M. Shetty. *Nonlinear Programming: Theory and Applications*. John Wiley and Sons, Inc., New York, N.Y., 1993.

[11] L. M. Beda, L. N. Korolev, N. V. Sukkikh, and T. S. Frolova. Programs for automatic differentiation for the machine BESM. Technical Report, Institute for Precise Mechanics and Computation Techniques, Academy of Science, Moscow, USSR, 1959. (In Russian).

[12] J. T. Betts. A sparse nonlinear optimization algorithm. *Journal of Optimization Theory and Applications*, 82(3), 1994.

[13] C. H. Bischof, P. Khademi, A. Bouaricha, and A. Carle. Efficient computation of gradients and jacobians by transparent exploitation of sparsity in automatic differentiation. *Optimization Methods and Software*, 7:1–39, 1996.

[14] Christian Bischof, Alan Carle, George Corliss, Andreas Griewank, and Paul Hovland. ADIFOR – Generating derivative codes from Fortran programs. *Scientific Programming*, 1(1):11–29, 1992.

[15] U. Block and B. Hegner. Development and application of a simulation model for three-phase distillation. *AIChE Journal*, 22:582, 1976.

[16] K. E. Brenan, S. L. Campbell, and L. R. Petzold. *Numerical Solution of Initial Value Problems in Differential–Algebraic Equations.* SIAM, Philadelphia, PA, 1996.

[17] B. P. Cairns and I. A. Furzer. Muticomponent three-phase azeotropic distillation. 1. Extensive experimental data and simulation results. *Industrial and Engineering Chemistry Research*, 29(7):1349–1363, 1990.

[18] B. P. Cairns and I. A. Furzer. Muticomponent three-phase azeotropic distillation. 2. Phase-stability and phase-splitting algorithms. *Industrial and Engineering Chemistry Research*, 29(7):1364–1382, 1990.

[19] B. P. Cairns and I. A. Furzer. Muticomponent three-phase azeotropic distillation. 3. Modern thermodynamic models and multiple solutions. *Industrial and Engineering Chemistry Research*, 29(7):1383–1395, 1990.

[20] M. D. Canon, J. Cullum, and E. Polak. *Theory of Optimal Control and Mathematical Programming.* McGraw-Hill, New York, N.Y., 1970.

[21] F. E. Cellier. *Combined Continuous/Discrete System Simulation by Use of Digital Computers: Techniques and Tools*. PhD thesis, Swiss Federal Institute of Technology, Zurich, Switzerland, 1979.

[22] R. G. Chapman and S. P. Goodwin. A general algorithm for the calculation of azeotropes in fluid mixtures. *Fluid Phase Equilibria*, 85:55–69, 1993.

[23] Bruce W. Char, Keith O. Geddes, Gaston H. Gonnet, Michael B. Monagan, and Stephen M. Watt. *MAPLE Reference Manual*. Watcom Publications, Waterloo, Ontario, Canada, 1988.

[24] C. R. Chavez, J. D. Seader, and T. L. Wayburn. Multiple steady-state solutions for interlinked separation systems. *Industrial and Engineering Chemistry Fundamentals*, 25:566, 1986.

[25] Shui-Nee Chow, John Mallet-Paret, and James A. Yorke. Finding zeroes of maps: Homotopy methods that are constructive with probability one. *Mathematics of Compuation*, 32(143):887–899, 1978.

[26] D. Bruce Christianson. Automatic Hessians by reverse accumulation. *IMA J of Numerical Analysis*, 12:135–150, 1992. Also appeared as Technical Report No. NOC TR228, The Numerical Optimisation Centre, University of Hertfordshire, U.K., April 1990.

[27] Thomas F. Coleman and Jorge J. Moré. Estimation of sparse jacobian matrices and graph coloring problems. *SIAM Journal on Numerical Analysis*, 20(1):187–209, 1983.

[28] George F. Corliss. Overloading point and interval Taylor operators. In Andreas Griewank and George F. Corliss, editors, *Automatic Differentiation of Algorithms: Theory, Implementation, and Application*, pages 139–146. SIAM, Philadelphia, Penn., 1991.

[29] Thomas H. Cormen, Charles E. Leiserson, and Ronald L. Rivest. *Introduction to Algorithms*. The MIT Press, Cambridge, Massachusetts, 1989.

[30] A. R. Curtis, M. J. D. Powell, and J. K. Reid. On the estimation of sparse jacobian matrices. *J. Inst. Maths. Applics.*, 13:117–119, 1974.

[31] J. E. Cuthrell and L. T. Biegler. Improved feasible path optimization for sequential modular simulators – I. The optimization algorithm. *Computers and Chemical Engineering*, 9:257, 1985.

[32] M. L. Cygnarowicz and W. E. Seider. Effect of retrograde solubility on the design optimization of supercritical extraction processes. *Industrial and Engineering Chemistry Research*, 28:1497–1503, 1989.

[33] E. J. Doedel. AUTO: Software for continuation and bifurcation problems in ordinary differential equations. California Institute of Technology, Pasadena, 1986.

[34] M. F. Doherty and G. A. Caldarola. Design and synthesis of homogeneous azeotropic distillation: III. The sequencing of columns for azeotropic and extractive distillations. *Industrial and Engineering Chemistry Fundamentals*, 24:474, 1985.

[35] M. F. Doherty and J. D. Perkins. On the dynamics of distillation processes - I. The simple distillation of multicomponent non-reacting, homogeneous liquid mixtures. *Chemical Engineering Science*, 33:281–301, 1978.

[36] M. F. Doherty and J. D. Perkins. On the dynamics of distillation processes - II. The simple distillation of model solutions. *Chemical Engineering Science*, 33:569–578, 1978.

[37] M. F. Doherty and J. D. Perkins. On the dynamics of distillation processes - III. The topological structure of ternary residue curve maps. *Chemical Engineering Science*, 34:1401–1414, 1979.

[38] D. B. Van Dongen and M. F. Doherty. Design and synthesis of homogeneous azeotropic distillations. 1. Problem formulation for a single column. *Industrial and Engineering Chemistry Fundamentals*, 24:454, 1985.

[39] I. S. Duff and J. K. Reid. MA48, a FORTRAN code for direct solution of sparse unsymmetric linear systems of equations. Technical Report RAL–93–072, Rutherford Appleton Laboratory, Oxon, UK, 1993.

[40] J. R. Fair. Distillation: Whither, not whether. *Institute of Chemical Engineering Symposium Series*, 104:A613–A627, 1987.

[41] W. W. Farr and R. Aris. "Yet who would have thought the old man to have so much blood in him?" — Reflections on the multiplicity of steady–states of the stirred tank reactor. *Chemical Engineering Science*, 41(6):1385–1402, 1986.

[42] William F. Feehery and Paul I. Barton. A differentiation–based approach to dynamic simulation and optimization with high–index differential–algebraic equations. In Martin Berz, Christian Bischof, George Corliss, and Andreas Griewank, editors, *Computational Differentiation: Techniques, Applications, and Tools*, chapter 21. SIAM, Philadelphia, Penn., 1996.

[43] William F. Feehery, John E. Tolsma, and Paul I. Barton. Efficient sensitivity analysis of large–scale differential–algebraic systems. *Applied Numerical Mathematics*, 25:41–54, 1997.

[44] G. B. Ferraris and M. Morbidelli. Mathematical modeling of multistaged separators with mixtures whose components have largely different volatilities. *Computers and Chemical Engineerng*, 6(4):303, 1981.

[45] Z. T. Fidkowski, M. F. Malone, and M. F. Doherty. Computing azeotropes in multicomponent mixtures. *Computers and Chemical Engineering*, 17(12):1141–1155, 1993.

[46] I. A. Furzer. *Distillation for University Students.* I. A. Furzer, Department of Chemical Engineering, University of Sydney: Sydney, NSW, Australia, 1986.

[47] S. Galan, W. F. Feehery, and P. I. Barton. Parametric sensitivity functions for hybrid discrete/continuous systems. *Applied Numerical Mathematics*, 1999. In press.

[48] P. E. Gill, W. Murray, M. A. Saunders, and M. H. Wright. A schur–complement method for sparse quadratic programming. Technical Report Technical Report SOL 87–12, Department of Operations Research, Stanford University, 1987.

[49] Victor V. Goldman, J. Molenkamp, and J. A. van Hulzen. Efficient numerical program generation and computer algebra environments. In Andreas Griewank and George F. Corliss, editors, *Automatic Differentiation of Algorithms: Theory, Implementation, and Application*, pages 74–83. SIAM, Philadelphia, Penn., 1991.

[50] A. Griewank, D. Juedes, and J. Utke. ADOL–C: A package for the automatic differentiation of algorithms written in C/C++. *ACM TOMS*, 22(2):131–167, 1996.

[51] Andreas Griewank. On automatic differentiation. In M. Iri and K. Tanabe, editors, *Mathematical Programming: Recent Developments and Applications*, pages 83–108. Kluwer Academic Publishers, Dordrecht, 1989.

[52] Andreas Griewank and Shawn Reese. On the calculation of Jacobian matrices by the Markowitz rule. In Andreas Griewank and George F. Corliss, editors, *Automatic Differentiation of Algorithms: Theory, Implementation, and Application*, pages 126–135. SIAM, Philadelphia, Penn., 1991. Also appeared as Preprint MCS–P267–1091, Mathematics and Computer Science Division, Argonne National Laboratory, Argonne, Ill., January 1992.

[53] S. P. Han. A globally convergent method for nonlinear programming. *Journal of Optimization Theory and Control*, 22:297, 1977.

[54] S. T. Harding and C. A. Floudas. Locating all heterogeneous and reactive azeotropes. presented at the 1998 AIChE Annual Meeting, November 1998.

[55] S. T. Harding, C. D. Maranas, C. M. McDonald, and C. A. Floudas. Locating all azeotropes in multicomponent mixtures. *Industrial and Engineering Chemistry Research*, 36(160), 1996.

[56] D. A. Harney and N. L. Book. A survey of path tracking algorithms for homotopy continuation methods. 1994.

[57] J. L. Hay, R. E. Crosbie, and R. I. Chaplin. Integration routines for systems with discontinuities. *The Computer Journal*, 17:175–278, 1974.

[58] A. C. Hearn. *REDUCE User's Manual, Version 3.3.* The Rand Corporation, Santa Monica, Calif., 1987.

[59] K. E. Hillstrom. Users guide for JAKEF. Technical Memorandum ANL/MCS–TM–16, Mathematics and Computer Science Division, Argonne National Laboratory, Argonne, Ill., 1985.

[60] L. H. Horsley. Azeotropic data III. Advances in Chemistry Series 116, American Chemical Society, Washinton DC, 1973.

[61] Jim E. Horwedel, Brian A. Worley, E. M. Oblow, and F. G. Pin. GRESS version 1.0 users manual. Technical Memorandum ORNL/TM 10835, Martin Marietta Energy Systems, Inc., Oak Ridge National Laboratory, Oak Ridge, Tenn., 1988.

[62] J. W. Kovach III and W. D. Seider. Heterogeneous azeotropic distillation: Homotopy-continuation methods. *Computers and Chemical Engineering*, 11(6):593, 1987.

[63] Masao Iri. History of automatic differentiation and rounding estimation. In Andreas Griewank and George F. Corliss, editors, *Automatic Differentiation of Algorithms: Theory, Implementation, and Application*, pages 1–16. SIAM, Philadelphia, Penn., 1991.

[64] Masao Iri, T. Tsuchiya, and M. Hoshi. Automatic computation of partial derivatives and rounding error estimates with applications to large-scale systems of nonlinear equations. *J. Computational and Applied Mathematics*, 24:365–392, 1988. Original Japanese version appeared in *J. Information Processing*, 26 (1985), pp. 1411–1420.

[65] B. R. Keeping and C. C. Pantelides. On the implementation of optimisation algorithms for large-scale transient systems on a mimd computer. presented at the 1995 AIChE Annual Meeting, Novemeber 1995.

[66] R. W. Klopfenstein. Zeroes of nonlinear functions. *Journal of the Association of Computing Machinery*, 8:366, 1961.

[67] T. G. Koup, E. H. Chimowitz, A. Blonz, and L. F. Stutzman. An equation analyzer package for the manipulation of mathematical expressions – I. *Computers and Chemical Engineering*, 5:151–159, 1981.

[68] Koichi Kubota. PADRE2, a FORTRAN precompiler yielding error estimates and second derivatives. In Andreas Griewank and George F. Corliss, editors, *Automatic Differentiation of Algorithms: Theory, Implementation, and Application*, pages 251–262. SIAM, Philadelphia, Penn., 1991.

[69] J. R. Leis and M. A. Kramer. Sensitivity analysis of systems of differential and algebraic systems. *Computers and Chemical Engineering*, 9:93–96, 1985.

[70] W. J. Lin. *Application of Continuation and Modeling Methods to Phase Equilibrium*. PhD thesis, University of Utah, 1988.

[71] M. H. Locke, A. W. Westerberg, and R. H. Edahl. Improved successive quadratic programming algorithm for engineering design problems. *AIChE Journal*, 29:871, 1983.

[72] L. L. Lynn, E. S. Parkin, and R. L. Zahradnik. Near–optimal control by trajectory approximation. *Industrial and Engineering Chemistry Fundamentals*, 9:58, 1970.

[73] T. Maly and L. R. Petzold. Numerical methods and software for sensitivity analysis of differential–algebraic systems. *Applied Numerical Mathematics*, 20:57–79, 1996.

[74] Hisayoshi Matsuyama. Restrictions on patterns of residue curves around heterogeneous azeotropes. *Journal of Chemical Engineering of Japan*, 11(6):427–431, 1978.

[75] Hisayoshi Matsuyama and H. Nishimura. Topological and thermodynamic classification of ternary vapor-liquid equilibria. *Journal of Chemical Engineering of Japan*, 10(3):181, 1977.

[76] M. C. McDonald and C. A. Floudas. Decomposition based and branch and bound global optimization approaches for the phase equilibrium problem. *Journal of Global Optimization*, 5, 1994.

[77] M. L. Michelsen. The isothermal flash problem: I. Stability. *Fluid Phase Equilibria*, 9:1–19, 1982.

[78] M. L. Michelsen. The isothermal flash problem: II. Phase split calculation. *Fluid Phase Equilibria*, 9:21–40, 1982.

[79] M. L. Michelsen. Phae equilibria and separation processes: Manual for UNIFLASH. Technical report, Instituttet for Kemiteknik, Danmarks Tekniske Hojskole, Lyngby, Denmark, 1982.

[80] R. E. Moore. A test for existence of solutions to nonlinear systems. *SIAM Journal on Numerical Analysis*, 14(4):611–615, 1977.

[81] R. E. Moore. *Methods and Applications of Interval Analysis*. SIAM, Philadelphia, 1979.

[82] B. A. Murtagh and M. A. Saunders. MINOS/AUGMENTED User's manual. Technical Report SOL 80–14 Systems Optimization Laboratory, Stanford University, 1980.

[83] C. C. Pantelides. Speedup — Recent advances in process simulation. *Computers and Chemical Engineering*, 12:745, 1998.

[84] Taeshin Park and Paul I. Barton. State event location in differential algegraic models. *ACM Transactions on Modelling and Computer Simulation*, 6(2), 1996.

[85] Richard Pavelle and Paul S. Wang. MACSYMA from F to G. *J. Symbolic Computation*, 1(1):69–100, March 1985.

[86] Hoanh N. Pham and Michael F. Doherty. Design and synthesis of heterogeneous azeotropic distillations–I. Heterogeneous phase diagrams. *Chemical Engineering Science*, 45(7):1823–1836, 1990.

[87] Hoanh N. Pham and Michael F. Doherty. Design and synthesis of heterogeneous azeotropic distillations–III. Column sequences. *Chemical Engineering Science*, 45(7):1845–1854, 1990.

[88] Hoanh N. Pham and Michael F. Doherty. Design and synthesis of heterogeneous azeotropic distillations–II. Residue curve maps. *Chemical Engineering Science*, 45(7):1837–1843, 1990.

[89] Peter C. Piela. *ASCEND: An Object-oriented Computer Environment for Modeling and Analysis*. PhD thesis, Carnegie-Mellon University, Pittsburg, PA, 1989.

[90] M. J. D. Powell. An efficient method for finding the minimum of a function of several variables without calculating derivatives. *Computer Journal*, 7:155–162, 1964.

[91] M. J. D. Powell. A fast algorithm for nonlinearly constrained optimization calculations. Technical Report, presented at Dundee Conference on Numerical Analysis, 1977.

[92] Louis B. Rall. *Automatic Differentiation: Techniques and Applications*, volume 120 of *Lecture Notes in Computer Science*. Springer Verlag, Berlin, 1981.

[93] Maurizio Ravaglio and Michael F. Doherty. Dynamics of heterogeneous azeotropic distillation columns. *AIChE Journal*, 36(1):39–52, 1990.

[94] G. V. Reklaitis, A. Ravindran, and K. M. Ragsdell. *Engineering Optimization*. Wiley, New York, N.Y., 1983.

[95] W. C. Rheinboldt and J. V. Burkardt. A locally parameterized continuation process. *ACM Transactions on Mathematical Software*, 9(2), 1983.

[96] B. A. Ross and W. D. Seider. Simulation of three-phase distillation towers. *Computers and Chemical Engineering*, 5:7, 1980.

[97] R. W. H. Sargent, Ding Mei, and Shi Cai Zhu. New developments in large–scale nonlinear programming. presented at ASPENWORLD 94, 1994.

[98] J. D. Seader, M. Kuno, W. J. Lin, S. A. Johnson, K. Unsworth, and J. W. Wiskin. Mapped continuation methods for computing all solutions to gen-

eral systems of nonlinear equations. *Computers and Chemical Engineering*, 14(1):71–85, 1990.

[99] W. D. Seider and L. H. Ungar. Nonlinear systems. *Chemical Engineering Education*, Fall:178–183, 1987.

[100] Rüdiger Seydel. *Practical Bifurcation and Stability Analysis: From Equilibrium to Chaos*. Springer–Verlag, New York, NY, second edition, 1994.

[101] F. A. H. Shreinemakers. Dampfdrucke im system: Benzol, tetrachlorkolenstol. *Z. Phys. Chem.*, 39:440, 1901.

[102] F. A. H. Shreinemakers. Dampfdrucke im system: Benzol, tetrachlorkolenstol und athylalkohol. *Z. Phys. Chem.*, 47:257, 1903.

[103] S. Smale. A convergent process of price adjustment and global newton methods. *Journal of Mathematical Econonmics*, 3:1, 1976.

[104] B. Speelpenning. *Compiling Fast Partial Derivatives of Functions Given by Algorithms*. PhD thesis, Department of Computer Science, University of Illinois at Urbana-Champaign, Urbana-Champaign, Ill., January 1980.

[105] Mark A. Stadtherr, Carol A. Schnepper, and Joan F. Brennecke. Robust phase stability analysis using interval methods. *AIChE Symposium Series 304*, 91:356–359, 1995.

[106] J. G. Stichlmair, J. R. Fair, and J. L. Bravo. Separation of azeotropic mixtures via enhanced distillation. *Chemical Engineering Progress*, 85:63, 1989.

[107] W. J. Stupin and F. J. Lockhart. Thermally coupled distillation — A case history. *Chemical Engineering Education*, 68:71, 1972.

[108] A. C. Sun and W. D. Seider. Mapped homotopy continuation algorithm for global optimization. In *Recent Advances in Global Optimization*. 1990.

[109] C. L. E. Swartz and W. E. Stewart. Finite-element steady state simulation of multiphase distillation. *AIChE Journal*, 33:1977, 1987.

[110] D. W. Tedder and D. F. Rudd. Parametric studies in industrial distillation. *AIChE Journal*, 24:303, 1978.

[111] A. S. Teja and J. S. Rowlinson. The prediction of thermodynamic properties of fluids and liquid mixtures: IV. Critical and azeotropic states. *Chemical Engineering Science*, 28(529), 1973.

[112] K. Tone. Revisions of constraint approximation in the successive QP method for nonlinear programming. *Mathematical Programming*, 26:144, 1983.

[113] S. Vasantharajan and L. T. Biegler. Large–scale decomposition for successive quadratic programming. *Computers and Chemical Engineering*, 12:1087, 1988.

[114] G. Vasudevan, L. T. Watson, and F. H. Lutze. Homotopy approach for solving constrained optimization problems. *IEEE Transactions on Automatic Control*, 36(4):494–498, 1991.

[115] O. M. Wahnschafft, J. W. Koehler, E. Blass, and A. W. Westerberg. The product composition regions of single–feed distillation columns. *Industrial and Engineering Chemistry Research*, 31:2345, 1992.

[116] J. C. Wang and W. B. Whiting. New algorithm for calculation of azeotropes from equations of state. *Industrial and Engineering Chemistry Process Design and Development*, 25, 1986.

[117] Layne T. Watson, Stephen C. Billups, and Alexander P. Morgan. Algorithm 652. HOMPACK: A suite of codes for globally convergent homotopy algorithms. *ACM Transactions on Mathematical Software*, 13(3):281–310, 1987.

[118] R. E. Wengert. A simple automatic derivative evaluation program. *Comm. ACM*, 7(8):463–464, 1964.

[119] S. Widagdo and W. Seider. Azeotropic distillation. *AIChE Journal*, 42(1):96–130, 1996.

[120] S. Widagdo, W. D. Seider, and D. H. Sebastian. Bifurcation analysis in heterogeneous azeotropic distillation. *AIChE Journal*, 35(9):1457–1464, 1989.

[121] S. Widagdo, W. D. Seider, and D. H. Sebastian. Dynamic analysis in heterogeneous azeotropic distillation. *AIChE Journal*, 38(8):1229–1242, 1992.

[122] D. S. H. Wong, S. S. Jang, and C. F. Chang. Simulation of dynamics and phase pattern changes for an azeotropic distillation column. *Computers and Chemical Engineering*, 15(5):325–335, 1991.

[123] J. W.Ponton. The numerical evaluation of analytical derivatives. *Computers and Chemical Engineering*, 6:331–333, 1982.

[124] W. T. Zharov and L. A. Serafimov. Physicochemical fundamentals of distillations and rectification, 1975.