Analysis and Modernization of Mixture Critical Point Calculation Methods

by

Vishnu Jayaprakash

A thesis submitted in partial fulfillment of the requirements for the degree of

Master of Science

in

Petroleum Engineering

Department of Civil and Environmental Engineering
University of Alberta

Abstract

Critical point calculations are a topic of great importance and a fundamental part of classical thermodynamics. While the basis of this field is hundreds of years old, the effective handling of complex, multicomponent fluid mixtures has been an ongoing area of study over the past 50 years. Two major critical point formulations exist (i.e., the root-finding method and the optimization method), each with many modifications and improvements. Despite this, mixture critical point calculation algorithms can be difficult to implement, unreliable, and slow. In recent years, the development of machine learning and modern computer science theory has led to many computational techniques that have the capacity to improve and streamline the mixture critical point calculations. This work investigates the application of modern computational techniques to both root-finding-based and optimization-based mixture critical point calculations. Firstly, we apply the automatic differentiation (AD) technique to both methods. We demonstrate the effectiveness of AD in calculating the thermodynamic derivatives that are involved in critical point calculations. Next, we compare the root-finding methods and the optimization methods in terms of their robustness and accuracy. We find that the root-finding methods are more robust and accurate for simple mixtures. Meanwhile, the global optimization methods are effective in computing the critical points of large, complex mixtures. Finally, we develop a novel procedure that utilizes deep learning models to create predictions of mixture critical points; this procedure can be used to initialize critical point calculations. Our procedure, when implemented into both the root-finding and the global optimization methods, leads to speed and robustness improvements in mixture critical point calculations.

Preface

This thesis is an original work by Vishnu Jayaprakash. No part of this thesis has been previously published. Chapter 3 of this thesis will be presented at the Canadian Chemical Engineering Conference (CSChE 2023).

Acknowledgements

I would like to thank Dr. Huazhou Li for his supervision throughout my master's research at the University of Alberta. Additionally, I would like to thank Dr. Xuehua Zhang at the University of Alberta and Dr. Jae Bem You at the Kyungpook National University, for supporting me in other research projects, which helped me develop the necessary skills for this work. In addition, I extend my sincere thanks to the members of both research groups I have participated in (i.e., Dr. Li's and Dr. Zhang's research groups); their group members were a constant source of guidance and inspiration throughout my studies.

Also, I greatly acknowledge the SciNet HPC Consortium as the computations in this work were performed on their Niagara supercomputer. SciNet is funded by: the Canada Foundation for Innovation; the Government of Ontario; Ontario Research Fund - Research Excellence; and the University of Toronto. Without access to the computational power from SciNet, this work would not have been possible.

Finally, I greatly acknowledge the Natural Sciences and Engineering Research Council (NSERC) of Canada for a Discovery Grant and Alberta Innovates for an NSERC Alliance – Alberta Innovates Advance Program grant to Dr. H. Li.

Table of Contents

List of Tables

List of Figures

List of Symbols

Superscripts and Subscripts

$\square^*$	Normalized Quantity
$\square^{DEP}$	Departure Function
$\square^k$	Iteration Index
$\square_c$	Critical Property
$\square_r$	Reduced Quantity
$\square_{i,j,k}$	Component Indices
$\square_{pred}$	Value Predicted by Machine Learning

Latin

A	Helmholtz Free Energy or Nondimensionalized Attraction Factor	J
a	Attraction Term in Cubic Equation of State	$\frac{Pa*m^6}{mol^2}$
B	Nondimensionalized Co-volume	
b	Co-volume in Cubic Equation of State	$\frac{m^3}{mol}$
C	Nondimensionalized Volume Translation Correction	
c	Volume Translation Correction	$\frac{m^3}{mol}$
D	Damping Factor	
d	Modified Tangent Plane Distance Function	
f	Fugacity	
G	Gibbs Free Energy	J
g^*	Gibbs' First Critical Point Criterion	
g_i	Gibbs' Second Critical Point Criterion	
H	Hessian Matrix	
h	Gibbs' Third Critical Point Criterion	

J	Jacobian Matrix	
k_d	Number of Iterations for Damping	
k_{ij}	Binary Interaction Coefficient between Component i and Component j	
N	Number of Components	
n	Mole Number	mol
P	Pressure	Pa
Q	Damping Coefficient	
R	Universal Gas Constant	$8.314463\ \frac{J}{mol*K}$
T	Temperature	K
u^*	Minimum Eigenvector of Hessian Matrix	
v	Molar Volume	$\frac{m^3}{mol}$
x	Mole Fraction	
Z	Compressibility Factor	

Greek

Δ	Shift Operator	
δ	Differential or Differential Error	
$\delta_{1,2,3,4}$	Equation of State Parameters	
μ	Chemical Potential	$\frac{J}{mol}$
∇	Gradient Operator	
ϕ	Fugacity Coefficient	
ρ	Molar Density	$\frac{mol}{m^3}$

Abbreviations

AD Automatic Differentiation.

BFS Brute Force Search.

DE DE.

DGF Dirichlet Generator Function.

DNN Deep Neural Network.

DNR Damped Newton-Raphson.

EOS Equation of State.

MARE Mean Absolute Relative Error.

NR Newton-Raphson.

PC-SAFT Perturbed Chain-Statistical Associating Fluid Theory.

PR Peng-Robinson.

RMSRE Root Mean Squared Relative Error.

SRK Soave-Redlich-Kwong.

Chapter 1

Introduction

1.1 Background

Thermodynamics is the area of study that explores the fundamental physics behind how chemical species interact with each other. As such, it is a key area of research when exploring new technologies and concepts in chemical and petroleum engineering. Within steady-state thermodynamics, studies on phase behaviour investigate how pure fluids or fluid mixtures exist under given temperature and pressure conditions. Understanding the phase behaviour of pure fluids or fluid mixtures is crucial to the study of their transport and storage in bulk spaces and porous media. Therefore, in petroleum engineering, accurate and nuanced descriptions of phase behavior play a fundamental role in the development of effective techniques to exploit the valuable reservoir fluids [1].

A key concept in phase behavior is the critical point (sometimes referred to as critical state). This point corresponds to a temperature and a pressure at which the interface between any two phases disappears. The most commonly seen critical point is the vapour-liquid critical point, where a vapour phase and a liquid phase merge into a uniform phase. Such a point can be used as a key landmark for the construction of phase envelopes (i.e., graphical representations of the states at which phase transition occurs) [2].

The critical points of pure substances can be experimentally measured using well-

established techniques. However, it is more challenging to measure the critical points of fluid mixtures. Additionally, it is a non-trivial task to predict the critical points of fluid mixtures through theoretical calculations. Fortunately, in the last 50 years, several critical point calculation methods have been established in the literature based on fundamental thermodynamic relationships. Several technical difficulties are associated with the critical point calculations for fluid mixtures. For example, the computational cost increases exponentially with the number of components, and the algorithms may occasionally not converge. Many researchers have attempted to mitigate or resolve these technical issues by using new formulations or computational techniques. Despite this, there is still room for further improvement of the existing critical point calculation algorithms. Research that improves the critical point calculations also has wider significance to phase envelope constructions and minimum miscibility pressure calculations.

1.2 Literature Review

The determination of critical points has been a key task in the phase behaviour modeling of pure substances and mixtures. In the field's seminal paper, "On the Equilibrium of Heterogeneous Substances", Gibbs established that at a critical point, a fluid mixture must obey two restrictions [3]. These conditions are related to the minimization of Gibbs free energy and are referred to in the literature as Gibbs' critical point criteria [4]. By solving the equations that result from Gibbs' critical point criteria, the critical points of pure substances are obtainable. Despite this major step forward, scientists at the time were still not able to accurately apply Gibbs' critical point criteria to fluid mixtures, due to the difficulty of describing multicomponent systems.

In fact, as recently as 1973, Spencer et al. [1] questioned the practicality of extending Gibbs' critical point criteria to mixture critical points. However, with the advent of their accurate two-constant cubic equation of state (EOS) [5], Peng and Robinson

were able to utilize Gibbs' critical point criteria to rigorously calculate the critical points of hydrocarbon mixtures [6]. Since this technique was established, many authors have made improvements and optimizations to the critical-point calculation algorithm, such as simplifying the formulations [4, 7], improving the convergence behavior [8, 9], and allowing for the nonexistence of critical points to be determined [9]. The method presented by Peng and Robinson [6], based on Gibbs' critical point criteria, leads to a system of non-linear equations that require solving via root-finding methods.

The principle root-finding method utilized is the Newton-Raphson (NR) method. It is easy to implement and works for a wide variety of mathematical problems. Many modified techniques have been developed to circumvent some of the convergence issues of NR methods, such as NR-bisection hybrid methods, interval Newton methods, and Krylov-based Newton methods [9–11]. Many of these methods have been applied to mixture critical point calculations with great success for simple mixtures [4, 7]. Additionally, Michelsen presented an interpolation method for approximating the mixture critical points via working out the full phase envelope near the critical region using NR-type calculations [2].

However, fluid mixtures with a large number of components can still cause potential convergence issues when using NR-type methods. Convergence problems typically arise in NR-type methods when an initial guess is not sufficiently close to the true root or when the function has erratic behaviour near the root [12]. Additionally, the standard NR method can only produce one critical point per initial guess and it is difficult to determine whether the lack of convergence is a result of a nonexistent critical point. Stradi developed a modified NR method using interval mathematics that could calculate all mixture critical points on a given range and confirm nonexistence [9]. This method addresses the major concerns of the NR-based mixture critical point calculations; however, its computational complexity becomes too great for large fluid mixtures. Dimitrakopolous et al. used a damped NR (DNR) calculation procedure to

calculate mixture critical points [8]. This method has robust convergence behaviour, even for mixtures of greater than 10 components. Still, this method does not handle multiple critical points or nonexistent critical points. Throughout the development of mixture critical point calculations, no universal NR-type method has emerged that is robust and fast, works for large mixtures, and can handle multiple or nonexistent critical points.

Viewing the critical point problem from a different perspective, Henderson et al. reformulated Gibbs' critical point criteria into an optimization problem [13]. This methodology utilizes a modified tangent plane distance function, which is related to Gibbs' critical point criteria to create an objective function that can be globally minimized to find the state with the minimum Gibbs free energy (i.e., the critical point). Global optimization techniques have been applied to a wide variety of phase behavior calculation problems [14]. Since the advent of Henderson et al.'s method, tunneling algorithms [15], simulated annealing [13], and differential evolution (DE) [16] have all been applied to mixture critical point calculations. These techniques have shown accurate calculation of critical points across a wide variety of mixtures, including complex mixtures. With appropriate modifications, global optimization techniques are capable of determining multiple critical points simultaneously or confirming the nonexistence of a critical point. Using robust global optimization techniques does have a high computational resource demand and is considerably slower than the NR-type calculations. Also, Henderson et al.'s formulation involves the third derivative of the fugacity coefficient, in contrast to the NR-type methods which only require the second derivative [13]. This results in greater mathematical complexity and subsequent implementation difficulties, which limits the current potential of global optimization as a method for calculating mixture critical points.

1.3 Motivation

Although many algorithms have been developed for mixture critical point calculations, they are still often limited by the fundamental nature of the problem. Various techniques have been developed to improve the robustness [8] and convergence behaviour of the mixture critical point calculations algorithms [9], and even allow for the calculation of multiple critical points [16]. Despite this, further development and improvement of critical point calculations is possible, especially using modern computation techniques.

Firstly, when used for mixture critical point calculations, the root-finding methods require second-order derivatives, while the optimization methods require third-order derivatives. This results in many authors relying on less accurate numerical derivatives to approximate the higher order derivatives that are involved in mixture critical point calculations. Additionally, complex EOSs, such as the perturbed-chain statistical associating fluid theory (PC-SAFT) EOS, are under-explored in the literature when investigating these algorithms due to the complexity of their higher-order derivative calculations. With numerical derivatives having reduced accuracy and analytical derivatives sometimes being prohibitively complex to implement, there is a need for an accurate but simpler alternative. For this, we consider automatic differentiation (AD), a method utilized to great effect in machine learning applications that can potentially be used in critical point calculations.

Next, root finding and optimization techniques in the literature have largely been investigated separately. This means that the root-finding-based literature often uses completely different fluid data, compositions, and implementations from the optimization-based literature. Therefore, there is a need for a comprehensive and direct comparison between the two techniques, as well as a detailed exploration of the strengths and limitations of each method. In particular, applying both methods to a wide variety of fluid mixtures over a wide range of compositions will provide further insights into

both techniques.

Finally, the convergence of both root finding and optimization methods for critical point calculations can be heavily dependent on tuning parameters or initial conditions. Furthermore, the rate of convergence of these techniques can be very slow for large, complex fluid mixtures. Addressing these concerns to allow for the development of algorithms that are robust and fast is a key component of compositional simulations. Machine learning is a powerful tool that has a wide variety of applications. Generally, the "black-box" nature of machine learning makes it disconnected from fundamental scientific theory. However, with careful implementation, machine learning can be leveraged to improve the robustness and speed of mixture critical point calculations, without compromising the analytical rigor of traditional calculation techniques.

1.4 Thesis Objectives

This thesis aims to apply techniques that utilize machine learning and related techniques to improve the existing methods dedicated to the calculation of mixture critical points. The research objectives of this thesis are:

1. Demonstrate the application of AD to accurately compute the thermodynamic derivatives involved in mixture critical point calculations.

2. Identify the strengths and limitations of the root-finding and optimization methods for critical point calculation in terms of robustness and accuracy.

3. Investigate the ability of deep neural networks (DNN) to generalize mixture critical point calculations from a sample of compositions to all possible compositions.

4. Utilize the trained DNN models to provide appropriate initializations and bounds that can help improve convergence speed and robustness of both root-finding-based and optimization-based critical point calculation methods.

1.5 Thesis Structure

In this work, we focus on the incorporation of modern computational techniques to both the root-finding and optimization procedures for the calculation of mixture critical points. Namely, we incorporate two computational techniques: AD and deep learning to simplify, accelerate, and improve the robustness of both procedures. In the first chapter, AD is introduced as a computational method for the calculation of "near-analytical" derivatives. This technique is applied to a DNR root-finding algorithm as well as a DE-based global optimization procedure for mixture critical point calculations. We show that AD incurs insignificant errors when compared to analytical derivatives. In addition to avoiding tedious analytical differentiation, we also find the technique can lead to performance improvements in the DE algorithm.

In the second chapter, having implemented both the DNR and DE algorithms in their analytical and AD forms, we make some comparisons between the two algorithms. First, the robustness of each algorithm is determined by attempting critical point calculations for many compositions of various fluid mixtures. Next, deviations from analytical results are calculated for each algorithm. Finally, mixtures with multiple critical points are tested. These results reveal the advantages and disadvantages of each methodology.

In the third chapter, we develop a DNN model for the prediction of mixture critical points. This is done by using a generator function to create a random sample of compositions for a given mixture. Mixture critical point calculations are then done via the conventional methods to build a training dataset. This training dataset is used to create a predictive model that works for a given mixture with any composition. We find that the predictions can be fairly accurate for a wide range of mixtures. Furthermore, we utilize these DNN predictions to provide highly accurate initial guesses to the DNR algorithm and tight bounding boxes to the DE algorithm. These machine-learning-based accelerated calculations are shown to have much faster convergence

behavior as well as increased robustness, especially for global optimization methods.

We believe these three chapters will provide meaningful contributions to the modernization of mixture critical point calculations. Firstly, we establish AD as a valid technique for the calculation of thermodynamic derivatives. This allows for more complex and accurate EOSs to be used without the burden of tedious or impossible analytical derivations. Secondly, while general knowledge of the limitations of the root-finding and global optimization methods is known, clear and specific demonstration of these issues will allow for the selection of the most appropriate critical point calculation technique for a given application. Finally, utilizing machine learning to accelerate analytical calculations simplifies and speeds up the critical point calculations for complex fluid mixtures without loss of accuracy.

Chapter 2

Theoretical Basis of Critical Point Calculation

2.1 The Gibbs' Critical Point Conditions

The theoretical model for mixture critical point calculations revolves around the minimization of Gibbs free energy. This was first described by Gibbs for single component fluids [3]. He presented two critical point criteria based on thermodynamic theories that were necessary for Gibbs free energy minimization. For many years, extending these equations to multicomponent mixtures was attempted; however, the results were inaccurate. When Peng and Robinson developed their two-constant equation of state (EOS) [5], they were finally able to demonstrate multicomponent mixture critical point calculations through the use of appropriate mixing rules and a sufficiently accurate EOS model [6].

Heidemann and Khalil later simplified the forms produced by Gibbs by rewriting them in terms of Helmholtz free energy [4] as presented in Equation (2.1) to Equation (2.3).

$$\sum_{k}^{N}\sum_{j}^{N}\sum_{i}^{N}\left(\frac{\partial^3 A}{\partial n_k \partial n_j \partial n_i}\right)\Delta n_i \Delta n_j \Delta n_k = 0 \tag{2.1}$$

$$\sum_{j}^{N}\left(\frac{\partial^2 A}{\partial n_j \partial n_i}\right)\Delta n_j = 0 \tag{2.2}$$

$$\sum_{i}^{N}\Delta n_i^2 - 1 = 0 \tag{2.3}$$

Equation (2.1) produces one equation per component in a multicomponent fluid mixture. Equation (2.1) to Equation (2.3) lead to a system of N+2 nonlinear equations that need to be solved. The Helmholtz free energy of a system is known to be related to the fugacity coefficient. Therefore, Equation (2.1) and Equation (2.2) can be rewritten as Equation (2.4) and Equation (2.5) [8], respectively.

$$g^* = \sum_{k}^{N}\sum_{j}^{N}\sum_{i}^{N}\left(RT\frac{\partial^2 \ln f_i}{\partial n_k \partial n_j}\right)\Delta n_i \Delta n_j \Delta n_k = 0 \tag{2.4}$$

$$g_i = \sum_{j}^{N}\left(RT\frac{\partial \ln f_i}{\partial n_j}\right)\Delta n_j = 0 \tag{2.5}$$

$$h = \sum_{i}^{N}\Delta n_i^2 - 1 = 0 \tag{2.6}$$

From these equations, utilizing an EOS, an expression for fugacity can be substituted to produce the full system of equations. As described in Appendix A, an expression of fugacity can be derived from any EOS in a compressibility-factor-explicit form. The natural logarithm of the fugacity coefficient of a component in a mixture ($ln(\phi_i)$) is defined as the chemical potential departure function [17]. This, in turn, is obtained by taking the compositional derivative of the Gibbs departure function for that component ($\frac{\delta G^{DEP}}{\delta n_i}$). The Gibbs departure function for an EOS is defined as a function of compressibility factor (Z). Therefore, using the Z-explicit form of any EOS, the Gibbs departure function can be determined and used to obtain an expression for the natural logarithm of fugacity coefficient. For a generalized two-parameter

cubic EOS, the fugacity coefficient equation is shown in Equation (2.7). This can be then used to determine fugacity from the definition of fugacity coefficient [17].

$$\ln \varphi_i = \beta_i(Z-1) - \ln(Z-B) - \frac{A}{B}\ln(\frac{Z+\delta_1 B}{Z+\delta_2 B})(\frac{2\alpha_i - \beta_i}{\delta_1 - \delta_2}) \tag{2.7}$$

For the analytical calculation of fugacity coefficient, the roots of the Z polynomial are required. Beginning with the Z-explicit form of a generalized two-constant cubic EOS, algebraic rearrangement gives the Z polynomial Equation (2.8). This form is presented using non-dimensionalized variables, with full derivations and definitions available in Appendix A. The procedure for solving the Z polynomial is outlined in Figure 2.1.

$$\begin{aligned} Z &= \frac{1}{1-b\rho} - \frac{a\rho}{RT}\frac{1}{(v+\delta_1 b)(v+\delta_2 b)} \\ &\text{with EOS parameters} \quad \delta_1, \quad \delta_2 \\ 0 &= Z^3 + [(\delta_3 - 1)B - 1]Z^2 \\ &+[A - (\delta_3 - \delta_4)B^2 - \delta_3 B]Z - AB - \delta_4(B^2 + B^3) \\ &\text{where} \quad \delta_3 = \delta_1 + \delta_2, \quad \delta_4 = \delta_1 * \delta_2 \end{aligned} \tag{2.8}$$

Firstly, a cubic discriminant is calculated to determine the nature of roots. A negative discriminant implies a single real root, which can be calculated using the single root formula in Appendix A. A positive discriminant implies three distinct real roots, which are calculable as described in Appendix A. When multiple real roots are obtained, the largest is selected for usage in the optimization algorithms in accordance with the convention used by Henderson et al. [13]. However, any of the roots can be used due to the region of the critical point being far away from the three-root region [13]. When the discriminant is strictly positive, there is at least one repeated real root. A further check of the coefficients involved in the discriminant is done ($c_2^2 = 3c_1c_3$). When this check passes, there is a triple multiplicity root. Otherwise,

there are a double multiplicity root and a distinct single multiplicity root, both of which can be calculated subsequently [18].

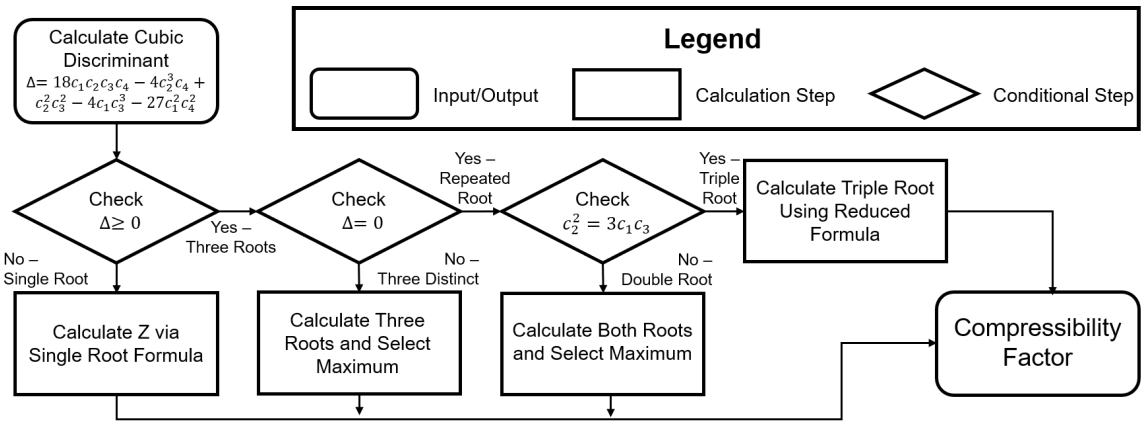

Figure 2.1: Algorithmic flowchart showing the procedure of calculating the compressibility factor (Z) based on a general two-constant cubic EOS. The discriminant of the cubic polynomial is calculated based on component critical properties. A series of checks are run on the parts of the discriminant to determine the nature of the roots (the number of unique and repeated roots as well as the number of real roots). When the nature of the polynomial roots is determined specific calculations can be done to obtain the cubic roots. The largest root of Z is used in critical point calculations.

As a final note, some formulations require fugacity rather than fugacity coefficient. For this, the following two relations are utilized [17].

$$\frac{\partial \ln \varphi_i}{\partial n_j} = \frac{\partial \ln f_i}{\partial n_j}, i \neq j$$

$$\frac{\partial \ln \varphi_i}{\partial n_i} = \frac{\partial \ln f_i}{\partial n_i} + \frac{1}{n_i}, i = j$$

2.2 Root Finding Techniques for Critical Point Calculation

Root finding methods can be broadly categorized into those that are derivative-based and those that are non-derivative-based [19]. Derivative-based methods tend to have much greater reliability and speed than non-derivative methods, given the appropriate usage. The majority of derivative-based methods are related to the Newton-Raphson

(NR) method [12]. In this work, we utilize a damped NR (DNR) procedure, originally presented by Dimitrakopolous et al., as a model root-finding algorithm [8]. In this algorithm, a system of N+2 nonlinear equations, in the form presented in Equation (2.9), is solved for its root:

$$\begin{bmatrix} \frac{\partial g_1}{\partial T_c} & \frac{\partial g_1}{\partial v_c} & \frac{\partial g_1}{\partial \Delta n_1} & \cdots & \frac{\partial g_1}{\partial \Delta n_N} \\ \frac{\partial g_2}{\partial T_c} & \frac{\partial g_2}{\partial v_c} & \frac{\partial g_2}{\partial \Delta n_1} & \cdots & \frac{\partial g_2}{\partial \Delta n_N} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ \frac{\partial g_N}{\partial T_c} & \frac{\partial g_N}{\partial v_c} & \frac{\partial g_N}{\partial \Delta n_1} & \cdots & \frac{\partial g_N}{\partial \Delta n_N} \\ \frac{\partial g^*}{\partial T_c} & \frac{\partial g^*}{\partial v_c} & \frac{\partial g^*}{\partial \Delta n_1} & \cdots & \frac{\partial g^*}{\partial \Delta n_N} \\ 0 & 0 & \frac{\partial h}{\partial \Delta n_1} & \cdots & \frac{\partial h}{\partial \Delta n_N} \end{bmatrix} \cdot \begin{bmatrix} \Delta T_c \\ \Delta v_c \\ \Delta \Delta n_2 \\ \vdots \\ \vdots \\ \Delta \Delta n_N \end{bmatrix} = - \begin{bmatrix} g_1(x^k) \\ g_2(x^k) \\ \vdots \\ \vdots \\ g_N(x^k) \\ g^*(x^k) \\ h(x^k) \end{bmatrix} \tag{2.9}$$

$$J\Delta x \quad = -F(x) \text{ with the update formula,}$$

$$x^{k+1} = x^k + D\Delta x^k, \text{ for some damping constant } D.$$

g_i, g^* and h are described in Equation (2.4), Equation (2.5) and Equation (2.6), respectively. The exact forms of these functions can be calculated for a given EOS by expansion and non-dimensionalization. The work of Dimitrakopolous et al. presents derivations for these forms for a generalized two-parameter cubic EOS [8]. As per their work, a damping constant is defined by Equation (2.10).

$$D = \frac{1}{1 + Q * e^{-0.5k_d}} \tag{2.10}$$

k_d is the iteration number and Q is the damping coefficient. Q is set to 518 for non-binary mixtures which, from the numerical studies presented by Dimitrakopolous et al. [8], is the best-performing value. In mixtures where convergence at Q=518 is not obtainable, particularly those with more than 10 components, Q=700 is tried.

The implementation of the DNR technique comes directly from the work of Dimitrakopolous et al. and is shown in Figure 2.2. An initial guess for the input vector

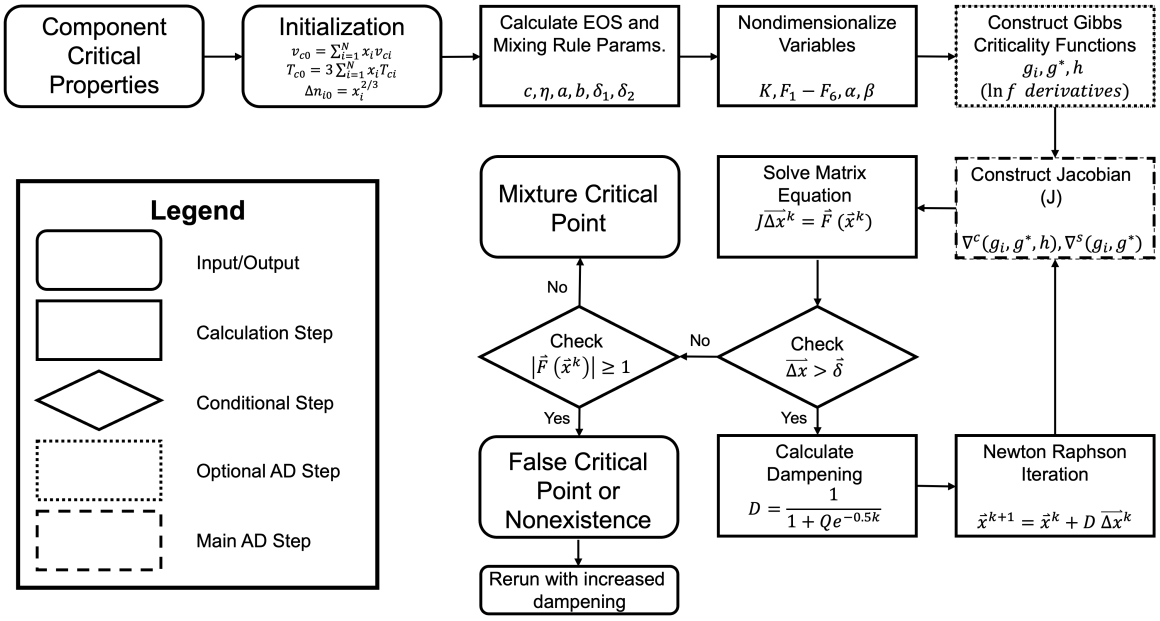

Figure 2.2: Algorithmic flowchart of DNR procedure used in this work. Component critical properties are used to compute the initial guess of the critical property vector and nondimensionalized EOS parameters. From there, the system of equations for the Gibbs critical point criteria is constructed and its Jacobian is taken. The Jacobian is used to solve a matrix equation that obtains the next iterate for the critical property vector. This iterate is damped and the next Jacobian matrix equation is constructed. When a tolerance is reached, the Gibbs critical point criteria is rechecked to determine if the answer is a correctly converged critical point.

is taken using Kay's mixing rule [20]. From this, EOS parameters and subsequent non-dimensionalizations are precomputed. Then, the necessary expressions in the Gibbs critical point criteria and its Jacobian are computed using the forms of Equation (2.4) derived by Dimitrakopolous et al. [8]. Once the Jacobian is obtained, the input vector is iterated upon by solving the matrix equation. Each iteration is damped based on the selected Q value until the tolerance (10^{-4} for T_c/v_c and 10^{-8} for Δn) is reached. When an input vector passes the tolerance check, the function value of the Gibbs critical point criteria is checked to determine if they are far from zero, which would indicate a false or nonexistent root. If this check is passed, then pressure can be calculated from the EOS and the critical point can be returned. There is a maximum of 30 iterations allowed for simple mixtures and 50 for complex mixtures (10+ components). If the maximum number of iterations is reached, it is possible

that the initial guess and damping factor are poorly suited for the mixture or that no critical point exists.

2.3 Global Optimization Formulation of the Gibbs Critical Point Conditions

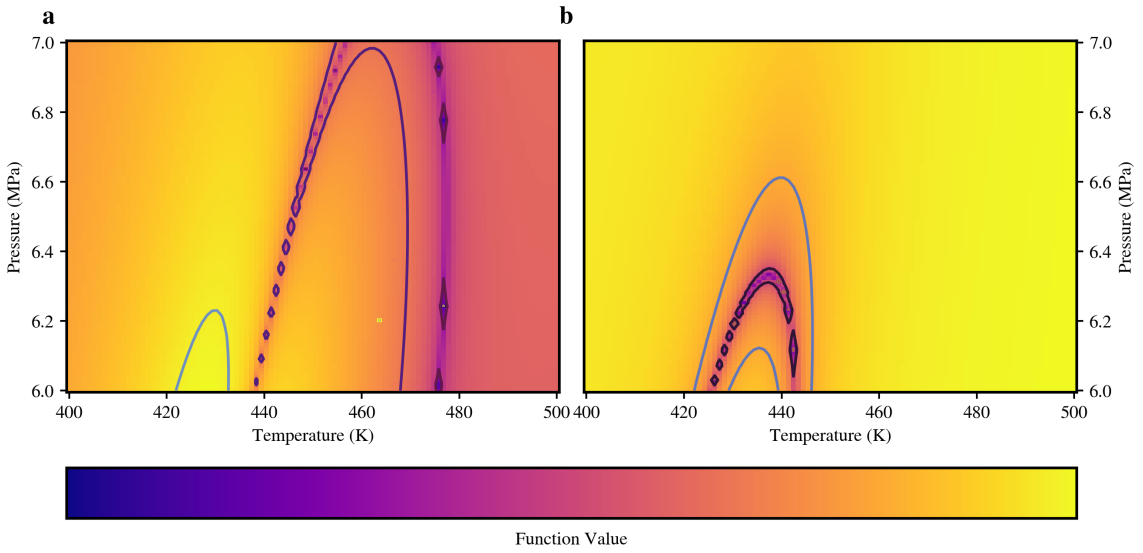

Figure 2.3: Example contour plot over temperature and pressure of the two components of the objective function (Equation (2.13)) for a ternary mixture (C2/C4/C7): a) c component corresponding to Equation (2.4); b) q component corresponding to Equation (2.5).

The global optimization methodology for mixture critical point calculations was developed by Henderson et al. [13]. This formulation uses the modified phase stability test based on the Gibbs tangent plane criterion presented by Equation (2.11).

$$d = \sum_{i=1}^{N} x_i[\mu_i(\vec{x}) - \mu_i(\vec{z})] \geq 0 \tag{2.11}$$

This considers the stability of a theoretical phase at $\vec{x}$, by comparing it to the global composition $\vec{z}$ on the Gibbs tangent plane. The third critical point condition Equation (2.6) imposes the restriction that the mole fractions must add to 1. This

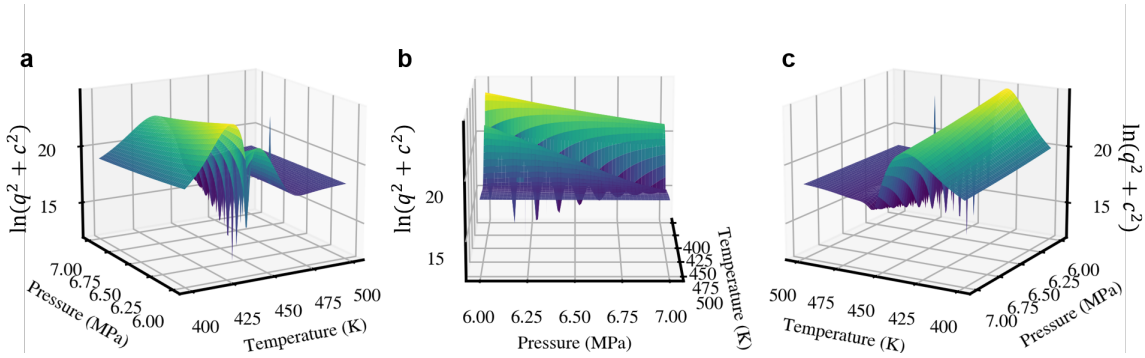

Figure 2.4: Natural logarithm of the objective function (Equation (2.13)) for a ternary mixture (C2/C4/C7) plotted as a function of temperature and pressure. The three subplots represent 3D views from three perspectives. The global minimum represents the critical point of this fluid mixture at this composition.

produces the form Henderson et al. referred to as the modified stability test condition (Equation (2.12)) [13].

$$d(\vec{x}) = \sum_{i=1}^{N-1} x_i \Big\{ [\mu_i(\vec{x}) - \mu_i(\vec{z})] - [\mu_r(\vec{x}) - \mu_r(\vec{z})] \Big\} + [\mu_r(\vec{x}) - \mu_r(\vec{z})] \geq 0 \tag{2.12}$$

By considering the Taylor expansion of the modified stability test function at the input composition, the phase stability problem can be recast to the following equation:

$$q = \frac{\nabla^2 d(x) \cdot \vec{u}^2}{2} \geq 0$$

Based on the Lagrange multiplier theory, Henderson et al. identified the vector u in these forms as the minimizing eigenvector of the modified stability test function in state space, $\vec{u}^*(T,P)$ [13]. From this analysis, Henderson et al. constructed an objective function that could be minimized to find the critical point, where both critical point conditions should be satisfied, as shown in Equation (2.13) [13].

$$\begin{aligned} q(x,T,P) &= \nabla^2 d(x) \cdot \vec{u}^{*^2}(T,P) \\ c(x,T,P) &= \nabla^3 d(x) \cdot \vec{u}^{*^3}(T,P) \\ F(x,T,P) &= q^2 + c^2 \end{aligned} \tag{2.13}$$

The objective function is a superposition of the functions q and c which are linked to Equation (2.4) and Equation (2.5), respectively. Both of these functions create an envelope of local minima on the state space, as shown in Figure 2.3. The intersection of these two envelopes will have a value near zero. This point, having a zero value for the minimum eigenvalue and its derivative, is identified as the critical point. Additionally, Henderson et al. proved that the second order derivatives of the modified tangent plane distance function could be obtainable through a relation with the fugacity coefficient (Equation (2.14)) [13].

$$\begin{aligned} \nabla^2 d = \frac{\delta d(x)}{\delta x_i \delta x_j} = \frac{\delta \mu_i}{\delta x_j} - \frac{\delta \mu_r}{\delta x_j} \\ \frac{\delta \mu_i}{\delta x_j} = RT \left[\frac{\delta ln(\phi_i)}{\delta x_j} + \frac{\delta \ln(x_i P)}{\delta x_j} \right] \end{aligned} \tag{2.14}$$

The objective function is highly jagged with many local minima and local maxima in the vicinity of the critical point, as demonstrated in Figure 2.4. This leads to local optimization algorithms failing to converge to the correct minimum. Therefore, stochastic global minimization methods are typically utilized to identify the critical point [13, 16]. For our calculations, a DE (DE) algorithm is utilized. The initial values of critical pressure and critical temperature from Kay's mixing rule are used to obtain search bounds for DE [20].

Figure 2.5 shows the overall procedure of global optimization used in this work. This algorithm contains three subroutines: the cubic root solver as described in Figure 2.1; the inverse iteration algorithm dedicated to the calculation of the minimal eigenvalue; and the DE algorithm for the minimization of the objective function. Firstly, the component critical properties are used to calculate the EOS parameters. Then, the cubic discriminant is calculated and used to determine the roots of the compressibility polynomial, as previously described. The calculated root and non-dimensionalized EOS parameters are used to construct the log-fugacity expressions. These expressions are then differentiated and then related to the elements of the Hes-

sian matrix via Equation (2.14). The minimal eigenvalue of the Hessian matrix is then calculated using the inverse iteration technique. For the cubic form of the objective function, the element-wise derivative of the Hessian matrix is taken. Based on the minimal eigenvector, the Hessian matrix, and the derivative of the Hessian matrix, we can calculate the objective function. This objective function can then be minimized via any global optimization technique. In our work, a simple DE algorithm is used. A detailed explanation of the implementation of the DE algorithm is explained in the literature [16][21].

The following parameters are utilized initially across all mixtures:

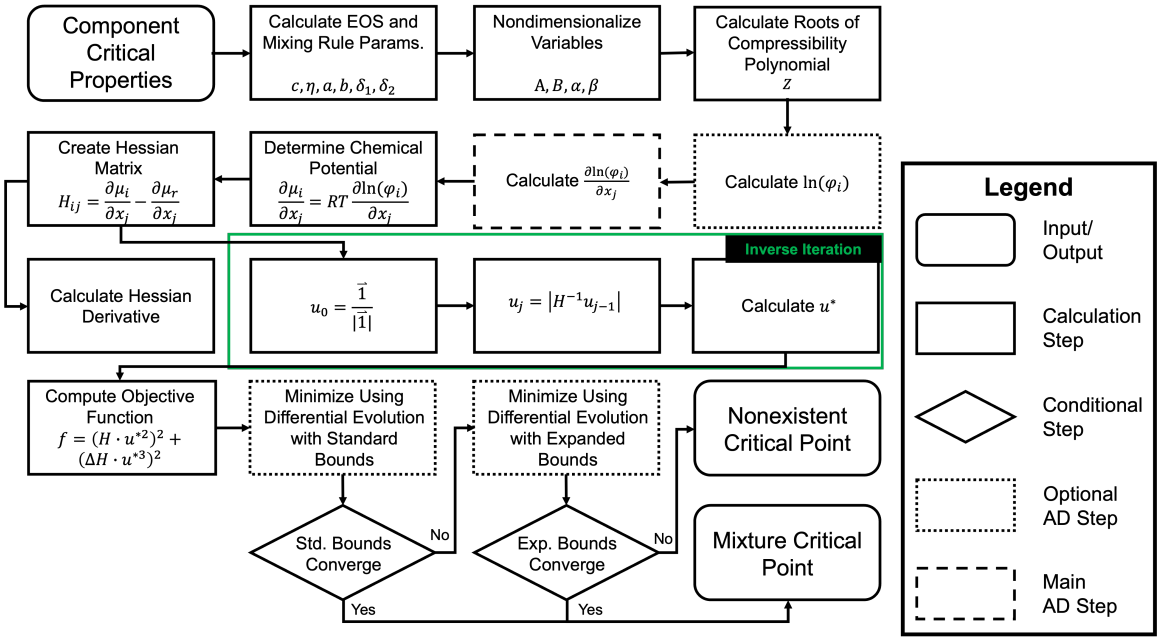

Figure 2.5: Algorithmic flowchart of the DE procedure used in this work. The component critical properties are used to compute nondimensionalized EOS parameters and the Z-polynomial. Roots of the Z-polynomial are computed using the algorithm shown in Figure 2.1. Next, mole fraction derivatives of the natural logarithms of the fugacity coefficients are computed. These are used to construct the Hessian matrix. The gradient of the Hessian matrix is then computed. Also, using inverse iteration, the smallest eigenvector of the Hessian matrix is computed. These elements are used to construct the objective function to prepare for minimization. Standard bounds and tuning parameters are used initially for the minimization of the objective function via DE. If these bounds fail to converge, then an expanded bound with more rigorous parameters is used. If both of these fail to converge, we deduce there is no critical point on the bounds; otherwise, we obtain a mixture critical point.

$T \in (\sum x_i T_{ci}, \quad \sum x_i T_{ci} + 100)$, $P \in (0.9 \sum x_i T_{ci}, \quad 2.5 \sum x_i T_{ci})$, a mutation constant $F = 0.95$, and an absolute tolerance $\eta = 10^{-14}$. The Sobol-sequence-based initialization is adopted. In cases where convergence is not achieved, a more rigorous parameter set is used: $T \in (\sum x_i T_{ci} - 50, \quad \sum x_i T_{ci} + 200)$, $P \in (0.75 \sum x_i T_{ci}, \quad 4 \sum x_i T_{ci})$, a mutation constant $F = 0.95$, a population number $N_{de} = 40$, an absolute tolerance $\eta = 10^{-16}$, and a crossover probability $P_{CR} = 0.4$. DE is done using Scipy's built-in routine based on the work of Storm and Price [21].

Chapter 3

Application of Automatic Differentiation to Mixture Critical Point Calculations

3.1 Introduction

Throughout its development, equilibrium thermodynamics has been reliant on relatively complex mathematical models that can describe a wide variety of physical phenomena. In these mathematical models, differentiation is a very common operation that is especially important in the context of thermodynamics. The differentiation of thermodynamic properties is a necessary component of nearly all mathematical calculations in the field.

In his foundational paper, Gibbs established that the phase transition of pure components is tied to the differential change in Gibbs free energy at a given state [3]. This established the basis for many of the phase behaviour advancements throughout the 1900s. A key part of the overall phase behaviour of a fluid is the calculation of its critical points. Using a highly accurate two-constant cubic equation of state (EOS) [5], Peng and Robinson were able to extend Gibbs' previous work to calculate the critical point of fluid mixtures of many components [6]. Heidemann and Khalil later reformulated the mixture critical point calculation algorithm [4]. Their formulation is based on the compositional derivatives of the Helmholtz free energy of the fluid

system. This has since become the most common formulation of the mixture critical point equations.

The formulation of Hiedemann and Khalil requires the calculation of first and second-order compositional derivatives of the system's Helmholtz energy [4]. Furthermore, the two common calculation algorithms that solve this non-linear system of equations are the Newton-Raphson (NR) method and tangent plane distance function minimization; both of these techniques require further compositional derivatives. Additionally, these and other computations can require derivatives of temperature, pressure, or volume (state derivatives). This can result in highly complex mathematical models with multivariable, higher-order differential forms.

Helmholtz energy is most often calculated using EOSs based on the critical properties of each component in a system. For a two-parameter cubic EOS, derivation of the Helmholtz energy form and its subsequent derivatives are tedious but well explored in the literature. However, for fluids containing many components and complex chemistry, more sophisticated EOSs are typically preferred. The Helmholtz energy equation of these complex EOSs can be convoluted, even more so for its derivatives. For certain EOS, the required derivatives can be implicit and pose even further issues in their analytical calculations [22]. This has led to many authors utilizing numerical derivatives, whose accuracy degrades as system complexity increases [13, 23].

In computer science and machine learning, automatic differentiation (AD) is an established technique for accurate differentiation of complex functions where analytical derivatives are impractical [24]. This technique takes advantage of the fact that any mathematical function calculated by a computer can be decomposed into a series of basic operations. AD applies chain rule to the algorithmic form of the function, resulting in "near-analytical" accuracy even for complex functions [25]. AD has been shown to be more accurate, more robust, and potentially faster than finite difference methods in machine-learning, especially for large or complex problems. In other areas of science, AD has been effectively used to allow for the simplification of complex cal-

culations and physics [26, 27]. The scalability and modular nature of this technique makes it an attractive option for calculating thermodynamic derivatives, which has previously been demonstrated in the literature [28].

In this work, we outline the principles and basic theory of AD. We apply AD to the calculation of fluid mixture critical points over a variety of fluid mixtures. Specifically, we consider both a damped Newton-Raphson (DNR) algorithm and a minimization via differential evolution (DE) algorithm. Various incorporations of AD into these algorithms, as well as implementation concerns, are discussed. We demonstrate that the errors introduced by AD are negligible when compared to analytical implementations. Additionally, we show that for complex mixtures, the computational cost is reduced by using AD over numerical derivatives. Finally, we discuss other potential advantages of AD when applied to thermodynamic derivatives in general, such as ease of model change and implementation of complex EOSs.

3.2 Theoretical Basis of Automatic Differentiation

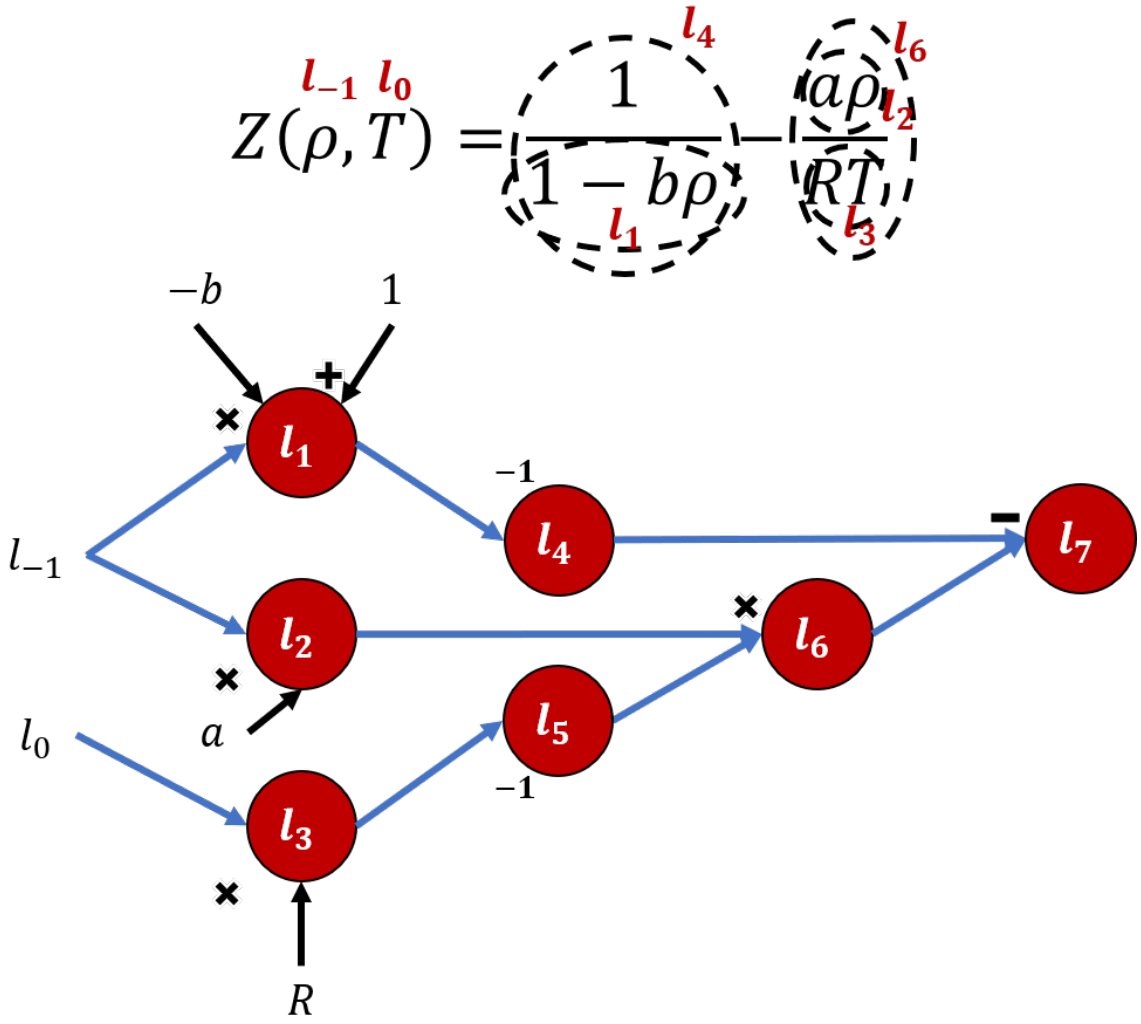

Figure 3.1: Computational graph for the calculation of compressibility factor for carbon dioxide at $0°C/22.4\frac{L}{mol}$, using the Van der Waals EOS. $l_{-1,\dots,7}$ represent the various terms in the equation. Simple mathematical operations are performed to combine terms and produce the overall result. This graph can be easily traced by an algorithm to allow for quick calculations. Additionally, an AD algorithm can compute the adjoint or tangent of this computational graph to obtain the function derivative.

AD is a computational technique for the calculation of "near-analytical" derivatives. AD takes advantage of the fact that any program implemented by a computer can be seen as a mathematical map consisting of simple mathematical operations. An example of such a map is presented in Figure 3.1. The chain rule is trivial to

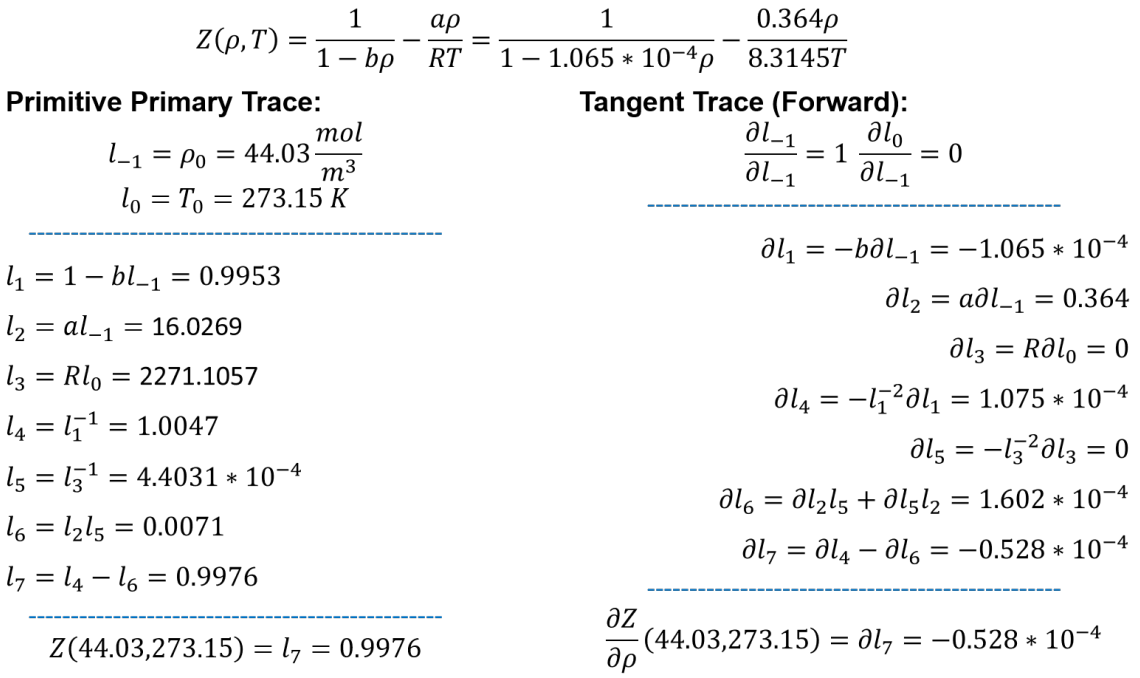

Figure 3.2: Example of a forward mode AD trace to obtain the molar density derivative of the compressibility factor using the Van der Waals EOS for carbon dioxide at $0°C/22.4\frac{L}{mol}$. On the left is a primary function trace, which consists of computations of each basic term in the equation. On the right is the tangent trace, which consists of the chain rule derivatives of the terms in the primary trace. Differentiation by each input variable has its own unique tangent trace.

implement for basic mathematical operations. Furthermore, its propagation is also simple when the machine is given a clear mathematical map between function input and output. With these facts together, implementations of AD instruct the machine to propagate the derivative of the algorithm alongside the function value itself [25]. This is done by constructing the mathematical map of the function (Figure 3.1) and then computing its adjoint or tangent step by step. An outline for this procedure is presented in Figure 3.2. Implementation of AD can be done manually or through a variety of packages available in most modern programming languages, such as JAX, pytorch, autodiff, ADF, etc.

As shown in Figure 3.2, forward mode AD begins with the function input variables seeding the primary (function) trace (l_{-1}, l_0). The tangent trace in a forward AD operation tracks the derivative of each variable with respect to the selected input

variable, l_{-1} in the case of the figure. The primary trace breaks the overall function into individual steps comprised of only one or two basic arithmetic operations ($l_1 - l_7$). Each step in the primary trace can then be differentiated via chain rule with respect to l_{-1} ($\delta l_1 - \delta l_7$). The resulting combination of these derivatives, following the primary trace, is the total derivative of the function with respect to l_{-1} (δl_7). The error on this approximation of the analytical derivative is the total propagated truncation error at each step [25]. This is what results in the "near-analytical" property of AD, as the associated error is not an approximation error but a propagated truncation error.

$$Z(\rho,T) = \frac{1}{1-b\rho} - \frac{a\rho}{RT} = \frac{1}{1-1.065*10^{-4}\rho} - \frac{0.364\rho}{8.3145T}$$

Primitive Primary Trace:

$$l_{-1} = \rho_0 = 44.03\frac{mol}{m^3}$$
$$l_0 = T_0 = 273.15\,K$$

$$l_1 = 1 - bl_{-1} = 0.9953$$
$$l_2 = al_{-1} = 16.0269$$
$$l_3 = Rl_0 = 2271.1057$$
$$l_4 = l_1^{-1} = 1.0047$$
$$l_5 = l_3^{-1} = 4.4031*10^{-4}$$
$$l_6 = l_2 l_5 = 0.0071$$
$$l_7 = l_4 - l_6 = 0.9976$$

$$Z(44.03,273.15) = l_7 = 0.9976$$

Adjoint Trace (Reverse):

$$\frac{\partial Z}{\partial \rho} = \frac{\partial l_4}{\partial l_{-1}} - \frac{\partial l_6}{\partial l_{-1}} = -0.528*10^{-4}, \qquad \frac{\partial Z}{\partial T} = \frac{\partial l_7}{\partial l_3}\frac{\partial l_3}{\partial l_0} = 2.583*10^{-5}$$

$$\bar{l_1} = \frac{\partial l_7}{\partial l_1} = \frac{\partial l_7}{\partial l_4}\frac{\partial l_4}{\partial l_1} = \bar{l_4} * \frac{-1}{l_1^2} = -1.0095$$
$$\bar{l_2} = \frac{\partial l_7}{\partial l_2} = \frac{\partial l_7}{\partial l_6}\frac{\partial l_6}{\partial l_2} = \bar{l_6} l_5 = -4.4031*10^{-4}$$
$$\bar{l_3} = \frac{\partial l_7}{\partial l_3} = \frac{\partial l_7}{\partial l_5}\frac{\partial l_5}{\partial l_3} = \bar{l_5} * \frac{-1}{l_3^2} = 3.107*10^{-6}$$
$$\bar{l_4} = \frac{\partial l_7}{\partial l_4} = 1$$
$$\bar{l_5} = \frac{\partial l_7}{\partial l_5} = \frac{\partial l_7}{\partial l_6}\frac{\partial l_6}{\partial l_5} = \bar{l_6} l_2 = -16.0269$$
$$\bar{l_6} = \frac{\partial l_7}{\partial l_6} = -1$$
$$\bar{l_7} = \frac{\partial l_7}{\partial l_7} = 1$$

Figure 3.3: Example of a reverse mode AD trace to obtain the molar density and temperature derivative of the compressibility factor using the Van der Waals EOS for carbon dioxide at $0°C/22.4\frac{L}{mol}$. On the left is a primary function trace, which consists of computations of each basic term in the equation. On the right is the adjoint trace, which consists of the chain rule derivative of the final term with respect to each of the terms in the primary trace. The adjoint trace computes the derivative of the function with respect to all input variables simultaneously.

In addition to the forward-mode AD in Figure 3.2, it is possible to instead calculate the adjoint trace to create reverse-mode AD as shown in Figure 3.3 [24]. In reverse AD, two calculations are required. Firstly, the primary trace is calculated as an evaluation of the function. Secondly, an adjoint trace is calculated using backpropagation.

The adjoint of each primary trace step is the derivative of the output with respect to that primary trace step ($\frac{\delta y}{\delta l_i}$). This means the adjoint of the final trace step is 1 and the adjoint of the variable of differentiation is $\frac{\delta y}{\delta x}$, i.e., the desired derivative. All intermediate adjoints are calculated via chain rule expansion, similar to the tangent trace. Note that the adjoint trace computes the derivative of the function with respect to all input variables simultaneously. As reverse AD requires two distinct passes through the function, it tends to be slower than forward AD. However, since reverse AD calculates the derivative with respect to all input variables simultaneously, it is much more efficient for some problems. Generally, forward AD's speed scales with the dimensionality of the output and reverse AD's speed scales with the dimensionality of the input, rendering AD additional flexibility to handle many disparate types of problems.

There are several advantages brought by the utilization of AD for derivative calculations. Firstly, AD is not numerical differentiation. The only error that an AD derivative incurs is the truncation error which, for simple algorithms, can be limited to some multiple of machine epsilon [29]. In addition to this, numerical derivatives are poorly conditioned and can be more costly to evaluate as the size and complexity of problems increases [25]. Analytical and symbolic derivatives can be used in place of numerical derivatives to avoid these issues. Symbolic derivatives come at a great computational cost and are limited in scope, which often makes them impractical [25]. Similarly, analytical derivatives are time-consuming, complicated, and prone to human errors, especially when functions are implicit [24]. For sufficiently complex functions, analytical derivatives are not feasible due to the difficulty of implementation. Even in the relatively simple derivatives of PR and SRK EOS, typographical and mathematical errors have been made in even major publications [8].

AD is also far more flexible than the other options. AD of similar functions generally does not need recomputation of identical traces. This leads to an improved performance when many similar derivatives are needed. Additionally, when making

slight changes to the differentiated function, AD traces are naturally recalculated without further user input. This can allow for fast changes in EOS model, mixing rule, and other thermodynamics models for testing or simulation purposes. However, as for analytical derivatives, rederivation of the derivatives when the EOS model is changed is required.

3.3 Application to Critical Point Calculation Algorithms

AD can be used at various points in the mixture critical point calculation algorithm. At a fundamental level, both algorithms (global optimization and root finding) involve the minimization of chemical potential. Therefore, AD can be used directly in Equation (3.1) to obtain the form of the chemical potential departure function. Integral functions are able to be processed by many modern AD packages by programming them as Riemann sums [30]. Alternatively, a compressibility factor equation might be simple to analytically integrate in ρ while being nontrivial to differentiate by component moles (dN_i). An example of this is a virial coefficients EOS, which is a simple polynomial in ρ but can have more complex mixing rules. Once a chemical potential departure function expression is found it can be related to fugacity and used in either procedure [17].

$$\ln\left(\varphi_i\right) = \mu^{DEP} = \left(\frac{\partial G^{DEP}\left(x,\ T,\ P\right)}{\partial n_i}\right)_{T,\ P,\ n_{j\cong i}} = \frac{\partial}{\partial n_i}\int_0^{\rho}\frac{Z-1}{\rho}d\rho + (Z-1) - \ln Z \tag{3.1}$$

In the DNR procedure, the main step in the algorithm that AD can contribute to is the construction of the Jacobian. In the work by Dimitrakopolous et al., the authors derived complex Jacobian expressions for a generalized cubic EOS [8]. These expressions can be replaced with AD, which is particularly helpful if more complex EOSs are needed. For this, Equation (2.4) is constructed analytically based on the

fugacity expression of a particular EOS. Then the composition gradient of this vector of functions is taken to generate the Jacobian. These implementations are described in Figure 2.2.

In the global optimization algorithm, AD can play a role in two major steps. Firstly, Henderson et al. analytically derived expressions for the derivatives of the logarithms of the fugacity coefficient of each component [13]. This is done for the Peng-Robinson EOS and involves complex differentials of compressibility and mixing rules. Instead, an automatic derivative of the $ln\phi_i$ can be taken directly from the expression derived from the EOS. In this work, a generalized cubic EOS is used to expand the work by Henderson et al. [13]. Secondly, Henderson et al. utilized a numerical differentiation methodology to obtain the Hessian gradient [13]. While this approximation is sufficiently accurate, it involves two evaluations of the gradient of the modified tangent plane distance function. Therefore we can instead take the automatic gradient of the Hessian matrix when it is constructed to reduce computational costs and improve accuracy. This is shown in Figure 2.5.

One key consideration when implementing AD in these algorithms is that some formulations or models require mole fraction derivatives to be taken with or without the restriction that the sum of mole fractions is equal to one. These are referred to as constrained mole fraction derivatives. Full considerations of constrained derivatives are available in Appendix A. For a general cubic EOS with Van der Waals mixing rules, we define AD-compliant constant definitions with constrained derivatives (Equation (3.2)).

$$\begin{aligned} a = a_{NN} + 2\sum_{i=1}^{N-1} x_i(a_{Ni} - a_{NN}) + \sum_{j=1}^{N-1}\sum_{i=1}^{N-1} x_i x_j(a_{ij} - 2a_{Ni} + a_{NN}) \\ b = b_N + \sum_{i=1}^{N-1} x_i(b_i - b_N) \end{aligned} \tag{3.2}$$

In this work the JAX package is used for AD [31]. Firstly, the original DNR

method utilizes manually computed analytical forms to populate the Jacobian matrix Equation (2.9). The AD version of the DNR method (AD-DNR) instead uses AD to calculate the complete Jacobian matrix from the original form of Equation (2.4).

Secondly, for the global optimization method proposed by Henderson et al., a third-order numerical approximation of the derivative of the cubic form Equation (2.13) is used [13]. The Hessian itself is populated via analytical computation of the chemical potential derivatives. In the AD-GO method, AD is used to directly calculate the derivative of $ln(\phi)$, which is used to compute chemical potential and populate the Hessian matrix. Then a second AD routine is run to differentiate the Hessian and obtain the cubic form. DE for minimization of the objective function is done via scipy's DE function [32]. Notice that this implementation utilizes AD to replace both the analytical derivations to obtain the Hessian and the numerical differentiation of the Hessian.

3.4 Results and Discussion

3.4.1 Performance of AD When Applied to DNR Method

In order to quantify the benefits of using AD in mixture critical point calculations, the DNR method is run with the methodology as reported by Dimitrakopolous et al. [8] and with the AD method.

By nature, AD versions of algorithms cannot meaningfully outperform the analytical counterparts [24]. However, from the 44 mixtures that are tested, the relative deviations yielded by AD-DNR from analytical results (as shown in Figure 3.4a), are less than 3×10^{-6}. Therefore, AD derivatives provide very accurate approximations of the analytical ones.

Additionally, from Figure 3.4b, it can be seen that the performance impact of utilizing AD is not associated with the number of components in a mixture. This indicates that mixture complexity does not have a significant impact on AD's speed.

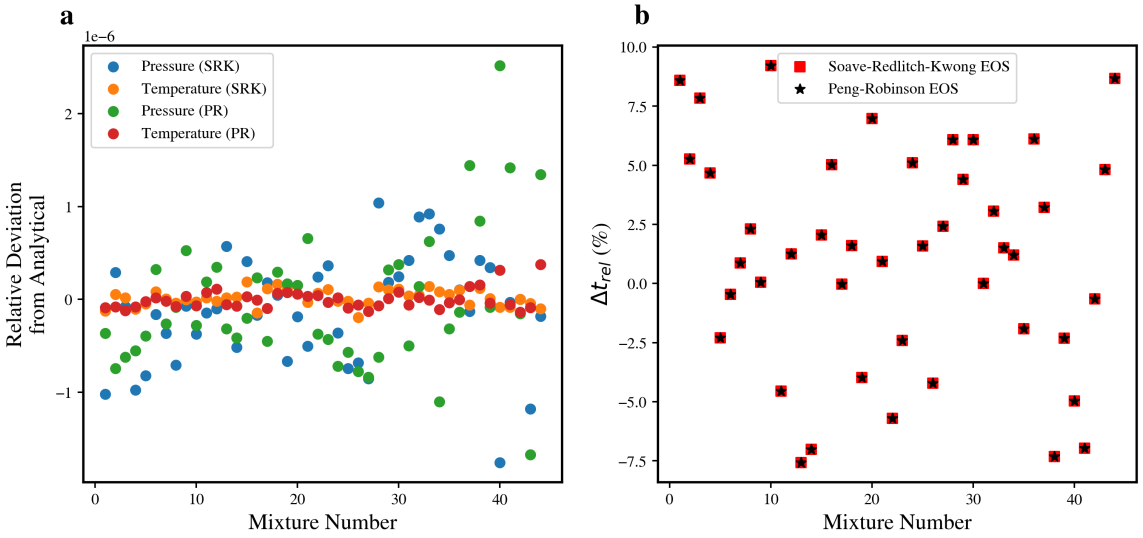

Figure 3.4: Accuracy and computational speed comparisons between the AD implementation of the DNR algorithm and the analytical implementation of the DNR algorithm for each of the 44 mixtures in Dimitrakopoulos et al. [8]: a) the relative deviation of AD from analytical results for each fluid mixture; b) the relative change in computational time consumed to reach convergence for each fluid mixture. The results include those for the SRK EOS and the PR EOS.

AD can offer up to a 7.5% speed up or 10% slowdown in terms of the overall computation time. For the 44 mixtures examined in this study, the number of iterations consumed by AD-DNR and analytical DNR are very similar.

The major advantage of AD when compared to analytical derivatives is the reduction in the complexity of the mathematics and coding. The Python code for AD-DNR consists of 527 lines of code, in comparison to the 848 lines of code needed for a fully analytical DNR. Also, the AD method does not require tedious mathematical calculations of the Jacobian matrix as expressed by equations 54-79 in the original work of Dimitrakopolous et al. [8].

3.4.2 Performance of AD When Applied to DE Method

Henderson et al. used a third-order finite difference to compute the derivative of the Hessian [13]. This introduces very little error to the critical pressure and critical temperature calculated by DE [13]. However, using finite differences does have an

impact on the computational speed. This prompts us to use AD in lieu of numerical derivatives.

Table 3.1 shows the changes in the number of iterations and the number of function evaluations yielded by the AD-DE algorithm as compared to the numerical-derivative-based DE algorithm. The use of AD results in an average reduction of 19.34 in the number of iterations and an average reduction of 2549.2 in the number of function evaluations. For some mixtures, AD increases these quantities; however, for most mixtures, it reduces them. Through the calculations, we observe that since AD-DE can normally converge over the narrower bounds ($T \in (\sum x_i T_{ci}, \quad \sum x_i T_{ci} + 100)$, $P \in (0.9 \sum x_i T_{ci}, \quad 2.5 \sum x_i T_{ci})$), AD-DE does not need to search for the minimum over the expanded bounds ($T \in (\sum x_i T_{ci} - 50, \quad \sum x_i T_{ci} + 200)$, $P \in (0.75 \sum x_i T_{ci}, \quad 4 \sum x_i T_{ci})$). Therefore, huge reductions in the number of iterations and function calls by AD-DE are observed for some mixtures (e.g., mixture No. 16). A considerable performance improvement is possible with the inclusion of an AD-compatible DE algorithm such as EvoSAX [33].

3.4.3 Easiness in Handling EOS Model Changes by AD

If a new EOS is used, new analytical derivatives have to be derived in order to perform critical point calculations based on the new EOS. A major advantage of AD is its superior adaptability and easiness in handling EOS changes.

To demonstrate this, we modify the original DNR system to utilize the volume-translated SRK developed by Peneloux et al. [34]. Peneloux et al. modified the SRK equation with a volume translation parameter as shown in Equation (3.3) [34].

$$\begin{aligned} P &= \frac{RT}{v-b} - \frac{a}{(v+c)(v+2c+b)} \\ c &= \sum_{i=1}^{n} z_i c_i, \\ c_i &= 0.40768 \frac{RT_{ci}}{P_{ci}} (0.008881 - 0.08775 w_i) \end{aligned} \tag{3.3}$$

Making the substitutions $c^* = \frac{c}{b}, C = \frac{Pc}{RT}, c^*B = C$ and putting compressibility factor into its cubic form, we obtain Equation (3.4). Similarly, the substitutions into the general fugacity coefficient form shown in Appendix B gives Equation (3.5).

$$\begin{aligned} 0 = Z^3 + [3C-1]\,Z^2 + \left[A - \left(B^2 + 2CB - 2C^2\right) - (B+3C)\right] Z \\ -AB - (CB+2C)\,(1+B) \end{aligned} \tag{3.4}$$

$$\ln(\varphi_i) = \beta_i\,(Z-1) - \ln(Z-B) + A\ln\left(\frac{Z+C}{Z+(B+2C)}\right)\left(\frac{2\alpha_i - \beta_i}{B+C}\right) \tag{3.5}$$

Typically, implementing this new model would require considerable modifications to the analytical derivatives associated with the critical point calculations due to the derivatives of Equation (3.5) needing to be calculated. Instead, with the use of AD, we can directly perform the derivative calculations without re-deriving the analytical derivatives. Figure 3.5 compares the calculated objective function (Equation (2.13)) with and without the application of volume translation in SRK EOS. Both results are calculated with the AD-DE algorithm.

While, herein, we only demonstrate how to apply AD in mixture critical point calculations, we can extend AD to many other thermodynamic calculation procedures. The overall flexibility makes AD an especially attractive choice for calculating thermodynamic derivatives. In thermodynamics, it is quite common practice to switch to different EOSs and mixing rules. Additionally, thermodynamic problems can involve both function derivatives of many variables and many function derivatives of a single variable, motivating the usage of both forward and reverse AD. As a specific example, the usage of complex EOS models in both root finding and global optimization methods for critical point calculations is limited by the difficulty of differentiating Helmholtz energy equations, which can often get very complex. Previous authors have attempted to use numerical derivatives to perform critical point calculations

Table 3.1: The change in the number of iterations and function evaluations yielded by the AD implementation of the DE algorithm as compared to the numerical implementation of the DE algorithm for each mixture from Dimitrakopolous et al. [8]. Large changes in function evaluations (> 10000) are a result of convergence being achieved over the standard bounds, preventing the need for calculations over the expanded bounds.

Mixture No.	**Change in the number of iterations**	**Change in the number of function evaluations**	**Mixture No.**	**Change in the number of iterations**	**Change in the number of function evaluations**
0	-6	-192	22	5	160
1	41	1312	23	14	448
2	2	64	24	-17	-544
3	-6	-192	25	-18	-576
4	18	576	26	-9	-288
5	-8	-256	27	7	224
6	4	128	28	9	288
7	9	288	29	5	160
8	-6	-207	30	-19	-608
9	-24	-768	31	-6	-192
10	-1	-32	32	-30	-960
11	2	64	33	147	4704
12	-32	-1024	34	6	192
13	-21	-672	35	-12	-384
14	-23	-736	36	-43	-1376
15	0	0	37	-8	-256
16	-327	-47667	38	2	64
17	-596	-67123	39	85	2720
18	78	2496	40	-31	-992
19	7	224	41	-419	-41584
20	-30	-960	42	-21	-672
21	0	0	43	421	41984

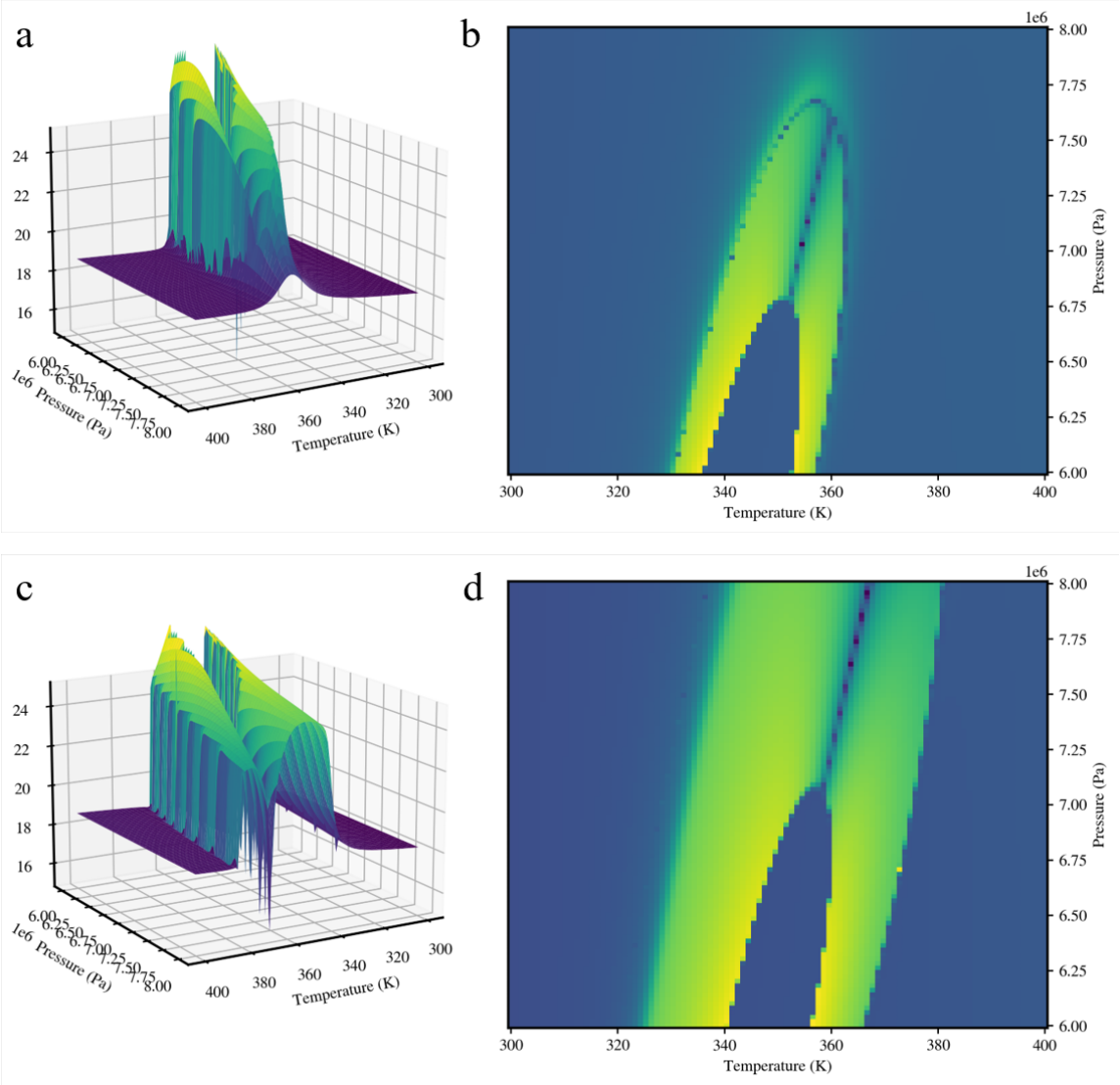

Figure 3.5: Impact of applying volume translation on critical point calculations: a) 3D view of the objective function (Equation (2.13)) calculated based on SRK EOS; b) 2D topography of the objective function calculated based on SRK EOS over the pressure-temperature plane; c) 3D topography of the objective function (Equation (2.13)) calculated based on volume-translated SRK EOS; d) 2D topography of the objective function calculated based on volume-translated SRK EOS over the pressure-temperature plane.

based on PC-SAFT EOS [23]. AD would be a much better choice in this case as its use is more straightforward and accurate.

3.5 Conclusions

In this work, AD is introduced in the context of thermodynamic mixture critical point calculations. AD is identified as a powerful computational technique that is compatible with a wide range of thermodynamic calculations. Firstly AD allows for accurate calculations of thermodynamic derivatives by avoiding tedious analytical calculations. Additionally, for complex thermodynamic models, AD can replace less accurate numerical approximations that are often used in the literature. Finally, the flexibility of AD means that changes in EOS models or mixing rules can be easily accommodated by AD.

These facts are demonstrated by implementing AD in both NR-based and DE-based algorithms. We find that for the fluid mixtures examined in this study, an AD implementation of the DNR method yields a relative deviation of $< 3 \times 10^{-6}$ in the calculated critical pressure and critical temperature in comparison to a fully analytical scheme. The computational cost of both AD and analytical implementations is of similar magnitude. We also show that replacing numerical derivatives with AD in the DE-based algorithm can result in fewer iterations and fewer function evaluations.

Chapter 4

Comparison of Root-Finding and Global Optimization Methods for Critical Point Calculations

4.1 Introduction

Critical points are of major interest to researchers in thermodynamics, as they are an important component of the phase envelope. In his 1879 work, Gibbs originally laid out the foundation for critical point calculations of pure substances [3]. For pure substances, Gibbs calculated the point at which the Gibbs free energy of phase separation is zero [3]. Above this point, a single phase exists. Meanwhile, below this point, the Gibbs free energy of phase separation is negative, and spontaneous phase separation occurs. This was later extended to allow for the calculation of critical points of multicomponent mixtures by Peng and Robinson [6]. The method developed by Peng and Robinson utilized a Newton-Raphson (NR) algorithm to solve the system of equations obtained from extending Gibbs' calculations to multicomponent fluids [6]. Heidemann and Khalil later simplified the algorithm by modifying the equations to use Helmholtz free energy [4]. Michelsen used an NR procedure to calculate full-phase envelopes, but interpolated the critical point on a given phase envelope [2].

NR techniques were the primary methodology for critical point calculations until Henderson et al. developed a new formulation [13]. In this new procedure, Henderson

et al. developed a modified tangent plane distance function for phase stability testing [13]. This results in the original Gibbs critical point criteria being re-framed as a minimization problem. This new method appears to have some key advantages over the original NR-based methods [13]. In accordance with the original criteria, the modified tangent plane distance also compares the free energy of a single phase to the free energy of two separate phases. Minimization of the objective function returns the state (temperature and pressure) at which the difference between the two free energies is zero.

Many authors have worked on improvements to the original NR method that Peng and Robinson had suggested [6]. After the Helmholtz formalization was developed by Hiedemann and Khalil [4], Michelsen and Heidemann extended the work to use a generalized cubic EOS [7]. Stradi attempted to resolve the two major issues of the NR method when applied to thermodynamic critical point calculations [9]. His application of interval mathematics allowed for the nonexistence of critical points to be determined and for multiple critical points to be found on a given temperature/-pressure range. While Stradi's method is effective, it is computationally expensive, especially for complex mixtures [9].

As root finding is a general computational technique, many other improvements and modifications have been made that are also applicable to thermodynamic computations. Combinations of the NR and bisection methods have been commonly explored in literature, such as Brent's method [35]. These hybrid techniques offer more efficient convergence, while maintaining the accuracy of standard NR methods [14]. Quasi-Newton methods, such as Broyden's Method, avoid the computational expense of computing the derivatives of complex functions by estimating the Jacobian or Hessian matrix [10]. Additionally, recent works have investigated utilizing Krylov subspaces to accelerate the convergence of traditional NR algorithms [11]. These modifications and improvements provide an abundance of algorithms that are suited for all manner of mathematical problems.

Dimitrakopoulos et al. provided a detailed explanation of a damped Newton Raphson (DNR) algorithm based on the work of Stradi and Jia et al. [8, 9, 36]. This damped method has highly robust convergence and only requires one parameter, a damping factor. This damping factor can be tuned to allow a wide range of mixtures to converge, even when the initial guess is not refined. Despite these many advancements in NR calculation algorithms, no universally superior algorithm has yet emerged.

The global optimization formulation, presented by Henderson et al., offers a set of advantages and disadvantages different from NR-based algorithms [13]. Various global optimization techniques have been successfully used to calculate mixture critical points, including simulated annealing [13], differential evolution (DE) [16], and stochastic optimization [14]. Global optimization techniques are generally slow compared to NR, due to their nondeterministic nature [14]. While this is can be a significant drawback, the optimization form of the problem has been shown to have some other advantages. Firstly, the optimization formulation can be fully visualized as a function of temperature and pressure. This can help identify multiple critical points, nonexistence of critical points, and give information about the type of critical point in a fluid mixture [13]. Additionally, for complex mixtures, global optimization may converge where NR does not. Therefore, the advantages of global optimization make it quintessential for the thorough modeling of mixture critical points.

In this work, a DNR algorithm and a DE algorithm are implemented to calculate critical points of fluid mixtures. Both methods are tested on a wide range of mixtures to evaluate their robustness of convergence, number of iterations, and overall performance. Our discussion identifies the key issues and advantages of each method using illustrative example mixtures. Additionally, this provides a comprehensive comparison of the two methods using the same mixtures, which is currently lacking in the literature.

4.2 Comparison of Root Finding and Global Optimization Methods

While implementing the DNR and DE algorithms for mixture critical point calculations, we investigate their respective robustness and accuracy. For a particular fluid system, the optimal algorithm may be unclear. In-depth analysis based on speed, robustness, and accuracy can allow for the appropriate calculation algorithm to be selected.

4.2.1 Evaluation of Algorithm Robustness

Firstly, the DNR algorithm is generally quite robust. It is able to compute the critical point of a wide range of mixtures at various compositions. In contrast, other NR methods from the literature, do not show this level of robustness [4, 9]. The DNR algorithm has the added advantage of having only 1 tuning parameter, the damping coefficient Q. This makes this method easy to tune to obtain convergence for a given fluid mixture. In contrast, methods that modify the bounds or initial guesses, such as the DE or interval NR algorithms, may require specific parameters for each unique mixture.

Dimitrakopoulos et al. [8], from numerical studies, suggested Q = 518 as an initial damping coefficient value. For the mixtures studied in this work, all but 2 of the simple mixtures converge with this damping coefficient. The failed simple mixtures can converge with Q = 700. Meanwhile, for complex mixtures of greater than 10 components, Q = 1000 is used to achieve convergence.

Table 4.1 shows the rate of convergence of the DNR method over 150 generated compositions for each unique mixture. The convergence of the DNR algorithm is dependent on the quality of the initial guess. For the mixtures we have studied, the initial guess that Dimitrakopoulos et al. suggest [8] is sufficiently accurate to enable convergence. In highly irregular mixtures, it is possible that regardless of the value

Q takes, the DNR algorithm will not converge for a particular initial guess. For the simple mixtures, all 150 compositions typically converge. The compositions that do not fully converge have nitrogen as a major component, which can lead to abnormal critical point behaviour. We obtain similar results for complex mixtures, with >98% of compositions converging.

Table 4.1: Rate of convergence yielded by the DNR method for 150 randomly generated compositions for each unique fluid mixture in the data sources. Generated compositions are created by a Dirichlet generator function with limited nitrogen mole fraction, as described in Chapter 5.

Mixture No.	**Convergence Rate (%)**	**Mixture No.**	**Convergence Rate (%)**	**Mixture No.**	**Convergence Rate (%)**
Simple Mixtures		8	100	17	100
0	100	9	100	18	98.667
1	98	10	100	19	100
2	100	11	100	20	100
3	100	12	100	21	100
4	100	13	100	**Complex Mixtures**	
5	100	14	100	0[1]	98.667
6	100	15	100	1[2]	98
7	100	16	100	2[3]	99.333

[1] Dimitrakopoulos et al. [8];
[2] Ghorayeb et al. [37];
[3] Xu and Li [38]

In contrast, the global minimization via DE could not achieve a $> 75\%$ convergence rate with a generalized set of parameters. We found that the DE algorithm can achieve convergence rates of $> 90\%$, but this requires mixture-specific bounds and tuning parameters. Henderson et al. proposed modified DE algorithms that seeded the search space more efficiently, which allowed for faster searches of large bounds [16]. These modified algorithms have improved robustness but are still much slower than NR-based methods. The DE algorithm, and other global optimization methods, have

the advantage of being able to search a much larger space than NR-based methods. This can be important in the case where a good initial guess is not possible.

Another issue with the DE algorithm is that, over the mixtures tested, DE yields a greater deviation from experimental values than DNR. Experimental results, as collected by Dimitrakopolous et al., are compared to the critical points found by both calculation algorithms [8]. These comparisons are shown in Table 4.2. The average absolute percentage deviation from the experimental results for the DNR algorithm is 1.96% for critical pressure and 0.81% for critical temperature. Meanwhile, the DE algorithm results in larger deviations, i.e., average absolute percentage deviations of 3.19% for critical pressure and 1.32% for critical temperature.

4.2.2 Handling of Complex Mixtures

The major advantage that global optimization methods have over NR-based methods is the ability to handle multiple critical points and confirm critical point nonexistence. The NR-based methods require different initializations to be able to identify multiple critical points. Furthermore, nonexistent critical points pose a major issue for the DNR algorithm. It is difficult to determine when the DNR algorithm fails to converge due to the nonexistence of a critical point rather than due to inappropriate initializations or damping coefficient values. In his work, Stradi proposed an interval NR-based method that could converge multiple critical points on a given range as well as confirm critical-point nonexistence [9]. Despite these advantages, the method is computationally inefficient for large mixtures (>5 components). This makes the global optimization methodology the most effective method for mixtures that may have multiple or no critical points.

An additional difficulty in the DNR algorithm is that in complex mixtures with multiple critical points, it can be impossible to predict which initial guess will lead to which critical point. This can lead to some critical points that are very difficult to obtain, due to few initial guesses converging to that particular critical point. This

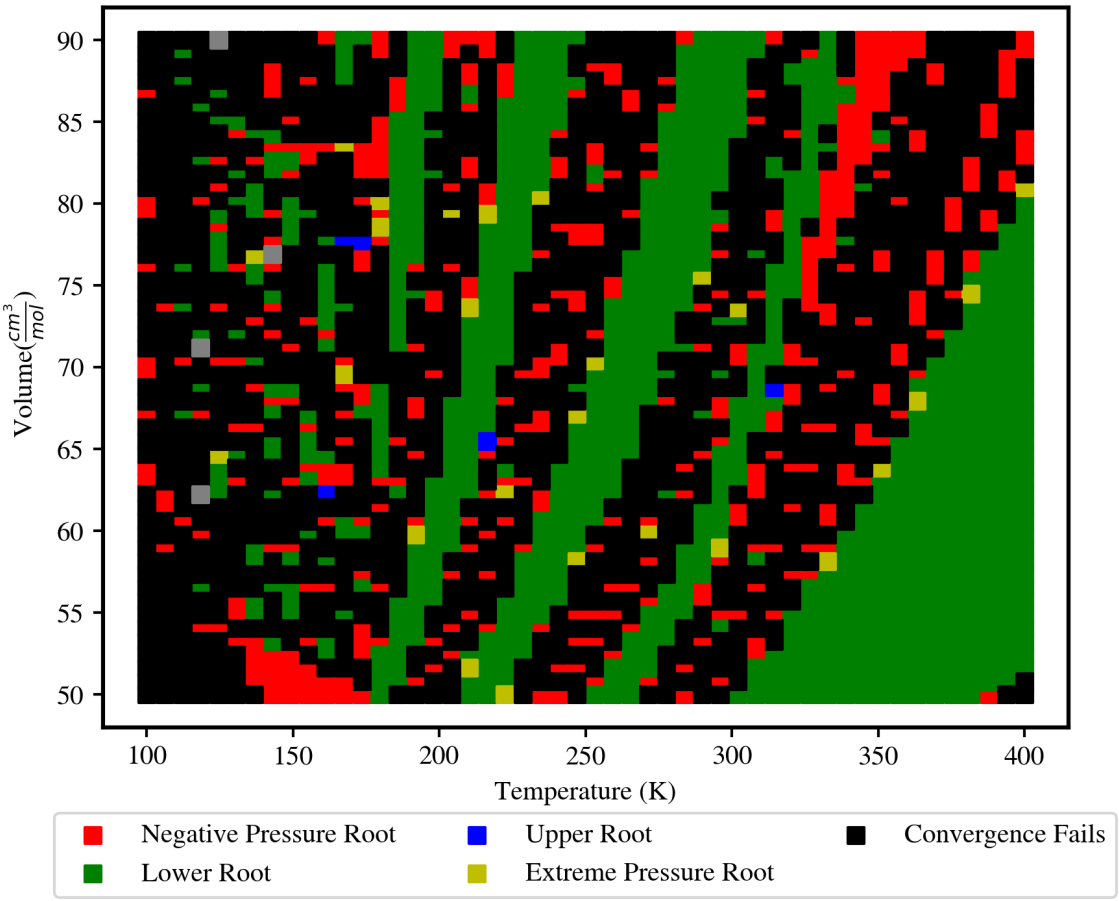

Figure 4.1: Root loci yielded by the DNR algorithm for Ghorayeb et al.'s first mixture at composition 0, where multiple critical points are present [37]. Initial guesses that lead to negative pressure roots are shown in red, convergence failures in black, the extreme pressure root in yellow, the upper root in blue, and the lower root in green. Initial guesses are tested over a wide range of molar volumes and temperatures. Convergence failures are declared when the algorithm does not converge after 50 iterations.

is a well-described mathematical issue with all NR-based root-finding methods [39]. Figure 4.1 shows the root loci of the first composition of the fluid mixture described by Ghorayeb et al., which has multiple roots. Henderson et al. identify a lower and upper root for this mixture [16]. In the figure, various initial guesses and the roots they converge to are shown. As shown, with a damping coefficient of 1000, very few initializations converge to the upper root in the wide range of initial conditions searched. Additionally, the algorithm can also converge to non-physical negative

pressure roots, and an extreme pressure root (T = 92.7 K, P = 191.1 MPa). These difficulties make the DNR algorithm an unreliable choice for calculating critical points of mixtures with multiple or nonexistent critical points.

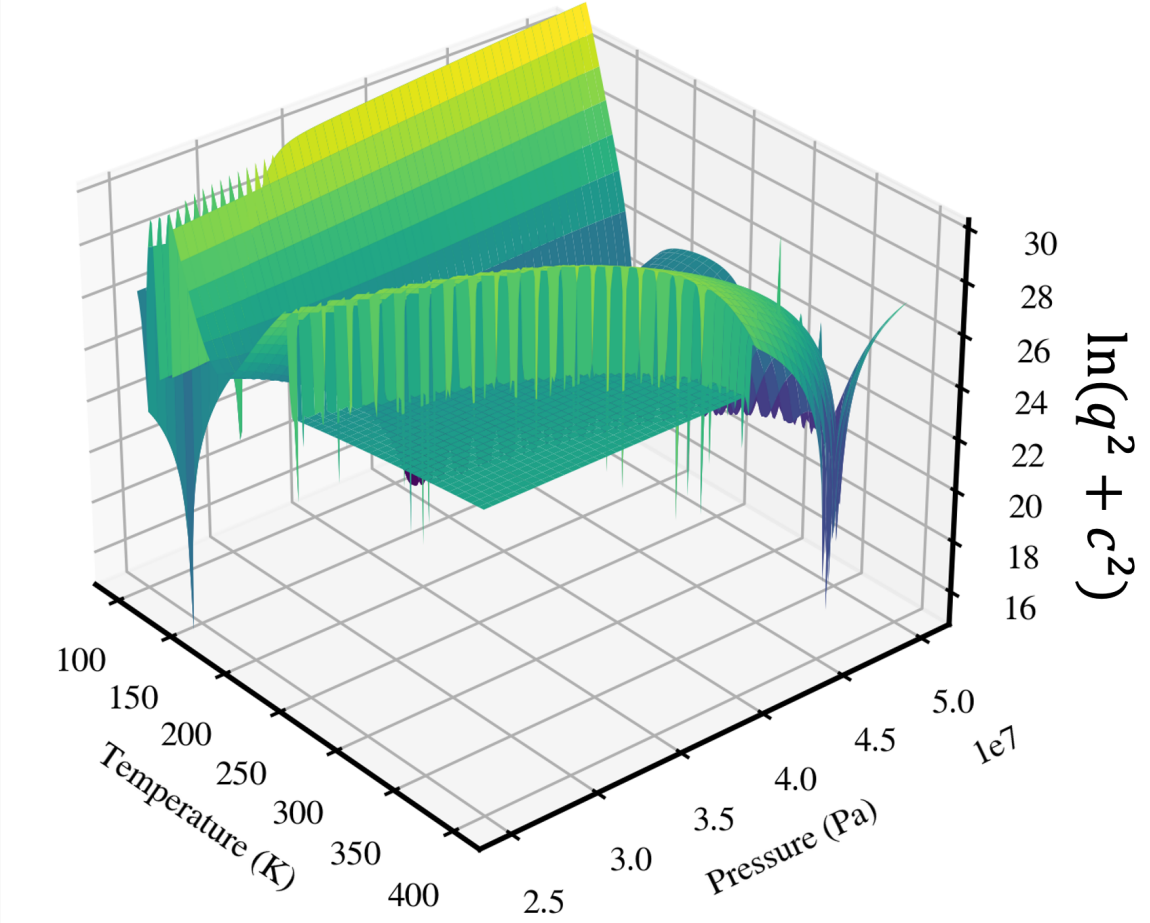

Figure 4.2: Objective function (Equation (2.13)) with marked upper and lower critical points for Ghorayeb et al.'s first mixture at composition 0 [37]. Using the DE algorithm, bounds can be set that only include one global minimum to predictably get either the lower or upper critical point. Visualization of the objective function can help set bounds and determine the existence of the two critical points.

Herein the DE algorithm has key advantages, as it searches an entire bound and is formulated based on temperature and pressure. Firstly, the pressure range is easier to set than the molar volume range. Secondly, pressure bounds avoid the issue of non-physical roots to the Gibbs critical point criteria. Thirdly, due to the complete search of the bound, when the DE algorithm fails to converge, it confirms nonexistence of

a critical point over the search bound. Most importantly, by setting an appropriate bound, every critical point of a fluid system can be obtained easily and predictably. For the Ghorayeb et al. fluid [37], we obtain both the lower (T = 105.8 K, P = 25.99 MPa) and upper critical point (T = 395.1 K, P = 45.92 MPa) using their respective bounds.

Another advantage of the global optimization formulation is that the objective function (Equation (2.13)) values can be calculated and plotted for a given bound in the temperature-pressure space. This can be used to graphically check for critical point nonexistence and multiple critical points. Additionally, the nature of a critical point can be discerned from the topography of the objective function surrounding it [16]. Also, Henderson et al. demonstrated modified DE methods that can converge to multiple critical points simultaneously on a given bound [16]. These advantages make the DE algorithm, and similar global optimization methods, extremely adept at handling fluid mixtures with complex critical point behaviour.

4.3 Conclusions

In this chapter, we compare the robustness, accuracy, and ability to handle the critical point phenomena of the DNR method with that of the global optimization via DE. To do this, we use both algorithms to calculate the critical points of various fluid mixtures. The convergence rate of 150 randomly generated compositions for a given mixture is used as a robustness metric. Next, we measure the accuracy of both methods by considering the absolute percentage deviations from experimental results. Finally, a complex mixture with two critical points is selected to illustrate the algorithmic difficulties in handling complex critical point phenomena.

We find that the DNR algorithm has a high convergence rate, even for complex mixtures. For most mixtures, all 150 compositions converge. For mixtures that do not have full convergence, the convergence rate is above 98%. These results are obtained without mixture-specific tuning of the damping coefficient. Meanwhile, the DE

algorithm can achieve a $> 90\%$ convergence rate only with mixture-specific bounds and tuning parameters. While both algorithms show acceptable deviations from experimentally obtained values, the DNR algorithm leads to smaller deviations than the DE algorithm.

Despite the robustness advantage of the DNR algorithm, we show that global optimization methods can be highly effective at calculations involving complex fluid mixtures. Firstly, critical-point nonexistence over a certain bound can be confirmed by the lack of convergence of global optimization methods. This is not possible for NR-based methods without incurring a large increase in computational costs. Additionally, the DE algorithm can effectively locate multiple critical points of complex mixtures by supplying different bounds or simultaneous computations. In contrast, some critical points may be difficult to obtain with the DNR algorithm, and the nature of the critical point yielded by the DNR algorithm is sometimes unclear. Finally, the global optimization methods enable us to visualize the critical points via plotting the objective function as a function of temperature and pressure in a 3D space. This allows manual confirmation of nonexistent critical points or multiple critical points and facilitates bound selections.

The above analysis suggests that the NR-based methods are more robust than the global optimization methods. Meanwhile, the global optimization methods appear to be more reliable at handling challenging critical point behavior, such as the existence of multiple critical points, the nonexistence of critical points, and non-physical critical points.

Table 4.2: Calculated and experimental critical points for unique fluid mixtures in the work of Dimitrakopoulos et al. [8]. Calculated results are shown for the DNR algorithm and the DE algorithm. Experimental results are obtained from Dimitrakopolous et al [8]. The DE algorithm does not converge for Mixture 7.

Mixture No.	**DNR**		**DE**		**Experimental**	
	P_c (MPa)	T_c (K)	P_c (MPa)	T_c (K)	P_c (MPa)	T_c (K)
0	5.3121	299.179	5.3121	299.180	5.3220	299.0
1	8.0601	300.570	8.0682	300.394	8.3000	302.0
2	6.3144	439.295	6.2269	441.459	6.6120	438.0
3	8.2624	392.507	8.0793	400.535	8.1010	391.0
4	9.1028	321.486	8.9369	323.033	9.2320	313.0
5	7.2634	358.785	7.1934	359.942	7.6400	354.0
6	10.1609	307.505	9.7740	317.577	10.3400	311.0
7	4.8969	306.105	-	-	4.9000	306.0
8	5.5452	403.769	5.5292	404.320	5.6000	397.0
9	4.1675	430.417	4.1670	430.445	4.1900	429.0
10	3.7911	450.486	3.7910	450.503	3.8800	450.0
11	7.0174	226.412	7.0192	228.834	6.8900	228.0
12	8.9871	316.132	8.8282	317.731	8.9630	313.0
13	7.3911	422.990	7.2280	427.155	7.4120	423.0
14	5.0595	410.264	5.0534	410.460	5.1130	406.0
15	4.4165	419.360	4.4157	419.451	4.5060	418.0
16	5.5979	388.242	5.5459	388.870	5.6200	385.0
17	7.0402	394.004	6.9709	394.905	7.2200	387.0
18	5.6343	201.753	5.7302	202.843	5.4560	200.0
19	6.4443	380.725	6.4243	381.324	6.5360	376.0
20	7.8514	318.063	7.8348	318.464	7.8460	314.0
21	5.8464	202.455	6.2783	206.511	5.5780	201.0
22	6.9610	204.278	7.1964	207.204	6.5840	204.0
Average Error (%)	1.9605	0.8084	3.1921	1.3198	-	-

Chapter 5

Machine-Learning-Based Acceleration of Mixture Critical Point Calculations

5.1 Introduction

Machine learning has proved itself as a fundamental tool for modern scientific and engineering practice. Despite the potential of machine-learning techniques, their black-box nature makes them difficult to use for fundamental scientific problems. Literature is beginning to explore machine learning approaches for solving thermodynamics problems [40, 41]. Typically, the lack of connection between machine-learning models and theoretical models reduces the practicality of machine-learning-based solutions in thermodynamics. This has been an additional barrier to the utilization of machine-learning techniques in the field.

In other chemical engineering fields, various implementations of machine learning have attempted to mitigate these concerns. Coupling physics-based modeling with machine learning has been a promising option to allow the integration of analytical models into machine-learning algorithms [42, 43]. These techniques are just emerging in the literature, and while they show promise, much work is needed before they are generally applicable to a wide variety of chemical engineering problems.

One other approach is the machine-learning-based acceleration of analytical cal-

culations. This has been shown to be a viable way of taking advantage of machine-learning techniques while still preserving a connection to theory [44]. In these types of approaches, machine learning is used to improve an existing theoretical framework rather than serve as a standalone solution. A machine-learning-based acceleration protocol can be implemented in a multitude of ways, e.g. making a first-pass estimate with machine learning [45], or using machine learning to sort problems into different categories which are to be solved by different analytical algorithms. This type of machine learning tends to rely less on complex, application-specific models and can prove to be more robust than standard predictive machine learning.

In this work, we investigate a machine-learning-based acceleration procedure to improve mixture critical point calculations. In our methodology, a series of training datasets for various fluid mixtures are created by computing critical points at varying compositions. From these training datasets, machine-learning models are trained for each unique fluid mixture. These trained models can predict the critical properties of their respective mixtures at any composition and can be easily saved, modified, and reused. The predictions are shown to have relatively good accuracy for a wide range of fluid mixtures at most compositions. The trained machine-learning models can be used in applications where a rough critical-point estimation is sufficient.

When higher accuracy is needed, the predictions made by the machine-learning models are used to initialize the traditional critical point computation algorithms (i.e., the root finding method or the global optimization method). We find that the resulting machine-learning-initialized algorithms have a superior convergence rate and generally improved robustness. The machine-learning-based initialization procedure is demonstrated using a damped Newton-Raphson (DNR) algorithm and a differential evolution (DE) algorithm. Additionally, we determine the impact of the size of the generated training dataset on the number of iterations needed to reach convergence. We quantify the convergence improvement achieved for both the DNR and DE algorithms, which have differing initialization procedures. We believe the framework that

is outlined in this work can contribute to compositional simulations by speeding up and front-loading the computational work required.

5.2 Computational Methods

5.2.1 Generation of Training Datasets for Machine Learning

In the procedure outlined by this work, the most computationally expensive step is the initial creation of a machine-learning training database. The training of machine-learning models can require 30-250 various compositions, along with their precomputed critical points, as training data. The computation of mixture critical points in the training data must be done via conventional critical point calculation algorithms (i.e., the DNR and DE algorithms). While it can be beneficial to pay this computational cost upfront, for complex mixtures (15+ components) this can involve a huge amount of computational time and memory. Therefore, we are well motivated to generate compositions for the training dataset in a way that maximizes the amount of valuable information per generated composition.

To this end, we recommend the usage of a Dirichlet Generator Function (DGF). The DGF randomly selects compositions from the total compositional space of the mixture via a Dirichlet distribution. The Dirichlet distribution is the multivariable extension of the well-known beta distribution [46]. A Dirichlet distribution function of N variables will produce N probabilities that sum to 1. This function has N tuning parameters $\alpha_{1,2,\ldots,N}$, allowing for the implementation of bias towards one or more of the variables. Figure 5.1 demonstrates an example of the generated training datasets from the DGF with various tuning parameters.

The tuning parameters allow for the creation of training datasets that are specialized for a given mixture or application. For example, some of the mixtures presented by Dimitrakopoulos et al. contain nitrogen gas [8]. Nitrogen gas, at high mole fractions, can produce open-phase envelopes without critical points or with abnormal

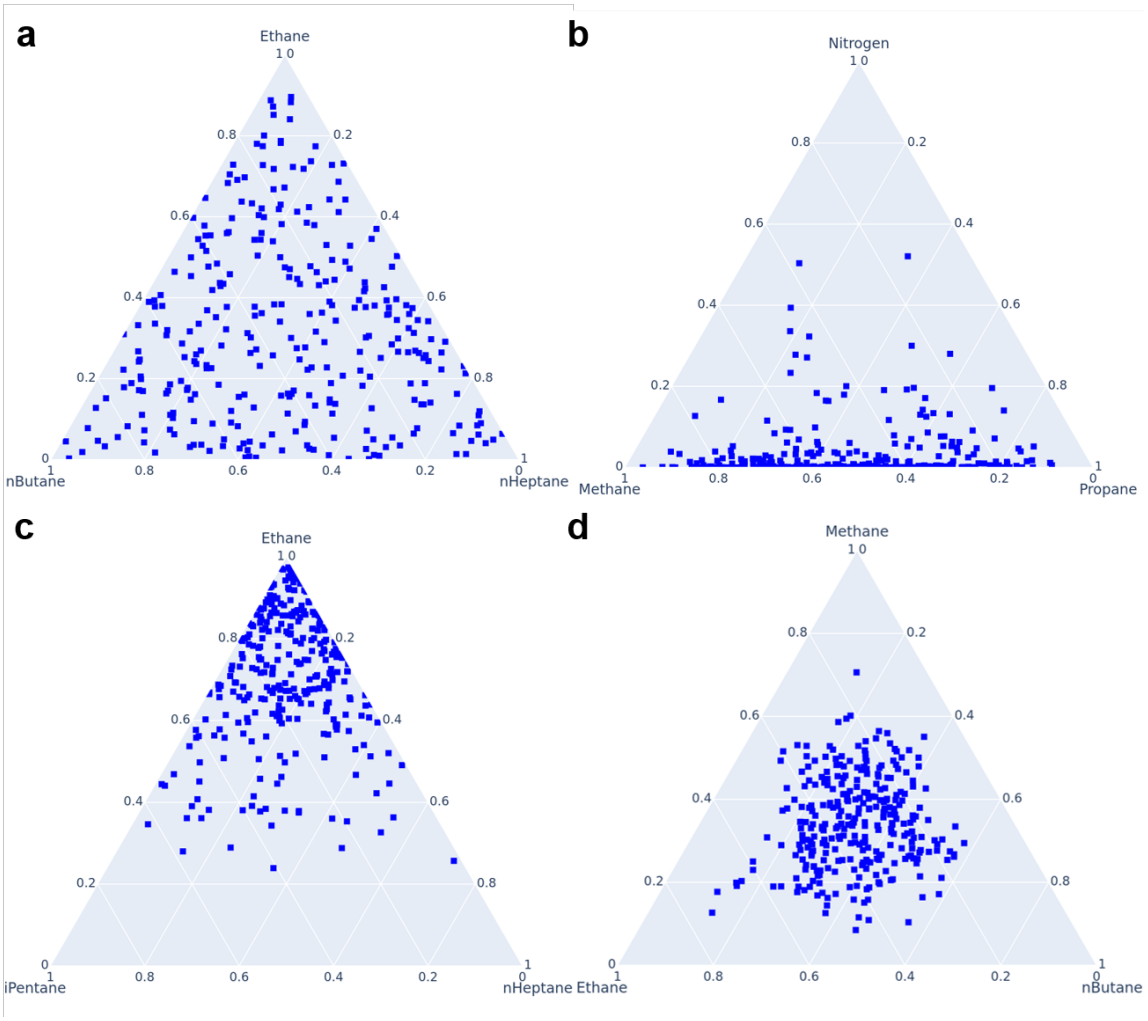

Figure 5.1: Example composition distributions of ternary mixtures from the DGF: a) a C2/C4/C7 system with an even composition distribution (α = [1, 1, 1]); b) a N2/C1/C3 system with a maximum nitrogen gas mole fraction of 0.3 (α = [0.2, 3, 3]); c) a C2/C5/C7 system with a dominant ethane component (α = [5, 1, 1]); d) a C1/C2/C4 system clustered around the point of equimolar composition (α = [6, 6, 6]).

critical points (such as critical points with extremely high critical pressures). This is particularly an issue when a fluid mixture contains mostly light hydrocarbon components. Therefore, for mixtures with nitrogen and light hydrocarbons, α can be selected such that the nitrogen mole fraction is limited to $< 0.3\%$ (b). In cases where trace heavy components are present in a light-oil-dominant mixture, tuning parameters can be selected to generate higher ratios of light-oil components (c). Alternatively, there

are mixtures with chemically similar components, in which near equimolar ratios are commonly expected; in this case, the tuning parameters can be modified such that small mole fractions are avoided for all components (d).

By tuning the DGF to limit the range of the training dataset and utilizing advanced computational techniques, training datasets can be generated in a reasonable amount of time for any fluid mixture. Once the DGF has generated the compositions, critical point calculations are performed to label the training and testing datasets. Two major calculation methodologies can be used for mixture critical point determinations: root-finding methods and global optimization methods.

The DNR algorithm is used to label the training dataset generated for the fluid mixtures presented in this work. It is worth noting that nonconverged or nonexistent critical points in the training dataset can cause issues associated with the training of the machine-learning models. Therefore, critical points that do not converge via the DNR algorithm are checked using the DE algorithm. If neither algorithm can converge, a nonexistent critical point at that composition is likely. We remove and replace such compositions using the DGF. This checking step is expensive when it occurs; therefore, it is preferable to tune the DGF to exclude compositions that can lead to critical-point nonexistence if known beforehand. For the results shown in this work, nitrogen fractions are limited to avoid nonexistent critical points, but no other DGF tuning is performed.

5.2.2 Deep Neural Network (DNN) for Critical Point Predictions

For the machine-learning model, a deep neural network is utilized, which is shown in Figure 5.2. The base DNN class that is used for all mixtures consists of two linear hidden layers with 50 neurons each. The input layer takes N variables representing the component mole fractions. Meanwhile, the output layer depends on the critical point calculation algorithm. For the DNR algorithm, critical temperature, critical volume,

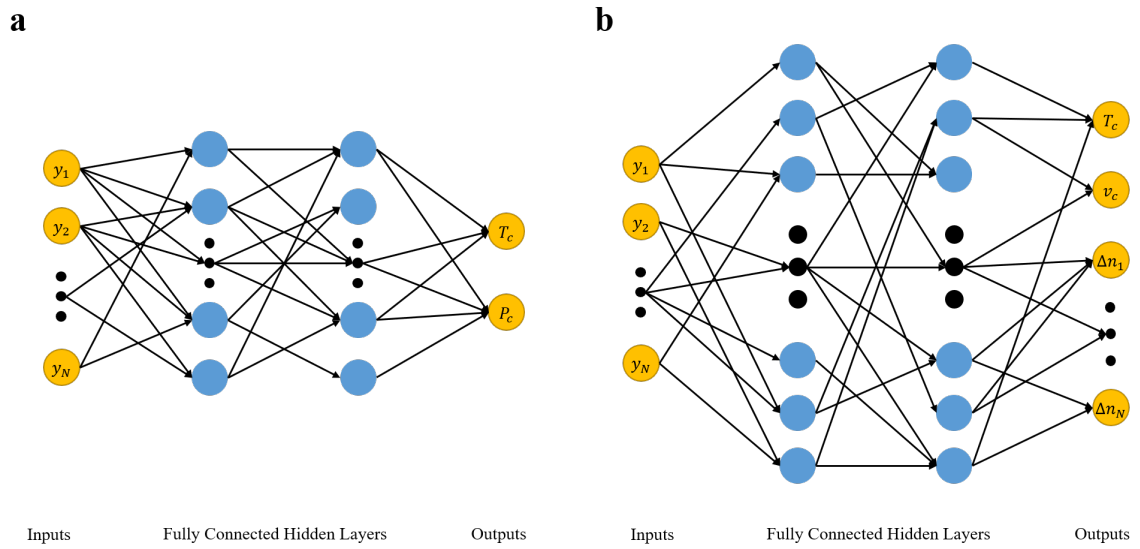

Figure 5.2: Diagram of the architecture of the DNN models used: a) DNN model that can potentially accelerate critical point calculations via the DE algorithm; b) DNN model that can potentially accelerate critical point calculations via the DNR algorithm. Both models consist of an input layer of mole fractions to describe the fluid mixture. Both models have two fully connected hidden layers of 50 neurons. Critical properties are normalized by the simple average of component critical properties before training.

and mole number perturbation for each component are outputs (N+2 variables in the output layer). For the DE algorithm, critical temperature and critical pressure are the output variables. The layers are initialized with a Xavier uniform distribution and both layers are activated via rectified linear units. Mean squared error is the loss parameter, with an 'Adam' optimizer. Training is done over 100 epochs with a learning rate of 1e-4.

The general base model does not take component critical properties as direct inputs. While the model's performance can be improved by including component critical properties, we still achieve adequate performance. The critical property values of a given mixture are normalized to improve training speed. We perform normalization of critical properties based on the simple average of the mixture components' critical properties (Equation (5.1)):

$$P_c^* = \frac{P_c}{\sum_{i=1}^{N} P_{ci}/N}, T_c^* = \frac{T_c}{\sum_{i=1}^{N} T_{ci}/N} \tag{5.1}$$

This base model is very generalized and significant accuracy improvements can be made via the tuning of hyperparameters and the inclusion of more input variables. Including chemical property data as inputs may not be viable for fluid systems with pseudo-components or other complex component chemistries. Additionally, machine-learning-based acceleration does not require a very high level of prediction accuracy. Therefore, our calculations are performed using the un-optimized base model to establish a baseline performance. However, when the application allows, chemical property data should be included as input to the models to improve predictions.

5.2.3 Initialization of Critical Point Calculations with Machine Learning

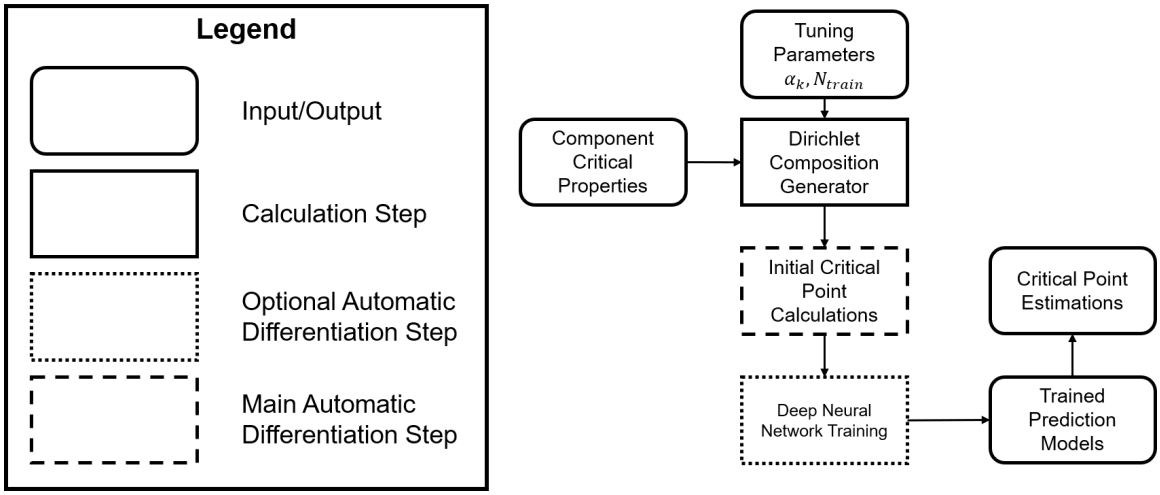

Figure 5.3: Flowchart showing the procedure adopted for training the DNN models dedicated to mixture critical point predictions. The DGF is used with tuning parameters to generate a dataset with random compositions. Component critical properties are then used with the DNR algorithm to compute a mixture critical point to act as a label for each random composition. Generated compositions and their labels are split into a training and a testing dataset by a 0.9/0.1 ratio. DNN models are trained with the training dataset, with prediction accuracy being evaluated on the testing set. The trained DNN models for each unique mixture have their weights and parameters saved.

Many circumstances call for accurate calculations of critical properties with a theoretical basis. When analytical accuracy is required, we utilize the trained machine-learning model to provide an initialization for the mixture critical point calculations. This initialization reduces the overall difficulty of the critical point calculations. Both the DNR and DE algorithms benefit from a reliable initialization scheme to reduce the number of iterations needed to reach convergence. The integration of the machine-learning-based initialization is slightly different for each algorithm. Figure 5.3 demonstrates the overall procedure used for generating machine-learning initializations.

Any NR-based method, including the DNR method, requires an initial guess. It is possible to prove, when supplied with an initial guess that is sufficiently close to the true answer, that NR-based methods have quadratic convergence [47]. As mentioned above, to generate an initial guess for the DNR method, we train the DNN models to predict critical temperature, critical volume (T_c and v_c), and component mole deviations (δn_i). The DNN-model predictions are normally very close to the true critical point, resulting in a faster convergence. Due to the convergence issues encountered by Newton's Method when handling multiple roots, as discussed in Chapter 4, mixtures with multiple critical points still create issues.

For the DE algorithm, machine-learning-based initializations can be implemented in two ways. Firstly, global optimization methods typically work over some bounding box. Therefore, it is simple to use machine-learning predictions and their associated errors to define a tight bounding box around the predicted critical point that should contain the true critical point. We compute the bounding box by adding and subtracting the computed prediction uncertainty ($\delta T_{pred}, \delta P_{pred}$) to the predicted critical point. The prediction uncertainty is based on the expected error in the machine-learning predictions, as described in Equation (5.2). The tight bounding box computed from machine-learning predictions will allow for the DE algorithm to be replaced with a brute force search (BFS) algorithm.

$$
\begin{aligned}
P^+ = P_{pred} + \delta P_{pred}, \quad P^- = P_{pred} - \delta P_{pred} \\
T^+ = T_{pred} + \delta T_{pred}, \quad T^- = T_{pred} - \delta T_{pred} \\
(T_{crit}, P_{crit}) = min(F(T,P)|_x), \\
T \in (T^-, T^+), \quad P \in (P^-, P^+)
\end{aligned}
\tag{5.2}
$$

Alternatively, for the objective function of a fluid mixture, there is a line on the temperature-pressure plane that contains the global minimum and the nearby local minima. By identifying this line, we can perform a BFS along the line to find the global minima. To identify this line, we begin with a DNN model prediction of critical pressure for the fluid mixture. Using the prediction (P_{pred}) and its associated uncertainty (δP_{pred}), we can get the estimated maximum and minimum expected pressures (P_+, P_-). The true critical pressure lies somewhere between these two values. At the maximum expected pressure (P_+), we can minimize the objective function as a 1-D function of temperature. This minimum will be one of the points on the line of local minima. The same procedure can be applied to the minimum expected pressure (P_-) to get a second point on the line of local minima. These two points can be used to write a line equation that contains the global minima. Finally, a BFS of the points on the line will identify the global minimum, which is the critical point. This methodology is described in Equation (5.3). All the above three initialization methods are shown in Figure 5.4.

$$
\begin{aligned}
(T_c, P_c) = min(F(T,P)|_x : (T,P) \in y_{min}), \\
y_{min} = mT + b \\
m = \frac{2\delta P}{min(F(T)_{x,P^+}) - min(F(T)_{x,P^-})} \\
b = P^+ - m \cdot min(F(T)_{x,P^+})
\end{aligned}
\tag{5.3}
$$

Utilizing the DGF, datasets with 150 data points are built for each unique mixture in two different mixture sets. The first mixture set is taken from the work of Dimitrakopolous et al., containing various mixtures of up to 10 components with alkanes,

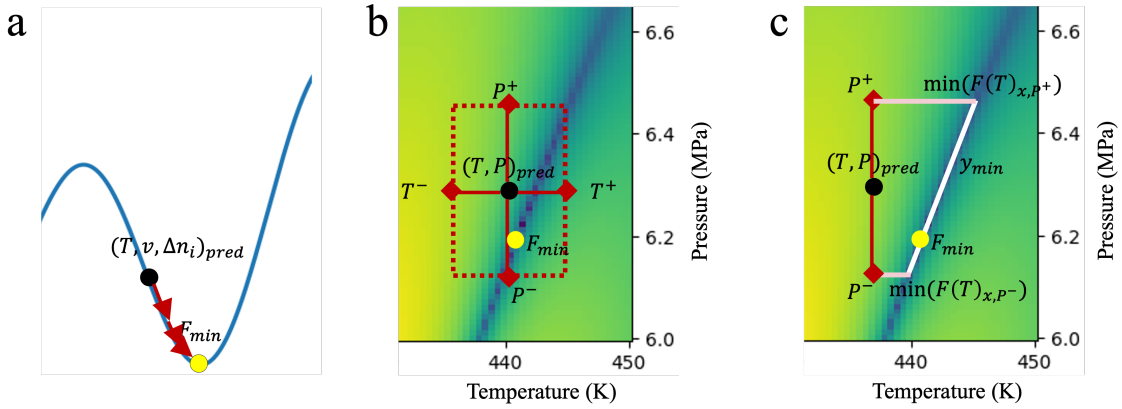

Figure 5.4: Use of machine learning to accelerate critical point calculations: a) machine-learning models can provide a refined initial guess for the NR-based root finding methods; b) machine-learning models can provide a tight bounding box for the global optimization methods; c) machine-learning models can provide a line constraint for the global optimization methods.

nitrogen, and carbon dioxide [8]. The second mixture set consists of various complex mixtures (>10 components) obtained from multiple sources, including petroleum fluids and pseudo-components. Mixture compositions and chemical properties can be found in Appendix B.

5.3 Results and Discussion

5.3.1 Predictive Performance of Stand Alone Deep Learning Models

Training dataset generation is done for each of the unique mixtures in the dataset provided by Dimitrakopolous et al. [8]. These training sets will train machine-learning models that are directly used for making critical point predictions. Each unique mixture has a unique training dataset and a unique trained model. We use the following evaluation metrics as performance indicators: root mean squared relative error (RMSRE) and mean absolute relative error (MARE). These metrics are defined in Equation (5.4) and the prediction errors are summarized in Table 5.1.

Table 5.1: Prediction errors of mixture critical points yielded by the trained machine-learning models for the unique fluid mixtures from Dimitrakopoulos et al. [8]

Mix	CO_2	N_2	CH_4	C_2H_6	C_3H_8	iC_4H_{10}	nC_4H_{10}	iC_5H_{12}	nC_5H_{12}	nC_6H_{14}	nC_7H_{16}	T_c **RMSRE**	P_c **RMSRE**	T_c **MARE**	P_c **MARE**
0			□	□								0.0010	0.0023	2.904E-06	2.760E-10
1	□	□										0.0042	0.0126	1.014E-05	8.539E-10
2				□			□				□	0.0071	0.0098	1.286E-05	1.123E-09
3				□				□			□	0.0086	0.0147	1.436E-05	1.742E-09
4		□	□		□							0.0203	0.0879	4.100E-05	5.782E-09
5			□	□			□					0.0052	0.0475	1.153E-05	2.937E-09
6			□	□					□			0.0172	0.1070	3.399E-05	6.157E-09
7				□	□		□					0.0016	0.0041	3.494E-06	7.388E-10
8				□	□				□			0.0035	0.0083	7.603E-06	1.394E-09
9					□		□		□			0.0040	0.0047	8.174E-06	8.857E-10
10							□		□	□		0.0032	0.0016	5.652E-06	3.518E-10
11			□	□	□							0.0032	0.0252	8.620E-06	1.445E-09
12		□	□	□	□							0.0113	0.0375	3.017E-05	3.492E-09
13				□			□	□			□	0.0076	0.0124	1.377E-05	1.486E-09
14				□	□		□		□			0.0037	0.0081	6.972E-06	1.310E-09
15					□		□		□	□		0.0034	0.0064	5.921E-06	1.334E-09
16				□	□		□		□	□		0.0096	0.0160	1.644E-05	2.219E-09
17			□	□	□		□		□			0.0129	0.0146	2.794E-05	1.575E-09
18		□	□	□	□		□					0.0521	0.3737	1.236E-04	2.095E-08
19			□	□	□		□		□	□		0.0219	0.0514	3.443E-05	4.017E-09
20		□	□	□	□		□		□			0.0496	0.0891	6.667E-05	6.236E-09
21		□	□	□	□		□		□	□		0.0288	0.0632	4.831E-05	4.837E-09

$$RMSRE = \sqrt{\frac{\sum_i^{N_{test}} \left(\frac{y_i^{DNN} - y_i^{NR}}{y_i^{NR}}\right)^2}{N_{test}}}, \qquad MARE = \frac{\sum_i^{N_{test}} \left|\frac{y_i^{DNN} - y_i^{NR}}{y_i^{NR}}\right|}{N_{test}} \tag{5.4}$$

For each mixture, a 0.9:0.1 ratio is applied to split the entire generated compositions into the training and testing datasets. This results in $N_{test} = 15$ testing compositions, across which RMSRE and MARE are calculated. Predictably, we see that mixtures with more components have higher prediction errors, and may benefit from larger training datasets. For mixtures that do not include nitrogen, the RMSRE in the critical temperature predictions is 0.71±0.58% and the RMSRE in the critical pressure predictions is $2.09 \pm 2.73\%$. Mixtures that include nitrogen tend to have higher prediction errors since a higher nitrogen fraction may lead to erratic critical point behaviour. For nitrogen-inclusive mixtures, the RMSRE in the critical temperature predictions is $2.82 \pm 1.81\%$ and the RMSRE in the critical pressure predictions is $10.8 \pm 12.1\%$.

For both nitrogen-inclusive and non-nitrogen-inclusive mixtures, RMSRE can exaggerate the effect of outliers and misrepresent the actual expected inaccuracy. The calculated MARE is considerably lower than the RMSRE, even after taking the squareroot. This indicates that the majority of the predictions are being made very accurately, with only some outlier compositions causing large errors.

The effect of outlier compositions on the prediction accuracy can be observed in Figure 5.5. There is a tendency for the machine-learning models to predict with larger errors at extreme normalized pressures. Prediction errors can be potentially reduced by tuning the generator function to avoid compositions that are unrealistic or abnormal. Alternatively, the generator can be biased towards the components that are most irregular to provide more training data in regions that are more difficult to predict.

Table 5.2 shows three mixtures, each with over ten components, that are selected to test the prediction performance of the trained DNN model for complex mixtures.

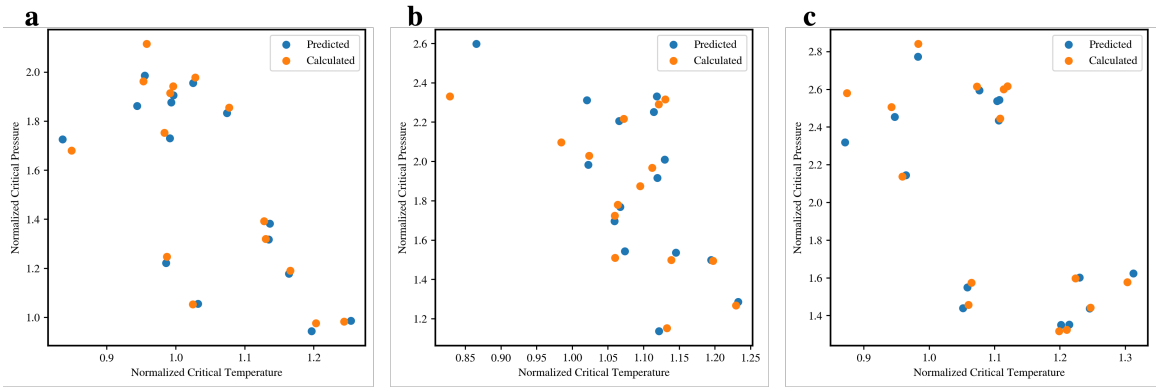

Figure 5.5: Comparison of true mixture critical points and DNN-predicted mixture critical points: a) mixture 2 - binary Ethane/nButane system; b) mixture 5 - ternary Methane/Ethane/nButane system; c) mixture 19 - 6-component Methane/Ethane/Propane/nButane/nHeptane/nHexane system.

The properties of these mixtures can be found in Appendix A. The prediction errors are similar to those shown in Table 5.1. We observe that the first mixture has more components than the other two mixtures and yet has a lower prediction error. This indicates that prediction accuracy may not directly correlate with the number of components in a mixture.

Table 5.2: DNN-model prediction accuracies for complex mixtures (10+ components).

Mixture	N_c	T_c **RMSRE**	P_c **RMSRE**	T_c **MARE**	P_c **MARE**
Ghorayeb et al. [37]	13	0.0138	0.0285	1.674E-05	3.842E-09
Xu and Li [38]	12	0.0345	0.0914	3.759E-05	5.899E-09
Dimitrakopoulos et al. [8]	11	0.0314	0.0932	3.260E-05	4.320E-09

5.3.2 DNR Algorithm Initialized by a Trained DNN Model

To improve speed and robustness of critical point calculations, we initialize the DNR algorithm with predictions that are made by machine-learning models. For the DNR algorithm, the trained machine-learning models predict critical temperature, critical molar volume, and mole number perturbations of individual components ($T_c, v_c, \Delta n_{1,2,\ldots,N}$). Due to the close proximity of the machine-learning prediction to

the true solution, the initialized algorithm converges for all mixtures that have a single root. This allows the damping coefficient to be set to zero for the DNR in initialized mixture critical point calculations.

We evaluate the performance improvements of the initialized DNR algorithm by the reduction in the number of iterations when compared to the original DNR algorithm without DNN-aided initialization. For each unique mixture, we calculate the critical points of the 15 compositions in the testing dataset using the original DNR algorithm and count the number of iterations needed to reach convergence. We predict the critical point for each of the 15 test compositions using the previously trained DNN model for that mixture. We then use the DNN-model predictions to initialize the DNR algorithm and again count the number of iterations needed for the critical point calculation to converge. For each mixture, the number of iterations required for both the DNN-model initialized calculations and standard initialized calculations (i.e., using Kay's mixing rule as shown in Figure 2.2) is averaged across the 15 test compositions. The difference in the average number of iterations between standard and DNN-initialized DNR calculations is taken as the evaluation metric. For the simple mixtures, with 150 generated data points, the average number of iterations is reduced from around 20 to 3-6, as shown in Figure 5.6.

Training dataset generation and DNN-model training can be done before the critical point calculation process. However, the generation of training data is still computationally resource-intensive. Therefore, we desire to avoid generating extraneous training data that may not yield significant performance improvement. To this end, we generated datasets with various sizes (30-250 compositions) and used them for DNN-model training to determine the necessary size of the training dataset. For each generated dataset, a 0.9/0.1 ratio is again applied to split the dataset into a training dataset and a testing dataset. The difference in the number of iterations between standard and DNN-initialized DNR calculations is determined and averaged across all the compositions in the respective testing dataset. The performance and

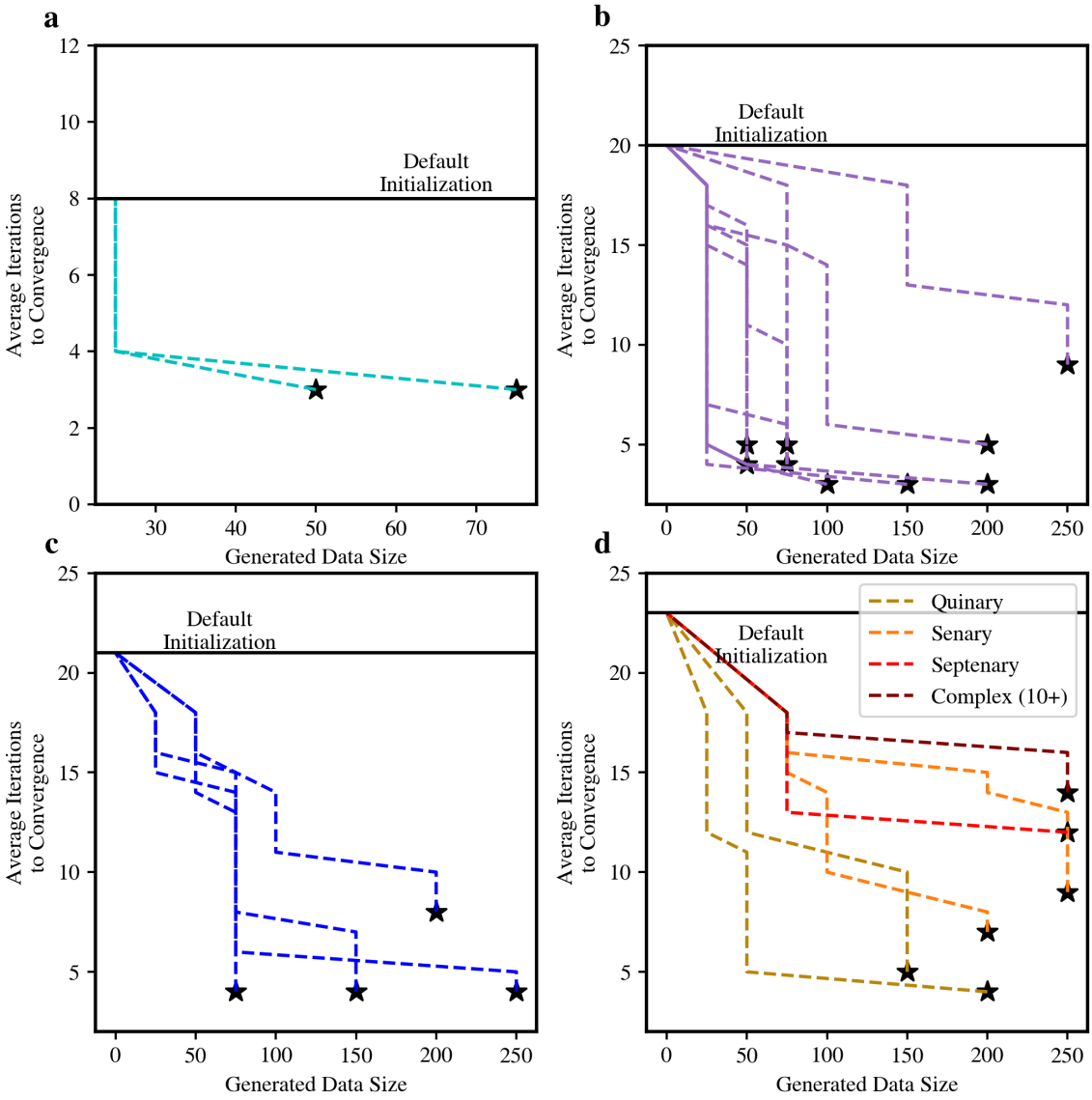

Figure 5.6: Average number of iterations needed to reach convergence for DNN-initialized DNR calculations with varied sizes of generated data: a) binary mixtures, b) tertiary mixtures, c) quaternary mixtures, and d) mixtures of 5 or more components. The iteration count yielded by the DNR algorithm initialized with Kay's mixing rule is shown as 'Default Initialization'. The number of iterations to reach convergence are averaged over the testing dataset for each unique fluid mixture at each generated data size.

dependence on the generated dataset size of the DNN-initialized critical point calculations are shown in Figure 5.6. Note that, the datasets are randomly generated from the DGF and randomly split into training and testing datasets. Therefore, the

calculation of the optimal generated dataset size will be stochastic and vary based on what compositions are generated by the DGF and how they are split into training and testing datasets.

As shown in Figure 5.6, the DNR algorithm with the standard initializations takes an average of 8 iterations to converge for binary mixtures. We observe that, with only a small dataset of 25-50 data points, a 62.5% reduction in the average number of iterations can be achieved. For larger mixtures of 3-8 components, approximately 200 data points are needed to achieve the greatest performance enhancement (63.6%-85.7%). Complex mixtures with 9+ components require 250+ data points for their peak performance enhancement (50%) but can achieve meaningful enhancement with datasets of 150 points. For the three complex mixtures investigated in this study, 150 generated data points allow for a 27.2-50% reduction in the number of iterations. Complex mixtures can benefit from a higher learning rate. Further tuning of the learning rate, epochs, and DNN-model architecture can yield a greater performance enhancement effect from DNN initializations.

5.3.3 Global Optimization Methods Initialized by a Trained DNN Model

Performance enhancement of the DE algorithm is achievable in a similar way. Global optimization methods are less reliable than NR methods, as they may converge to local minima rather than the global minimum. This makes most of the simple optimization methods ineffective for training dataset generations. Sufficiently robust global minimization methods, such as simulated annealing, can be very computationally expensive. Therefore, for smaller mixtures, we use root-finding calculations to calculate the critical properties of the generated compositions. For larger, more complex mixtures with multiple or nonexistent critical points, global optimization is necessitated. For these mixtures, slow but robust optimization methods should be used to avoid convergence issues in the training dataset.

When applying DNN-model initializations to global optimization methods via bounding box (method 1, Equation (5.2)) or determining the line of local minima (method 2, Equation (5.3)), the search space is drastically reduced. Therefore, it becomes feasible to perform a faster, deterministic BFS of the entire search space rather than relying on stochastic methods such as DE.

Table 5.3 displays the performance improvement achieved by switching from DE minimization to DNN-initialized BFS minimization. The number of iterations and computation time are calculated for the basic DE algorithm (without DNN-aided initializations) using the default DE parameters from Henderson et al. [13]. The DNN-initialized BFS uses machine-learning predictions with an uncertainty of 5% (based on the average uncertainty) to define a bounding box for BFS minimization, as described in Figure 5.4a. Method 2, as shown in Figure 5.4b, is also tested and found to yield similar results for the mixtures we have investigated. Across all mixtures, substantial reductions in the number of iterations and computation time (>90%) are possible when using the DNN-model predictions to initialize mixture critical point calculations. Additionally, we see that two of the fluid mixtures that do not converge when using the DE algorithm with the default parameters do converge with the DNN-initialized BFS.

5.4 Conclusions

In this work, we explore the usage of machine learning to predict mixture critical points and improve mixture critical point calculations. Using a specialized generator function, databases of random compositions are created for several fluid mixtures. Traditional critical point calculation algorithms are used to label each of these compositions. These databases are then used to train simple DNN models to predict mixture critical properties. These predictions are then further utilized to initialize the conventional critical point calculations, resulting in a reduction in the number of iterations needed to reach convergence and improved robustness.

Table 5.3: Performance improvements yielded by using the DNN-initialized DE instead of the standard DE algorithm for mixture critical point calculations. Mixtures with * require a higher learning rate (1e-2) to converge.

Mixture No.	**Convergence of Standard DE**	**Convergence of DNN-Initialized DE**	**Iteration Reduction (%)**	**Computation Time Reduction (%)**
0	✓	✓	91.258	93.906
1	✓	✓	89.224	93.936
2	✓	✓	92.119	92.140
3	✓	✓	88.814	91.556
4	✓	✓	88.793	91.381
5		✓	36.431	53.434
6	✓	✓	89.615	92.140
7*		✓	61.930	79.592
8	✓	✓	90.712	93.137
9	✓	✓	89.364	93.355
10	✓	✓	89.912	93.022
11	✓	✓	89.766	92.337
12	✓	✓	92.681	91.556
13	✓	✓	89.600	91.989
14	✓	✓	89.852	92.521
15	✓	✓	90.190	93.174
16	✓	✓	93.094	92.242
17	✓	✓	88.839	91.498
18*	✓	✓	89.265	93.178
19	✓	✓	93.832	92.145
20	✓	✓	89.259	91.989
21*	✓	✓	92.677	91.837

The prediction accuracy of the standalone DNN models, trained with 150 generated compositions, is found to be sufficient for many purposes. The standalone DNN models yield an average RMSRE of $0.71 \pm 0.58\%$ for critical temperature predictions and $2.09 \pm 2.73\%$ for critical pressure predictions for mixtures without nitrogen. For nitrogen-containing mixtures, the average RMSRE for critical temperature predictions is $2.82 \pm 1.81\%$, and the average RMSRE for critical pressure predictions is $10.8 \pm 12.1\%$. For complex mixtures with more than ten components, the average RMSRE for critical temperature predictions is less than 3.5%, while the average RMSRE for critical pressure predictions is less than 10%.

The trained DNN models are applied to initialize the DNR and DE algorithms. We can significantly reduce the number of iterations to reach convergence (30-85%) if we initialize the DNR algorithm with the DNN model that is trained with a dataset size of 150 generated compositions. For mixtures with less than six components, datasets of 50 to 150 points can result in significant iteration reductions. But for more complex mixtures, datasets of >200 points are more favorable. Meanwhile, for the global optimization formulation, using the DNN-model predictions to define a tight bound, and using a BFS algorithm instead of standard stochastic methods, allows for an 80-90% reduction in the required number of iterations.

In this work, we have shown that DNN models can be easily trained for the prediction of mixture critical points. DNN models have a fair prediction accuracy and can be easily saved, tuned, and shared. Additionally, DNN-model predictions can be used to initialize standard critical point calculation algorithms, leading to faster analytical calculations. We believe the procedure outlined in this work can improve the robustness and speed of critical point calculation algorithms for potential applications such as compositional reservoir simulations.

Chapter 6

Conclusions, Recommendations, and Future Work

6.1 Conclusions and Recommendations

In this thesis project, we explore the incorporation of modern computational techniques into mixture critical point calculations. The critical point problem is solved using the two main calculation methods presented in the literature - root finding and global optimization. Root finding is explored in the form of a damped Newton-Raphson (DNR) algorithm, while global optimization is conducted via a differential evolution (DE) algorithm. Calculations and comparisons are performed using a set of 43 (21 unique) hydrocarbon mixtures of 2-8 components, as well as a set of three complex fluids (>10 components).

Firstly, we discuss the implementation of automatic differentiation (AD) in mixture critical point calculations. We find that using AD to replace analytical derivatives causes negligible accuracy reduction and results in similar calculation speeds. Additionally, we observe faster calculations for some fluid mixtures when we use AD instead of numerical derivatives in the DE algorithm. We find AD highly effective in the computation of thermodynamic derivatives, with great potential to simplify algorithmic implementations and accommodate easy changes in equation of state (EOS) models.

Secondly, a comparison between the DNR and DE algorithms is carried out. Our

investigation shows that, with a generalized damping coefficient, the DNR algorithm can achieve a convergence rate of $> 98\%$ for 21 different fluid mixtures. Meanwhile, the DE algorithm can only achieve similar results when mixture-specific tunings are done. Additionally, the DNR algorithm yields lower deviations from experimental values than the DE algorithm (about 2 and 3%, respectively). We show that for complex mixtures with multiple or nonexistent roots, the global optimization method is more reliable due to its predictable convergence behaviour. Overall, the DNR algorithm appears to be more accurate and robust, but the DE algorithm performs well for mixtures with complex critical point phenomena (such as nonexistent or multiple critical points).

Finally, the application of deep neural networks (DNNs) to critical point calculations as prediction and acceleration techniques is investigated. DNN models have fairly good prediction accuracy for mixture critical points with less than 5% error for critical temperature predictions and less than 10% error for critical pressure predictions. These accuracies can be further improved by tuning the generated training datasets and model architectures. Next, with the DNN-aided initializations, the average number of iterations to reach convergence for the DNR algorithm decreases by 50-85%, depending on the training dataset size and the number of components in a mixture. Similarly, for the global optimization method, we obtain an 80-90% reduction in the number of iterations and enhanced robustness by using the DNN-aided initializations.

Incorporating AD and DNN-aided initialization strategies not only improves the calculations of mixture critical points but also can be extended to other thermodynamic computations. As a result, much faster, more sophisticated, and more robust algorithms can be developed. Overall, we believe this thesis work helps open the door to incorporating modern computational techniques into classical thermodynamic computations.

6.2 Future Work

Firstly, recent years have seen a rising application of the perturbed chain statistical associating fluid theory (PC-SAFT) EOS to critical point calculations. With some reformulation and utilizing AD, the DNR and DE algorithms can be implemented to allow for calculations based on the PC-SAFT EOS. Secondly, in this study, the predictions and subsequent initialization of the mixture critical point calculations are performed with no significant tuning of the training dataset and DNN architecture. The generation of the training datasets and DNN architectures can be further optimized in future studies. Thirdly, the DNN models used in this work use only the compositional data as the input data; incorporating critical properties and molecular structures as part of the input data may greatly improve the accuracy and robustness of the DNN-model predictions. Finally, in the current procedure, a unique model is needed for each unique mixture. With the incorporation of critical properties and molecular structures, a generalized mixture critical point prediction model could be potentially built.

Bibliography

[1] C. F. Spencer, T. E. Daubert, and R. P. Danner, "A critical review of correlations for the critical properties of defined mixtures," *AIChE Journal*, vol. 19, no. 3, pp. 522–526, 1973. DOI: 10.1002/aic.690190316.

[2] M. L. Michelsen, "Calculation of phase envelopes and critical points for multicomponent mixtures," *Fluid Phase Equilibria*, vol. 4, no. 1-2, pp. 1–10, 1980. DOI: 10.1016/0378-3812(80)80001-X.

[3] J. W. Gibbs, "On the equilibrium of heterogeneous substances," *American Journal of Science*, vol. 3-16, no. 96, pp. 241–458, 1878. DOI: 10.2475/ajs.s3-16.96.441.

[4] R. A. Heidemann and A. M. Khalil, "The calculation of critical points," *AIChE Journal*, vol. 26, no. 5, pp. 769–779, 1980. DOI: 10.1002/aic.690260510.

[5] D.-Y. Peng and D. B. Robinson, "A new two-constant equation of state," *Industrial and Engineering Chemistry Fundamentals*, vol. 15, no. 1, pp. 59–64, 1976. DOI: 10.1021/i160057a011.

[6] D.-Y. Peng and D. B. Robinson, "A rigorous method for predicting the critical properties of multicomponent systems from an equation of state," *AIChE Journal*, vol. 23, no. 2, pp. 137–144, 1977. DOI: 10.1002/aic.690230202.

[7] M. L. Michelsen and R. A. Heidemann, "Calculation of critical points from cubic two-constant equations of state," *AIChE Journal*, vol. 27, no. 3, pp. 521–523, 1981. DOI: 10.1002/aic.690270326.

[8] P. Dimitrakopoulos, W. Jia, and C. Li, "An improved computational method for the calculation of mixture liquid–vapor critical points," *International Journal of Thermophysics*, vol. 35, no. 5, pp. 865–889, 2014. DOI: 10.1007/s10765-014-1680-7.

[9] B. A. Stradi, "Interval mathematics applied to critical point transitions," *Revista de Matemática: Teoría y Aplicaciones*, vol. 12, no. 1/2, pp. 29–44, 2005. DOI: 10.15517/rmta.v12i1-2.248.

[10] E. Kvaalen, "A faster broyden method," *BIT Numerical Mathematics*, vol. 31, no. 1, pp. 369–372, 1991. DOI: ff10.1007/BF01931297ff.

[11] M. H. Scott and G. L. Fenves, "Krylov subspace accelerated newton algorithm: Application to dynamic progressive collapse simulation of frames," *Journal of Structural Engineering*, vol. 136, no. 5, pp. 473–480, 2010. DOI: 10.1061/ASCEST.1943-541X.0000143.

[12] J. Verbeke and R. Cools, "The newton-raphson method," *International Journal of Mathematical Education in Science and Technology*, vol. 26, no. 1, pp. 177–193, 1995. DOI: 10.1080/0020739950260202.

[13] N. Henderson, L. Freitas, and G. M. Platt, "Prediction of critical points: A new methodology using global optimization," *AIChE Journal*, vol. 50, no. 6, pp. 1300–1314, 2004. DOI: 10.1002/aic.10119.

[14] H. Zhang, A. Bonilla-Petriciolet, and G. Rangaiah1, "A review on global optimization methods for phase equilibrium modeling and calculations," *The Open Thermodynamics Journal*, vol. 5, no. Supp 1-M7, pp. 72–92, 2011. DOI: 10.2174/1874396x01105010071.

[15] D. V. Nichita and S. Gomez, "Efficient and reliable mixture critical points calculation by global optimization," *Fluid Phase Equilibria*, vol. 291, no. 2, pp. 124–140, 2010. DOI: 10.1016/j.fluid.2009.12.023.

[16] N. Henderson, W. F. Sacco, N. E. Barufattin, and M. M. Ali, "Calculation of critical points of thermodynamic mixtures with differential evolution algorithms," *Industrial and Engineering Chemistry Research*, vol. 49, no. 1, pp. 1872–1882, 2010. DOI: 10.1021/ie900948z.

[17] J. R. Elliott and C. T. Lira, *Introductory Chemical Engineering Thermodynamics*. Pearson, 2012.

[18] P. Borwein and T. Erdélyi, *Polynomials and Polynomial Inequalities*. Springer, 1995.

[19] J. McNamee, *Numerical Methods for Roots of Polynomials - Part I*. Elsevier, 2007.

[20] W. Kay, "Gases and vapors at high temperature and pressure - density of hydrocarbon," *Industrial and Engineering Chemistry*, vol. 28, no. 9, pp. 1014–1019, 1936. DOI: 10.1021/ie50321a008.

[21] R. Storm and K. Price, "Differential evolution – a simple and efficient heuristic for global optimization over continuous spaces," *Journal of Global Optimization*, vol. 11, no. 1, pp. 341–359, 1997. DOI: 10.1023/a:1008202821328.

[22] G. M. Kontogeorgis, M. L. Michelsen, G. K. Folas, S. Derawi, N. von Solms, and E. H. Stenby, "Ten years with the cpa (cubic-plus-association) equation of state. part 1. pure compounds and self-associating systems," *Ind. Eng. Chem. Res*, vol. 45, no. 14, pp. 4855–4868, 2006. DOI: 10.1021/ie051305v.

[23] D. N. Justo-García, F. García-Sánchez, N. L. Díaz-Ramírez, and A. Romero-Martínez, "Calculation of critical points for multicomponent mixtures containing hydrocarbon and nonhydrocarbon components with the pc-saft equation of state," *Fluid Phase Equilibria*, vol. 265, no. 1-2, pp. 192–204, 2008. DOI: 10.1016/j.fluid.2007.12.006.

[24] C. C. Margossian, "A review of automatic differentiation and its efficient implementation," *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery*, vol. 9, no. 4, pp. 1–19, 2019. DOI: 10.1002/widm.1305.

[25] A. G. Baydin, B. A. Pearlmutter, A. A. Radul, and J. M. Siskind, "Automatic differentiation in machine learning: A survey," *Journal of Machine Learning Research*, vol. 18, no. 1, pp. 1–43, 2018. DOI: 10.5555/3122009.3242010.

[26] V. Alves, J. R. Kitchin, and F. V. Lima, "An inverse mapping approach for process systems engineering using automatic differentiation and the implicit function theorem," *AIChE Journal*, vol. 69, no. 9, e18119, 2023. DOI: 10.1002/aic.18119.

[27] Y. Yang, S. K. Achar, and J. R. Kitchin, "Evaluation of the degree of rate control via automatic differentiation," *AIChE Journal*, vol. 68, no. 6, e17653, 2022. DOI: 10.1002/aic.17653.

[28] A. S. Silva and M. Castier, "An inverse mapping approach for process systems engineering using automatic differentiation and the implicit function theorem," *Computers and Chemical Engineering*, vol. 17, no. S1, pp. 473–478, 1993. DOI: 10.1016/0098-1354(93)80268-R.

[29] M. Iri and K. Kubota, "Encyclopedia of optimization," in Springer, 2008, ch. Automatic Differentiation: Introduction, History and Rounding Error Estimation.

[30] J. Kitchin, *Autograd and the derivative of an integral function*, Internet Article, 2018.

[31] J. Bradbury *et al.*, *JAX: Composable transformations of python+numpy programs*, version 0.3.13, 2018. [Online]. Available: http://github.com/google/jax.

[32] P. Virtanen *et al.*, "Scipy 1.0: Fundamental algorithms for scientific computing in python," *Nature Methods*, vol. 17, pp. 261–272, 2020. DOI: 10.1038/s41592-019-0686-2.

[33] R. T. Lange, "Evosax: JAX-based evolution strategies," arXiv preprint arXiv:2212.04180, 2022.

[34] A. Péneloux, E. Rauzy, and R. Fréze, "A consistent correction for redlich-kwong-soave volumes," *Fluid Phase Equilibria*, vol. 8, no. 1, pp. 7–23, 1982. DOI: 10.1016/0378-3812(82)80002-2.

[35] R. P. Brent, "Some efficient algorithms for solving nonlinear systems of equations," *SIAM Journal on Numerical Analysis*, vol. 10, no. 2, pp. 327–344, 1973. DOI: 10.1137/0710031.

[36] W. Jia, C. Li, and X. Wu, "Computation of liquid-vapor critical points for multi-component mixtures," *International Journal of Thermodynamics*, vol. 15, no. 3, pp. 149–156, 2012. DOI: 10.5541/ijot.380.

[37] K. Ghorayeb, A. Firoozabadi, and T. Anraku, "Interpretation of the unusual fluid distribution in the yufutsu gas-condensate field," *SPE Journal*, vol. 8, no. 2, pp. 114–123, 2003. DOI: 10.2118/84953-PA.

[38] L. Xu and H. Li, "A modified multiple-mixing-cell method with sub-cells for mmp determinations," *Energies*, vol. 14, no. 23, p. 7846, 2021. DOI: 10.3390/en14237846.

[39] P. D. S. Jr., "Tools for teaching," in COMAP, 1991, ch. Newton's Method and Fractal Patterns.

[40] D. Rosenberger, K. Barros, T. C. Germann, and N. Lubbers, "Machine learning of consistent thermodynamic models using automatic aifferentiation," *Physical Review E*, vol. 105, no. 1, p. 045 301, 2022. DOI: 10.1103/PhysRevE.105.045301.

[41] Y. Liu, W. Hong, and B. Cao, "Machine learning for predicting thermodynamic properties of pure fluids and their mixtures," *Energy*, vol. 188, no. 1, p. 116 091, 2019. DOI: 10.1016/j.energy.2019.116091.

[42] J. Willard, X. Jia, S. Xu, M. Steinbach, and V. Kumar, "Integrating scientific knowledge with machine learning for engineering and environmental systems," *ACM Computing Surveys*, vol. 55, no. 4, pp. 1–37, 2022. DOI: 10.1145/1122445.1122456.

[43] G. E. Karniadakis, I. G. Kevrekidis, L. Lu, P. Perdikaris, S. Wang, and L. Yang, "Physics-informed machine learning," *Nature Reviews Physics*, vol. 3, no. 1, pp. 422–440, 2021. DOI: 10.1038/s42254-021-00314-5.

[44] D. Kochkov, J. A. Smith, A. Alieva, Q. Wang, M. P. Brenner, and S. Hoyer, "Machine learning–accelerated computational fluid dynamics," *Proceedings of the National Academy of Sciences*, vol. 118, no. 21, e2101784118, 2021. DOI: 10.1073/PNAS.2101784118.

[45] H. Wang, Y. Ji, and Y. Li, "Simulation and design of energy materials accelerated by machine learning," *Wiley Interdisciplinary Reviews: Computational Molecular Science*, vol. 10, no. 1, e1421, 2019. DOI: 10.1002/wcms.1421.

[46] K. W. Ng, G.-L. Tian, and M.-L. Tang, *Dirichlet and Related Distributions: Theory, Methods and Applications*. Wiley, 2011.

[47] B. Polyak, "Newton's method and its use in optimization," *European Journal of Operational Research*, vol. 181, no. 3, pp. 1086–1096, 2007. DOI: 10.1016/j.ejor.2005.06.076.

[48] H. Li, *Multiphase Equilibria of Complex Reservoir Fluids*. Springer, 2022.

[49] R. Taylor, H. Kooijman, J. van Baten, and R. Baur, "The need for mole fraction derivatives," ChempSep, Tech. Rep., 2006.

Appendix A: Derivations of Key Equations

A.1 Generalized Cubic Polynomial

A.1.1 Polynomial Form

For a two-constant cubic equation of state with no additional terms we have the generic form for compressibility factor (Z) [7]:

Definition 1 $Z = \frac{1}{1-b\rho} - \frac{a\rho}{RT}\frac{1}{(1+\delta_1 b\rho)(1+\delta_2 b\rho)}$

Define non-dimensionalized cubic EOS constants:

Definition 2 $A = \frac{aP}{(RT)^2}, \quad B = \frac{bP}{RT}, \quad b\rho = \frac{B}{Z}$

Substituting and dividing both sides by a factor of Z,

$Z = \frac{1}{1-\frac{B}{Z}} - \frac{A}{Z}\frac{1}{(1+\delta_1\frac{B}{Z})(1+\delta_2\frac{B}{Z})}$

$1 = \frac{1}{Z-B} - \frac{A}{(Z+\delta_1 B)(Z+\delta_2 B)}$

Rewriting this equation as a polynomial gives the expression,

$(Z-B)(Z+\delta_1 B)(Z+\delta_2 B) = (Z+\delta_1 B)(Z+\delta_2 B) - A(Z-B)$

$0 = -Z^3 + [1-(\delta_1+\delta_2-1)B]Z^2 + [(\delta_1+\delta_2-\delta_1\delta_2)B^2 +$

$(\delta_1+\delta_2)B - A]Z + AB + \delta_1\delta_2(B^2+B^3)$

Then we define combinations of EOS parameters as additional EOS parameters to obtain he final form:

Definition 3 $\delta_3 = \delta_1 + \delta_2, \quad \delta_4 = \delta_3\delta_4, \quad \delta_5 = \delta_1 - \delta_2$

Theorem 1 $0 = Z^3 + [(\delta_3 - 1)B - 1]Z^2 + [A - (\delta_3 - \delta_4)B^2 - \delta_3 B]Z - AB - \delta_4(B^2+B^3)$

A.1.2 Cubic Roots

Solving the compressibility factor cubic equation is necessary for both critical point calculation methods.

Algorithm 1 *Roots of cubic compressibility factor equation for a generalized two constant EOS.*

$0 = Z^3 + [(\delta_3 - 1)B - 1]Z^2 + [A - (\delta_3 - \delta_4)B^2 - \delta_3 B]Z - AB - \delta_4(B^2 + B^3)$

$0 = c_1 Z^3 + c_2 Z^2 + c_3 Z + c_4$

Discriminant Δ *indicates root types.* $\Delta = 18c_1c_2c_3c_4 - 4c_2^3c_4 + c_2^2c_3^2 - 4c_1c_3^3 - 27c_1^2c_4^2$

$\Delta = 0$*: Three real roots with repeated root(s).*

$c_2^2 == 3c_1c_3$*: Triple multiplicity root which is calculated by:*

$Z_1, Z_2, Z_3 = -\frac{c_2}{3c_1}$

$c_2^2 \neq 3c_1c_3$*: Distinct root and repeated Root, calculated by:*

$Z_1 = \frac{4c_1c_2c_3 - 9c_1^2c_4 - c_2^3}{c_1(c_2^2 - 3c_1c_3)}, Z_2, Z_3 = \frac{9c_1c_4 - c_2c_3}{2*(c_2^2 - 3c_1c_3)}$

$\Delta > 0$*: Three distinct real roots.*

$p_1 = \frac{3c_1c_3 - c_2^2}{3c_1^2}, \quad p_2 = \frac{2c_2^3 - 9c_1c_2c_3 + 27c_1^2c_4}{27c_1^3}, \quad \theta = \frac{3p_2}{2p_1}\sqrt{\frac{-3}{p_1}}$

$Z_{j=1,2,3} = \sqrt{\frac{-2p_1}{3}}cos(\frac{cos^{-1}(\theta)}{3} - \frac{2(j-1)\pi}{3})$

$\Delta < 0$*: One real root with two complex conjugates.*

$d_0 = c_2^2 - 3c_1c_3, \quad d_1 = 2c_2^3 - 9c_1c_2c_3 + 27c_1^2c_4$

$C = \sqrt[3]{\frac{d_1 \pm \sqrt{d_1^2 - 4d_0^3}}{2}}$

$Z_1 = \frac{-1}{3c_1}(c_2 + C + \frac{d_0}{C})$

A.2 Fugacity Coefficient Derivation

A.2.1 Differentiation of Mixing Rules

Standard Van Der Waals mixing rules are traditionally combined with cubic EOS [48]. For the Henderson et al. formation, these mixing rules are differentiated with the relation $\sum_i^N x_i = 1$ [13]; therefore, to allow for compatibility with AD, we define the mixing rules as follows:

Definition 4 $a_i = \frac{\Omega_1 (RT_{ci})^2}{P_{ci}}[1 + m_i(1 - \sqrt{(}\frac{T}{T_{ci}}))]^2, \quad m_i = p_0 + p_1 + \omega_i + p_2\omega_i^2$
$a_{ij} = (1 - k_{ij})\sqrt{(}a_i a_j), \quad b_i = \frac{\Omega_2 RT_{ci}}{P_{ci}}$
$a = a_{NN} + 2\sum_{i=1}^{N-1} x_i(a_{Ni} - a_{NN}) + \sum_{j=1}^{N-1}\sum_{i=1}^{N-1} x_i x_j(a_{ij} - 2a_{Ni} + a_{NN}),$
$b = b_N + \sum_{i=1}^{N-1} x_i(b_i - b_N),$
$\alpha_i = \frac{a_{Ni} + \sum_{j=1}^{N-1} x_j(a_{ji} - a_{Ni})}{a}, \quad \beta_i = \frac{b_i}{b}$

Derivatives of EOS constants then become the forms below, and their respective mole number derivative can be calculated as described in the literature [49]:

$\frac{\partial a}{\partial x_k} = 2a(\alpha_k - \alpha_N), \quad \frac{\partial b}{\partial x_k} = b(\beta_k - \beta_N)$
$\frac{\partial A}{\partial x_k} = 2A(\alpha_k - \alpha_N), \quad \frac{\partial B}{\partial x_k} = B(\beta_k - \beta_N)$
$\frac{\partial \alpha_i}{\partial x_k} = \frac{a_{ik} - a_{Ni}}{a} - 2\alpha_i(\alpha_k - \alpha_N), \quad \frac{\partial \beta_i}{\partial x_k} = \beta_i(\beta_k - \beta_N)$
$\frac{\partial f}{\partial n_k} = -\frac{1}{n_t}\sum_{j \neq k} \frac{\partial f}{\partial x_j}$

A.2.2 Gibbs Departure Form

The fugacity coefficient of a component in a mixture can be defined in terms of chemical potential as follows [17]:

Definition 5 $ln(\phi_i) = ln(\frac{f_i}{y_i P}) = \frac{\mu_i - \mu_{i0}}{RT}$

The right-hand side of the equation is the chemical potential departure function. This is further expanded as the compositional derivative of the Gibbs' departure function [17].

$ln(\phi_i) = \mu_i^{DEP} = (\frac{\partial G^{DEP}}{\partial n_i})|_{T,P,n_{i \neq j}}$

In turn, the Gibbs departure function is defined based on the selected equation of state as,

Definition 6 $G^{DEP} = \int_0^\rho \frac{Z-1}{\rho}\, d\rho + (Z-1) - ln(Z)$

Utilizing the generic cubic definition of compressibility factor we obtain:

$ln(\phi_i) = \frac{\partial}{\partial n_i} \int_0^\rho \frac{Z-1}{\rho}\, d\rho + (Z-1) - ln(Z)$

$Z = \frac{1}{1-b\rho} - \frac{a\rho}{RT} \frac{1}{(1+\delta_1 b\rho)(1+\delta_2 b\rho)}$

$G^{DEP} = \int_0^\rho \frac{b}{1-b\rho} - \frac{a}{RT} \frac{1}{(1+\delta_1 b\rho)(1+\delta_2 b\rho)}\, d\rho + (Z-1) - ln(Z)$

$G^{DEP} = ln(1-b\rho) - \frac{a}{bRT} \frac{1}{\delta_1 - \delta_2} ln(\frac{1+\delta_1 b\rho}{1+\delta_2 b\rho}) + (Z-1) - ln(Z)$

Introducing non-dimensionalized variables from Definition 2, we can obtain:

$G^{DEP} = ln(\frac{Z-B}{Z^2}) - \frac{A}{B} \frac{1}{\delta_1 - \delta_2} ln(\frac{Z+\delta_1 B}{Z+\delta_2 B}) + (Z-1)$

A.2.3 Fugacity Coefficient Expression

We can take the compositional partial derivatives to obtain the fugacity coefficient expression:

$ln(\phi_i) = \frac{\partial}{\partial n_i}(ln(\frac{Z-B}{Z^2}) - \frac{A}{B\delta_5}ln(\frac{Z+\delta_1 B}{Z+\delta_2 B}) + (Z-1)|_{T,P,n_{i\neq j}}$

$ln(\phi_i) = -\frac{1}{n_t}\sum_{j\neq i}\frac{\partial}{\partial x_j}x_i(E_1 - \frac{E_2E_3}{\delta_5} + E_4)$

$ln(\phi_i) = -\frac{1}{n_t}\sum_{j\neq i}(\frac{\partial}{\partial x_j}(x_iE_1 - \frac{x_iE_2E_3}{\delta_1-\delta_2} + x_iE_4))$

$\frac{\partial Z}{\partial x_i} = \frac{\partial b}{\partial x_i}(\frac{1}{1-b\rho})^2 - \frac{\rho}{RT(1+\delta_1 b\rho)^2(1+\delta_2 b\rho)^2}$
$\left(\frac{\partial a}{\partial x_i}(1+\delta_1 b\rho)(1+\delta_2 b\rho) - ((1+\delta_2 b\rho)\delta_1 + (1+\delta_1 b\rho)\delta_2)\, a\rho\frac{\partial b}{\partial x_i}\right)$
$= \frac{\partial b}{\partial x_i}(\frac{1}{1-b\rho})^2 - \frac{\rho}{RT(1+\delta_3 b\rho+\delta_4 b^2\rho^2)^2}\left(\frac{\partial a}{\partial x_i}(1+\delta_3 b\rho+\delta_4 b^2\rho^2) - (\delta_3+2\delta_4 b\rho)a\rho\frac{\partial b}{\partial x_i}\right)$
$= \frac{\partial B}{\partial x_i}\frac{Z}{(Z-B)^2} - \frac{Z^3}{(Z^2+\delta_3 ZB+\delta_4 B^2)^2}\left(\frac{\partial A}{\partial x_i}(Z^2+\delta_3 ZB+\delta_4 B^2) - A\frac{\partial B}{\partial x_i}(Z\delta_3+2\delta_4 B)\right)$
$\frac{\partial Z}{\partial x_i} = B(\beta_i-\beta_N)\left[\frac{Z}{(Z-B)^2} - \frac{AZ^3}{Z^2+\delta_3 ZB+\delta_4 B^2}\left(\frac{2(\alpha_i-\alpha_N)}{B(\beta_i-\beta_N)} - \frac{Z\delta_3+2\delta_4 B}{Z^2+\delta_3 ZB+\delta_4 B^2}\right)\right]$

$\frac{\partial E_1}{\partial x_i} = \frac{2\frac{\partial Z}{\partial x_i}B - Z(\frac{\partial Z}{\partial x_i}+\frac{\partial B}{\partial x_i})}{Z(Z-B)} = \frac{\frac{\partial Z}{\partial x_i} - B(\beta_i-\beta_N)}{Z-B}$
$\frac{\partial E_2}{\partial x_i} = \frac{1}{B^2}(\frac{\partial A}{\partial x_i}B - A\frac{\partial B}{\partial x_i})$
$\frac{\partial E_3}{\partial x_i} = \frac{\delta_5(Z\frac{\partial B}{\partial x_i} - \frac{\partial Z}{\partial x_i}B)}{Z^2+\delta_3 ZB+\delta_4 B^2}$
$\frac{\partial E_4}{\partial x_i} = \frac{\partial Z}{\partial x_i}$

Theorem 2 $\ln\varphi_i = \beta_i(Z-1) - \ln(Z-B) - \frac{A}{B}\ln(\frac{Z+\delta_1 B}{Z+\delta_2 B})(\frac{2\alpha_i-\beta_i}{\delta_5})$

Further simplification of the derivatives of E_{1-4} is described in "*Introductory Chemical Engineering Thermodynamics*" by Elliot and Lira [17].

Appendix B: Mixture and Chemical Properties

B.1 Simple Mixture Set

Table B.1: Mixtures from Dimitrakopolous et al. [8]. Unique mixtures used for machine learning are in bold.

Mixture No.	CO_2	N_2	Methane	Ethane	Propane	iButane	nButane	iPentane	nPentane	nHexane	nHeptane
1			0.1	0.9							
2	0.95	0.05									
3				0.429			0.373				0.198
4				0.726			0.171				0.103
5				0.514			0.412				0.074
6				0.801				0.064			0.135
7				0.612				0.271			0.117
8				0.615				0.296			0.089

Table B.1: Mixtures from Dimitrakopolous et al. [8]. Unique mixtures used for machine learning are in bold.

Mixture No.	CO_2	N_2	Methane	Ethane	Propane	iButane	nButane	iPentane	nPentane	nHexane	nHeptane
9		0.0465	0.453		0.5005						
10			0.193	0.47			0.337				
11			0.391	0.354			0.255				
12			0.04	0.821			0.139				
13			0.007	0.879			0.114				
14			0.461	0.443					0.095		
15			0.196	0.758					0.045		
16				0.996	0.001		0.003				
17				0.99	0.004		0.006				
18				0.98	0.016		0.004				
19				0.97	0.027		0.003				
20				0.3414	0.3421				0.3165		
21					0.3276		0.3398		0.3326		
22					0.201		0.399		0.4		
23					0.201		0.298		0.501		
24					0.198		0.106		0.696		

Table B.1: Mixtures from Dimitrakopolous et al. [8]. Unique mixtures used for machine learning are in bold.

Mixture No.	CO_2	N_2	Methane	Ethane	Propane	iButane	nButane	iPentane	nPentane	nHexane	nHeptane
25							0.6449		0.2359	0.1192	
26			0.833	0.13	0.035						
27			0.8	0.039	0.161						
28		0.049	0.4345	0.0835	0.433						
29				0.6168			0.1376	0.0726			0.173
30				0.2542	0.2547		0.2554		0.2357		
31					0.4858		0.3316		0.1213	0.0613	
32		0.033	0.91	0.056	0.0012						
33		0.015	0.959	0.026	0.0001						
34		0.016	0.95	0.026	0.0078						
35				0.3977	0.2926		0.1997		0.0713	0.0369	
36			0.2019	0.2029	0.2033		0.2038		0.1881		
37		0.016	0.945	0.026	0.0081		0.0052				
38			0.1015	0.3573	0.2629		0.1794		0.0657	0.0332	
39		0.022	0.316	0.388	0.223		0.043		0.008		
40		0.014	0.943	0.027	0.0074		0.0049		0.001	0.0027	

Table B.1: Mixtures from Dimitrakopolous et al. [8]. Unique mixtures used for machine learning are in bold.

Mixture No.	CO_2	N_2	Methane	Ethane	Propane	iButane	nButane	iPentane	nPentane	nHexane	nHeptane
41	0.0109	0.0884	0.8286	0.0401	0.0174	0.003	0.0055	0.0019	0.0012	0.0014	0.0006
42	0.002	0.24	0.7364	0.012	0.0053	0.001	0.0015	0.0005	0.0004	0.00004	0.0005
43	0.003	0.113	0.858	0.015	0.006	0.0012	0.0018	0.0006	0.0004	0.0004	0.0006
44	0.01	0.1611	0.7625	0.0369	0.016	0.0028	0.0051	0.0018	0.0011	0.0012	0.0015

Table B.2: Pure component critical properties as described by Dimitrakopolous et al. [8].

Component	**Pc (MPa)**	**Tc (K)**	**Vc (L/mol)**	ω
CO_2	7.374	304.12	0.0939569	0.225
N_2	3.398	126.2	0.0892421	0.037
Methane	4.599	190.56	0.0985305	0.011
Ethane	4.872	305.32	0.145375	0.099
Propane	4.248	369.83	0.199785	0.152
iButane	3.65	408.2	0.2585	0.183
nButane	3.796	425.12	0.255136	0.2
iPentane	3.39	460.4	0.307143	0.227
nPentane	3.37	469.7	0.310571	0.252
nHexane	3.025	507.6	0.368329	0.3
nHeptane	2.74	540.2	0.427839	0.35

Table B.3: Binary interaction parameters as described by Dimitrakopolous et al. [8].

Component	CO_2	N_2	Methane	Ethane	Propane	iButane	nButane	iPentane	nPentane	nHexane	nHeptane
CO_2	0.0000	-0.0200	0.1000	0.1298	0.1350	0.1298	0.1298	0.1250	0.1250	0.1250	0.1199
N_2	-0.0200	0.0000	0.0360	0.0500	0.0800	0.0950	0.0900	0.0950	0.1000	0.1490	0.1439
Methane	0.1000	0.0360	0.0000	0.0022	0.0068	0.0131	0.0123	0.0176	0.0179	0.0235	0.0289
Ethane	0.1298	0.0500	0.0022	0.0000	0.0013	0.0046	0.0041	0.0074	0.0076	0.0114	0.0153
Propane	0.1350	0.0800	0.0068	0.0013	0.0000	0.0010	0.0008	0.0026	0.0027	0.0051	0.0079
iButane	0.1298	0.0950	0.0131	0.0046	0.0010	0.0000	0.0000	0.0003	0.0004	0.0019	0.0036
nButane	0.1298	0.0900	0.0123	0.0041	0.0008	0.0000	0.0000	0.0005	0.0005	0.0019	0.0036
iPentane	0.1250	0.0950	0.0176	0.0074	0.0026	0.0003	0.0005	0.0000	0.0000	0.0004	0.0015
nPentane	0.1250	0.1000	0.0179	0.0076	0.0027	0.0004	0.0005	0.0000	0.0000	0.0004	0.0014
nHexane	0.1250	0.1490	0.0235	0.0114	0.0051	0.0019	0.0019	0.0004	0.0004	0.0000	0.0003
nHeptane	0.1199	0.1439	0.0289	0.0153	0.0079	0.0036	0.0036	0.0015	0.0014	0.0003	0.0000

B.2 Complex Mixture Set

Table B.4: Mixture compositions for the 13-component mixture from Ghorayeb et al. [16].

Mixture No.	C30+	C25-29	C20-24	C16-19	C13-15	C10-12	C7-9	C6	n/i-C5	n/i-C4	C3	C2	C1,CO_2,N_2
1	0.0049	0.0134	0.0059	0.0074	0.0088	0.0126	0.0297	0.0065	0.0093	0.0228	0.0378	0.0815	0.7696
2	0.007	0.0191	0.0203	0.0182	0.0172	0.0202	0.0404	0.008	0.0108	0.0248	0.0389	0.0806	0.6945
3	0.004	0.0389	0.0334	0.026	0.0223	0.0242	0.0449	0.0086	0.0111	0.0247	0.0382	0.0784	0.6453

Table B.5: Mixture compositions for the 12-component mixture from Xu and Li [38].

Mixture No.	CO_2	CH_4	C2	C3	C4	C5
1	0.121556	0.379469	0.120906	0.128456	0.026601	0.009350
2	0.133343	0.220989	0.141543	0.157692	0.033048	0.012349
Mixture No.	C6	C7+(1)	C7+(2)	C7+(3)	C7+(4)	C7+(5)
1 (cont.)	0.010900	0.089554	0.045502	0.030252	0.022351	0.015101
2 (cont.)	0.014899	0.126244	0.064247	0.042748	0.031548	0.021349

Table B.6: Pure component critical properties for the 13 component mixture as described by Ghorayeb et al. [16]. V_c is calculated based on Peng-Robinson critical compressibility.

Components	**Pc (Mpa)**	**Tc (K)**	**Vc (L/mol)**	ω
C30+	0.686	830	3.08939	1.505
C25-29	1.108	818.92	1.88721	1.399
C20-24	1.328	778.92	1.49766	1.168
C16-19	1.679	726.96	1.10555	0.94
C13-15	2.031	679.05	0.85371	0.776
C10-12	2.453	622.29	0.64776	0.611
C7-9	2.969	554.13	0.47656	0.428
C6	3.396	506.35	0.38072	0.299
n/i-C5	3.33	469.6	0.36008	0.251
n/i-C4	3.799	425.18	0.28577	0.193
C3	4.19	369.8	0.22536	0.152
C2	4.883	305.4	0.15970	0.098
C1,CO_2,N_2	4.6	190.6	0.10580	0.008

Table B.7: Pure component critical properties for the 12-component mixture as described by Xu and Li [38]. V_c is calculated based on Peng-Robinson critical compressibility.

Components	**Pc (Mpa)**	**Tc (K)**	**Vc (L/mol)**	ω
CO_2	7.374	304.14	0.10542	0.228
CH_4	4.592	190.6	0.10609	0.008
C2	4.875	305.4	0.16011	0.098
C3	4.238	369.8	0.22302	0.152
C4	3.793	425.2	0.28652	0.193
C5	3.368	469.6	0.35636	0.251
C6	2.964	507.4	0.43753	0.296
C7+(1)	2.883	616.2	0.54628	0.454
C7+(2)	1.932	698.9	0.92458	0.787
C7+(3)	1.659	770.4	1.18688	1.048
C7+(4)	1.527	853.1	1.42790	1.276
C7+(5)	1.467	1001.2	1.7443	1.299

Table B.8: Binary interaction parameters for the 13-component fluid from Ghorayeb et al. [16].

Component	C1,CO_2,N_2
C30+	0.024
C25-C29	0.02
C20-C24	0.02
C16-C19	0.019
C13-15	0.015
C10-12	0.01
C7-9	0.009

Table B.9: Binary interaction parameters for the 12-component fluid from Xu and Li [38]. CO_2 binary interactions are taken from Xu and Li [38], and other parameters are taken from Dimitrakopolous et al. [8].

Component	CO_2	CH_4	C2	C3	C4	C5	C6	**Component**	CO_2
CO_2	0	0.12	0.15	0.15	0.15	0.15	0.15	C7+(1)	0.15
CH_4	0.12	0	0.0022413	0.0068288	0.01270905	0.01777645	0.0288643	C7+(2)	0.15
C2	0.15	0.0022413	0	0.0012579	0.004335	0.0075114	0.0114138	C7+(3)	0.15
C3	0.15	0.0068288	0.0012579	0	0.00092975	0.00264195	0.005142	C7+(4)	0.15
C4	0.15	0.01270905	0.004335	0.00092975	0	0.000444625	0.0018663	C7+(5)	0.15
C5	0.15	0.01777645	0.0075114	0.00264195	0.000444625	0	0.0004167		
C6	0.15	0.0288643	0.0114138	0.005142	0.0018663	0.0004167	0		

Appendix C: Automatic Differentiation Compliant Code

C.1 Dampened Newton-Raphson

The following code implements the Damped Newton-Raphson Method as described by Dimitrakopolous et al. [8]. This code utilizes Jax for automated differentiation. All packages are the property of their respective authors.

The following packages are required:

- funtools.partial
- jax
- jax.numpy
- time

Algorithm C.1: Automatic Differentiation Compliant Damped Newton-Raphson

```
#N_bar, the total change in mols.
def N_bar(dnvec):
    import jax.numpy as jnp
    return jnp.sum((dnvec))

#nu, a constant dependant on EOS, that affects a.
def nu(EOS):
    if EOS == 'SRK':
        return 0.42748
    if EOS == 'PR':
        return 0.45724

#c, the accentricity polynomial.
def c(i, EOS, w):
    if EOS == 'SRK':
        ci = 0.48
        ci += 1.574*w[i]
        ci += -0.176*w[i]**2
        return ci
    if EOS == 'PR':
        ci = 0.37464
        ci += 1.54226*w[i]
        ci += -0.26992*w[i]**2
        return ci

#ai, the attraction parameter of component i.
def ai(i, Tc_mx, Tc, R, EOS, Pc, w):
    ai = (R*Tc[i])**2*nu(EOS)/Pc[i]
    ai = ai*(1 + c(i, EOS, w)*(1-(Tc_mx/Tc[i])**0.5))**2
    return ai

#aij, the binary attraction parameter of component system i-j.
def aijf(Tc_mx, Tc, R, EOS, Pc, w, k, C):
    import jax.numpy as jnp
    aij = jnp.zeros([C, C])
    for i in range(C):
        for j in range(C):
            aij = aij.at[i,j].set((ai(i, Tc_mx, Tc, R, EOS, Pc, w)*ai(j, Tc_mx, Tc, R, EOS, Pc, w))**0.5*(1-k[i]
    return aij

#bi, the covolume parameter of component i.
def bi(i, EOS, R, Tc, Pc):
    if EOS == 'SRK':
        bi = 0.08664*R*Tc[i]/Pc[i]
        return bi
    if EOS == 'PR':
        bi = 0.07780*R*Tc[i]/Pc[i]
        return bi

#D1 and D2, parameters that defines the EOS.
def D1(EOS):
    import jax.numpy as jnp
    if EOS == 'SRK':
        u0 = 1
        w0 = 0
    if EOS == 'PR':
        u0 = 2
        w0 = -1
    D1 = (u0 + jnp.sqrt(u0**2-4*w0))/2
    return D1
```

```
def D2(EOS):
    import jax.numpy as jnp
    if EOS == 'SRK':
        u0 = 1
        w0 = 0
    if EOS == 'PR':
        u0 = 2
        w0 = -1
    D2 = (u0 - jnp.sqrt(u0**2-4*w0))/2
    return D2

#a_tot, the weighted sum of binary attraction interactions.
def a_tot(n, n_tot, aij, C):
    a_t = 0
    for i in range(C):
        for j in range(C):
            a_t += n[i]*n[j]/n_tot**2*aij[i, j]
    return a_t

#b_tot, the weighted sum of covolume interactions.
def b_tot(y, EOS, R, Tc, Pc, C):
    b_t = 0
    for i in range(C):
        b_t += y[i]*bi(i, EOS, R, Tc, Pc)
    return b_t

#alphak, the percentage of total attraction parameter from component k.
def alpha(j, y, aij, n, n_tot, C):
    alpk = 0
    for i in range(C):
        alpk += y[i]*aij[i, j]
    return alpk/a_tot(n, n_tot, aij, C)

#alpha_bar, total change in the alphaks due to delta_n.
def alpha_bar(dnvec, y, aij, n, n_tot, C):
    alp_bar = 0
    for i in range(C):
        alp_bar += dnvec[i]*alpha(i, y, aij, n, n_tot, C)
    return alp_bar

#a_bar, the relative change in a_tot due to delta_n.
def a_bar(dnvec, aij, n, n_tot, C):
    a_b = 0
    for i in range(C):
        for j in range(C):
            a_b += dnvec[i]*dnvec[j]*aij[i,j]
    a_b = a_b/a_tot(n, n_tot, aij, C)
    return a_b

#betai, the percentage of total covolume parameter from component k.
def beta(i, EOS, R, Tc, Pc, y, C):
    return bi(i, EOS, R, Tc, Pc)/b_tot(y, EOS, R, Tc, Pc, C)

#beta_bar, the total change in betai due to delta_n.
def beta_bar(dnvec, EOS, R, Tc, Pc, y, C):
    bet_bar = 0
    for i in range(C):
        bet_bar += dnvec[i]*beta(i, EOS, R, Tc, Pc, y, C)
    return bet_bar

#K, the nondimensional ratio of critical molar volume of mixture to covolume parameter.
def K(Vc_mx, y, EOS, R, Tc, Pc, C):
```

```
    return Vc_mx/b_tot(y, EOS, R, Tc, Pc, C)

#F1-F6 are EOS based factors which are f(D1, D2, K).
def F1(Kv, EOS):
    return 1/(Kv-1)
def F2(Kv, EOS):
    F2 = 2/(D1(EOS)-D2(EOS))
    F2 *= (D1(EOS)/(Kv+D1(EOS))-D2(EOS)/(Kv+D2(EOS)))
    return F2
def F3(Kv, EOS):
    F3 = 1/(D1(EOS)-D2(EOS))
    F3 *= ((D1(EOS)/(Kv+D1(EOS)))**2-(D2(EOS)/(Kv+D2(EOS)))**2)
    return F3
def F4(Kv, EOS):
    F4 = 1/(D1(EOS)-D2(EOS))
    F4 *= ((D1(EOS)/(Kv+D1(EOS)))**3-(D2(EOS)/(Kv+D2(EOS)))**3)
    return F4
def F5(Kv, EOS):
    import jax.numpy as jnp
    F5 = 2/(D1(EOS)-D2(EOS))
    F5 *= jnp.log((Kv+D1(EOS))/(Kv+D2(EOS)))
    return F5
def F6(Kv, EOS):
    import jax.numpy as jnp
    F6 = 2/(D1(EOS)-D2(EOS))
    F6 *= (D1(EOS)/(Kv+D1(EOS))-D2(EOS)/(Kv+D2(EOS)))-jnp.log((Kv+D1(EOS))/(Kv+D2(EOS)))
    return F6

"""Generalized vector function that calculates the various equations in the N+2 system
   based on the Gibbs Criticality Conditions.
Inputs:
        xvec - A vector consisting of
               [Mixture Critical Temp. (K), Mixture Critical Molar Volume (cum/mol), Mol Number Deviations]
        y - A vector of component compositions.
        Tc - A vector of component critical temperatures in the order of y. (K)
        Pc - A vector of component critical pressures in the order of y. (Pa)
        w - A vector of component accentricity factors in the order of y.
        EOS - Equation of state. 'PR' or 'SRK' implemented.
        C - Number of components in mixture.
        n - A vector of component mole number. Unused, only for compatibility with other functions. (mol)
        n_tot - Total moles. (mol)
        R - Gas constant. (J/(molK))
        Vc - A vector of component crit. molar volumes.
             Unused, only for compatibility with other functions. (cum/mol)
        k - Matrix of binary interaction parameters with component
            in order of y as columns and rows. Main diagonal of 0.

Outputs:
        fvec - A vector of the values of each of the Gibbs criticality functions.
        """
def GeneralizedCubicFunction(xvec, y, Tc, Pc, w, EOS, C, n, n_tot, R, Vc, k):
    import jax.numpy as jnp
    #Unpack input xvec.
    Tc_mx = xvec[0]
    Vc_mx = xvec[1]
    dnvec = xvec[2:]

    fvec = jnp.zeros([1,len(xvec)])

    #Calculate universal variable values
    aij = aijf(Tc_mx, Tc, R, EOS, Pc, w, k, C)
    a_totv = a_tot(n, n_tot, aij, C)
```

```
    b_totv = b_tot(y, EOS, R, Tc, Pc, C)
    N_barv = N_bar(dnvec)
    a_barv = a_bar(dnvec, aij, n, n_tot, C)
    alpha_barv = alpha_bar(dnvec, y, aij, n, n_tot, C)
    beta_barv = beta_bar(dnvec, EOS, R, Tc, Pc, y, C)

    Kv = K(Vc_mx, y, EOS, R, Tc, Pc, C)

    F1v = F1(Kv, EOS)
    F2v = F2(Kv, EOS)
    F3v = F3(Kv, EOS)
    F4v = F4(Kv, EOS)
    F5v = F5(Kv, EOS)
    F6v = F6(Kv, EOS)

    #First GCC - First derivative of each component's Helmholtz Energy should be 0. N non-linear equations.
    for i in range(C):
        betav = beta(i, EOS, R, Tc, Pc, y, C)
        pA = R*Tc_mx/n_tot
        pB = a_totv/(b_totv*n_tot)
        A1 = dnvec[i]/y[i]
        A2 = F1v*(betav*N_barv + beta_barv)
        A3 = betav*F1v**2*beta_barv
        B1 = betav*beta_barv*F3v
        B2 = 0
        for j in range(C):
            B2 += dnvec[j]*aij[i,j]
        B2 = -F5v/a_totv*B2
        B3 = F6v*(betav*beta_barv-alpha(i, y, aij, n, n_tot, C)*beta_barv-alpha_barv*betav)
        A = pA*(A1+A2+A3)
        B = pB*(B1+B2+B3)
        fvec = fvec.at[0,i].set(A+B)

    #Second GCC - Sum of all second derivatives of Helmholtz Energy should be 0. 1 non-linear equation.
    pA = R*Tc_mx/n_tot**2
    pB = a_totv/(b_totv*n_tot**2)
    A1 = 0
    for i in range(C):
        A1 += dnvec[i]**3/y[i]**2
    A1 = -A1
    A2 = 3*(N_barv*(beta_barv*F1v)**2)
    A3 = 2*((F1v*beta_barv)**3)
    B1 = 3*(beta_barv**2)*((2*alpha_barv-beta_barv)*(F3v+F6v))
    B2 = -2*(beta_barv**3)*F4v
    B3 = -3*beta_barv*a_barv*F6v
    A = pA*(A1+A2+A3)
    B = pB*(B1+B2+B3)
    fvec = fvec.at[0,C].set(A+B)

    #Third GCC - Euclidean distance of delta_n should be 1. 1 non-linear equation.
    norm_mols = 0
    for i in range(C):
        norm_mols += dnvec[i]**2
    norm_mols += -1
    fvec = fvec.at[0,C+1].set(norm_mols)
    return fvec

"""Initialization for NR method with no provided initialization.
   Based in Kay's Mixing Rule as described by Dimitrakopolous et al.
Inputs:
        y - A vector of component compositions.
        Vc - A vector of component critical molar volumes in the order of y. (cum/mol)
```

```
        Tc - A vector of component critical temperatures in the order of y. (K)
        C - Number of components in mixture.


Outputs:
        Tc0 - Initial mixture critical temperature guess.
        Vc0 - Initial mixture critical molar volume guess.
        dn0 - Initial mixture mole number deviations.
        """
def InitializeNR(y, Vc, Tc, C):
    import jax.numpy as jnp
    #Initialize first guess based on composition and component critical values.
    dn0 = jnp.zeros(jnp.shape(y))
    Tc0 = 0
    Vc0 = 0
    for i in range(C):
        Vc0 += y[i]*Vc[i]
        Tc0 += y[i]*Tc[i]
        dn0 = dn0.at[i].set(y[i]**(2/3))
    Tc0 = 3*Tc0
    return Tc0, Vc0, dn0

"""Generalized vector function that calculates the various equations
   in the N+2 system based on the Gibbs Criticality Conditions.
Inputs:
        y - A vector of component compositions.
        Tc - A vector of component critical temperatures in the order of y. (K)
        Pc - A vector of component critical pressures in the order of y. (Pa)
        w - A vector of component accentricity factors in the order of y.
        C - Number of components in mixture.
        R - Gas constant. (J/(molK))
        Vc - A vector of component critical molar volumes in the order of y. (cum/mol)
        k - Matrix of binary interaction parameters with component
            in order of y as columns and rows. Main diagonal of 0.
        n_tot - Total moles. (mol)
        EOS - Equation of state. 'PR' or 'SRK' implemented.
        ini - Optional initialization, overwriting Kay's Mixing Rule.
        printflag - Optional iteration printouts to external file.
Outputs:
        Fmat - Function values at calculated root
        xvec - Final x vector values in format: [Temp., Mol. Vol., Mole Dev.]
        Pc - Calculated critical pressure using EOS. (Pa)
        itr - Iterations to convergence.
        rn_time - Run time. (s)
        """
def CompNewtonRaphson(y, Tc, Pc, w, C, R, Vc, k, n_tot, EOS, ini = None, printflag=True):
    from functools import partial
    import jax
    import jax.numpy as jnp
    import time
    import KeyFunctions as me
    from IPython.display import clear_output

    n = y*n_tot

    #Initialize x0vec with default initialization or provided intialization.
    temptc, tempvc, tempdn = InitializeNR(y, Vc, Tc, C)
    if ini is None:
        x0vec = jnp.array([temptc, tempvc])
        x0vec = jnp.append(x0vec, tempdn)
    elif len(ini) == 2:
        x0vec = jnp.array(ini)
```

```
        x0vec = jnp.append(x0vec, tempdn)
    else:
        x0vec = jnp.array(ini)


    if printflag:
        print("Initial X-Vector Guess:")
        print(x0vec)
        print("--------------------------------------------")

    start_time = time.time()


    #Create Jacobian matrix function and initialize F matrix.
    Fmat = jnp.zeros([1, len(x0vec)])
    jitf = partial(jax.jit, static_argnames=['EOS', 'C', 'n_tot', 'R'])
    Jmat = jax.jacfwd(jitf(GeneralizedCubicFunction))

    #Iterate with x(k+1) = x(k) + D*dx.
    xvec = x0vec
    dxvec = jnp.ones(jnp.shape(xvec))
    itr = 0
    itr_time = 0

    if printflag:
        print("dxVector Magnitudes:")
    while (abs(dxvec[0])>1e-4 or abs(dxvec[1])>10e-8 or any(abs(dxvec[2:])>1e-4)):
        #Count iterations for dampening factor.
        itr_start = time.time()
        itr += 1

        #Calcualte function values and Jacobian at xvec(k).
        F = GeneralizedCubicFunction(xvec, y, Tc, Pc, w, EOS, C, n, n_tot, R, Vc, k)
        cubtime = time.time()
        Fmat = jnp.append(Fmat, F, axis = 0)
        F = -1*jnp.transpose(F)
        J = Jmat(xvec,y, Tc, Pc, w, EOS, C, n, n_tot, R, Vc, k)[0]

        #Solve dxvec, uses LU decomposition with partial pivoting.
        dxvec = jnp.linalg.solve(J, F)
        dxvec = jnp.transpose(dxvec)
        dxvec = jnp.reshape(dxvec, [jnp.shape(J)[0]])

        #Define Q, the damping factor. 0 for binary mixtures, and 518 for ~20 iterations for non-binary.
        if C ==2:
            Q = 0
        else:
            Q =518
        #Apply dampening.
        D = 1/(1+Q*jnp.exp(-0.5*itr))
        xvec = xvec + D*dxvec

        #Max iterations is set as 30.
        if itr>30:
            print("Convergence not achieved in 30 iterations.")
            Pc = None
            return Fmat, xvec, Pc

        #Calculate iteration time for full report.
        itr_end = time.time()
        itr_time = itr_end-itr_start
        if printflag:
```

```
        print(str(itr)+"    "+str(round(jnp.linalg.norm(dxvec), 4))+"    "+str(round((itr_time), 4)))

#Calculate mixture runtime for summary report.
end_time = time.time()
rn_time = end_time-start_time

#Calculate final function values for evaluation of convergence.
F = GeneralizedCubicFunction(xvec, y, Tc, Pc, w, EOS, C, n, n_tot, R, Vc, k)
Fmat = jnp.append(Fmat[1:, :], F, axis = 0)

#Check convergence and calculate Pc.
conv_flag = jnp.linalg.norm(F-jnp.zeros(jnp.shape(F)))
if conv_flag >= 1:
    print("Convergence achieved, function values high. Critical point may be false or nonexistent.")
else:
    if printflag:
        print("Magnitude of final function vector: " + str(round(conv_flag, 4)))
if printflag:
    print("-----------------------------------------------")
aij = aijf(xvec[0], Tc, R, EOS, Pc, w, k, C)
Pc = R*xvec[0]/(xvec[1]-b_tot(y, EOS, R, Tc, Pc, C))\
- a_tot(n, n_tot, aij, C)/((xvec[1]+D1(EOS)*b_tot(y, EOS, R, Tc, Pc, C))\
                           *(xvec[1]+D2(EOS)*b_tot(y, EOS, R, Tc, Pc, C)))

return Fmat, xvec, Pc, itr, rn_time
```

C.2 Global Optimization - Differential Evolution

The following code implements the differential evolution method for optimization-based critical point calculations (as described by Henderson et al.) [13]. This code utilizes Jax for automated differentiation and Scipy's differential evolution procedure for minimization. All packages are the property of their respective authors.
The following packages are required:

- scipy.optimize
- jax
- jax.numpy
- time

Algorithm C.2: Automatic Differentiation Compliant Global Optimization via Differential Evolution

```
#nu, a constant dependant on EOS, that affects a.
def nu_(EOS):
    if EOS == 'SRK':
        O1 = 0.42748
        O2 = 0.08664
        return [O1, O2]
    if EOS == 'PR':
        O1 = 0.45724
        O2 = 0.07780
        return [O1, O2]

#D1, D2, D3, and D4, parameters that define the EOS.
def d_1(EOS):
    import jax.numpy as jnp
    if EOS == 'SRK':
        u0 = 1
        w0 = 0
    if EOS == 'PR':
        u0 = 2
        w0 = -1
    D1 = (u0 + jnp.sqrt(u0**2-4*w0))/2
    return D1
def d_2(EOS):
    import jax.numpy as jnp
    if EOS == 'SRK':
        u0 = 1
        w0 = 0
    if EOS == 'PR':
        u0 = 2
        w0 = -1
    D2 = (u0 - jnp.sqrt(u0**2-4*w0))/2
    return D2
def d_3(D1, D2):
    return D1+D2
def d_4(D1, D2):
    return(D1*D2)

#m, the accentricity polynomial.
def m_(w, C, EOS):
    import jax.numpy as jnp

    m = jnp.zeros([C])
    if EOS == 'SRK':
        m = m.at[:].set(0.48+1.574*w-0.176*w**2)
        return m
    if EOS == 'PR':
        m = m.at[:].set(0.37464+1.54226*w-0.26992*w**2)
        return m

#ai, the attraction parameter of each component i.
def a_i(T, M, NU, Tc, Pc, R, C):
    import jax.numpy as jnp
    ai = jnp.zeros([C])
    for i in range(C):
        ai = ai.at[i].set((R*Tc[i])**2*NU[0]/Pc[i]*(1 + M[i]*(1-(T/Tc[i])**0.5))**2)
    return ai

#aij, the binary attraction parameter of component system i-j.
def a_ij(ai, C, k):
    import jax.numpy as jnp
```

```
    aij = jnp.zeros([C, C])
    for i in range(C):
        for j in range(C):
            aij = aij.at[i,j].set((ai[i]*ai[j])**0.5*(1-k[i][j]))
    return aij

#bi, the covolume parameter of each component i.
def b_i(NU, Tc, Pc, R, C):
    import jax.numpy as jnp
    bi = jnp.zeros([C])
    for i in range(C):
        bi = bi.at[i].set(NU[1]*R*Tc[i]/Pc[i])
    return bi

#a_tot, the weighted sum of binary attraction interactions.
def a_t(y, aij, C):
    import jax.numpy as jnp
    import jax
    a = jnp.array(0)

    #maintain the sum(y_i) =1 relation needed for Henderson's formulation.
    yn = jnp.append(y[:-1], (1-jnp.sum(y[:-1])))

    r = C-1
    for i in range(C):
        for j in range(C):
            a += jnp.multiply(jnp.multiply(yn[i],aij[i,j]), yn[j])
    return a

#b_tot, the weighted sum of covolume interactions.
def b_t(y, bi, C):
    import jax.numpy as jnp
    import jax
    bt = jnp.array(0)
    r = C-1
    for i in range(C):
        #maintain the sum(y_i) =1 relation needed for Henderson's formulation.
        bt += y[i]*(bi[i]-bi[r])
    bt += bi[r]
    return bt

#A_dim, nondimensionalized a.
def A_dim(a, P, R, T):
    import jax.numpy as jnp
    return a*P/(R*T)**2

#Alphai, percentage of total attraction parameter of component i.
def ALPHA_(y, aij, a, C):
    import jax.numpy as jnp
    import jax
    #maintain the sum(y_i) =1 relation needed for Henderson's formulation.
    yn = jnp.append(y[:-1], (1-jnp.sum(y[:-1])))
    ALPHA = jnp.zeros(C)
    for i in range(C):
        for k in range(C):
                ALPHA  = ALPHA.at[i].add(yn[k]*aij[k, i])

    ALPHA = ALPHA.at[:].set(ALPHA[:]/a)

    return ALPHA

#Betai, percentage of total covolume parameter of component i.
```

```
def BETA_(bi, b, C):
    BETA = bi/b
    return BETA

#B, nondimensionalized b.
def B_dim(b, P, R, T):
    import jax.numpy as jnp
    return b*P/(R*T)

#Z, compressibility factor.
def CompFact(A, B, EOS):
    import jax.numpy as jnp
    import jax
    #EOS cubic compressibility coefficients
    D1 = d_1(EOS)
    D2 = d_2(EOS)
    D3 = d_3(D1, D2)
    D4 = d_4(D1, D2)

    c1 = jnp.array(1)
    c2 = jnp.array((D3-1)*B-1)
    c3 = jnp.array(A-D3*B-(D3-D4)*B**2)
    c4 = jnp.array(-D4*(B**3+B**2)-A*B)

    #Piecewise functions must be coded using JAX-compliant conditionals.

    #Cubic discriminant
    disc = 18*c1*c2*c3*c4 - 4*c2**3*c4 + c2**2*c3**2 - 4*c1*c3**3 - 27*c1**2*c4**2
    def threeroot(c1, c2, c3, c4, disc):
        #Three real roots.
        def three_distinct(c1, c2, c3, c4):
            p1 = (3*c1*c3 - c2**2)/(3*c1**2)
            p2 = (2*c2**3 - 9*c1*c2*c3 + 27*c1**2*c4)/(27*c1**3)
            arg = 3*p2/(2*p1)*jnp.sqrt(-3/p1)
            arg = jax.lax.complex(arg, jnp.array(0.0))

            Z1 = 2*(-p1/3)**(1/2)*jnp.cos(jnp.arccos(arg)/3)
            Z2 = 2*(-p1/3)**(1/2)*jnp.cos(jnp.arccos(arg)/3-2*jnp.pi/3)
            Z3 = 2*(-p1/3)**(1/2)*jnp.cos(jnp.arccos(arg)/3-4*jnp.pi/3)

            Z1 = jnp.real(Z1 - c2/(3*c1))
            Z2 = jnp.real(Z2 - c2/(3*c1))
            Z3 = jnp.real(Z3 - c2/(3*c1))
            return jnp.maximum(jnp.maximum(Z1, Z2), Z3)
        #At least one repeated root.
        def repeated_root(c1, c2, c3, c4):
            def one_repeated(c1, c2, c3, c4):
                #Double multiplicity root.
                Z1 = (4*c1*c2*c3-9*c1**2*c4-c2**3)/(c1*(c2**2-3*c1*c3))
                Z2 = (9*c1*c4-c2*c3)/(2*(c2**2-3*c1*c3))
                return jnp.maximum(Z1, Z2)
            def two_repeated(c1, c2, c3, c4):
                #Triple multiplicity root.
                Z3 = -c2/3*c1
                return Z3
            return jax.lax.cond(c2**2 == 3*c1*c3, two_repeated, one_repeated, c1, c2, c3, c4)
        return jax.lax.cond(disc != 0, three_distinct, repeated_root, c1, c2, c3, c4)

    def oneroot(c1, c2, c3, c4, disc):
        #One Real Root, Two Complex Conjugates

        d0 = c2**2 - 3*c1*c3
```

```
        d1 = 2*c2**3 - 9*c1*c2*c3 + 27*c1**2*c4

        C0 = jnp.cbrt((d1 + jnp.sqrt(d1**2-4*d0**3))/2)
        #Select other root if needed
        def proot(d1, d0):
            C = jnp.cbrt((d1 + jnp.sqrt(d1**2-4*d0**3))/2)
            return C
        def nroot(d1, d0):
            C = jnp.cbrt((d1 - jnp.sqrt(d1**2-4*d0**3))/2)
            return C

        C = jax.lax.cond(C0 == 0, nroot, proot, d1, d0)

        Z1 = -1/(3*c1)*(c2+C+d0/C)

        return Z1

    return jax.lax.cond(disc >= 0, threeroot, oneroot, c1, c2, c3, c4, disc)

#Calculate Cubic EOS params based on inputs.
def EOS_Params(y, T, P, Tc, Pc, w, k, R, C, EOS):
    nu = nu_(EOS)
    m = m_(w, C, EOS)
    d1 = d_1(EOS)
    d2 = d_2(EOS)

    ai = a_i(T, m, nu, Tc, Pc, R, C)
    aij = a_ij(ai, C, k)
    a = a_t(y, aij, C)
    bi = b_i(nu, Tc, Pc, R, C)
    b = b_t(y, bi, C)


    A = A_dim(a, P, R, T)
    B = B_dim(b, P, R, T)



    Z = CompFact(A, B, EOS)

    ALPHA = ALPHA_(y, aij, a, C)
    BETA = BETA_(bi, b, C)
    return ai, aij, a, bi, b, A, B, d1, d2, Z, ALPHA, BETA

#ln(f/xP), the chemical potential departure for component i.
def lnfc(ai, aij, a, bi, b, A, B, d1, d2, Z, ALPHA, BETA, C):
    import jax
    import jax.numpy as jnp

    lnfc = jnp.zeros(C)
    E0 = jnp.log((Z+d1*B)/(Z+d2*B))
    lnfci = BETA*(Z-1) -jnp.log(Z-B) - 2*(A/B)*ALPHA/(d1-d2)*E0 + (A/B)*BETA/(d1-d2)*E0
    lnfc = lnfc.at[:].set(lnfci)

    return lnfc

#Chemical potential of component i as calculated by Henderson et al.
def ChemicalPotential(y, T, P, Tc, Pc, w, k, R, C, EOS):
    import jax
    import jax.numpy as jnp

    def lnf_(y, T, P, Tc, Pc, w, k, R, C, EOS):
```

```
        ai, aij, a, bi, b, A, B, d1, d2, Z, ALPHA, BETA = EOS_Params(y, T, P, Tc, Pc, w, k, R, C, EOS)
        lnf = lnfc(ai, aij, a, bi, b, A, B, d1, d2, Z, ALPHA, BETA, C)
        return lnf

    lnf = lnf_(y, T, P, Tc, Pc, w, k, R, C, EOS)
    dlnf_ = jax.jacfwd(lnf_, argnums = 0)
    dlnf = dlnf_(y, T, P, Tc, Pc, w, k, R, C, EOS)

    r = C-1
    dmu = jnp.zeros([C, C])
    for i in range(C):
        for j in range(C):
            if i != j:
                if i != r:
                    dmuij = dlnf[i, j]*R*T
                elif i == r:
                    dmuij = (dlnf[r, j] - 1/y[r])*R*T
            else:
                dmuij = (dlnf[j, j] + 1/y[j])*R*T

            dmu = dmu.at[i, j].set(dmuij)
    return dmu

#Compositional Hessian of the objective function.
def GeneralizedCubicHessian(y, yr, T, P, Tc, Pc, w, k, R, C, EOS):
    import jax.numpy as jnp
    dmu = ChemicalPotential(jnp.append(y, yr), T, P, Tc, Pc, w, k, R, C, EOS)
    r = C-1
    H = jnp.zeros([r, r])

    for i in range(r):
        for j in range(r):
            Hij = dmu[i, j]-dmu[r, j]
            H = H.at[i, j].set(Hij)
    return H


"""Inverse power iteration for computation of minimum eigenvector.
   Other calculation methods have not yet been made jax-compliant.
Inputs:
        H - A matrix whose dominant eigenvalues are needed.
Outputs:
        l - The minimum eigenvalue, the dominant eigenvalue of inverse(H).
        umin - The eigenvector associated with l.
        """
def l_min(H):
    import jax.numpy as jnp
    import jax

    #Inverse power iteration
    H_inv = jnp.linalg.inv(H)
    umin = jnp.ones(len(H))
    umin = umin/jnp.linalg.norm(umin)
    for t in range(50):
        umin = jnp.dot(H_inv, umin)
        umin = umin/jnp.linalg.norm(umin)

    l = jnp.dot(jnp.dot(H, umin), umin)
    return l, umin

"""Objective function for minimization based on Cubic EOS.
Inputs:
```

```
        TP - A vector of state in the format [T (K), P(Pa)]
        y - A vector of component compositions.
        Tc - A vector of component critical temperatures in the order of y. (K)
        Pc - A vector of component critical pressures in the order of y. (Pa)
        w - A vector of component accentricity factors in the order of y.
        k - Matrix of binary interaction parameters with component
            in order of y as columns and rows. Main diagonal of 0.
        R - Gas constant. (J/(molK))
        C - Number of components in mixture.
        EOS - Equation of state. 'PR' or 'SRK' implemented.
Outputs:
        Q+C - Q, the value of the first criticality condition.
              C, the value of the second criticality condition.
        """
def CostFunction(TP, y, Tc, Pc, w, k, R, C, EOS):
    import jax
    import jax.numpy as jnp
    T = TP[0]
    P = TP[1]
    yr = y[-1]
    y = y[0:-1]
    Hf = jax.jit(GeneralizedCubicHessian, static_argnames=['C', 'R', 'EOS'])
    H = Hf(y, yr, T, P, Tc, Pc, w, k, R, C, EOS)

    lmin = jax.jit(l_min)
    l, u = lmin(H)
    Q = l**2

    dHf = jax.jacfwd(Hf, argnums = 0)
    dH = dHf(y, yr, T, P, Tc, Pc, w, k, R, C, EOS)
    C = jnp.dot(jnp.dot(jnp.dot(dH, u), u), u)**2

    return Q+C

#Rho, calculated from the compressibility factor.
def CalcRho(y, T, P, Tc, Pc, w, k, R, C, EOS):
    import jax
    import jax.numpy as jnp
    ai, aij, a, bi, b, A, B, d1, d2, Z, ALPHA, BETA = EOS_Params(y, T, P, Tc, Pc, w, k, R, C, EOS)
    rho = P/(Z*R*T)
    return rho


"""Objective function for minimization based on Cubic EOS.
Inputs:
        MxN - Number associated with selected composition.
        DataSet - DataSet folder to obtain chemical data from.
        EOS - Equation of state. 'PR' or 'SRK' implemented.
        y_given - vector of composition as an alternative to default y.
        TP_bound - Alternative search bound, overwrites Kay's Mixing.
Outputs:
        minima - Optimize object with attributes x, fun, nfev, nitr.
        rho - Critical density (mol/cum).
        rn_time - Runtime. (s)
        TP_bound - Selected bound (testing purposes).
        """
def minimizer(MxN, DataSet, EOS, y_given = None, TP_bound=None):
    import jax.numpy as jnp
    import KeyFunctions as me
    import jax
    import numpy as np
    import scipy as sp
```

```
    from time import time as t

    jax.config.update("jax_enable_x64", True)
    #Function for obtaining chemical data.
    [y, Tc, Pc, w, C, R, Vc, k, mxNames] = me.LookUpMix(MxN, DataSet, EOS)

    if y_given is not None:
        y = y_given

    ts = t()
    TP_guess = [jnp.dot(y, Tc), jnp.dot(y, Pc)]
    if TP_bound is None:
        TP_bound = [(TP_guess[0]*0.75, TP_guess[0]*1.25), (TP_guess[1], TP_guess[1]*2)]


    #Initial minimization with simple parameters.
    minima = sp.optimize.differential_evolution(CostFunction, \
              bounds = TP_bound, args = (y, Tc, Pc, w, k, R, C, EOS),\
                                                  mutation = 0.95, atol = 1e-14, init = 'sobol')

    if minima.fun > 1:
        #Expanded bound and more stringent parameters for improved convergence.
        TP_bound = [(TP_guess[0]-50, TP_guess[0]+200), (TP_guess[1], TP_guess[1]*2)]
        minima = sp.optimize.differential_evolution(CostFunction,\
                        bounds = TP_bound, args = (y, Tc, Pc, w, k, R, C, EOS),\
                                      mutation = 0.95, popsize = 40, tol = 1e-16, recombination = 0.4)
        if minima.fun > 1:
            display('Erroneous Minima')
    tn = t()
    rn_time = tn-ts
    rho = CalcRho(y, minima.x[0], minima.x[0], Tc, Pc, w, k, R, C, EOS)

    return minima, rho, rn_time, TP_bound
```