The Thesis Committee for Jose Luis Hernandez Mejia
Certifies that this is the approved version of the following Thesis

Application of Artificial Neural Networks for Rapid Flash Calculations

APPROVED BY
SUPERVISING COMMITTEE:

Ryosuke Okuno, Supervisor

Kamy Sepehrnoori

Application of Artificial Neural Networks for Rapid Flash Calculations

by

Jose Luis Hernandez Mejia

Thesis

Presented to the Faculty of the Graduate School of

The University of Texas at Austin

in Partial Fulfillment

of the Requirements

for the Degree of

Master of Science in Engineering

The University of Texas at Austin

August 2019

Dedication

To my mom Elvira, my dad Jose, and my brother Noe.

Acknowledgements

First, I would like to express my genuine gratitude to my advisor Dr. Ryosuke Okuno for his guidance, inspiration, and patience, during my studies at the University of Texas at Austin. My admiration for Dr. Okuno for greatly inspiring me with his example to be the best professional and person that I can be. I also would like to thank Dr. Kamy Sepehrnoori for his time to review my thesis and for his valuable feedback.

I would like to thank my research group and the staff of the Hildebrand Department of Petroleum Engineering for welcoming me and supporting me during the course of my degree. To my friends Sofiane, Ricardo, Javier, Fernando, Chelsea, Hongeeun Jo, Francisco, Julia, Mayra, Gustavo, and Mauricio, thank you for the long study sessions and the wonderfull memories collected outside the campus. To professor Dr. Michael Pyrcz, thank you for your comments on the development of artificial neural networks.

I also would like to thank, Thais McComb, John Pederson and Evelyn Vilchez my mentors from Chevron for playing a crucial role in my professional growth.

Additionally, I would like to thank M.S.E Maria Isabel Villegas Javier, Dr. Pedro Silva, Dr. Simon Lopez, and Dra. Cecilia de Los Angeles Duran Valencia for always believe in me and supporting me through my petroleum engineering career.

To my friends and family whom I did not cite here explicitly, thank you for your love and support. I also would like to thank the Fulbright Commission in Mexico and Conacyt, for the financial support during my studies at UT Austin.

Last but not least, I would like to thank my mom Elvira, my brother Noe and my dad Jose for always supporting me and encourage me to always follow my dreams no matter what. Without all of you, this work would have not been possible. Thank you

Abstract

Application of Artificial Neural Networks for Rapid Flash Calculations

Jose Luis Hernandez Mejia, MSE

The University of Texas at Austin, 2019

Supervisor: Ryosuke Okuno

Compositional reservoir simulation is widely used as an important tool for optimization of enhanced oil recovery processes. In compositional reservoir simulation, flash calculations are performed to solve for phase properties and amounts for each grid-block and each time step by use of a cubic equation of state (EOS). EOS flash calculation is one of the most time-consuming operations during compositional reservoir simulation. There has been a critical need for more efficient EOS flash for practical compositional reservoir simulation.

The central idea tested in this thesis is to use artificial neural networks (ANNs) to replace the most fundamental, but time-consuming portion of EOS flash; that is, the evaluation of fugacity coefficients. ANNs are used for efficient feedforward approximation of the EOS fugacity coefficient function with a series of weights, bias, and activation functions. A set of weights and bias is found by using an algorithm that minimizes the mean squared error between the predicted and real values. This type of approximation is called supervised learning in machine learning applications. The

thermodynamic model used is the Peng – Robinson equation of state with the van der Waals mixing rules and solved by the successive substitution algorithm for flash calculations.

The implementation of the ANN-based fugacity coefficient function is straightforward because it only replaces the EOS-based fugacity coefficient in conventional flash calculation algorithms. Once an ANN-based fugacity coefficient function is built based on a cubic EOS, the EOS is required only when phase densities are calculated, usually at the final convergence. That is, ANN-based flash does not use an EOS during the iterative solution. We show comparisons between the conventional EOS flash calculations and the ANN flash calculations in terms of computational efficiency. Use of ANN flash can reduce on average 89.83% of the time needed by the conventional EOS flash for the cases studied in this thesis.

Table of Contents

List of Tables

List of Figures

CHAPTER 1

INTRODUCTION

This chapter describes the flash calculation problems to be solved in this thesis, following by the objectives of this research and finally an outline of the thesis is presented.

1.1 PROBLEM DESCRIPTION

Compositional reservoir simulation is widely used for designing and optimizing enhanced oil recovery (EOR) processes. Such simulators must be able to predict phase behavior, volumetric sweep efficiency, and incremental oil recovery of miscible gas/CO_2 flooding. Compositional simulators often use a fluid model based on a cubic equation of state (EOS) to predict the phase behavior of reservoir fluid mixtures at operating conditions.

The phase behavior in EOS compositional simulation is determined by the use of stability analysis and flash calculations. Stability analysis will indicate if the hydrocarbon mixture of interest is in stable single phase at specific conditions of pressure and temperature. If more than one phase is necessary for phase equilibrium, the subsequent flash calculation is to calculate the amounts and properties of the coexisting phases. Those phase equilibrium calculations are performed in an iterative manner at each grid – block and at each iteration step, and therefore, can consume a non-trivial amount of the simulation time during EOS compositional reservoir simulation.

This problem of compositional reservoir simulation is intensified when the number of components used for the simulation is increased, when there are more than two phases or when the thermodynamic conditions are in the critical region in gas flooding simulations. Additionally, compositional simulation becomes challenging when the

resolution of the reservoir model is increased by grid refinement since flash calculation has to be performed in each gridblock. The development of algorithms that decrease the time spent in EOS flash calculations would facilitate the decision making and the design of more efficient oil production for EOR projects.

1.2 RESEARCH OBJECTIVES

The computational time in compositional reservoir simulation depends on the algorithm used for phase equilibrium calculations and the number of equations to be solved during the simulation run. With the advances in computer science and artificial intelligence, new tools have been developed to predict complex and nonlinear functions. The aim of this thesis is to generate a fast flash calculation algorithm using artificial neural networks (ANNs) to replace the most fundamental, but time-consuming portion of the flash calculation; that is, the evaluation of the fugacity coefficient. The fugacity coefficient function by a cubic EOS (e.g., Peng and Robinson) can be accurately represented by a simpler function within a given thermodynamic domain by use of ANNs. If that was done, flash calculation usually does not require solving the EOS during the iterative solution. This is the central idea that motivated this thesis project.

The objectives of this research are the following:

1. Develop ANN models of the fugacity coefficient based on the Peng-Robinson EOS.
2. Integrate the ANN-based fugacity coefficient function into a flash algorithm based on successive substation.
3. Evaluate the accuracy of the ANN flash in comparison to the original EOS flash by using different reservoir fluids.
4. Quantify the computational efficiency of the ANN flash in comparison to the EOS flash.

5. Identify implementation problems for ANN flash and propose practical remedies to them.

To achieve the first objective, we create ANN models for the fugacity coefficient, without model overfitting, with low generalization error for predictions in unseen data. For the second objective, we integrate the ANN models into the traditional flash calculation algorithm using successive substitution. The generated algorithm can increase the speed of the flash calculation because we replace the fugacity coefficient with ANNs avoiding calculation of the fugacity coefficient during the iteration. Cases studies using the ANN and EOS flash calculations are used for the 3rd, 4th, and 5th objectives.

1.3 OUTLINE OF THESIS:

This thesis consists of 6 chapters. Chapter 2 presents background information of the conventional flash calculations and the basics of the phase equilibrium. Additionally, a literature review on the different methods for speeding up compositional reservoir simulations is presented.

In Chapter 3, theory behind artificial neural networks is presented. Here, we define the concepts used to generate, train and evaluate the performance of ANNs.

In Chapter 4, we describe the methodology to generate ANNs for the fugacity coefficient. Additionally, the ANN flash formulation and algorithm developed in this research is presented.

In Chapter 5, we present the results obtained with the new algorithm as well as comparisons in terms of accuracy, robustness and efficiency against the conventional EOS flash calculation.

Chapter 6 summarizes and concludes the research. Suggestions for future work are also presented.

CHAPTER 2

PHASE EQUILIBRIUM CALCULATIONS

This chapter defines the fundamental concepts involved in phase equilibrium calculation. The first portion of this chapter describes the fundamental equations involved in phase equilibrium and the conventional methods of flash calculation. The second portion of this chapter provides a literature review about the different methods to speed up flash calculations for compositional reservoir simulators.

2.1 BASICS OF PHASE EQUILIBRIUM CALCULATIONS

This section describes the fundamental equations used in flash calculations. Then, the methodology to solve the flash calculation is presented followed by equations of state (EOSs). In this thesis, the Peng – Robinson equation of state (1978) is used with the van der Waals mixing rules.

2.1.1 Equilibrium Conditions

Thermodynamics analysis of mixtures involves the partial molar properties of each component in the mixture. Therefore, the partial molar Gibbs free energy is one of the most important parameters for phase equilibrium modeling. Since the Gibbs free energy of a multicomponent mixture is a function of temperature, pressure and mole number of the components in the mixture, the total differential of the Gibbs free energy function can be written as

$$dG = \left(\frac{\partial G}{\partial T}\right)_{P,N_i} dT + \left(\frac{\partial G}{\partial P}\right)_{T,N_i} dP + \sum_{i}^{Nc} \left(\frac{\partial G}{\partial N_i}\right)_{T,P,N_{j\neq i}} dN_i \qquad 2.1$$

$$dG = -SdT + VdP + \sum_{i}^{Nc} \bar{G}_i dN_i \qquad 2.2$$

where T is temperature, P is pressure, $\bar{G}_i$ is the partial molar Gibbs energy, Nc is the number of components, and N_i is the number of moles of component i. Using the commutative properties of second derivatives of the thermodynamic functions,

$$\left.\frac{\partial}{\partial N_i}\right|_{T,P,N_{j\neq i}} = \left.\frac{\partial}{\partial T}\right|_{P,N_j} \left(\frac{\partial G}{\partial N_i}\right)_{T,P,N_{j\neq i}} \quad 2.3$$

and

$$\left.\frac{\partial}{\partial N_i}\right|_{T,P,N_{j\neq i}} = \left.\frac{\partial}{\partial T}\right|_{P,N_j} \left(\frac{\partial G}{\partial N_i}\right)_{T,P,N_{j\neq i}} \quad 2.4$$

To obtain the two equations

$$\bar{S}_i = -\left(\frac{\partial \bar{G}_i}{\partial T}\right)_{P,N_j} \quad 2.5$$

and

$$\bar{V}_i = \left(\frac{\partial \bar{G}_i}{\partial P}\right)_{T,N_j} \quad 2.6$$

where $\bar{S}_i$ and $\bar{V}_i$ are the partial molar entropy and partial molar volume of component *i* respectively. Therefore, the fugacity equation can be derived with relation to the partial Gibbs free energy (Okuno 2009). Integration from a reference pressure P_1 to a higher pressure P_2 results in

$$\bar{G}_i(T_1, P_1, \underline{x}) - \bar{G}_i(T_1, P_2, \underline{x}) = \int_{P_1}^{P_2} \bar{V}_i \, dP \quad 2.7$$

where $\underline{x}$ is the vector representing the composition of the phase mixture of interest. For ideal gas mixtures (IGM), the EOS for ideal gas can be used with equation 2.2 to obtain

$$\bar{G}_i^{IGM}(T_1, P_1, \underline{x}) - \bar{G}_i^{IGM}(T_2, P_2, \underline{x}) = RT \ln\left(\frac{P_2}{P_1}\right) \quad 2.8$$

where R is the universal gas constant. In analogy with equation 2.8, the fugacity coefficient can be defined as

$$\bar{G}_i(T_1, P_1, \underline{x}) - \bar{G}_i(T_1, P_2, \underline{x}) = RT \ln\left(\frac{f_i}{f_i^0}\right) \qquad 2.9$$

where f_i and f_i^0 are the fugacities at P_2 and P_1 respectively. The fugacity f_i accounts for the deviation in the partial pressure P_i generated by the non-ideal behavior of molecules in the mixture. Subtracting equation 2.8 from 2.9

$$f_i = x_i \mathrm{P} \exp\left[\frac{\bar{G}_i(T, P, \underline{x}) - \bar{G}_i^{IGM}(T, P, \underline{x})}{RT}\right] \qquad 2.10$$

$$f_i = x_i \mathrm{P} \exp\left[\frac{1}{RT}\int_0^P (\bar{V}_i - \bar{V}_i^{IG})dP\right] \qquad 2.10$$

Assuming that $f_1 = x_i P_1 \equiv P_i$ and $\bar{G}_i(T_1, P_1, \underline{x}) = \bar{G}_i^{IGM}(T_1, P_1, \underline{x})$ when $P_1 \longrightarrow 0$. Here, P_i is the partial pressure of component i, and the superscript IGM indicates an ideal gas mixture property. When the pressure goes to 0, all mixtures become ideal and the fugacity can be express as.

$$f_i = x_i \mathrm{P} \exp\left[\frac{\bar{G}_i(T, P, \underline{x}) - \bar{G}_i^{IG}(T, P, \underline{x})}{RT}\right] \qquad 2.11$$

Equation 2.11 is provided as a definition of the fugacity in textbooks (e.g. Sandler, 2006) and its derivation with its relationship with the partial molar Gibbs free energy is given by Okuno (2009). The fugacity coefficient for a component in a mixture is defined as

$$\varphi_i = \frac{f_i}{x_i \mathrm{P}} \qquad 2.12$$

which is intensively used during phase equilibrium calculations. To calculate the fugacity of the components in the mixture from equation 2.10, a volumetric equation of state needs

to be solved explicitly for volume in terms of pressure and temperature (Sandler, 2006). However, most of the equations of state are usually pressure explicit. Therefore, the fugacity equation can be re written starting with

$$dP = \frac{1}{\bar{V}} d\left(P\underline{V}\right) - \frac{P}{\underline{V}} d\underline{V} \qquad 2.13$$

And the triple product rule

$$\left(\frac{\partial V}{\partial N_i}\right)_{T,P,N_{j\neq i}} \left(\frac{\partial P}{\partial V}\right)_{T,N_j} \left(\frac{\partial N_i}{\partial P}\right)_{T,V,N_{j\neq i}} = -1 \qquad 2.14$$

in the form

$$\left(\frac{\partial V}{\partial N_i}\right)_{T,P,N_{j\neq i}} dP = -\left(\frac{\partial P}{\partial N_i}\right)_{T,V,N_{j\neq i}} dV \qquad 2.15$$

The symbolic fugacity coefficient for a component in a mixture from an EOS is

$$\ln\varphi_i = \ln\frac{f_i(T,P,\underline{x})}{x_i \mathrm{P}} = \frac{1}{RT}\int_{\underline{V}=\infty}^{\underline{V}=ZRT/P}\left[\frac{RT}{\underline{V}} - \left(\frac{\partial P}{\partial N_i}\right)_{T,V,N_{j\neq i}}\right] d\underline{V} - \ln Z \qquad 2.16$$

Finally, the criterion for equilibrium between two phases is that $\bar{G}_i^I = \bar{G}_i^{II}$ for all components *i* in the mixture with the condition that temperature and pressure are equal in both phases. Therefore, at equilibrium conditions, the fugacity of each component in the mixture must be the same in the two phases.

2.1.2 Conventional formulations for flash calculations

There are two main approaches to performing phase equilibrium calculations: a) direct minimization of the Gibbs free energy and b) solution of the fugacity equations. In this section both methods will be discussed, and common algorithms will be presented.

2.1.2.1 Direct minimization of the Gibbs Free energy

Flash calculations are used in this thesis to find the equilibrium phase compositions and phase amounts at conditions of pressure P, temperature T, and overall composition $\underline{z}$.

The fundamental formulation in flash calculations is the minimization of the Gibbs free energy at given pressure and temperature. As described in Okuno (2009), the molar Gibb free energy of a multicomponent mixture is defined as

$$\underline{G} = \sum_{j=1}^{Np}\sum_{i=1}^{Nc}\beta_j x_{ij}\bar{G}_{ij} \tag{2.17}$$

where x_{ij} is the mole fraction of component *i* in phase *j* and β_j is the mole fraction of phase *j*. Combining equation 2.12 with equation 2.17, we get the following:

$$\underline{G} = RT\sum_{j=1}^{Np}\sum_{i=1}^{Nc}\beta_j x_{ij}\ln(x_{ij}\varphi_{ij}) + \underline{G}^{IG}\sum_{j=1}^{Np}\sum_{i=1}^{Nc}\beta_j x_{ij} \tag{2.18}$$

$$= RT\sum_{j=1}^{Np}\sum_{i=1}^{Nc}\beta_j x_{ij}\ln(x_{ij}\varphi_{ij}) + \underline{G}^{IG}$$

In equation 2.18, the molar Gibbs free energy of the ideal gas $\underline{G}^{IG}$ is a function of pressure and temperature that are fixed in flash calculations. Therefore, the minimization of the Gibbs free energy can be solved using the following dimensionless equation:

$$\underline{G}_R = RT\sum_{j=1}^{Np}\sum_{i=1}^{Nc}\beta_j x_{ij}\ln(x_{ij}\varphi_{ij}) \tag{2.19}$$

Thus, the constrained global optimization problem for the Gibbs free energy minimization is

$$\min \underline{G}_R = RT\sum_{j=1}^{Np}\sum_{i=1}^{Nc}\beta_j x_{ij}\ln(x_{ij}\varphi_{ij})$$

subject to

$$\sum_{j=1}^{Np}\beta_j x_{ij} = z_i \tag{2.20}$$

$$\sum_{i=1}^{Nc} z_i = 1.0 \tag{2.21}$$

$$\sum_{j=1}^{Np} \beta_j = 1.0 \text{ and } \beta_j \geq 0 \text{ for j} = 1, \dots, \text{Np}, \tag{2.22}$$

$$\sum_{i=1}^{Nc} x_{ij} = 1.0 \text{ and } x_{ij} \geq 0 \text{ for i} = 1, \dots, \text{Nc and j} = 1, \dots, \text{Np}. \tag{2.23}$$

As shown by Baker *et al*. (1982), when the fluid system exhibits multiple phases, the Gibbs free energy using an EOS might have false solutions. Therefore, the global minimum is the correct solution while the other local minima are false solutions.

As explain by Okuno (2009), the standard minimization is performed in terms of $\text{Nc}(\text{Np} - 1)$ independent variables.

$$w_{ij} = \beta_j x_{ij} \text{ where i} = 1, \dots, \text{Nc and j} = 1, \dots, (\text{Np} - 1) \tag{2.19}$$

since $\sum_{j=1}^{Np} w_{ij} = z_i$. The mole fraction can be express as $x_{ij} = \frac{w_{ij}}{\sum_{i=1}^{Nc} w_{ij}}$ because $\sum_{i=1}^{Nc} w_{ij} = \beta_j$. The minimization problem can be re-arranged as follows

$$\min \underline{G_R} = RT \sum_{j=1}^{Np} \sum_{i=1}^{Nc} \beta_j x_{ij} \ln(x_{ij} \varphi_{ij}) \tag{2.20}$$

Subjected to:

$$w_{mn} = \beta_n x_{nm} \geq 0 \tag{2.21}$$

$$\sum_{n=1}^{Np} w_{mn} = z_m \text{ where m} = 1, \dots, \text{Nc and n} = 1, \dots, \text{Np} \tag{2.22}$$

For a function F to be a local minimum at x^*, the sufficient conditions for optimality are that $\Delta \text{F}(x^*) = 0$ and the Hessian of the matrix $\Delta^2 \text{F}(x^*)$ is positive definite. The gradient and the Hessian of the matrix are used in Newton's method of minimization

with a line search technique. The material balance has to be satisfied at each iteration to fix a reference value of the function and the dependent variables for the remaining phases can be calculated as

$$w_{iNp} = z_i - \sum_{j=1}^{Np-1} w_{ij}, \text{where i} = 1, \dots, \text{Nc} \qquad 2.23$$

The algorithm of Perschke *et al.* (1989) is the following:

1. Obtain the initial estimate of the $Nc(\text{Np} - 1)$ independent variables in equation 2.19
2. Calculate dependent variables using w_{iNp} with equation 2.23
3. Calculate fugacity coefficients and compressibility factors using an EOS for Np phases. When the solution of the cubic EOS, select the root that gives the lowest Gibbs Free energy (Evelein et at. 1976)
4. Calculate the gradient vector and the Hessian matrix analytically from equation 2.20
5. Decompose the Hessian matrix using the modified Cholesky decomposition algorithm.
6. If the Hessian matrix is positive definite in step 5 check for convergence of the max norm of the gradient vector. If convergence is achieved, stop. Otherwise, continue to step 7.
7. Obtained search direction using systems of equations decomposed in step 5
8. Calculate step length using the line search algorithm.
9. Update the $Nc(\text{Np} - 1)$ and go to step 2.

Since Newton's method is used for minimization of the Gibbs free energy the convergence behavior is quadratic close to the solution.

2.1.2.2 Successive Substitutions Iteration Method

A second alternative to flash calculation is the solution of the fugacity equations as explained in Okuno (2009). Assuming no capillary or gravity effect on phase equilibrium, the criterion for phase equilibrium is that the chemical potential of each component *i* in phase I is the same for component *i* in phase II for all components in the mixture. A useful expression for the chemical potential is the fugacity, f_i. Therefore, the constraint for phase equilibrium can be written as

$$lnf_{ij} - lnf_{ij} = 0, \text{where i} = 1, \dots, \text{Nc and j} = 1, \dots, \text{Np} \qquad 2.24$$

This constraint can be solved by successive substitution (SS) until a convergence criterion is achieved. Successive substitution is widely applied because of its simplicity and robustness. The independent variable in successive substations is k-values. k-values represent the tendency of a component to partition among different phases and is defined by

$$K_{ij} = \frac{x_{ij}}{x_{iNp}} \text{ where i} = 1, \dots, \text{Nc and j} = 1, \dots, (\text{Np} - 1) \qquad 2.25$$

The Np^{th} phase is a reference phase in equation 2.25. Successive substitution solves for equation 2.24 for K_{ij} with material balance equations 2.20, 2.22 and 2.23. Therefore, equation 2.24 can be express as

$$\ln {K_{ij}}^{k+1} = \ln {\varphi_{iNp}}^{k} - \ln {\varphi_{ij}}^{k} \text{ where i} = 1, \dots, \text{Nc and j} = 1, \dots, (\text{Np} - 1) \qquad 2.26$$

In equation 2.24, the superscript indicates iteration step. In equation 2.24, the fugacity coefficient φ_{ij} should be calculated at each iteration step to update the k – value. Because fugacity coefficient is a function of component *i* in phase *j* at pressure and temperature conditions, it is necessary to calculate phase composition for a given set of k – values. The conventional procedure for a constant K-value flash calculation was originally proposed by Rachford and Rice (1952), but more comprehensive and rigorous

treatment of the multiphase material balance was presented by Okuno (2009) and Okuno et al. (2010c). The objective of the constant k-value flash calculation is to determine the phase mole fraction and phase composition for a fixed overall composition. The Rachford and Rice equation is given by

$$f(\beta) = \sum_{1}^{Nc} \frac{(1 - K_i)z_i}{1 - (1 - K_i)\beta} = 0 \qquad 2.27$$

The Newton Rapson algorithm is commonly used to find the solution of $f(\beta)$ and the first guess of $f(\beta)$ can be chosen arbitrarily as 0.5 or the average between K_{max} and K_{min}. The first derivative of $f(\beta)$ and the Newton Rapson iteration step are express as

$$f'(\beta) = -\sum_{1}^{Nc} \frac{(1 - K_i)^2 z_i}{[1 - (1 - K_i)\beta]^2} \qquad 2.28$$

$$f(\beta)^{n+1} = f(\beta)^n - \frac{f(\beta)^n}{f'(\beta)^n} \qquad 2.29$$

where the correct solution of $f(\beta)$ lies in the region between $f(\beta)_{min}$ and $f(\beta)_{max}$ defined as

$$f(\beta)_{min} = \frac{1}{1 - K_{max}} < 0 \qquad 2.30$$

$$f(\beta)_{max} = \frac{1}{1 - K_{min}} > 1 \qquad 2.31$$

Whitson and Brulé (2000) showed that $f(\beta)_{min} < 0$ and $f(\beta)_{max} > 1$ if at least one k-value of is < 1 and one k-value is > 1. This implies that the solution of for $f(\beta) = 0$ is always limited to the region $f(\beta)_{min} < f(\beta) < f(\beta)_{max}$.

The phase compositions are calculated using the material balance as follows

$$x_i = \frac{z_i}{f(\beta)(K_i - 1) + 1} \; where \;\; i = 1, .., Nc \tag{2.32}$$

$$y_i = \frac{z_i K_i}{f(\beta)(K_i - 1) + 1} = z_i K_i \; where \;\; i = 1, .., Nc \tag{2.33}$$

In the successive substitution (SS) method, the inner iteration solves for equation 2.27 to determine the phase composition and phase mole fraction and the outer loop solves for equation 2.26 with the use of an EOS. The SS method is generally initialized with the use of Wilson's correlation to provide the initial estimates of the k-values as

$$K_i = \frac{x_i}{y_i} \approx \frac{\exp[5.373(1 + w_i)(1 - T_{ri}^{-1})]}{P_{ri}} \;\; where \;\; i = 1, .., Nc \tag{2.34}$$

where

$$P_{ri} = \frac{P_{ci}}{P} \;\; \text{where i} = 1, \ldots, \text{Nc} \tag{2.35}$$

$$T_{ri} = \frac{T_{ci}}{T} \;\; \text{where i} = 1, \ldots., \text{Nc} \tag{2.36}$$

where P_{ci}, T_{ci} are the critical pressure and critical temperature for component *i* respectively, and w_i is the acentric factor of component *i*. However, K – values from Wilson's correlation are not accurate at high pressures. Results from stability analysis can provide better initial k -values estimates to initialize two-phase flash calculations.

The successive substitution algorithm can be summarized as:

1. Specify T, P and feed mole fraction of the hydrocarbon mixture
2. Calculate the initial guess of the equilibrium constant using Wilson's correlation (2.34)

3. Solve the vapor fraction equation proposed by Rarchford and Rice (1952). Equation 2.27
4. Calculate the liquid and vapor molar fraction at those conditions of pressure and temperature, equation 2.32 and 2.33
5. Calculate the cubic equation parameters from an EOS
6. Solve the cubic EOS for Z for vapor and liquid and discard all middle roots as they are unstable
7. Calculate the fugacity coefficients $\ln\varphi_{iL}$ and $\ln\varphi_{iV}$
8. Check for convergence of based on the residuals of the fugacity coefficient equations $\|\ln x_i\varphi_{iL} - \ln y\varphi_{iv}\| < \varepsilon\ (e.g, \varepsilon = 10^{-6})$. If convergence is achieved stop. Otherwise, go to step 9
9. Update the equilibrium constants $lnK_i^{k+1} = (\ln\varphi_{iL} - \ln\varphi_{iV})^k$
10. Go to step 3

The successive substation method is linearly convergent, and convergence becomes slow in the near-critical region (Michelsen, 1982b).

2.1.3 Equations of State (EOS)

Cubic equations of state are simple equations that relate pressure, volume and temperature (PVT). EOS can predict volumetric phase behavior using the critical properties of the hydrocarbon mixture and acentric factors. One of the earliest attempts to represent phase behavior of real gases was the van der Waals (1873) equation of state.

$$\left(p + \frac{a}{V_{M^2}}\right)(V_M - b) = RT \qquad 2.37$$

The difference of this equation from the ideal gas equation is the addition of the term $\frac{a}{V_{M^2}}$ to pressure and the subtraction of b from molar volume. The term $\frac{a}{V_{M^2}}$ is a

pressure correction due to the attraction forces of the molecules. The constant b is a correction to the molar volume related to the volume occupied by the molecules. Constants a and b are characteristic of the particular gas of study. The van der Waals equation is an improvement from the ideal gas equation. However, its practical use is limited. Since the introduction of the van der Waals EOS, many cubic EOSs have been introduced in attempts to improve the accuracy of modeling fluids at a wide range of pressure and temperature (e.g. The Redlich and Kwong EOS in 1994 and the Peng – Robinson EOS in 1976). Changes in the molecular attraction term were commonly proposed.

Most of petroleum engineering applications use the Peng Robinson EOS or modifications of the Redlich Kwong EOS. To apply EOSs to hydrocarbon-rich mixtures, the van der Waals mixing rules are widely used, although more complex mixing rules can be also used.

$$\mathrm{A} = \sum_{i=1}^{Nc} \sum_{j=1}^{Nc} x_i x_j a_{ij} \tag{2.38}$$

$$\mathrm{B} = \sum_{i=1}^{Nc} x_i b_i \tag{2.39}$$

Parameters a_m and b_m are used in the EOS as a pure fluid. The combining rules for a_{ij} are defined as

$$a_{ij} = \sqrt{a_i a_j}(1 - k_{ij}) \tag{2.40}$$

where the terms k_{ij} are the binary interaction parameter (BIP), which are assumed to be independent of pressure and temperature. Values of the BIPs are usually obtained by fitting the equation of state to gas – liquid equilibria data for each mixture.

The Peng-Robinson EOS (Peng and Robinson 1978) with the van der Waals mixing rules are used for the development of training data for the ANN models in this thesis. The Peng-Robinson EOS is

$$\mathrm{P} = \frac{RT}{(\underline{V} - b)} - \frac{a(T)}{[\underline{V}(\underline{V} + b) + b(\underline{V} - b)]} \tag{2.41}$$

where

$$a(T) = 0.45724R^2{T_c}^2\alpha(T)/P_c \tag{2.42}$$

$$b = \frac{0.07780RT_c}{P_C} \tag{2.43}$$

$$\alpha(T) = 1 + \kappa(1 - \sqrt{T/T_c}) \tag{2.44}$$

$$\kappa = 0.37464 + 1.54226\omega - 0.26992\omega^2 \text{ for } \omega < 0.49 \tag{2.45}$$

$$\kappa = 0.37464 + \omega\big(1.48503 + \omega(-0.16442 + 0.01667\omega)\big) \text{ for } \omega > 0.49 \tag{2.46}$$

The fugacity coefficient of component i in a mixture using the Peng-Robinson EOS is defined as

$$\ln\varphi_{ij}\big(P, T, \underline{x}\big) = (Z - 1)\frac{B_i}{B} - \ln(Z - B) - \frac{A}{(\delta_1 - \delta_2)B}\left(\frac{2\sum_{j=1}^{Nc} x_j A_{ij}}{A_m} - \frac{B_i}{B}\right) ln\left[\frac{Z + \delta_1 B}{Z + \delta_2 B}\right]; \tag{2.47}$$

where $\delta_1 = 1 + 0.2^{0.5}$ and $\delta_2 = 1 - 0.2^{0.5}$.

2.2 Methods to speed up compositional reservoir simulators.

Because of the large computational cost involved in compositional reservoir simulations, a number of methods have been developed to accelerate phase equilibrium

calculations. Methods to achieve faster phase equilibrium calculations include reduction methods, tie-line methods, and most recently machine – learning methods. This section presents a literature review for the methods reported in the literature to speed up compositional reservoir simulation.

Reduction methods are used to accelerate phase fluid equilibrium calculations. The idea of the reduction method is to take advantage of the rank of the BIP matrix using only the most significant eigenvalues. The first reduction method was proposed by Michelsen (1986). Michelsen demonstrated that flash calculation and stability analysis can be performed using only three and two variables respectively when all the BIPs are zero independently of the number of components of the mixture.

The idea was then expanded by Jensen and Fredenslund (1987) to handle non – zero BIPs by applying truncated spectral expansion of the matrix $1 - k_{ij}$ from the Van der Waals mixing rules. Hendricks (1988) proved that two-phase equilibrium calculations can be performed with a reduced number of variables than the number of components including non-zero BIPs.

Hendriks and Van Bergen (1992) applied their reduction method to two-phase flash calculation. In their method, they approximate the BIP matrix using spectral expansion and then reduce the number of eigenvectors such that both, the error of the BIP matrix is small and the number of parameters is small. Its success is achieved because the large separation between the dominant eigenvalue of the k_{ij} matrix with the rest of the eigenvalues.

Kaul and Trasher (1996) proposed a parameter-based approach for two-phase equilibrium predictions. In their approach the reduce the number of variables to three or fourth depending on whether the BIPs are zero or not. Their method reduced the

minimization problem by taking advantage of the special mathematical form of the ideal mixing and excess parts of the Gibbs Free Energy.

Pan and Firoozabadi (2003) introduced a method based on the reduction of the dependent variables of the Gibbs Free Energy through the spectral theory of linear algebra. In their formulation, they reduced the number of eigenvalues of the $(1 - k_{ij})$ matrix as Hendriks and Van Bergen (1992) and tested on stability analysis and flash calculations focusing on robustness.

Nichita et al. (2004) presented a new reduction formulation with a selection of independent variables as suggested by Hendriks (1988). Their implementation includes non – zero BIPs in a $2\,Nc$ +3 equation system implemented with the Successive Substitution algorithm (SS) followed by the second-order minimum variable Newton Rapson close to the solution. Later, Nichita et al (2006) used the truncated expectral expantion of the attraction parameter of the EOS to reduce the number of equations to be solved during flash calculations.

Li and Johns (2006) implement a reduced flash calculation by decomposing the BIPs matrix into two parts using a quadratic expression. The number of independent parameters in their method is reduced to six for fluids with non-zero BIPs, three for zero BIPs and five for cases when exits only BIPs between a single component and the others. Okuno et al. (2010a) used their approximation of the BIPs matrix to develop a reduce method with only five and six independent variables for flash calculations and stability analysis respectively regardless of the number of components. Additionally, they demonstrate the robustness and efficiently of their method for two and three phases in UTCOMP a compositional reservoir simulator originally developed by Chang et al. (1990).

Okuno et al. (2010b) extend the algorithm for reduce two-phase flash calculations to three-phase flash calculations and showed the efficiency and robustness of the algorithm

in stand-alone calculations and during compositional reservoir simulations for various case studies.

Gaganis and Varotsis (2013) proposed a BIP matrix decomposition to a set of basis vectors that approximate the original BIP matrix by the minimization of an energy function. The reduction methods have shown to save computational time without losing a high degree of accuracy.

Another strategy that aims to reduce the computational cost of flash calculations is the use of tie-lines. The idea behind it is to solve the non-linear phase equilibrium equations for each gridlock separately from the reservoir simulation. Voskov and Tchelepi (2007) proposed a compositional space parametrization approach for simulation of gas flooding processes. In their method, flash calculations are performed and results are stored as a preprocessing stage for reservoir simulation. During the simulation, if the concentration lies on the compositional tie line, a tie-line table is used to look up the flash results. The performance of tie-line-based methods is improved by the use of information from a previous time step to determine the phase state at the current step.

ANNs have been applied to speed up flash calculations. Gaganis and Varotsis (2012) proposed an integrated method using classification and regression models. First, they used support vectors to classify the mixture as stable or unstable for a given pressure, temperature and feed composition. If the mixture is unstable, they use ANNs to predict the equilibrium coefficicients for a given pressure, temperature and composition. Later, they expanded their method in Gaganis and Varotsis (2014), introducing ANNs to estimate the reduced variables of Nichita and Graccia (2011) for flash calculations.

Kashinath et al. (2018). extended the work of Gaganis and Varotsis (2014) to supercritical phase determination, subcritical phase stability analysis and flash calculations. In their method, they used support vector machines to classify the

hydrocarbon mixture as supercritical, or subcritical with composition and pressure as inputs. Additionally, they solved the phase – split problem by applying ANNs to predict K-values for a given pressure, temperature, and composition.

Wang et al. (2019a) proposed using ANNs to assist the conventional flash calculation. In their approach, three ANNs are constructed. First two models predict the bubble point pressure and dew point pressure of the system. If the pressure of the system is between the bubble point pressure and the dew point pressure, the system is unstable. The second ANN is to predict the vapor mole fraction $f(\beta)$ and K – values that are used as an initial guess for the subsequent flash calculation. As a result, it requires less iterations to converge. The difference between Kashinath et al. (2018) and Wang et al. (2019) is that the use of ANNs is easier to calculate than the use of support vectors to label the stability of the system at a given pressure, temperature, and feed composition.

Wang et al. (2019b) implemented ANNs to speed up compositional reservoir simulation of a tight oil and shale gas reservoir. In their work, the initial estimates of the equilibrium constants, and capillary pressure are calculated using ANNs for a given pressure, temperature, and feed composition instead of Wilson's correlation. Their implementation reduced the number of iterations needed to converge to the solution during flash calculations because of the prediction of the initial guess using ANNs is close to the solution.

CHAPTER 3

ARTIFICIAL NEURAL NETWORKS

This chapter defines the basic concepts used in the development of ANNs, more specifically feed forward neural networks. This chapter is divided into three sections. The first part describes the fundamental concepts used in the development of ANNs. Secondly, the methodology to train ANNs is described, and finally, data preparation is discussed.

3.1 BASICS OF ARTIFICIAL NEURAL NETWORKS

Priddy et at. (2005) described that ANNs are networks of simple processing elements mapping an input space to an output space. One of the most outstanding capabilities of artificial neural networks is that they can perform complex non – linear mappings. There are many different types of ANNs that are used for different applications, but the principles are similar. Feed forward neural networks (FNNs) are used in this thesis. FNNs are composed of one input layer, one or more hidden layers, and one final layer that is called the output layer. Every layer, except for the output layer, includes a bias neuron and it is connected to the next layer. When a feed forward neural network has two or more hidden layers, it is called a deep neural network.

ANNs are conformed by neurons that receive information from previous neuros, then process the information internally through an activation function to generate an output response. However, some inputs to the neuron may be more relevant than others. Therefore, this process is modeled by weights in the input of the neuron. The output of a neuron can be express as

$$v = \sum_{i=0}^{Ni} w_i x_i + b_i \tag{3.1}$$

$$\alpha = f(v) \tag{3.2}$$

where f represents the activation function, v is the net stimuli of the neuron, N_i is the number of inputs of the neuron, x_i is the input value, w_i is the corresponding weight and b_i is the bias term. Equation 3.2 is given as the fundamental equation involved in ANNs by various textbooks (e.g. Aggwargal 2015, Goodfellow, 2016, Pedregosa 2011, Priddy 2005). Figure 3.1 shows a representation of a neuron can be seen as a small engine that takes the weighted inputs, process them and then transmit an output.

Priddy et al. (2005) showed that activation functions f are used in neural networks to control the firing rate or action potential of each neuron. There are different types of activation functions that can be used depending on the application of the neural network. For example, if the target value to predict is a real number, then the identity activation function is recommended. In another scenario, if the target value is to predict the probability of a binary classification system, a sigmoid activation function has to be selected.

In multilayer ANN, non-linear activation functions help to create more powerful compositions of different types of functions. If an ANN only uses linear activation functions, it would not provide better estimations than a single layer ANN. Additionally, the non – linear activation functions help to map the non-linear relationship with the inputs and the target values. Figures 3.3 to 3.5 show examples of widely used activation functions reported in the literature, such as sing, sigmoid, and hyperbolic tangent functions. Most recently, however, a number of piecewise activation functions, such as the rectified linear unit (RELU) and its variants, have become popular because they facilitate the training process of the neural networks as demonstrated by Goodfellow (2016). The most widely used activation functions are the following

$$f(v) = v\ (identity\ function) \tag{3.3}$$

$$f(v) = +1 \; if \; x \geq 0, -1 \; if \; x < 0 \;\; (sing \; function) \tag{3.4}$$

$$f(v) = \frac{1}{1 + e^{-v}} (sigmoid \; function) \tag{3.5}$$

$$f(v) = \frac{e^{2v} - 1}{e^{2v} + 1} (tanh \; function) \tag{3.6}$$

$$f(v) = \max\{v, 0\} \, (RELU \; function) \tag{3.7}$$

$$f(v) = \max\{\min[v, 1], 0\} \, (hard \; tanh \; function) \tag{3.8}$$

The specific architecture of a multilayer neural network is referred to as feed forward networks (FNN) because successive layers feed into the one another in a forward direction from input to output. The architecture of an artificial neural network is fully connected when all the neurons are connected to the neurons in the next layer. Therefore, the dimension of an ANN is defined by the number of layers and the number of neurons in each layer. Priddy et al. (2005) defined the structure of ANNs as composition function in a multilayer ANN as follows:

$$g^1 = f^1(v), \; g^2 = f^2(g^1), \ldots, \overline{y} = f^{\ell}\left(g^{\ell-1}\right) \tag{3.8}$$

where g represent the functions computed in layer ℓ and $\overline{y}$ represent the output of the neural network. The use of non-linear activation functions is key to increase the power of multiple layers. As a result, ANNs are often referred to as universal approximators.

The learning process of ANNs occurs by changing the weights connecting different neurons. As an analogy, a stimulus is needed to learn in biological organisms. In the case of ANNs, the external stimuli are provided by the training data containing examples of input – output response. The pairs of data are feed into the ANN to adjust the weights based on the error between the target value and the predicted one. The function used to calculate this error is called the loss function. There are several loss functions reported in

the literature. Goodfellow (2016) reported that their use depends on the application of the ANN. For FNN, the mean square error J_{MSE} and the mean absolute error (MAE) are commonly used.

$$J_{MSE} = \frac{1}{n}\sum_{i=1}^{n}(y_i - \overline{y_i})^2 \quad 3.9$$

$$J_{MAE} = \frac{1}{n}\sum_{i=1}^{n}(y_i - \overline{y_i}) \quad 3.10$$

where J is the value of the loss function, n is the number of data samples, y is the target value, and $\overline{y}$ is the predicted value at instance i.

The goal of changing the weights is to modify the computed function to make the predictions more accurate in future calculations. Therefore, the weights are carefully adjusted using an algorithm called backpropagation. Once the neural networks are properly trained, they are capable of making accurate predictions on unseen data over a finite set of inputs and outputs. This ability is referred to as model generalization. Figure 3.2 shows a fully connected ANN.

3.2 Training artificial neural networks using backpropagation algorithm

The objective of the backpropagation algorithm is to adjust the weights in the ANN model until the desired output is obtained by minimizing the output error. Goodfellow. (2016) described the backpropagation as an algorithm consisting of two main steps: the forward phase and backward phase. During the forward phase, the training data are fed into the ANN to calculate the response of each neuron and the output response with the current set of weights. Priddy et al. (2005) showed the derivation of the backpropagation algorithm starting with the comparison of the value predicted by the ANN and the real value through a loss function. To reduce the error of the ANN, it is necessary to minimize the loss function

J with respect to the weights in all the neurons of the network and force it to zero. The gradient descent error minimization is defined by

$$\frac{\partial \mathrm{J}}{\partial w_{ij}^{\ell}} \equiv 0 \tag{3.11}$$

Now, the chain rule can be used to break down the partial derivate into two parts. The first term corresponds to the change in the loss function with respect to the stimuli and the second term represent the change in the stimuli with respect to the weights.

$$\frac{\partial \mathrm{J}}{\partial w_{ij}^{\ell}} = \frac{\partial \mathrm{J}}{\partial v_{ij}^{\ell}} \frac{\partial v_{ij}^{\ell}}{\partial w_{ij}^{\ell}} \tag{3.12}$$

The second term of the partial derivate can be solved by substituting the stimuli given in equation 3.1. This results in the output of a neuron as shown in equation 3.13

$$\frac{\partial v_{ij}^{\ell}}{\partial w_{ij}^{\ell}} = \frac{\partial}{\partial w_{ij}^{\ell}} \sum_{i=0}^{Ni} (w_{ij}^{\ell} x_{ij}^{\ell-1}) + b_{ij}^{\ell} = x_{ij}^{\ell-1} \tag{3.13}$$

Substituting equation 3.13 into equation 3.12, we obtain equation 3.14. By defining the change of the loss function with respect to the stimuli as a delta term, we obtain equation 3.15. Substituting it back into equation 3.14 results in equation 3.16.

$$\frac{\partial \mathrm{J}}{\partial w_{ij}^{\ell}} = \frac{\partial \mathrm{J}}{\partial v_{ij}^{\ell}} x_{ij}^{\ell-1} \tag{3.14}$$

$$\Delta_{ij}^{\ell} = -\frac{\partial \mathrm{J}}{\partial v_{ij}^{\ell}} \tag{3.15}$$

$$\frac{\partial \mathrm{J}}{\partial w_{ij}^{\ell}} = -\Delta_{ij}^{\ell} x_{ij}^{\ell-1} \tag{3.16}$$

The derivative of the loss function with respect to the stimuli can be breakdown using the chain rule to a term that measures the change of the loss function with respect to the neuron output in any layer times the change of the neuron output with respect to its own stimuli. This gives

$$\frac{\partial \mathrm{J}}{\partial v_{ij}^{\ell}} = \frac{\partial \mathrm{J}}{\partial x_{ij}^{\ell}} \frac{\partial x_{ij}^{\ell}}{\partial v_{ij}^{\ell}} \tag{3.17}$$

The solution of the change of the neuron output with respect to its own stimuli can be solved by substituting equation 3.2 to obtain the derivative of the activation function evaluated at the total stimuli.

$$\frac{\partial x_{ij}^{\ell}}{\partial v_{ij}^{\ell}} = \frac{\partial}{\partial v_{ij}^{\ell}} f^{\ell}\left(v_{ij}^{\ell}\right) = f'^{\ell}\left(v_{ij}^{\ell}\right) \tag{3.18}$$

Finally, we have the derivative of the output error with respect to any neuron's output. Considering the mean square error between the predicted value with respect to the target value for a single layer,

$$\frac{\partial \mathrm{J}}{\partial x_{ij}^{\ell}} = \frac{\partial}{\partial x_{ij}^{\ell}} \left[\frac{1}{n} \sum_{i=1}^{n} (\overline{x_i} - x)^2 \right] = -(\overline{x_i} - x) \text{ for } \ell = \mathrm{L} \tag{3.19}$$

where L indicates the output layer. However, each neuron is connected to all neurons in the following layer. Therefore, a variation in weight affects the subsequent layers in the network. The variation in the loss function with respect to the internal neuron's output is determined by the variation on its own stimuli. Then, it is necessary to sum over all the variations in the downstream network to determine the variation in the final outcome with respect to the output of a hidden neuron. This can be express as

$$\frac{\partial \mathrm{J}}{\partial x_{ij}^{\ell}} = \sum_{k=1}^{N^{\ell+1}} \frac{\partial \mathrm{J}}{\partial v_{ij}^{\ell+1}} \frac{\partial v_{ij}^{\ell+1}}{\partial x_{ij}^{\ell+1}} \quad \text{when } \ell < \mathrm{L} \text{ (Hidden layers)} \tag{3.20}$$

The first term in the summation corresponds to the delta term defined in equation 3.15.

$$\frac{\partial \mathrm{J}}{\partial v_{ij}^{\ell+1}} = -\Delta_{ij}^{\ell+1} \text{ when } \ell < \mathrm{L} \text{ (Hidden layers)} \tag{3.21}$$

The second term in the summation can be solved by substituting the net stimuli as follows:

$$\frac{\partial v_{ij}^{\ell+1}}{\partial x_{ij}^{\ell+1}} = \frac{\partial}{\partial x_{ij}^{\ell+1}}\left(\sum_{i=0}^{N^{\ell}} (w_{ij}^{\ell+1} x_{ij}^{\ell}) + b_{ij}^{\ell+1}\right) = w_{ij}^{\ell+1} \text{ when } \ell < \mathrm{L} \tag{3.22}$$

Now, substituting equations 3.21 and 3.22 into equation 3.20 results in

$$\frac{\partial \mathrm{J}}{\partial x_{ij}^{\ell}} = -\sum_{k=1}^{N^{\ell+1}} \Delta_{ik}^{\ell+1}\, w_{ik}^{\ell+1} \text{ when } \ell < \mathrm{L} \tag{3.23}$$

The delta term defined in equation 3.15 can be defined for the hidden layers and the output layers as follows:

$$\Delta_{ij}^{L} = f'^{L}\left(v_{ij}^{L}\right)(y_i - \overline{y_i}) \text{ where } \ell = \mathrm{L} \text{ (Output layer)} \tag{3.24}$$

$$\Delta_{ij}^{\ell} = f'^{\ell}\left(v_{ij}^{\ell}\right) \sum_{k=1}^{\ell+1} \Delta_{ik}^{\ell+1}\, w_{ik}^{\ell+1} \text{ when } \ell < \mathrm{L} \text{ (Hidden layer)} \tag{3.25}$$

From equation 3.25, we can see that we need to calculate the delta term for the output layer first, and then go backwards from the output layer to the input layer calculating the delta term for the hidden neurons. Since the loss function is directly related to the delta term of the output layer, the error is propagated backwards to the neural network.

The minimization of the output error with respect to the weights of the network can be express as

$$\Delta w_{ijk}^{\ell} = -\mu\, \frac{\partial \mathrm{E}_k}{\partial w_{ijk}^{\ell}} \tag{3.26}$$

Finally, combining equation 3.26 with 3.16 gives the formula for weight update after each iteration

$$\Delta w_{ijk}^{\ell} = -\mu\, x_{ij}^{\ell} x_{ij}^{\ell-1} \qquad 3.27$$

where the subscripts $i, j, \ell,$ and k indicate neuron, connecting neuron, layer number, and iteration step respectively. The term μ corresponds to the step size in the minimization problem. Equation 3.27 is the working equation used to obtain the set of weights and bias that allow ANNs to model complex nonlinear functions. The backpropagation algorithm is the most common technique to train ANNs and more detailed derivations can be found in several textbooks (e.g. Rumelhart 1986, Pao 1989, Haykin 1994).

The backpropagation training procedure can be summarized as follows.

1. Uniformly random Initialize weights w_{ij}^{ℓ} for the entire neural network
2. Propagate training data forward through the network and generate an output response
3. Calculate the error between the predicted value and the target value by using equation 3.9
4. Propagate the error backwards to the neural network and calculate the gradients of each neuron by the weight-update formula with equation 3.27
5. If the output error is high or the maximin number of iterations has not been reached, go to step 2.

The weights of the neural network are updated after each sample in the training data is processed. However, if the adjustment of the weights is performed after all the samples have been processed, the method is called batch training. Both methods are performed with a large number of training samples until the error converges to a minimum. There are several programs available to train artificial neural networks using the backpropagation

algorithm (e.g. Torch, Theano, TensorFlow, and Keras). In this thesis, Keras was used to train the artificial neural networks for the fugacity coefficient.

3.3 Data preparation.

The objective of ANN training is to find a set of the parameters (e.g. weights, bias terms, and activation functions) that results in the best performance with data that have not been used during the training stage. This process is called "model generalization" and it means how well the model performed in new data within an acceptable limit of the input feature space.

During neural network training, we can find two cases. Undertraining, that occurs when the model did not reach the a minimum and it performs poorly with testing data. Undertraining can occur because of the sensitivity of the model to limited data, model bias, irreducible error due to missing variables or the range of the variables of the training data. Overfitting is the opposite of underfitting. In this case, the neural network will memorize the training data set, but it will perform poorly in testing data. The goal is to find the configuration with the best performance with independent data. To avoid underfitting or overfitting, datasets have to be prepared before neural network training. This section describes the two main steps for data preparation for neural network training.

3.3.1 Data splitting

To find the optimal neural network configuration, the general approach is to randomly sample the population to generate three different and independent data sets: training data set, validation data set, and testing data set (Pedregosa 2011).

The training data set is used with the backpropagation training algorithm to adjust the weights that conform to the neural network architecture to produce the desired output.

In practice, the training data set consists of pairs of an input vector that contains the variables to reproduce and output response. Successively, a validation data set is used to make predictions using the fitted model to tune the model hyperparameters (e.g. the number of neurons in each layer, the number of layers in the model) until the desired accuracy is achieved. Finally, the testing data set is used to provide an unbiased evaluation of the fully trained model performance. The neural network is biased towards the validation and testing data sets. Therefore, the test set is used to estimate the generalization error of the neural network.

There are several ratios of how to partition the data set into training, validation, and test data set reported in the literature. However, the conventional approach follows the pareto principle that states that for many events, roughly 80% of effects come from 20% of the causes. Mathematically, the pareto principle is roughly followed by a power-law distribution for a particular set of parameters. Therefore, the complete data set can be split in 80% for training, 10% for validation, and the remaining 10% of the data set for testing.

3.3.2 Data normalization

Once the dataset has been split into different sets, normalization of the data is required before starting the training process. The goal of normalization is to change the values of the variables into a common scale without distorting the range of the values. Data normalization is necessary during the development of ANNs to minimize the bias of the neural network towards one feature or another. Gulli et al (2017) explain that data normalization can speed up the training process, and it is especially useful for modeling applications when the range of the variables are generally on different scales. There are several techniques of data normalization reported in the literature (e.g. Pedregosa 2011, Priddy 2015, Gulli 2017), but the most widely used is the Min – Max normalization. This

is performed by rescaling all the variables to the same range of 0 to 1 or -1 to 1. The rescaling is calculating using linear interpolation as given by equation 3.28

$$\overline{x_i} = \left[\frac{x_i - x_{min}}{x_{max} - x_{min}}\right] \tag{3.28}$$

where $\overline{x_i}$ is the scaled feature, x_i represents the original value of the variable, and x_{min} and x_{max} correspond to the upper and lowest limits of the variable in the data set before splitting into 3 sub datasets. When Min – Max normalization is applied, each feature will lie in a new range of values, but the underlying data distribution will remain intact. The main advantage of using Min – Max normalization is that the dataset will keep its distribution, and no bias will be added to the neural network. Finally, for applications that require data in ranges outside of the neuron input, the data must be rescaled to its original data range as shown in equation 3.29.

$$x_i = \overline{x_i}\,(x_{max} - x_{min}) + \; x_{min} \tag{3.29}$$

A summary of the training and testing neural networks is the following

1. Randomly sample the dataset into 3 categories: training, validation and testing.
2. Choose a neural network model and define its architecture (Number of neurons, layers and activation function)
3. Normalize the datasets using min-max normalization using equation 3.28
4. Train the neural network using the backpropagation algorithm described in section 3.2 or with the use of available software to train neural networks (e.g. Keras, Tensorflow, Pythorch)
5. Evaluate the neural network using the validation dataset and tune hyperparameters if necessary

6. Repeat steps 2 to 5 with different neural networks architectures until the desired accuracy is achieved.
7. Select the neural network model with the lowest validation error
8. Make predictions using the chosen neural network models on the testing data set
9. Rescale the predictions of the neural network to its original scale using equation 3.29
10. Evaluate the performance of the ANNs using mean square error and mean absolute error from equations 3.9 and 3.10, respectively.

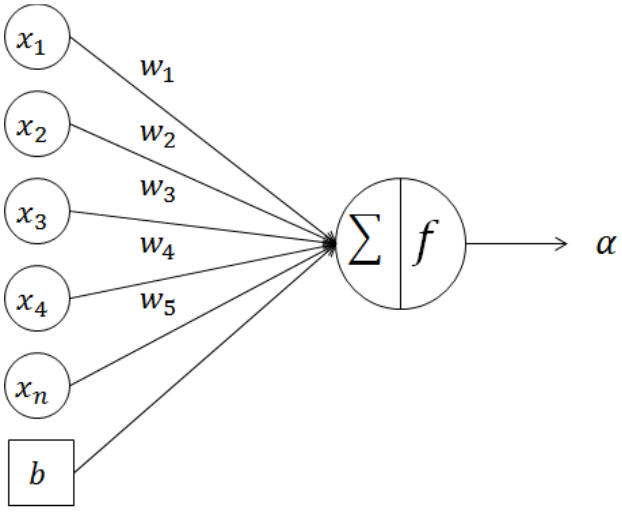


Figure 3.1 Basic architecture of a neuron

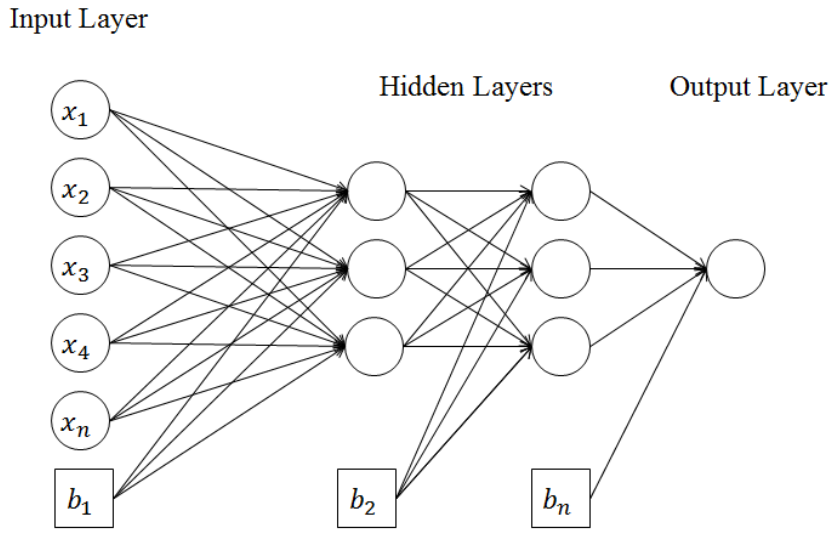


Figure 3.2 Fully connected feedforward neural network structure

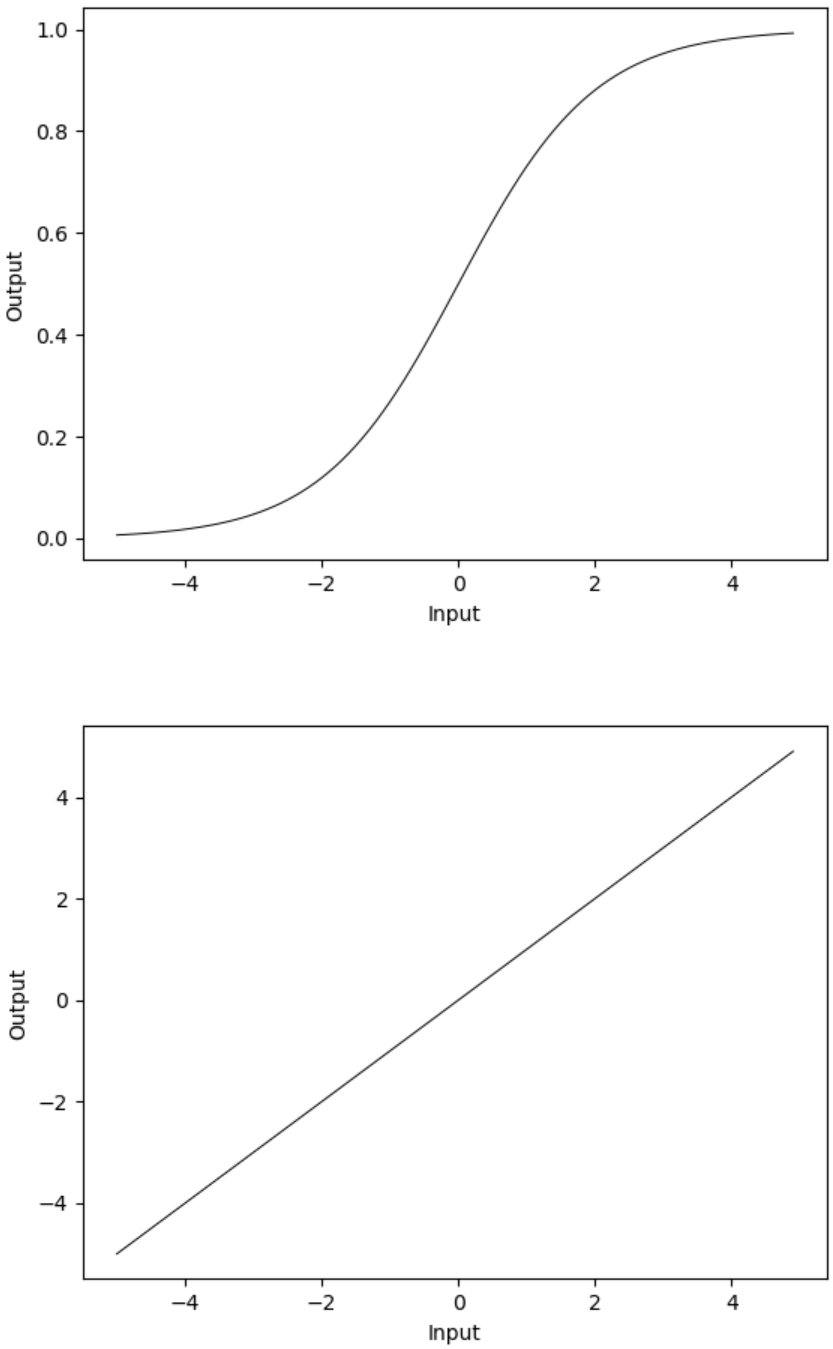


Figure 3.3 Commonly used activation functions used in ANNs.
Top: Sigmoid. Bottom: Identity.

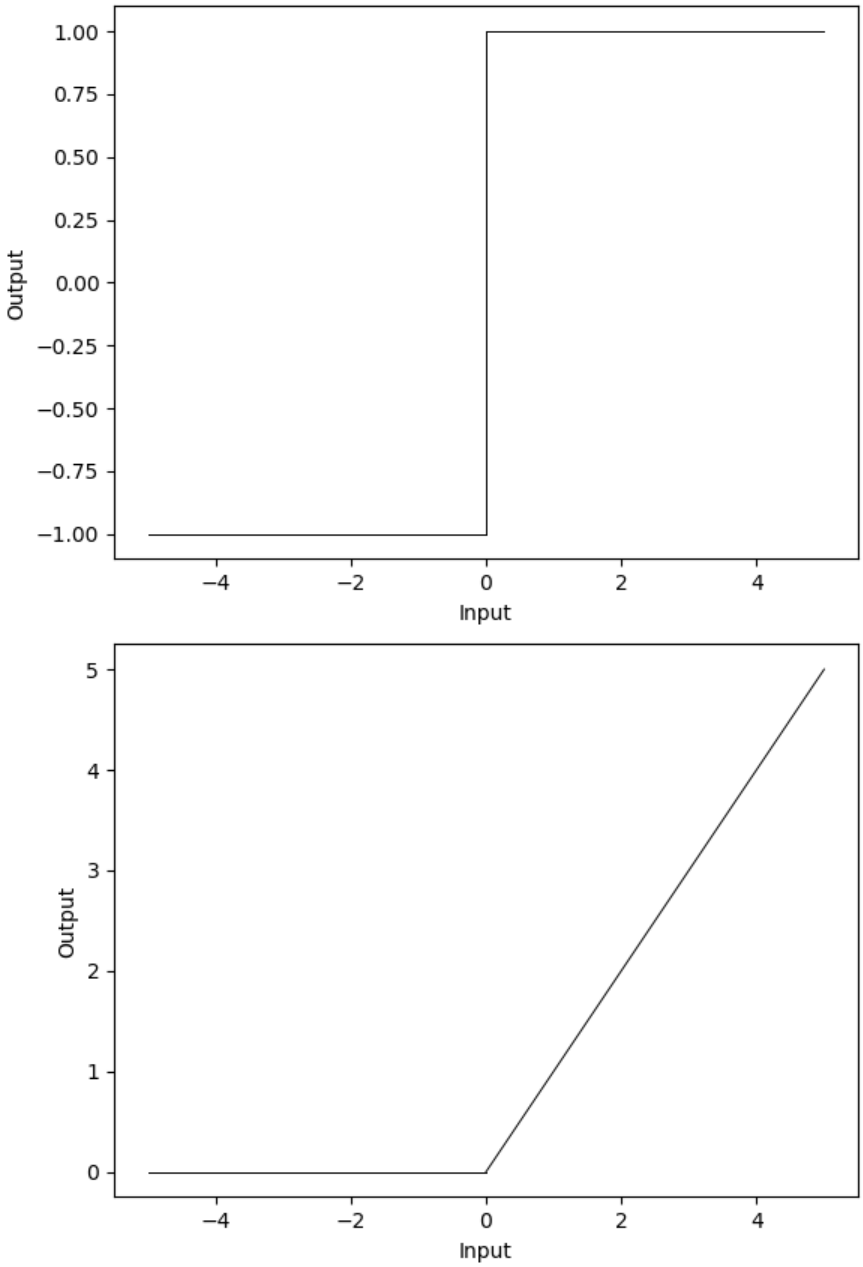


Figure 3.4 Commonly used activation functions used in ANNs. Top: Sign. Bottom: Rectified Linear Unit (ReLU).

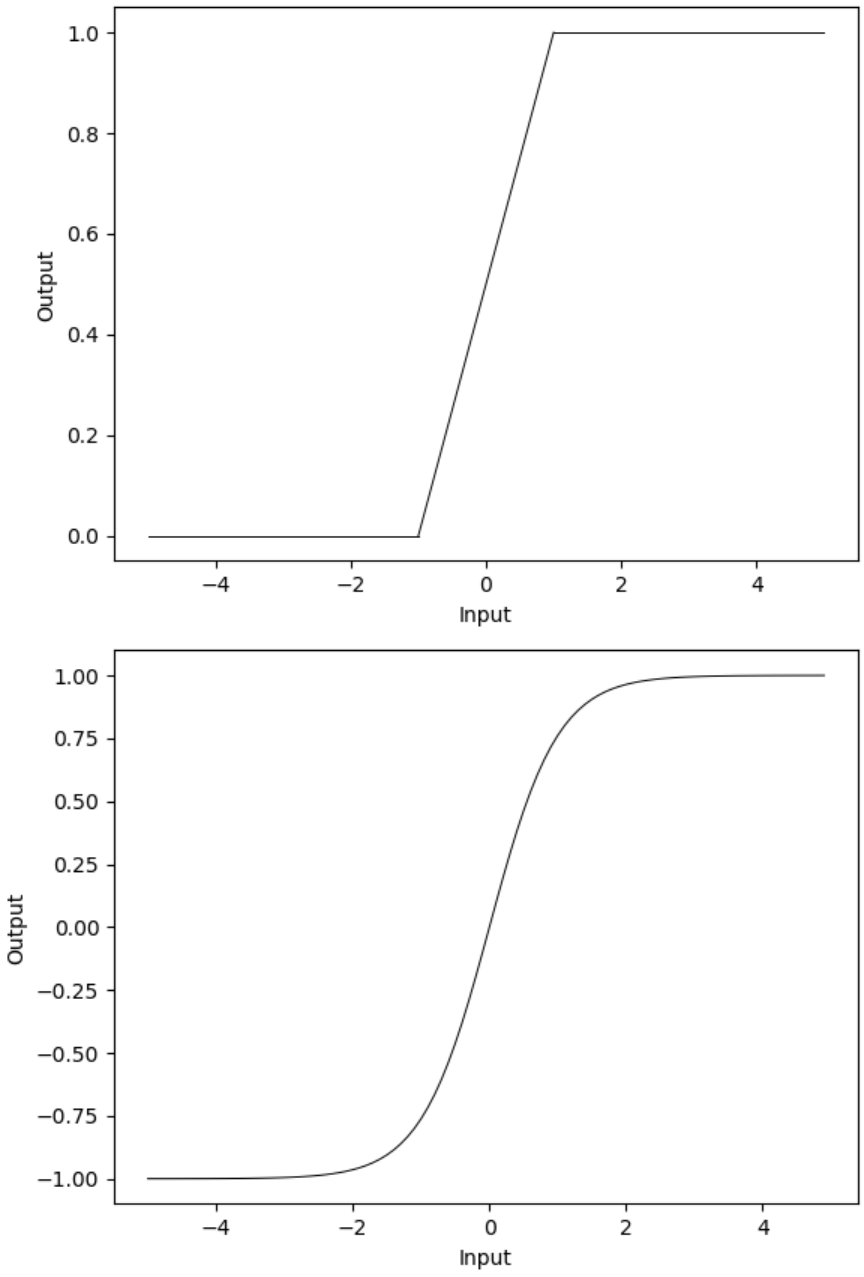


Figure 3.5 Commonly used activation functions used in ANNs.
Top: Hard Tanh. Bottom: Tanh

CHAPTER 4

METHODOLOGY, FORMULATION AND ALGORITHM

Efficient phase equilibrium calculations are important for practical compositional reservoir simulation studies. A number of methods have been proposed and tested for speeding up flash calculations, as discussed in Chapter 2. In this thesis, we test use of artificial neural networks (ANNs) to replace the most fundamental, but time-consuming function in EOS flash calculations, namely the fugacity coefficient. This chapter is divided into two sections. The first part describes the methodology to generate ANNs for the fugacity coefficient. The second portion describes the formulation and implementation of the ANN flash.

4.1 ARTIFICIAL NEURAL NETWORKS FOR THE FUGACITY COEFFICIENT

In this section, we present the methodology to generate ANN models to represent the fugacity coefficient during the minimization of the Gibbs free energy with successive substitution. The first step is to determine the range of application of the ANN and create a database for a range of pressure and an evenly distributed composition space. Since temperature of the reservoirs is keep constant, the only variables in the ANN are pressure and composition of the fluid system of interest.

The methodology to generate the dataset for the ANN is the following:

1. With the critical properties of the fluid and acentric factor calculate b, κ and the temperature-independent part of a using equations 2.43 and 2.45
2. At reservoir temperature, calculate the values for a and α with equations 2.42 and 2.44
3. Calculate A and B for the fluid mixture using equations 2.38 and 2.39

4. Calculate the parameters $\varepsilon, \beta,$ and γ using equations 2.53, 2.54 and 2.55
5. Solve the cubic equation of state to obtain the compressibility factor with equation 2.52
6. Calculate the fugacity coefficient using equation 2.48. When the cubic EOS has multiple roots of the compressibility factors, the correct root corresponds to the root with the lowest Gibbs free Energy (Evelein *et al*. 1976)
7. Store the fugacity coefficient value of each component and the corresponding pressure and concentration conditions.
8. Repeat steps 1 through 7 for all pressures and composition in the application range

Once a database is generated, the second step consists of determining the architecture of the ANNs (e.g. activation function, number of hidden layers, number of neurons per hidden layer). In this thesis, the architecture for all the ANN models was fixed as described in Table 4.1. The fully connected feed forward neural network for the fugacity coefficient is shown in Figure 4.1.

Depending on the number of components in the mixture, an ANN model for each component in the mixture as a function of pressure and composition has to be calculated using the methodology described in section 3.3.2. Finally, the performance of each neural network is evaluated by calculating the global percentage error, mean square error and the correlation coefficient for each model. The ones with the lowest error are selected.

4.2 ARTIFICIAL NEURAL NETWORK FLASH CALCULATION

This section describes the formulation of the proposed algorithm using the ANNs with the successive substitution to solve fugacity equations. The first portion of the section

describes the formulation of the fugacity coefficient using ANNs. The second part describes the ANN flash algorithm.

4.2.1 Formulation

The Peng-Robinson Equation of State (Peng and Robinson, 1978) with the van der Waals mixing rules is used for the formulation of the closed-form solution of the fugacity coefficient. However, any equation of state can be used depending on the type of thermodynamic modeling required. The Peng-Robinson equation of state is

$$\mathrm{P} = \frac{RT}{(\underline{V} - b)} - \frac{a(T)}{[\underline{V}(\underline{V} + b) + b(\underline{V} - b)]} \tag{4.1}$$

where

$$a(T) = 0.45724 R^2 {T_c}^2 \alpha(T)/P_c \tag{4.2}$$

$$b = \frac{0.07780 RT_c}{P_C} \tag{4.3}$$

$$\alpha(T) = 1 + \kappa(1 - \sqrt{T/T_c}) \tag{4.4}$$

$$\kappa = 0.37464 + 1.54226\omega - 0.26992\omega^2 \text{ for } \omega < 0.49 \tag{4.5}$$

$$\kappa = 0.37464 + \omega\big(1.48503 + \omega(-0.16442 + 0.01667\omega)\big) \text{ for } \omega > 0.49 \tag{4.6}$$

The van der Waals rules are as follows:

$$\mathrm{A} = \sum_{i=1}^{Nc} \sum_{j=1}^{Nc} x_i x_j a_{ij} \tag{4.7}$$

$$\mathrm{B} = \sum_{i=1}^{Nc} x_i b_i \tag{4.8}$$

where A and B represent the attraction and co-volume parameters, respectively. The combining rules for a_{ij} are defined as

$$a_{ij} = \sqrt{a_i a_j}(1 - k_{ij}) \tag{4.9}$$

The EOS can be expressed as a cubic equation with parameters $\varepsilon, \beta,$ and γ for the Peng – Robinson EOS as:

$$Z^3 + \varepsilon Z^2 + \beta Z + \gamma \tag{4.10}$$

$$\varepsilon = -1 + B \tag{4.11}$$

$$\beta = A - 3B^2 - 2B \tag{4.12}$$

$$\gamma = -AB + B^2 + B^3 \tag{4.13}$$

where $Z = P\underline{V}/RT$ is the compressibility factor.

The fugacity coefficient of component i in the mixture using the Peng-Robinson EOS is defined as

$$\ln\varphi_{ij}\left(P, T, \underline{x}\right) = (Z - 1)\frac{B_i}{B} - \ln(Z - B) - \frac{A}{(\delta_1 - \delta_2)B}\left(\frac{2\sum_{j=1}^{Nc} x_j A_{ij}}{A_m} - \frac{B_i}{B}\right) ln\left[\frac{Z + \delta_1 B}{Z + \delta_2 B}\right]; \tag{4.14}$$

where $\delta_1 = 1 + 0.2^{0.5}$ and $\delta_2 = 1 - 0.2^{0.5}$. As discussed in chapter 2, the fugacity coefficient is used repeatedly during phase equilibrium calculations. Therefore, the key to obtaining an efficient algorithm for phase equilibrium calculation is to reduce the operations required for the fugacity coefficient calculation. A simple approximation for the fugacity coefficient has been developed by using ANNs in this thesis. The approximation can be express as

$$\underline{v} = [\bar{P}, x_i\,]\; for\; n = 1, \ldots, Nc \tag{4.15}$$

$$\sigma = \sum_{n=1}^{Nv} v_n w_n + b_2 \tag{4.16}$$

$$g_i = max[\sigma, 0]_i \quad for\ i = 1, \dots, Nn \tag{4.17}$$

$$\tau = \sum_{n=1}^{Nn} g_n w_n + b_2 \tag{4.18}$$

$$h_i = max[\tau, 0]_i \quad for\ i = 1, \dots, Nn \tag{4.19}$$

$$ln\varphi_{ij}(P, \underline{x}) = \left(\sum_{n=1}^{Nn} h_n + b_3 \right)_{ij} \quad for\ \ i = 1, Nc\ and\ j = 1, \dots, Np \tag{4.20}$$

where $\underline{v}$ corresponds to the vector composed of normalized pressure $\overline{P}$, and x_i is the mole fraction of component i. The values of w and b correspond to the weights and bias that conform the neural network, and Nv and Nn represent the number of inputs of the neural network and the number of neurons in the hidden layer, respectively.

In this approach, the fugacity coefficient is calculated directly by the ANN models for a given pressure and feed composition. The advantages of using ANNs for the fugacity coefficient is that there is no need to solve the EOS during the iterative flash calculation. The computationally expensive calculations (e.g. logarithms, divisions, and root square) involved in the fugacity coefficient calculation are replaced by computationally less expensive calculations (e.g. additions and multiplications) used in the ANNs. Therefore, the fugacity coefficient approximation using ANNs can decrease the computation time of the fugacity coefficient calculation. A more detailed procedure to generate ANN models for the fugacity coefficient is given in case 1.

4.2.2 Algorithm

The application of ANNs to flash calculation (ANN flash) is a combination of the conventional flash calculation with the use of ANNs for efficient calculation of the fugacity coefficient without using an EOS. For a flash calculation at a given pressure and temperature, the solution of the minimum Gibbs free energy is found in the composition space. The first-order necessary conditions for the minimization of the Gibbs free energy are the solution of the fugacity equations as

$$lnf_{ij} - lnf_{ij} = 0, \text{where i} = 1, \dots, \text{Nc and j} = 1, \dots, \text{Np} \tag{4.21}$$

where Np represents the reference phase. The use of successive substitutions is used to find the phase equilibrium. As mention in section 2, the independent variable in successive substitutions is the constant equilibrium K-values.

$$K_{ij} = \frac{x_{ij}}{x_{iNp}} \text{ where i} = 1, \dots, \text{Nc and j} = 1, \dots, (\text{Np} - 1) \tag{4.22}$$

Therefore, successive substitution solves for equation 4.21 subjected to material balance constraints define in equations 4.24, 4.25 and 4.26.

$$\ln {K_{ij}}^{k+1} = \ln {\varphi_{iNp}}^{k} - \ln {\varphi_{ij}}^{k}, \text{where i} = 1, \dots, \text{Nc and j} = 1, \dots, (\text{Np} - 1) \tag{4.23}$$

$$\sum_{j=1}^{Np} \beta_j x_{ij} = z_i \tag{4.24}$$

$$\sum_{i=1}^{Nc} z_i = 1.0 \tag{4.25}$$

$$\sum_{i=1}^{Nc} x_{ij} = 1.0 \text{ and } x_{ij} \geq 0 \text{ for i} = 1, \dots, \text{Nc and j} = 1, \dots, \text{Np.} \tag{4.26}$$

In equation 4.23, the superscripts indicate the iteration step and the value of the fugacity coefficient. In this algorithm, the fugacity coefficient is calculated using the equation 4.20 at each iteration step for a given pressure and feed composition. Since the fugacity coefficient is a function of component i in phase j the Rachford – Rice equation is used to find the phase mole fractions as

$$f(\beta) = \sum_{1}^{Nc} \frac{(1 - K_i)z_i}{1 - (1 - K_i)\beta} = 0 \qquad 4.27$$

Finally, the phase compositions are calculated using the material balance defined as

$$x_i = \frac{z_i}{f(\beta)(K_i - 1) + 1} \; where \;\; i = 1, .., Nc \qquad 4.28$$

$$y_i = \frac{z_i K_i}{f(\beta)(K_i - 1) + 1} = z_i K_i \; where \;\; i = 1, .., Nc \qquad 4.29$$

The ANN flash calculation algorithm can be summarized as:

1. Specify T, P and feed mole fraction of the mixture of interest.
2. Calculate the initial guess of the equilibrium constant using Wilson's correlation using equation 2.34.
3. Solve Equation 4.27 for the vapor fraction.
4. Calculate the liquid and vapor compositions using equation 4.28 and 4.29.
5. Calculate the fugacity coefficients using the ANN model for $\ln\varphi_{iL}$ and $\ln\varphi_{iV}$ using equation 4.20.
6. Check for convergence based on the residuals of the fugacity coefficient ANN models $\|\ln x_i\varphi_{iL} - \ln y\varphi_{iv}\| < \varepsilon \; (e.g, \varepsilon = 10^{-6})$. If convergence is achieved, stop. Otherwise, go to step 7.

7. Update the equilibrium constants $lnK_i^{k+1} = (\ln\varphi_{iL} - \ln\varphi_{iV})^k$.

8. Go to step 3.

Note again that the above algorithm does not use an EOS during the iteration. It does not require the calculation of the van der walls mixing rules, the solution of the cubic equation of state and the evaluation of different roots when the EOS has more than one root. All those calculations were performed during the database generation for the training process of the ANN. Appendix C shows the code implementation of ANNs with successive substitutions.

Table 4.1. ANN architecture for fugacity coefficients

Setting	Value
Activation function in Hidden layers	ReLU, equation 3.7
Number of hidden layers	2
Number of neurons per hidden layer	20
Input	Mixture composition and pressure
Outputs	Fugacity coefficient

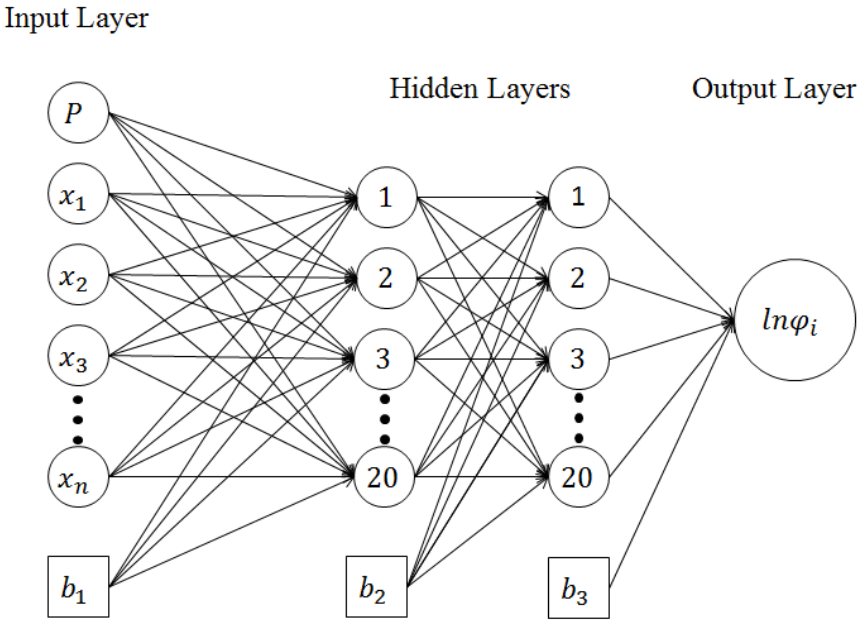


Figure 4.1. Fully connected neural network for the fugacity coefficient

CHAPTER 5

CASE STUDIES

This chapter presents four different hydrocarbon-rich mixtures for ANN-based standalone flash calculations. The mixtures represent different scenarios commonly found in petroleum engineering applications. We compare the ANN flash and the conventional EOS flash in terms of the accuracy of the fugacity coefficient, calculate the fluid properties, and analyze the convergence behavior.

5.1 STAND-ALONE FLASH CALCULATIONS

EOS flash often becomes more challenging and time-consuming with increasing number of components and increasing level of miscibility. The fugacity coefficient becomes more non-linear with increasing non-zero BIPs. In this chapter, therefore, ANN flash is compared with the conventional EOS flash for four fluid models with different component numbers, miscibility levels, and BIPs.

5.1.1 Case 1

The reservoir oil used for this study case was made after the BSB West Texas oil at 105°F (Khan et al. 1992). First, CO_2 was removed from the original model of Khan et al. (1992). Then, all BIPs were set to zero as described in Table 5.1. Okuno (2017) demonstrated that the fugacity coefficient with zero BIPs is less non-linear in composition space than when non-zero BIPs are involved. This is mainly because the fugacity coefficient with zero BIPs is expressed by only two reduced parameters as follows:

$$\theta_1 = \sum_{i=1}^{Nc} z_i B_i = B_m \tag{5.1}$$

$$\theta_2 = \sum_{i=1}^{Nc} z_i \sqrt{A_i} = \sqrt{A_m} \qquad 5.2$$

The objective of this case study was to test the idea of using ANNs for approximation of the fugacity coefficient during flash calculations. Hence, this case is designed to be relatively straightforward for phase equilibrium algorithms. The thermodynamic conditions are also far from a critical point. The phase diagram for the adapted BSB west Texas oil is shown in Figure 5.1.

The specific procedure used for this study case is as follows:

1. Determine the application range of the ANNs. The phase diagram indicates that the two-phase region is from 0 to 27.89 bar. The number of components in the mixture is six.
2. Create a database with the selected range of application. For each component in the mixture, calculate the fugacity coefficient in the composition space from 0 to 1 and the pressure range. The pressure intervals in this case were set to be 0.5 bar and the concentration intervals 0.01. However, the accuracy of the ANN model can be improved with larger datasets. Thus, the resolution in pressure and concentration act as parameters to increase the accuracy of the neural network if needed. The dataset is stored in an array of pressure, $x_1, x_2, x_3, x_4, x_5, x_6, ln\varphi_1, ln\varphi_2, ln\varphi_3, ln\varphi_4, ln\varphi_5,$ and $ln\varphi_6$. Here, the ANNs will take pressure and composition as inputs, and the fugacity coefficient will be the output.
3. Determine the maximum and minimum values for the pressure interval and the fugacity coefficients of each component.
4. Once the database has been generated, randomly split it into 3 different datasets: The training dataset is composed of 80% of the rows randomly selected from the

database. For the remaining 20% of the database, randomly select 50% of the rows to form the validation dataset and the remaining 50% correspond to the testing dataset.

5. Normalize the pressure and fugacity coefficient for each dataset using min-max normalization with equation 3.28. Since the concentration is a value between 0 and 1, no further normalization is needed.
6. Specify the ANN architecture. In this case, the ANN consists of 20 neurons and 2 hidden layers. The activation function is ReLU.
7. Train the ANN model for each component. In this work, Keras was used to perform the backpropagation algorithm. The Keras setup used in this research is shown in Table 5.2.
8. Make predictions using the trained ANNs with the testing dataset.
9. Re-scale the output of the neural network ($ln\varphi_i$) to its original range using equation 3.29.
10. Calculate the mean square error, average percentage error, and correlation coefficient R^2 of the predicted value and the original value.

Appendix A shows the code used to generate the database. Appendix B describes the code in Python that uses Keras to train the ANN models. The generalization errors for the different ANN models used in this case study are shown in Table 5.3. The mean average error of the fugacity coefficient neural network prediction ranges from 0.30% to 0.95%. It was observed that the ANNs models predict fugacity coefficients with low deviations from the original values calculated with EOS. The accuracy comparison between the EOS and the ANNs is shown in Figures 5.2, 5.3, and 5.4. The accuracy comparison between the ANN and EOS fugacity coefficients indicate that we can achieve a high accuracy in fugacity coefficients along different mixture concentrations and different pressures in the

two-phase region. The ANN fugacity coefficient was implemented in flash calculation with the successive substitution method. The phase compositions are compared between the ANN flash and EOS in Figures 5.5 to 5.10. The accuracy in the composition calculations will impact directly the phase property calculations. It can be observed that the deviation in phase compositions between the EOS and the ANN flash are negligible. The calculation comparisons of fluid properties, such as vapor and liquid fraction, saturation and densities, are shown in Figures 5.11 to 5.13. The ANN flash calculations show essentially the same results as the EOS for this case (Table 5.3). The time per iteration comparison between EOS and ANN flash is shown in table 5.4. Here, we can observe that the time per iteration in the ANN flash is reduced by 90.13%. The time reduction was calculated using equation 5.3.

$$\text{Time reduction} = \left(\frac{\text{EOS flash time} - \text{ANN flash time}}{\text{EOS flash time}}\right) \times 100\% \qquad 5.3$$

The fugacity coefficient execution time using ANNs is reduced by 96.85% for six components with respect to conventional EOS fugacity coefficient execution time, as shown in Figures 5.14 and 5.15. Finally, the total flash calculation time (the product of the number of iterations and the time per iteration) between the EOS flash and ANN flash is reduced by 85.96% on average as shown in Figure 5.16. For this case study we can conclude that the fugacity coefficients from ANNs can be used with successive substitutions with negligible variations on fluid property calculations at a lower computational cost.

5.1.2 Case 2

For the second study case, the number of components is increased to 10. This is a gas condensate (Al-Meshary, 2014) characterized by using the perturbation of n-alkanes method developed by Kumar (2016) and Kumar and Okuno (2016). The gas condensate fluid is composed of 6 pure components and 4 pseudo components, as described in Table 5.4. The reservoir temperature for this study case is 525 K, and the critical point is located at 425.30 K and 265.79 bars. This fluid model is closer to the critical point as compared with the first case, as shown in the phase diagram in Figure 5.17.

The generalization error for the ANNs for the fugacity coefficient is shown in t Table 5.5. The average percentage error for the ANN fugacity models goes from 0.0097% for the pseudo component number four to 0.1483% for ethane. Figures 5.18 to 5.22 show the accuracy comparison between the fugacity coefficients from ANN models and the EOS. As in the previous study case, the fugacity coefficients from ANNs have a low percentage error as compared to EOS fugacity coefficient. The advantage of using different ANNs for each component in the hydrocarbon mixture is that the average percentage error is kept low and the single fugacity coefficient predictions are more accurate. Therefore, when the different ANNs models are used together during iterations, we can predict accurate phase compositions and fluid properties.

The phase compositions using the conventional EOS and the ANN flash are compared in Figures 5.23 to 5.32. The results from the ANN flash overlap those calculated with the EOS flash. Fluid properties, such as vapor and liquid fraction, fluids' saturations and fluid densities, using the ANN flash and the EOS flash are compared in Figures 5.33 to 5.35, respectively. A good agreement between the ANN and EOS flash results is observed.

The efficiency of the ANN flash was evaluated by comparing the execution time to compute the fugacity coefficient using the conventional EOS method and that using the ANN method. The execution time of the fugacity coefficient by the use of ANNs is reduced by on average 96.56% of the time required by the original EOS for the 10 components in the mixture as shown in Figures 5.36 and 5.37. Table 5.7 presents the comparison of the time per iteration between the conventional method and the ANN flash. The ANN flash time per iteration is reduced by 81.58% with respect to the conventional method. Finally, the total flash time (the number of iterations multiplied by the time per iteration) of the ANN flash is reduced by 84.35% with respect to the EOS method as shown in Figure 5.39. The convergence behavior of the two methods is presented in Figure 5.40 for 220 bars where it is observed that the ANN flash follows similar convergence behavior as the EOS. For this case study, we can conclude that ANN flash can work independently of the number of components in the mixture with a higher miscibility degree while keeping high accuracy and faster calculations as compared with the conventional EOS flash

5.1.3 Case 3

The fluid model for this case study consists of 12 components with 2 non-hydrocarbon components (CO_2 and N_2) and contains non-zero BIPs. This fluid is a volatile oil (Al-Meshary, 2014) characterized by using the method, perturbation of n-alkanes, by Kumar and Okuno (2016). The critical point of this model is at 522.65 K and 324.22 bar. The reservoir temperature for this reservoir fluid model is 393.70 K, as shown in figure 5.41. The fluid model is described in table 5.8, and the BIPs are shown in Table 5.9.

In this case study, 12 ANNs were generated to predict the fugacity coefficient of each component in the mixture. Table 5.10 shows that the generalization error of each model for prediction is lower than 1.09%. As explain by Okuno (2017) the use of non –

zero BIPs increase the non – linearity of the fugacity coefficient in composition space. However, ANNs are capable to predict accurate fugacity coefficients using non – zero BIPs as shown in Figures 5.42 to 5.47. The phase compositions from the two flash methods are compared in Figures 5.48 to 5.59. The difference between the phase compositions with the two methods is insignificant for fluid property calculations (Fluid saturations, phase amount and fluid densities) as shown in Figures 5.60 to 5.62.

The fugacity coefficient execution time using the ANN model is 96.94% of that using the original EOS for 12 components in the mixture as shown in Figures 5.63 to 5.64. The time per iteration in the ANN flash is reduced by 94.01% with respect to the EOS flash as shown in Table 5.11. The total number of iterations needed to reach convergence between the two methods shown in Figures 5.65. It can be observed that the ANN flash converges with fewer iterations as compared with the EOS flash. Figure 5.66 shows the total flash time comparison (Number of iterations multiplied by the time per iteration) comparison between the two methods. The total flash time is reduced by 94. 66% with the use of ANN for the fugacity coefficient as compared with the EOS flash. Finally, Figure 5.67 presents the convergence behavior of the two methods where it is shown that the convergence behavior of the ANN flash is similar to the EOS flash.

For this case study we can conclude that ANN flash calculation can deal with fluid models including non-zero BIPs and non-hydrocarbon components such as CO_2 and N_2 resulting on accurate fluid properties calculations. The ANN architecture for all components was kept as the one used for the previous cases. That is, the universal approximation of the artificial neural networks can work even with complicated and non-continuous functions, like the fugacity coefficient, at a lower computational cost.

5.1.4 Case 4

EOS flash becomes more challenging near the critical region, where the physical properties of the liquid and vapor phase are sensitive to the thermodynamic conditions. The fluid model for this case is a near critical oil (Al-Meshary, 2014) characterized by using the methodology of perturbation from n-alkanes (Kumar and Okuno 2016). The model is composed of ten components: 6 pure components and 4 pseudo-components. The mixture shows a critical point at 477.03 bar at 276.49 K, and the reservoir temperature is 462 K. The fluid model is described in Table 5.12. Figure 5.68 shows the phase diagram of the mixture.

For this case, we generate 10 ANNs to model the fugacity coefficient of each component for a range of pressure from 150 bar to the bubble point pressure at 275.1 bar and with a uniformly spaced concentration distribution. The generalization error of each neural network model is shown in Table 5.13. The accuracy comparison is shown in Figures 5.69 to 5.73. The ANN fugacity coefficient can match on average 4 significant figures with the EOS fugacity coefficients.

The phase compositions of this case study are shown in Figures 5.74 to 5.83. The ANN flash model performs well, except for pressures close to the critical point. The total number of iterations needed to converge in this near-critical zone is more than 2000 with the conventional EOS flash. Figure 5.84 shows the comparison between the total number of iterations with the conventional EOS and that with the ANN flash calculations. Figure 5.85 shows the convergence behavior comparison between the two methods at 272 bar at which the ANN flash converges to an incorrect solution. Figure 5.86 shows the comparison at 180 bars, at which both methods converge to the correct solution. The ANN flash converges to an incorrect solution at pressures above 267 bars. Figures 5.87 to 5.89 show

the impact of the ANN flash miscalculation in reservoir fluid properties: vapor and liquid fraction, fluid saturation, and densities, respectively.

This miscalculations near the critical region is generated by the effect of the ANN fugacity coefficients on the RR function. Near the critical region, the gradient of the RR functions tends to 0. Therefore, the vapor fraction calculation become very sensitive to the accuracy of the K values calculated from ANN fugacity coefficients. This small variations on the K values affect the solution of the RR function leading to an incorrect vapor fractions calculation approaching the critical point.

Therefore, a switching criterion based on the gradient of the RR functions was implemented. The switching criterion will indicate when the ANN flash will stop to calculate the fugacity coefficients using the ANNs models and use the EOS based fugacity coefficients instead. For this case study, the switching criterion was set for the gradient of the RR function to be 0.05. When the gradient of the RR function is smaller than 0.05, the ANN flash is switched to the EOS for the fugacity coefficients. Results of the implementation of this switching criterion are shown in Figures 5.97 to 5.99. It can be observed that with the implementation of the switching criterion, the ANN flash converges to the correct solution near the critical point, and the fluid properties are calculated correctly. Note that this switching, if it occurs, does not start over the flash calculation using the EOS; hence, the calculations using ANNs prior to the switching are not wasted.

The fugacity coefficient execution time comparison between the ANN flash and the EOS method is shown in Figures 5.90 and 5.91. The fugacity coefficient execution time for 10 components using ANNs is reduced by 96.63% as compared with the EOS method.

The time per iteration comparison between the two methods is presented in Table 5.14. The ANN method shows a time reduction of 93.806% per iteration with respect to the EOS method. Finally, the total ANN flash calculation time (Number of iterations times

time per iteration) is reduced by 94.34% on average with fewer iterations to reach solution as compared with the EOS flash shown in Figure 5.92.

5.2 DISCUSSION

The implementation of ANNs to calculate the fugacity coefficient in different fluid models demonstrates that the method performs successfully for different fluid types: fluid models with a large number of components, non – zero BIPs, and relatively complex compositions with CO_2. The advantage of using ANNs is that they can calculate the fugacity coefficient with a low generalization error using simple neural network architectures that does not require a large number of neurons in the hidden layers. As a result, the training of the ANN models becomes simple, and also the risk of overfitting is avoided. ANNs were combined with the successive substitution algorithm to solve for the phase compositions and amounts in flash calculations.

The first case study indicates that the ANN fluid model implemented with successive substitutions can predict phase compositions and phase amounts with the same accuracy as the conventional EOS while reducing the total flash calculation time by 85.96%. This reduction in computational time comes from the ANN-based fugacity coefficients that require a smaller number of operations than the EOS-based ones. The ANN-based flash is EOS-free during the iteration; it solves the EOS for compressibility factors only at the final convergence when phase saturations are computed. This means that the complex operations involved in the fugacity coefficient calculation (e.g. logarithms, roots, exp) are replaced by computationally efficient calculations involved in the ANN-based feedforward calculations (additions and multiplications). ANN flash becomes more advantageous over EOS flash for a larger number of components because the difference between the two approaches becomes more significant with increasing number of components in the

mixture. It is expected that the advantage of ANN flash over EOS flash increases with increasing number of grid- blocks in reservoir simulations because the number of flash calculations increases. Note that the accuracy of fluid phase behavior representation is not reduced by the ANN flash. When the ANN flash may converge to an inaccurate solution in a near-critical region, EOS flash can be activated on the fly during the ANN flash calculation when a small gradient of the RR function is detected.

In the third case study, we analyzed the capability of the ANN flash with mixtures with more components and the use of non – zero BIPs. Results from this case study indicated that the application of the ANN-based flash was not affected by non – zero BIPs. The use of ANNs could handle the non-linearity introduced by non-zero BIPs with no modification of the ANN architecture (e.g. number of hidden neuros and number of hidden layers). The same accuracy was kept as the conventional EOS flash with an improvement of 94. 66% in computational time.

Finally, the ANN flash was tested for a volatile oil near the critical point. Fluids near the critical point are challenging and require the conventional EOS flash to take more iterations to converge to the solution. Also, convergence issues of flash calculations often arise in near-critical regions as presented in the literature. The ANN flash calculates the phase compositions and phase amounts accurately for most of the pressure ranges in the case studies. However, when the pressure is very close to a critical point, the ANN flash may deviate from the correct solution. Near a critical point, the gradient of the Rachford – Rice (RR) function tends to zero around the solution. The ANN-based fugacity coefficient is an approximation of an equation of state. Small variations in the K values using the ANN fugacity coefficient have a greater impact on the RR function near the critical point, resulting in an incorrect RR solution. Figure 5.93 shows the vapor fraction solutions from

the EOS and ANN flash when the gradient of the RR tends to zero in the critical region for case 4 at 272 bars.

The incorrect ANN flash calculations arose when the gradient of the RR function was smaller than 0.05179 for case 4. As shown in Figures 5.94 and 5.95, the gradient tends to zero when it gets close to a critical point as is the case with the original EOS used. In practical applications, the gradient of the RR function is calculated as part of the flash calculation and will indicate whether to use EOS or ANN flash. That is, it is possible to set a switching criterion as a minimum gradient of the RR function. Since the gradient of the RR is part of the solution method, no additional calculations are needed. Finally, figure 5.96 shows the deviation of the RR solution near the critical region at a pressure of 272 bar (Close to the critical point) for case 4. Phase diagrams for the four study cases were generated with PVTsim and fluid properties, phase concentrations, and fugacity coefficients using the code developed in this work were verified with PVTsim.

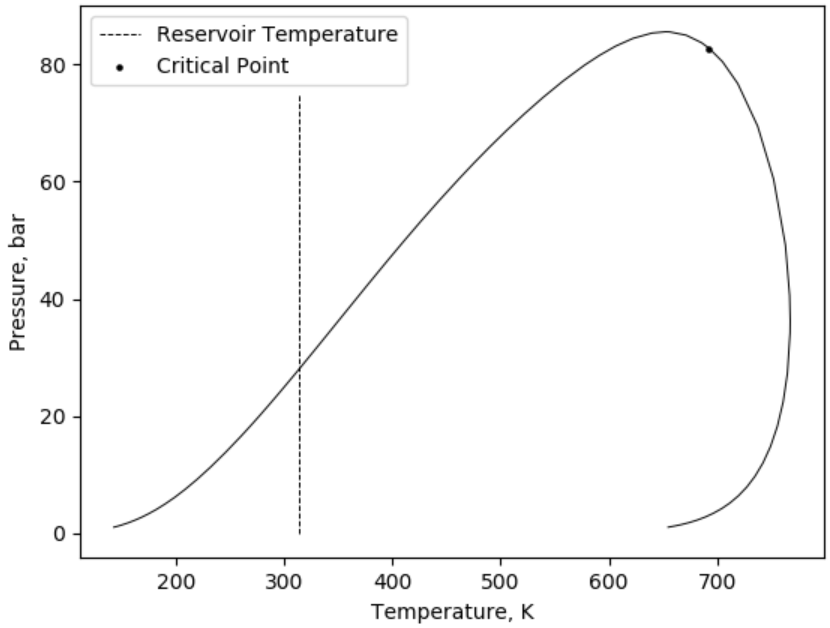


Figure 5.1. Phase diagram of adapted BSB west Texas oil fluid model.

Table 5.1. Fluid properties for case 1. BIPs are all zero.

Component	Overall composition	Molecular Weight, g/mol	Pc, Bar	Tc, K	Acentric Factor
C_1	0.0917	16.04	46.0017	190.6000	0.0080
C_{2-3}	0.1559	37.2	44.9923	344.2056	0.1310
C_{4-6}	0.1727	69.5	33.9959	463.2222	0.2400
C_{7-15}	0.3360	140.96	21.7487	605.7500	0.6180
C_{16-27}	0.1667	280.99	16.5404	751.0167	0.9570
C_{28+}	0.0769	519.62	16.4177	942.4778	1.2680

Table 5.2. Keras setup to train neural networks.

Optimizer	Adam
Loss function	Mean square error
Epochs	20
Batch size	100

Table 5.3. Generalization error of fugacity coefficient from ANNs.

Component	Mean square error (MSE)	Average percentage error (MAE)	R^2
C_1	6.70850106051E-06	0.308447055769539	0.999997519525688
C_{2-3}	3.848290256829E-06	0.320329972589314	0.999992396154579
C_{4-6}	3.430544575051E-06	0.634021871766337	0.999996884179634
C_{7-15}	0. 81966110180E-06	0.308447055769539	0.999999401780076
C_{16-27}	2.615210785704E-5	0.436224573336516	0.999999593942976
C_{28+}	0.000210430013068	0.955529562436639	0.999999064147138

Table 5.4. Time per iteration comparison for study case 1.

Time per iteration, msec	EOS Flash	ANN Flash
	11.51606089	1.135802257

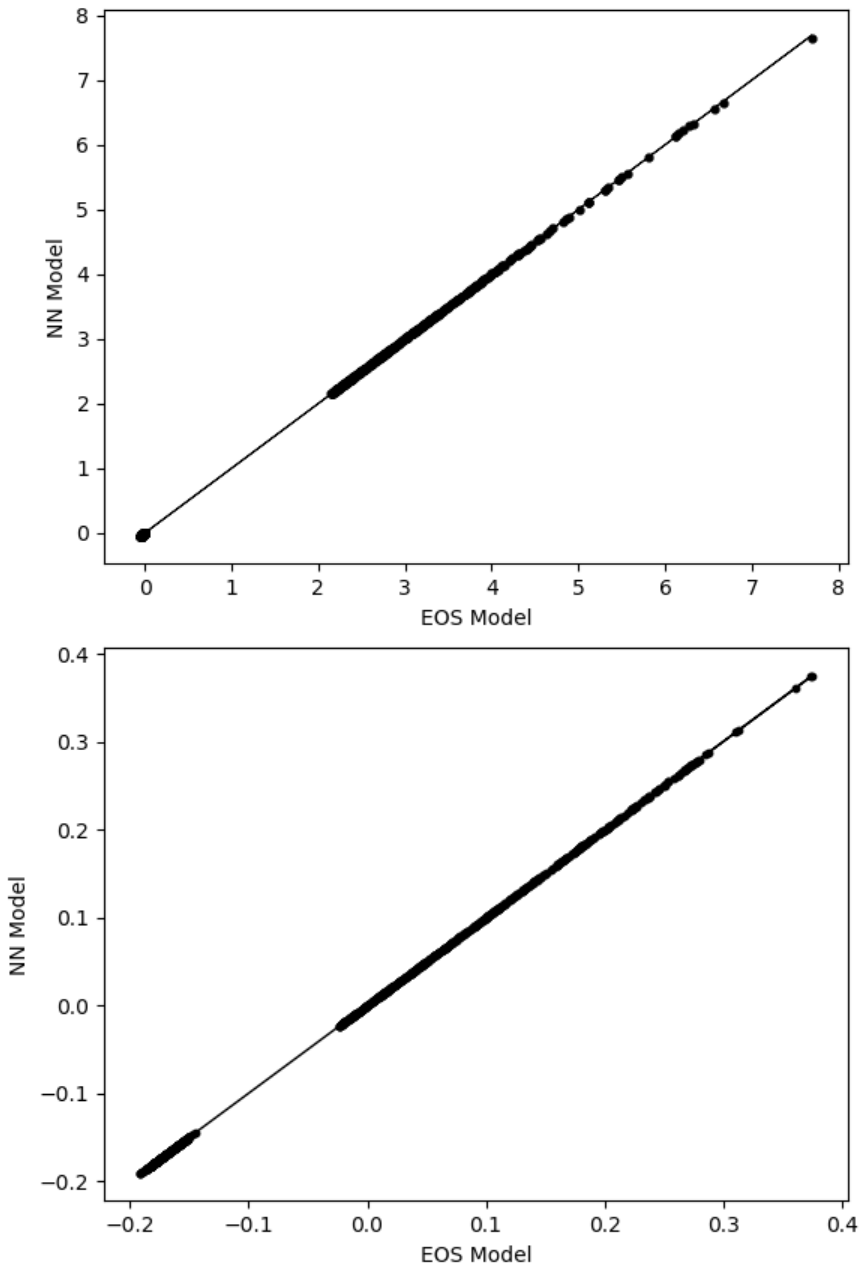
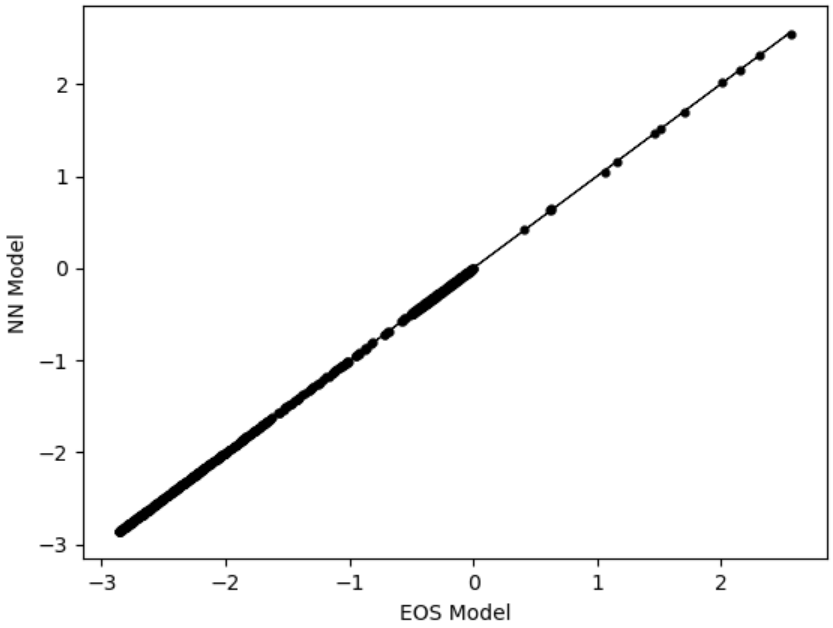


Figure 5.2 Accuracy comparison between fugacity coefficient calculated with EOS and ANN.
Top: Component C_1. Bottom: Pseudo-Component $C_{2\text{-}3}$.



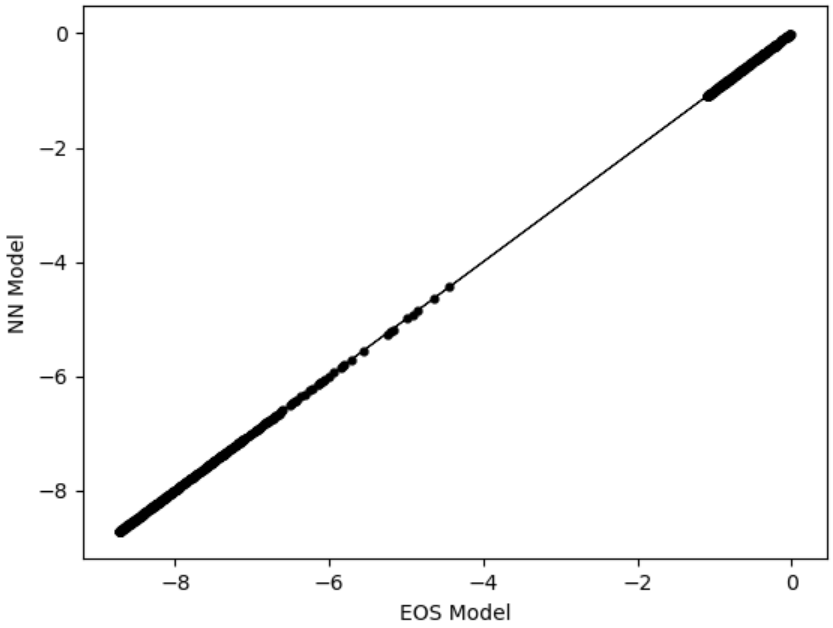
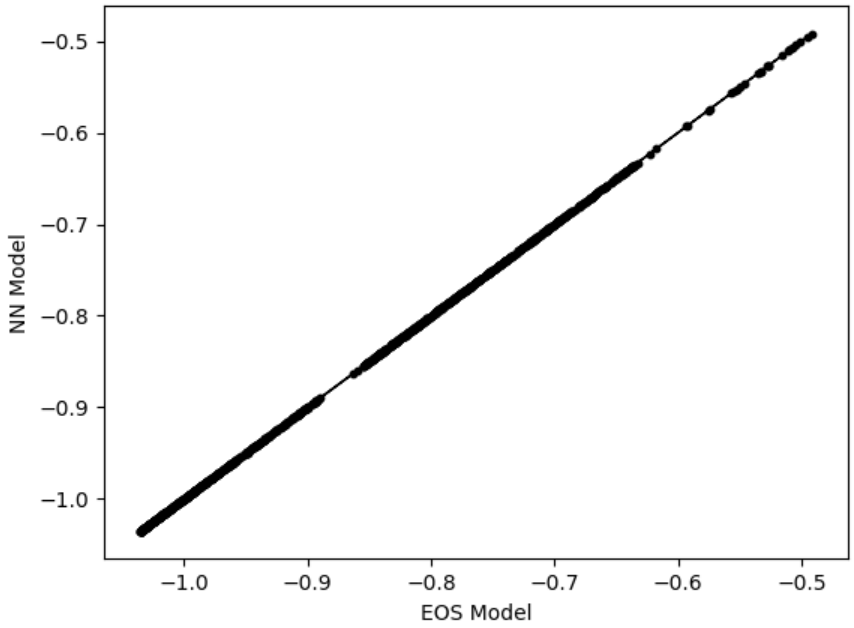


Figure 5.3 Accuracy comparison between fugacity coefficient calculated with EOS and ANN.
Top: Pseudo-Component $C_{4\text{-}6.}$ Bottom: Pseudo-Component $C_{7\text{-}15.}$



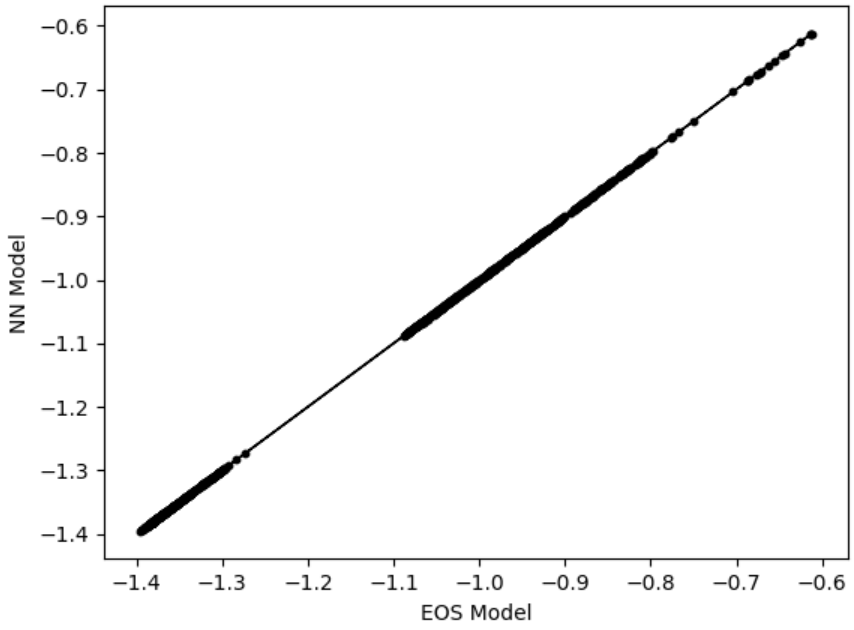


Figure 5.4 Accuracy comparison between fugacity coefficient calculated with EOS and ANN.
Top: Pseudo-Component $C_{16\text{-}28}$. Bottom: Pseudo-Component C_{28+}.

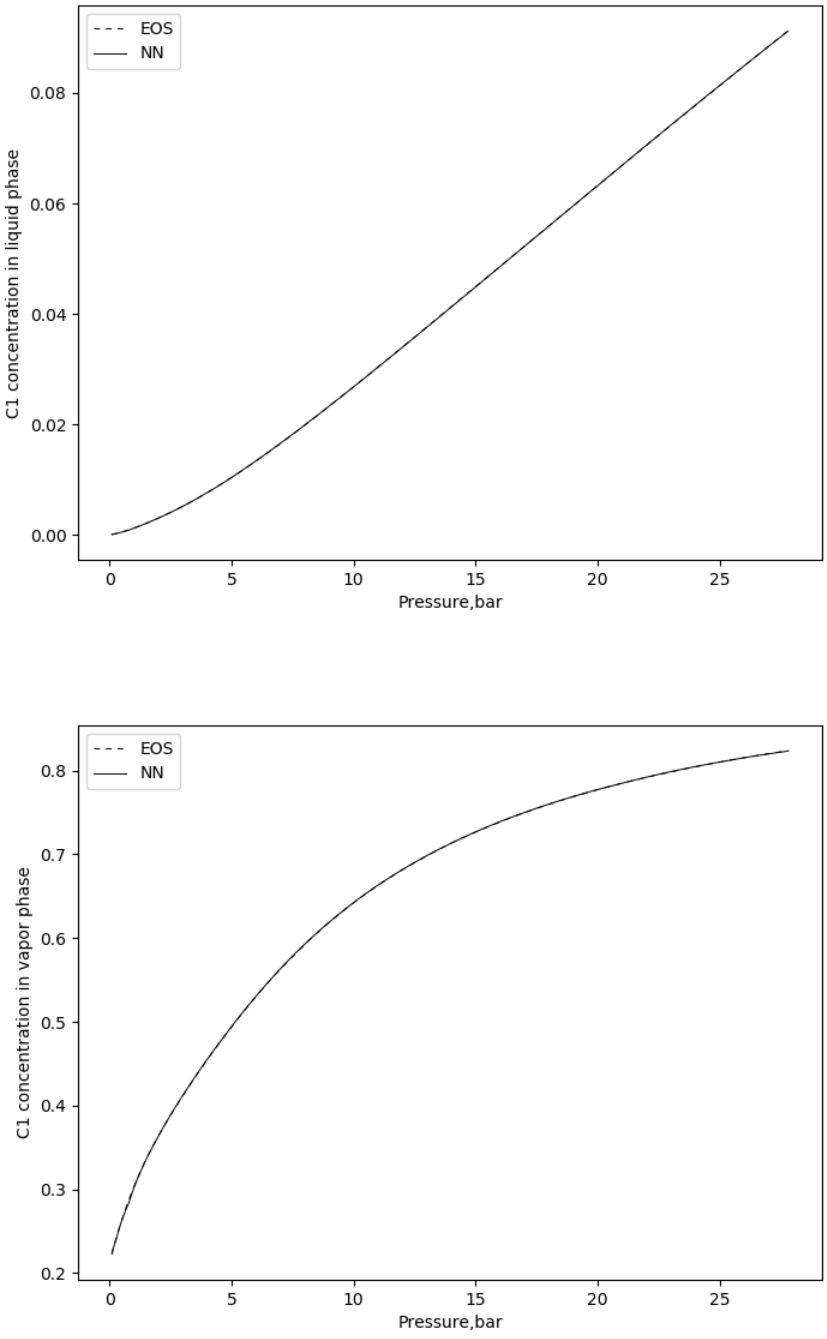


Figure 5.5 Phase mole fraction calculations comparison between EOS and ANN. Top: C_1 liquid concentration. Bottom: C_1 vapor concentration.

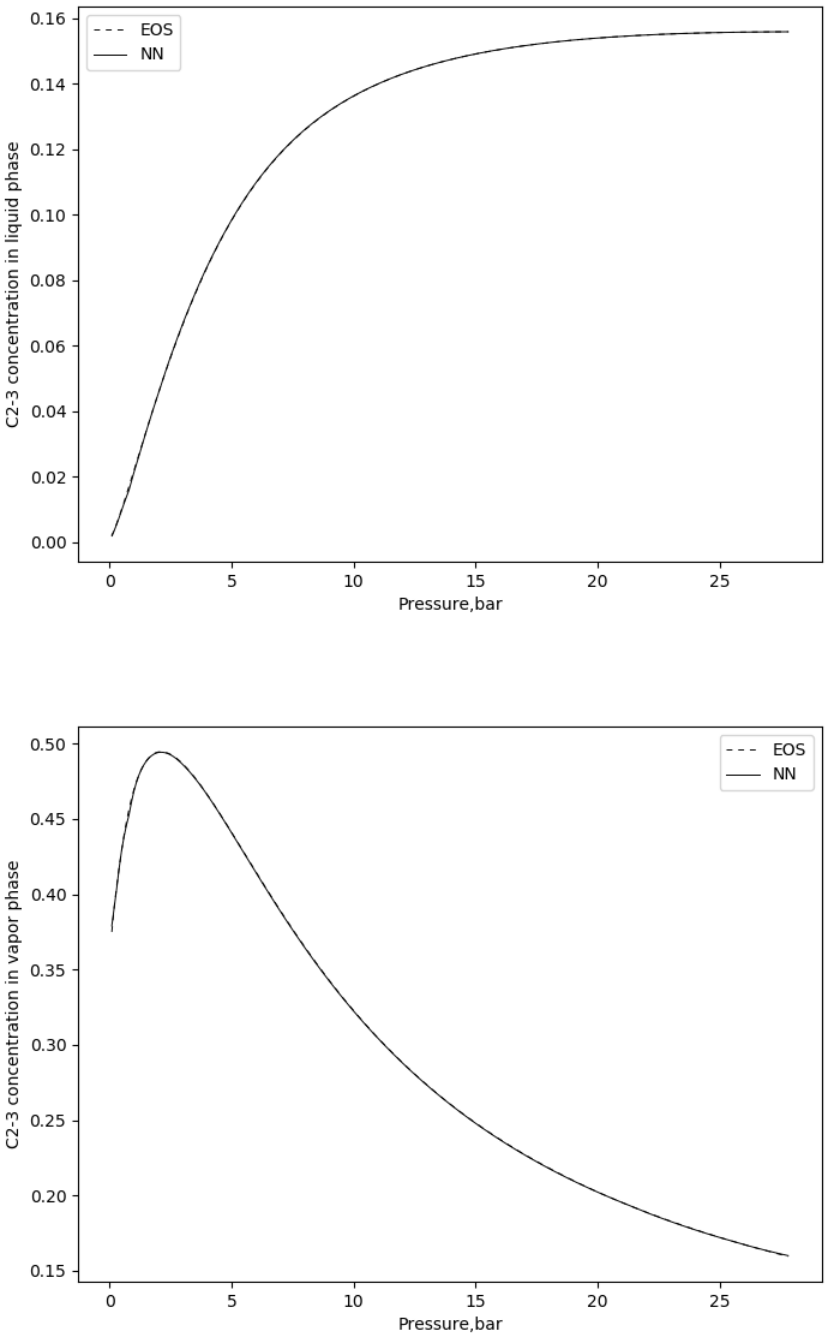
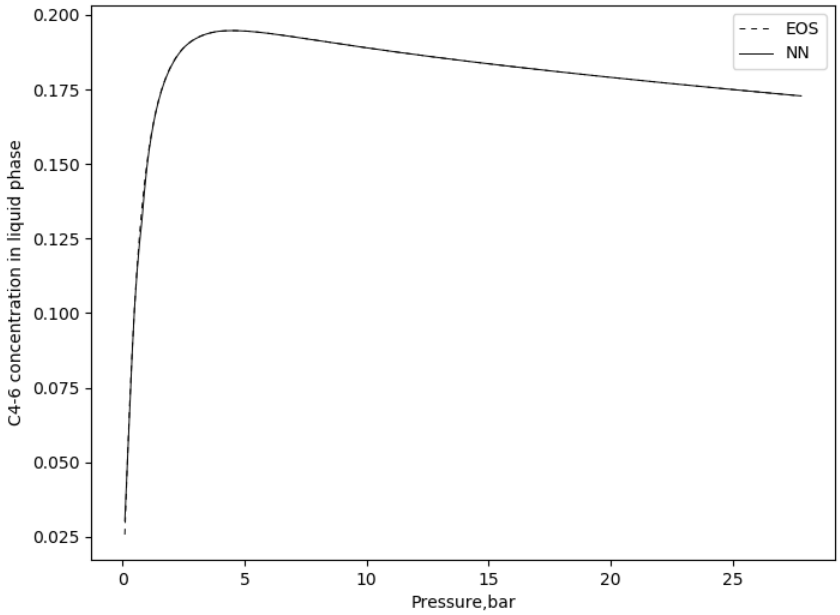


Figure 5.6 Phase mole fraction calculations comparison between EOS and ANN Top: Liquid phase for $C_{2\text{-}3.}$ Bottom: Vapor phase for $C_{2\text{-}3}$



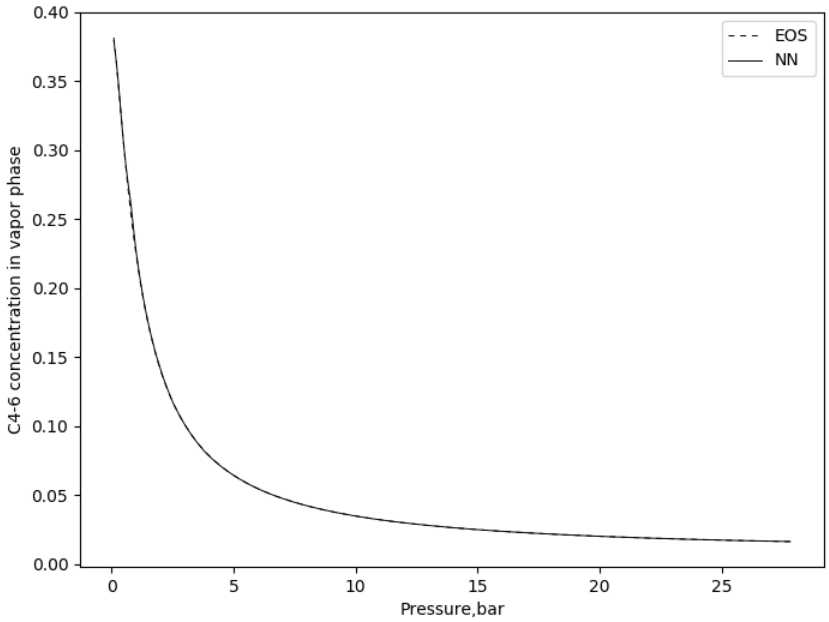


Figure 5.7 Phase mole fraction calculations comparison between EOS and ANN. Top: Liquid phase for $C_{4\text{-}6.}$ Bottom: Vapor phase for $C_{4\text{-}6.}$

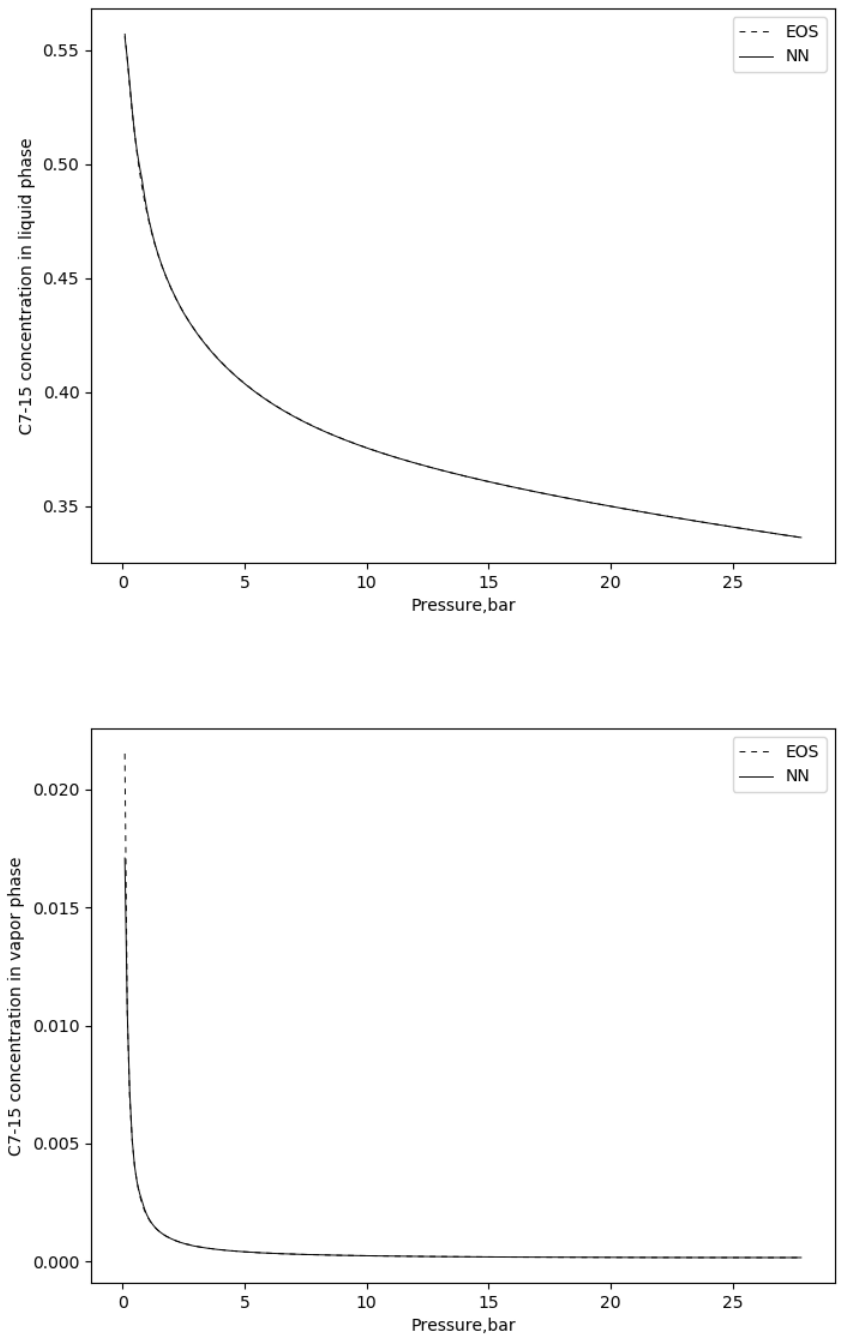


Figure 5.8 Phase mole fraction calculations comparison between EOS and ANN. Top: Liquid phase for $C_{7\text{-}15.}$ Bottom: Vapor phase for $C_{7\text{-}15.}$

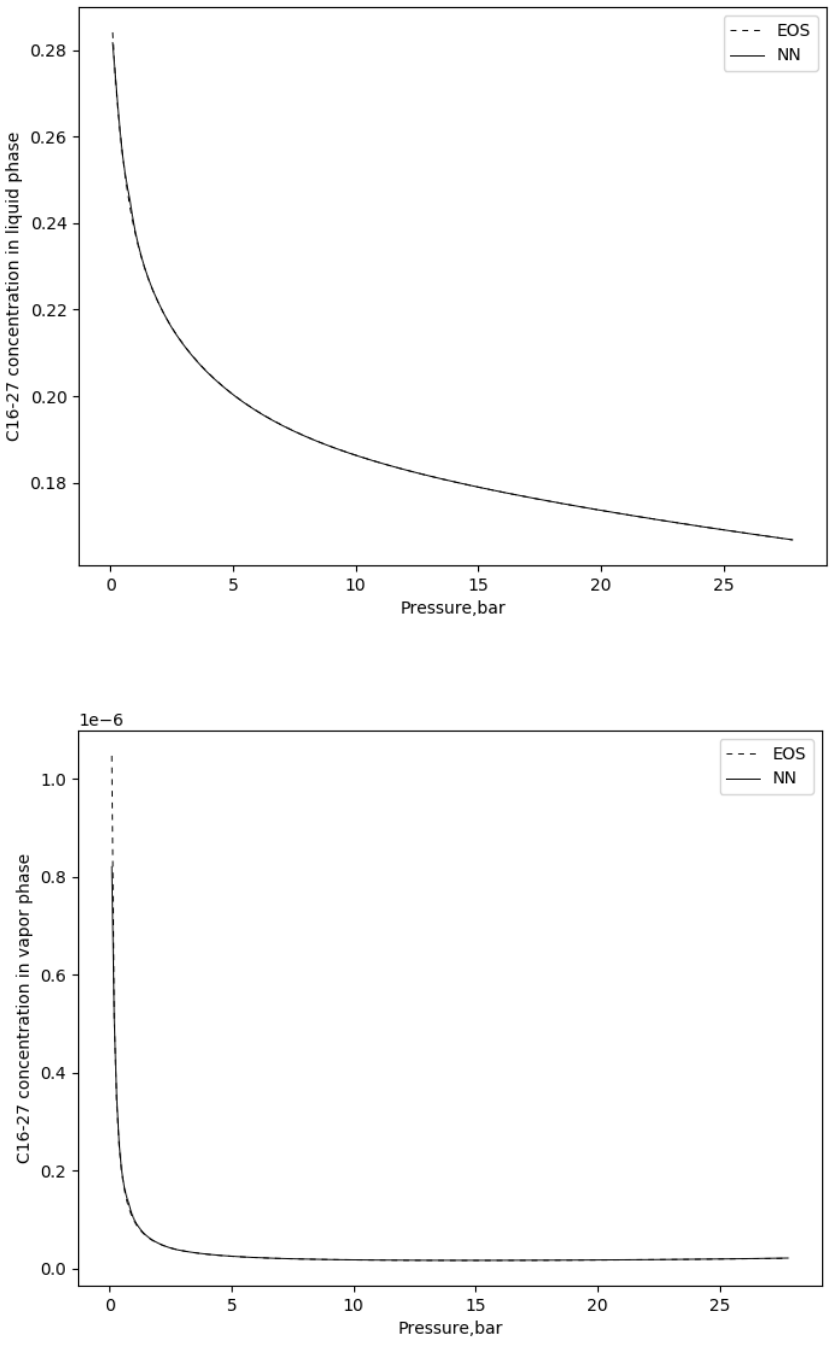


Figure 5.9 Phase mole fraction calculations comparison between EOS and ANN. Top: Liquid phase for $C_{16\text{-}27.}$ Bottom: Vapor phase for $C_{16\text{-}27.}$

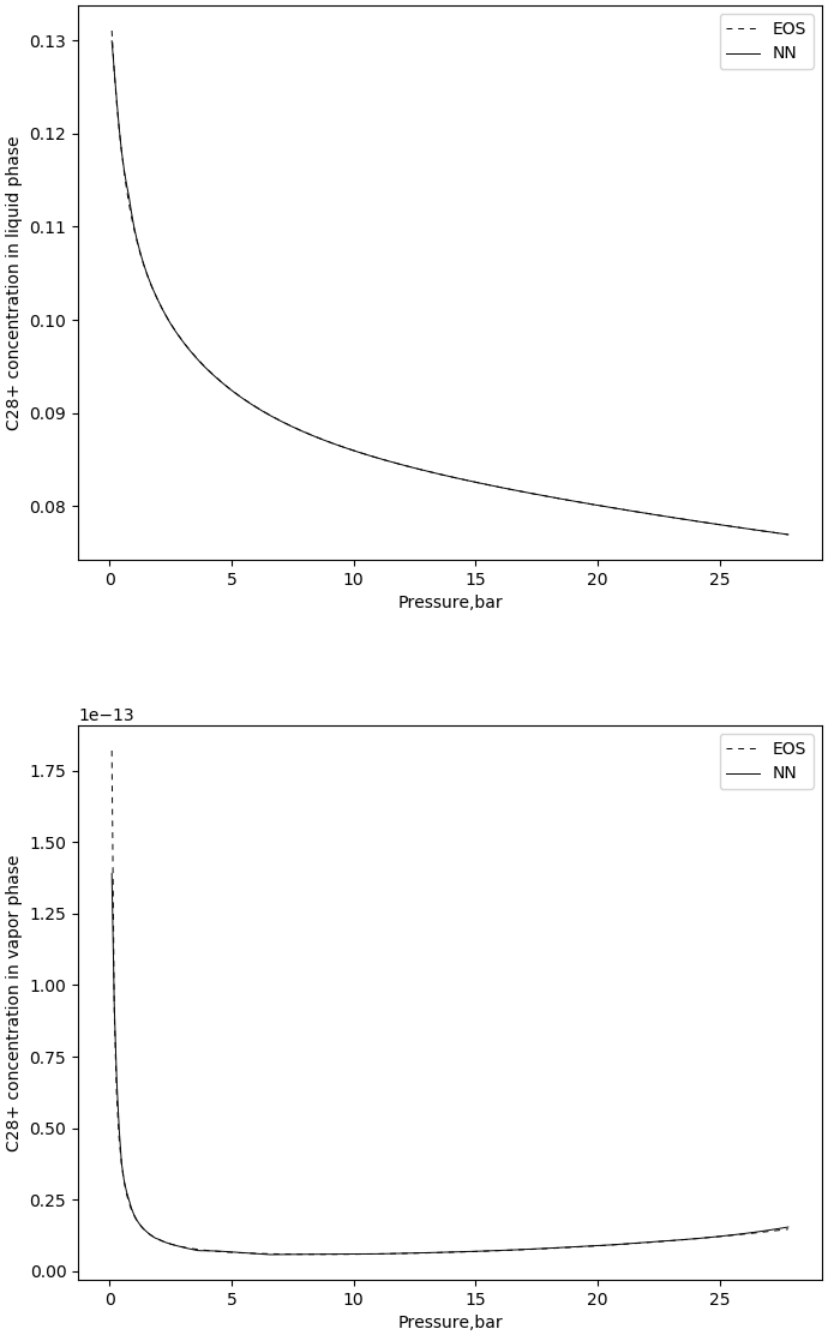


Figure 5.10 Phase mole fraction calculations comparison between EOS and ANN. Top: Liquid phase for C_{28+}. Bottom: Vapor phase for C_{28+}.

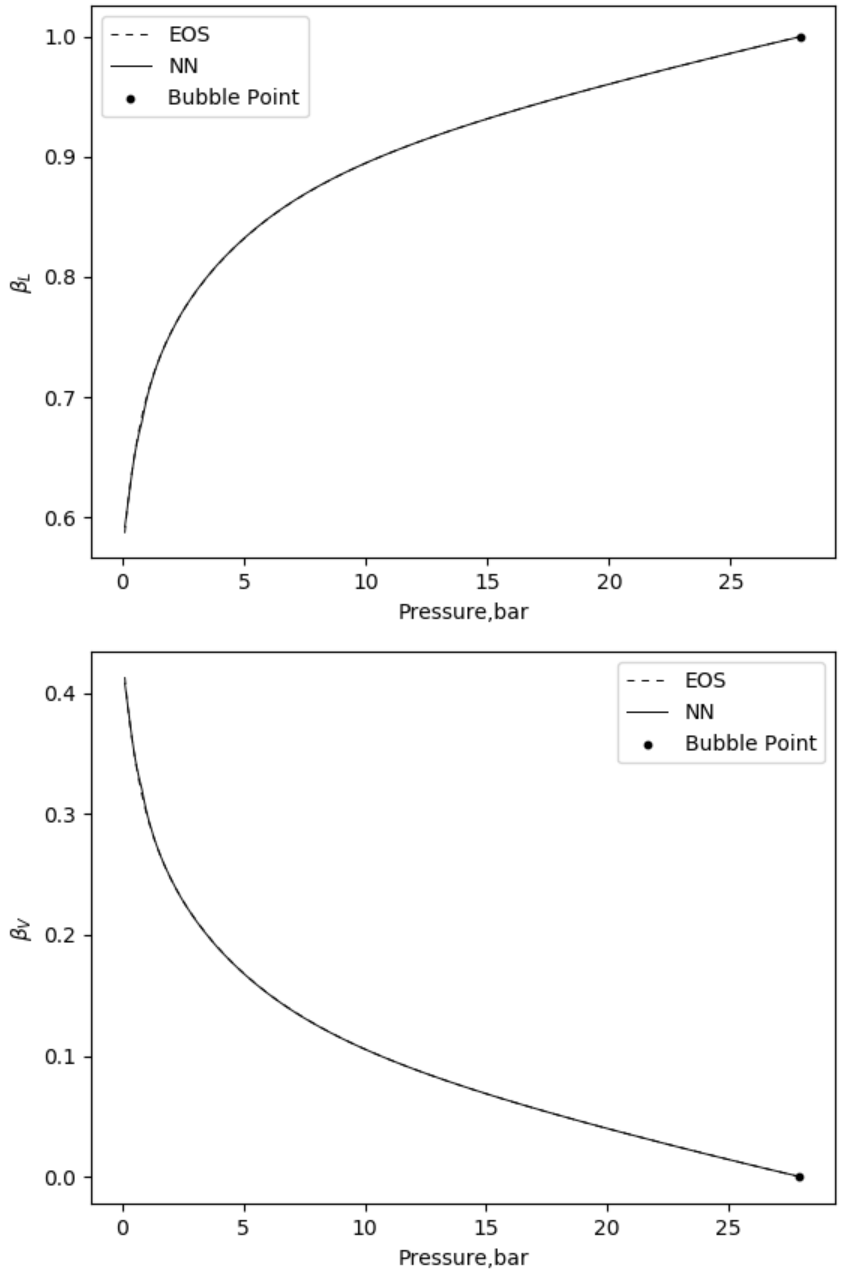
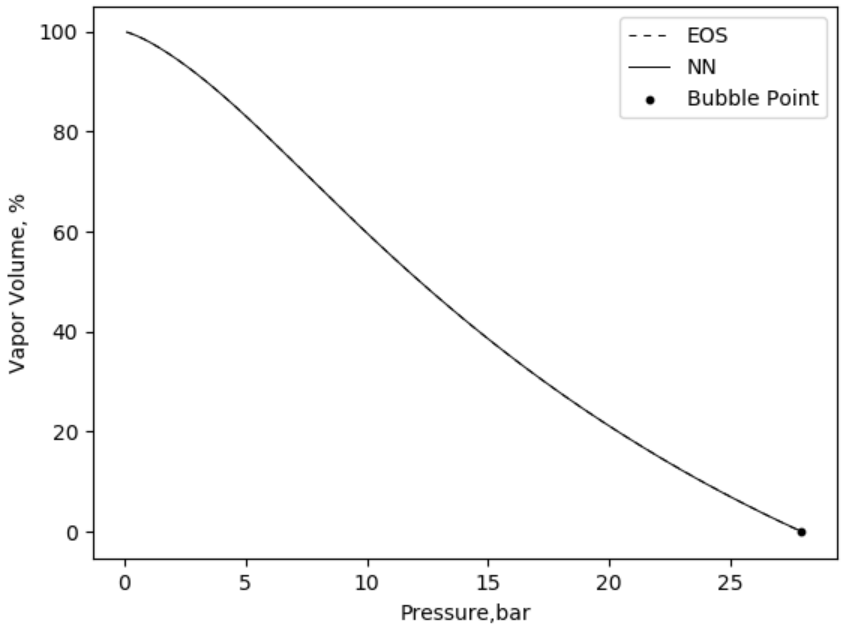


Figure 5.11 Phase mole fraction calculation comparison between EOS and ANN. Top: Liquid mole fraction. Bottom: Vapor mole fraction.



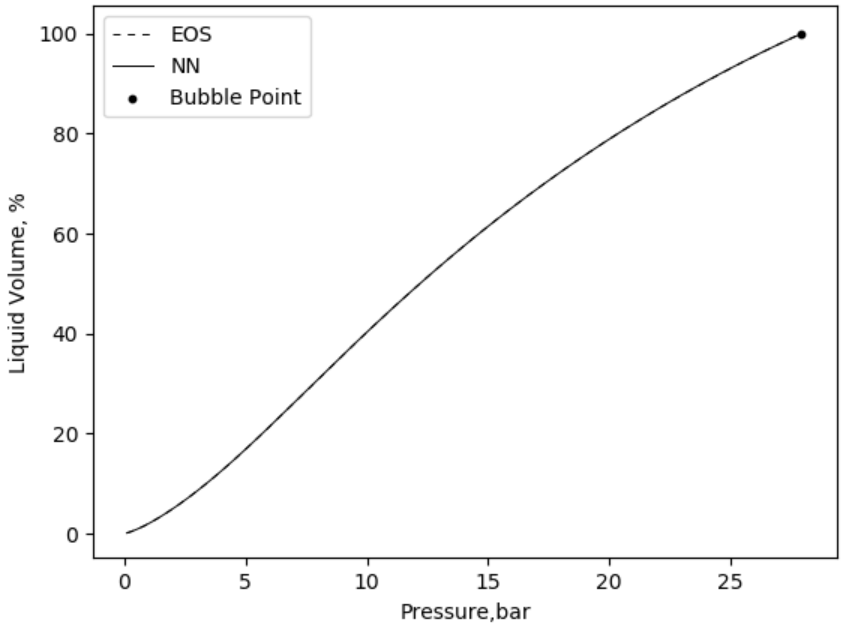


Figure 5.12 Fluid saturation comparison between EOS and ANN. Top: Vapor saturation. Bottom: Liquid Saturation.

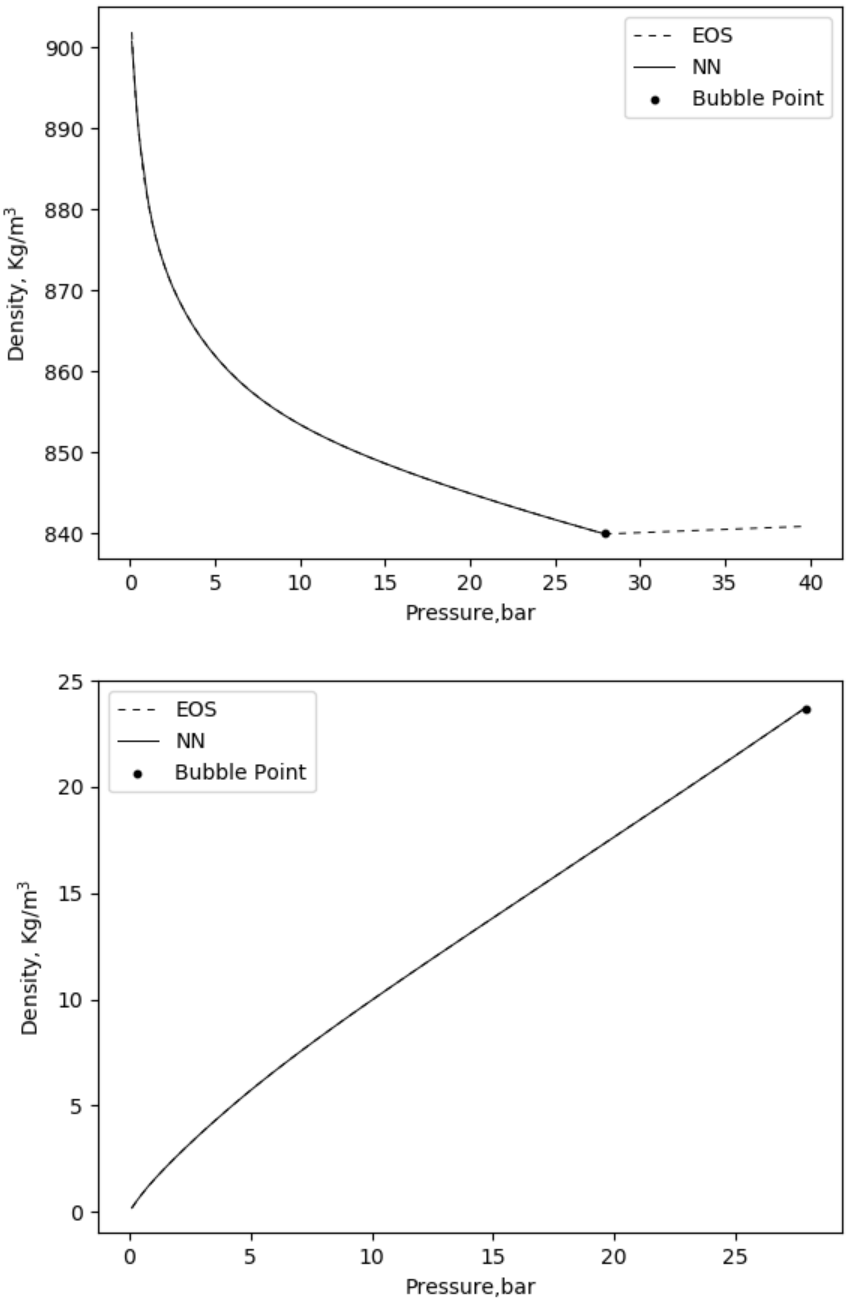


Figure 5.13 Fluid density calculation comparison between EOS and ANN. Top: Liquid density. Bottom: Vapor density.

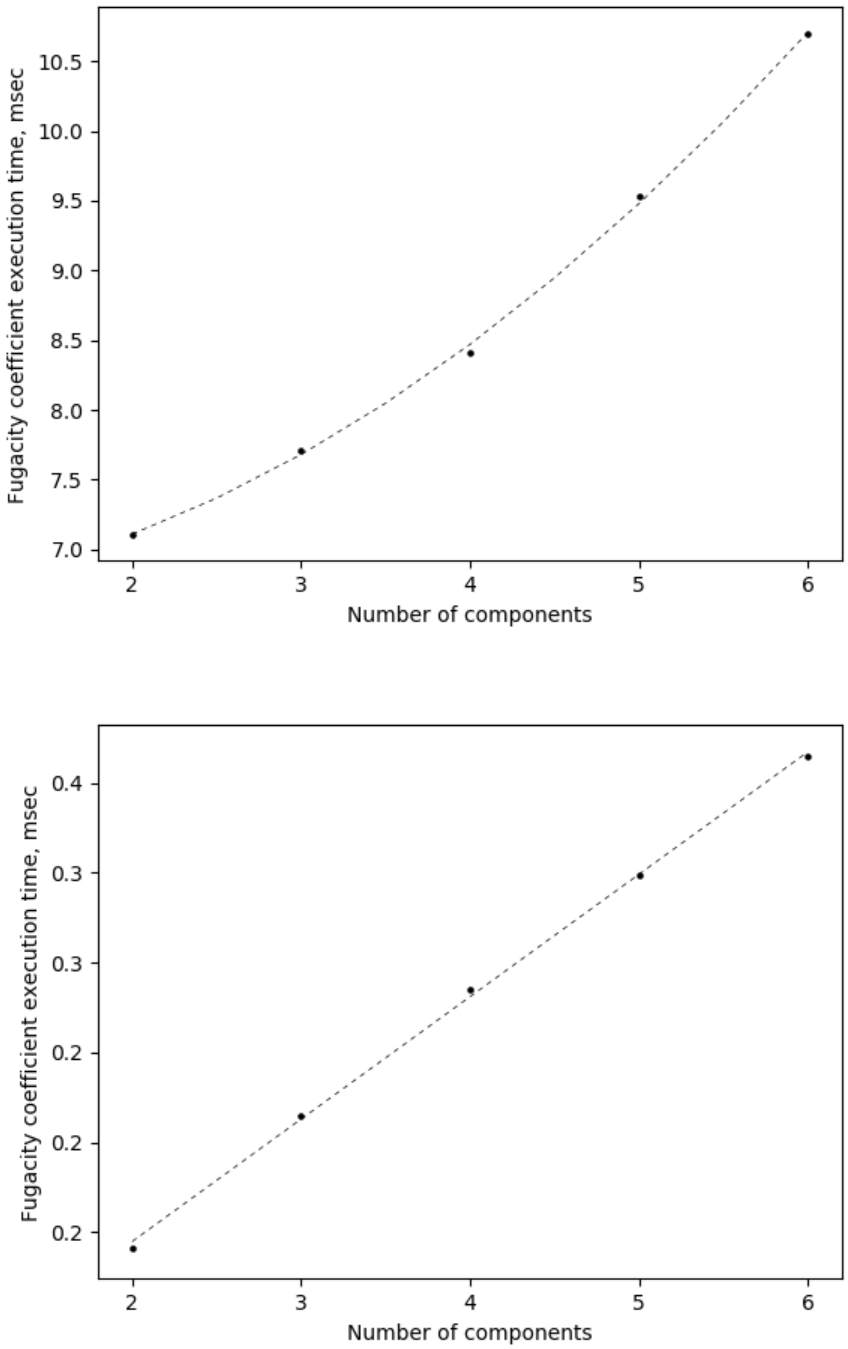


Figure 5.14 Fugacity coefficient execution time.
Top: EOS. Bottom: ANNs.

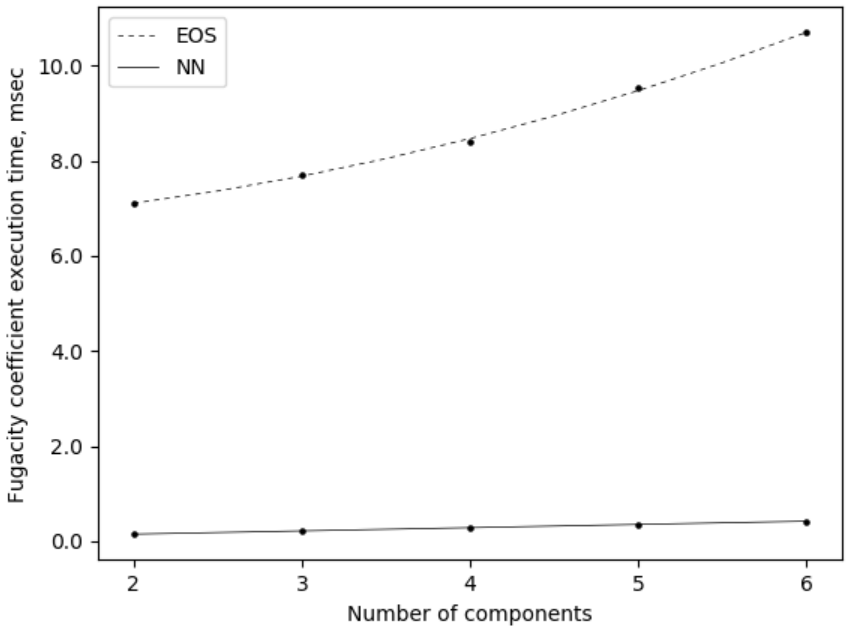


Figure 5.15 Fugacity coefficient execution time comparison.

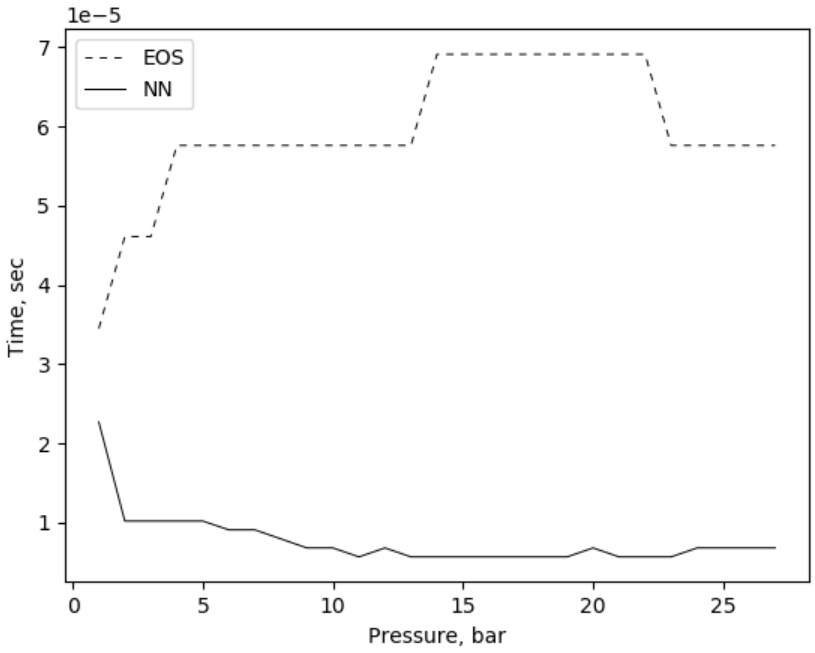


Figure 5.16 Flash calculation execution time comparison.

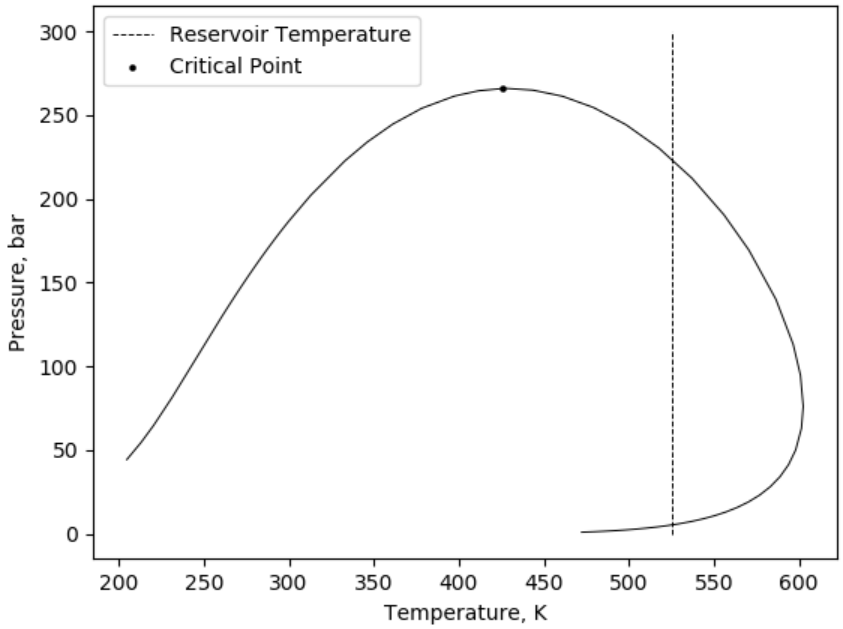


Figure 5.17. Gas Condensate phase envelope.

Table 5.5 Fluid properties for case 2.

Component	Overall composition	Molecular Weight, g/mol	Pc, Bar	Tc, K	Acentric Factor
C1	0.617632733	16.04	190.6	46	0.008
C2	0.113513628	30.07	305.4	48.84	0.098
C3	0.063427857	44.1	369.8	42.46	0.152
C4	0.036531905	58.12	419.95	37.53	0.1878
C5	0.019483682	72.15	465.25	33.79	0.2397
C6	0.018213007	86.18	507.4	29.69	0.296
PC-1	0.047967979	108.21	594.94	30.51	0.1519
PC-2	0.037379021	138.84	646.31	27.41	0.1971
PC-3	0.028272517	183.71	715.87	24.05	0.265
PC-4	0.01757767	294.82	878.5	18.96	0.4391

Table 5.6. Generalization error of fugacity coefficient from ANNs.

Component	Mean Square Error	Average percentage error	R^2
C_1	2.54311325475e-08	0.07234457797421	0.999999665740048
C_2	2.66136957059e-08	1.148322228690716	0.999998801603921
C_3	3.1167560168e-08	0.044787534277102	0.999993897126990
C_4	1.37826100055e-08	0.015488466508183	0.999996900515374
C_5	1.91603159707e-08	0.009869041485957	0.999998906088491
C_6	4.50596740515e-08	0.013688432331599	0.999999012191636
PC-1	1.0675432431e-07	0.015403556687481	0.999999285755196
PC-2	7.9619158355e-08	0.00977703684872	0.99999968709666
PC-3	4.58442675124e-07	0.021012046757658	0.999999025705854
PC-4	5.6781995651e-07	0.013882962556325	0.999999622923520

Table 5.7. Time per iteration comparison between EOS Flash and ANN Flash.

Time per iteration, msec	EOS Flash	ANN Flash
	16.3719777073514	3.01555897189149

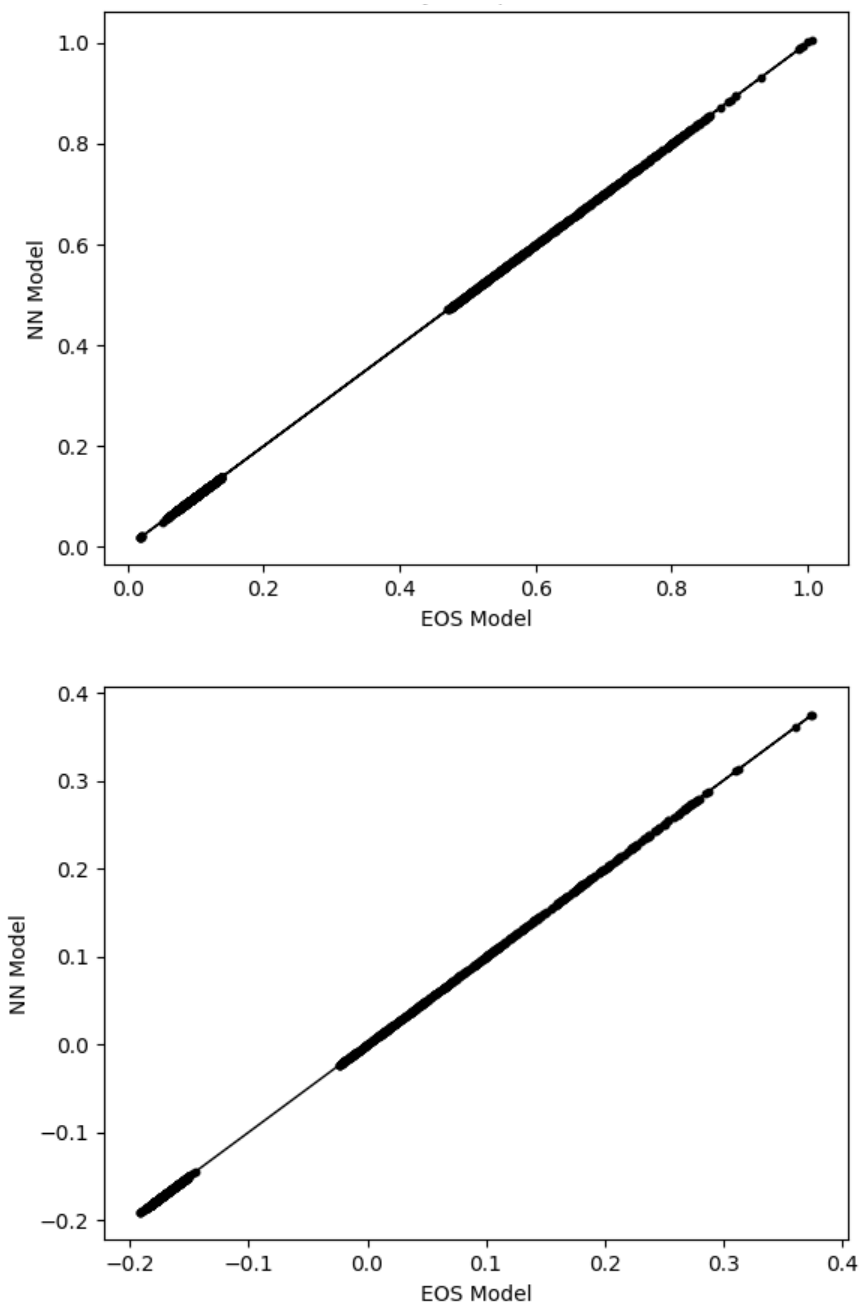


Figure 5.18 Accuracy comparison between fugacity coefficient calculated with EOS and ANN.
Top: Component C_1. Bottom: Component C_2.

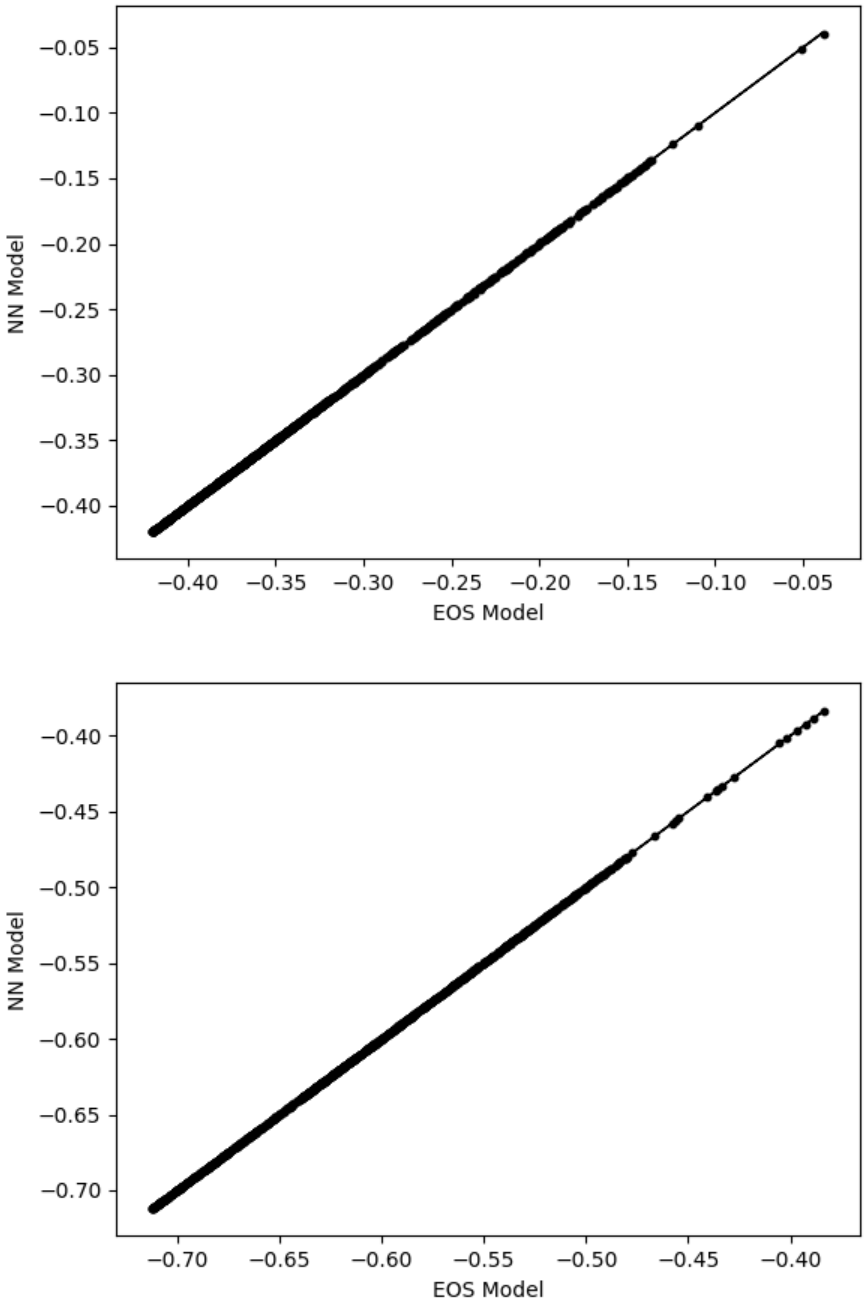


Figure 5.19 Accuracy comparison between fugacity coefficient calculated with EOS and ANN.
Top: Component C_3. Bottom: Component C_4.

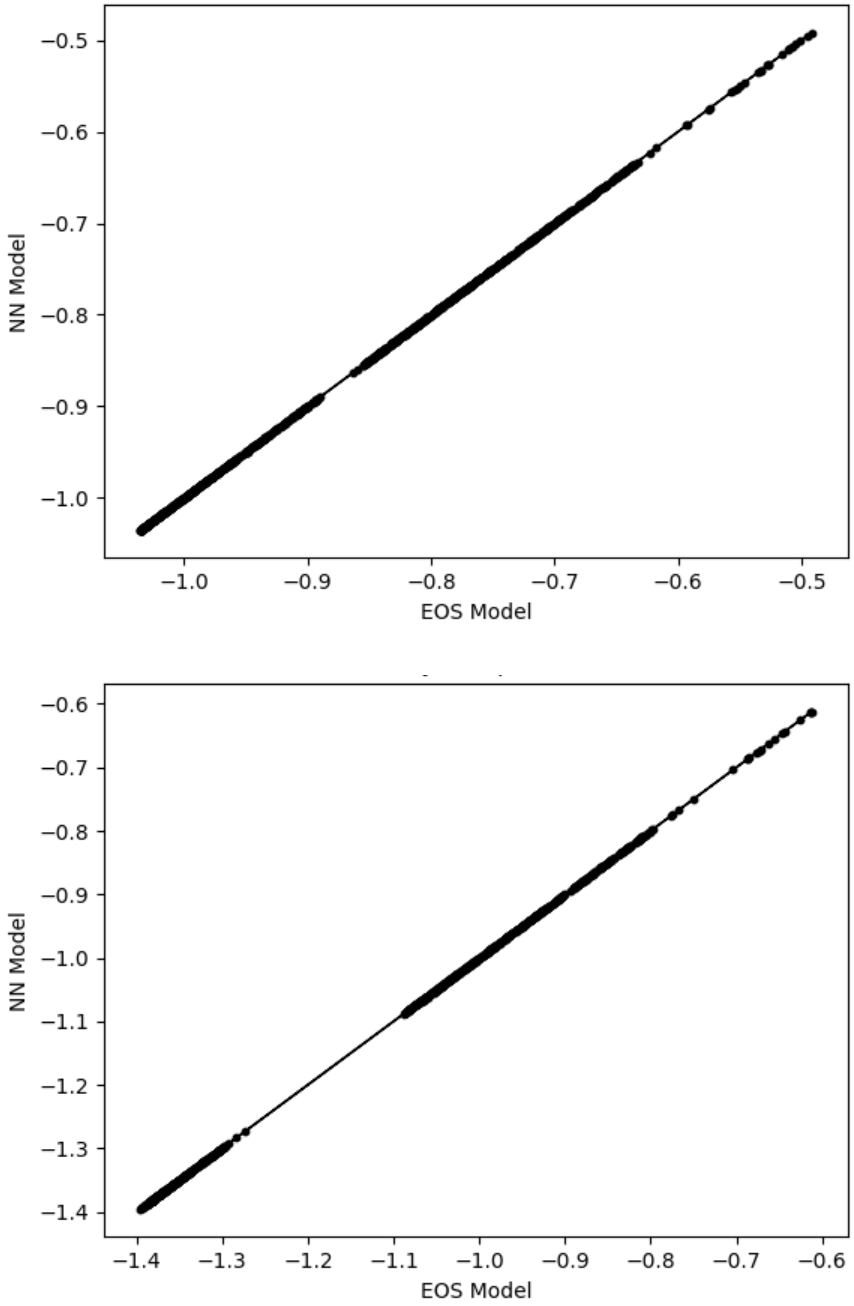


Figure 5.20 Accuracy comparison between fugacity coefficient calculated with EOS and ANN.
Top: Component C_5. Bottom: Component C_6.

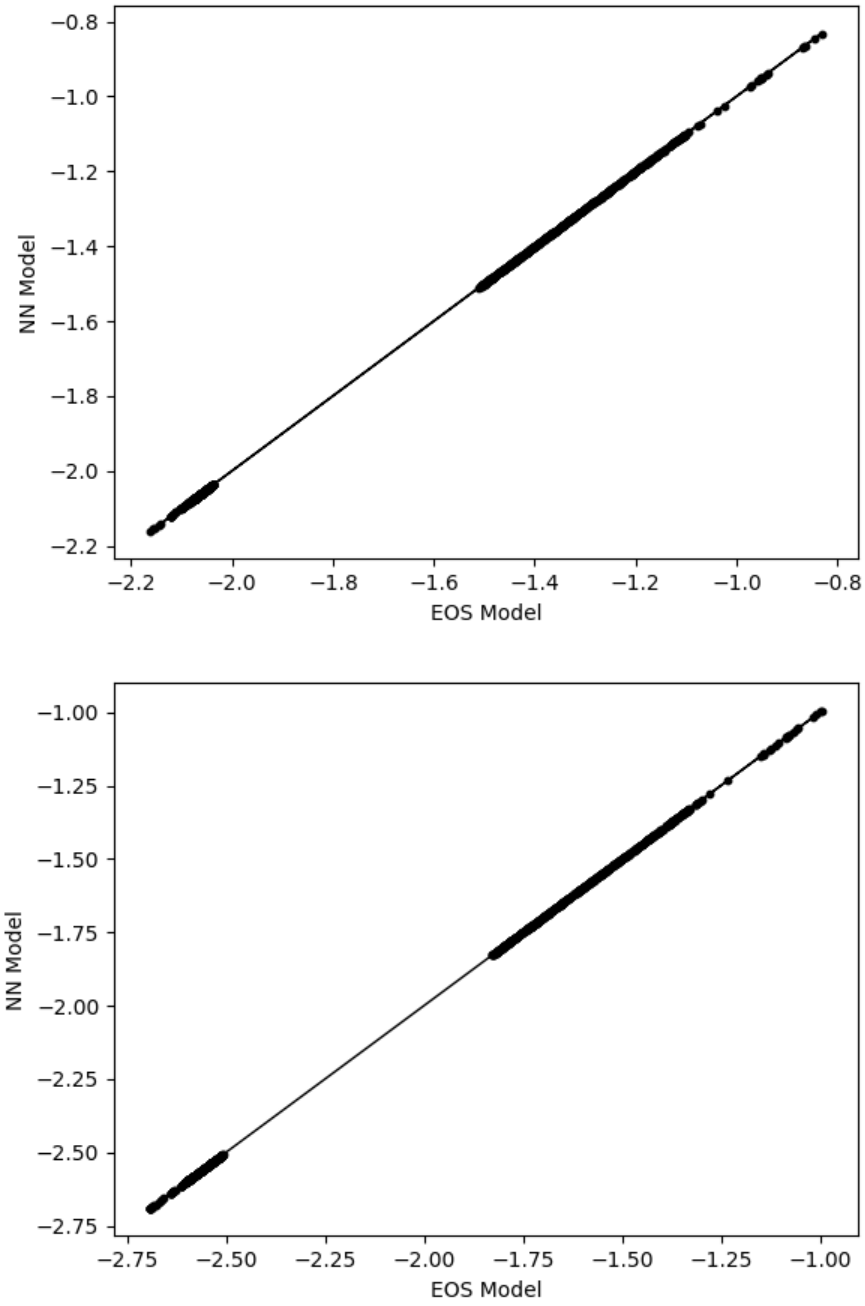


Figure 5.21 Accuracy comparison between fugacity coefficient calculated with EOS and ANN.
Top: Pseudo-Component 1. Bottom: Pseudo-Component 2.

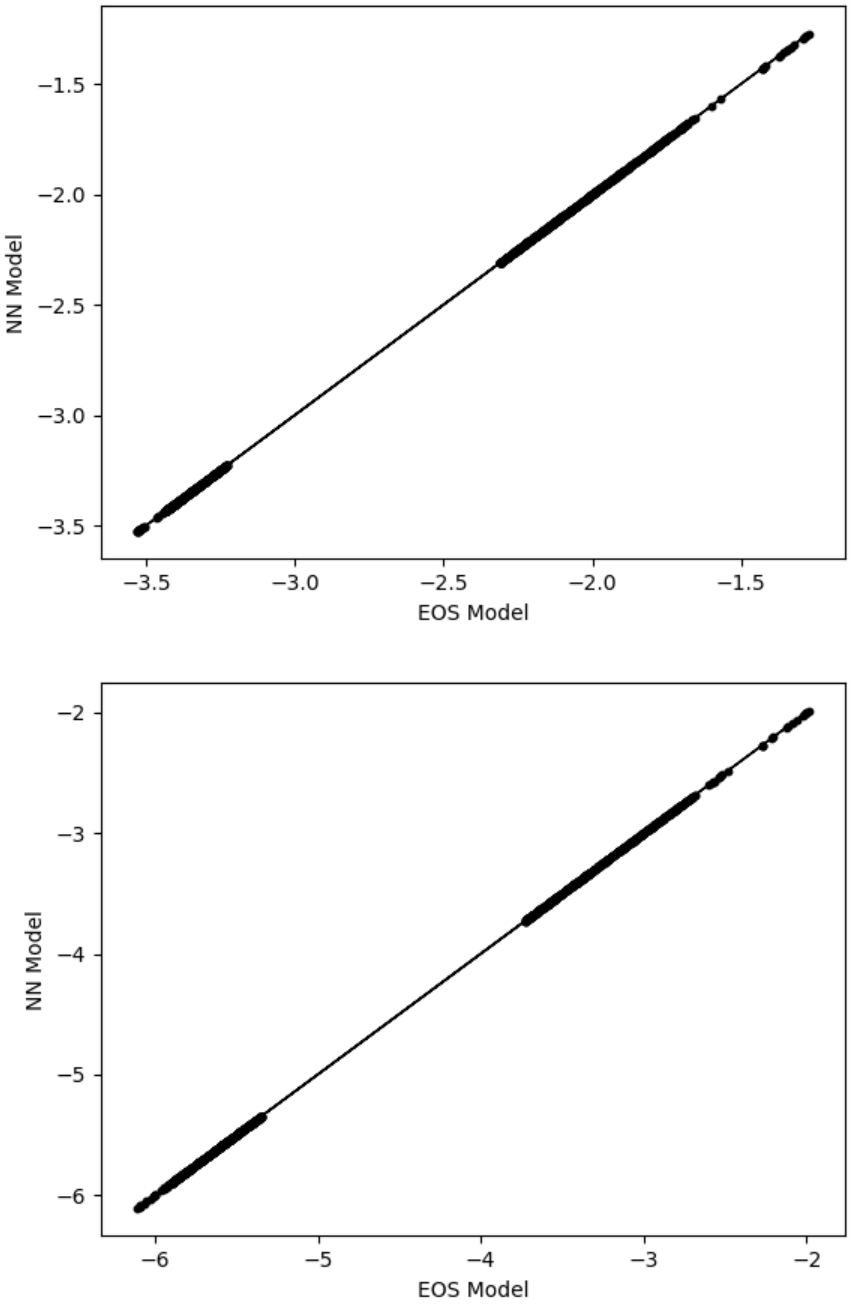


Figure 5.22 Accuracy comparison between fugacity coefficient calculated with EOS and ANN.
Top: Pseudo-Component 3. Bottom: Pseudo-Component 4.

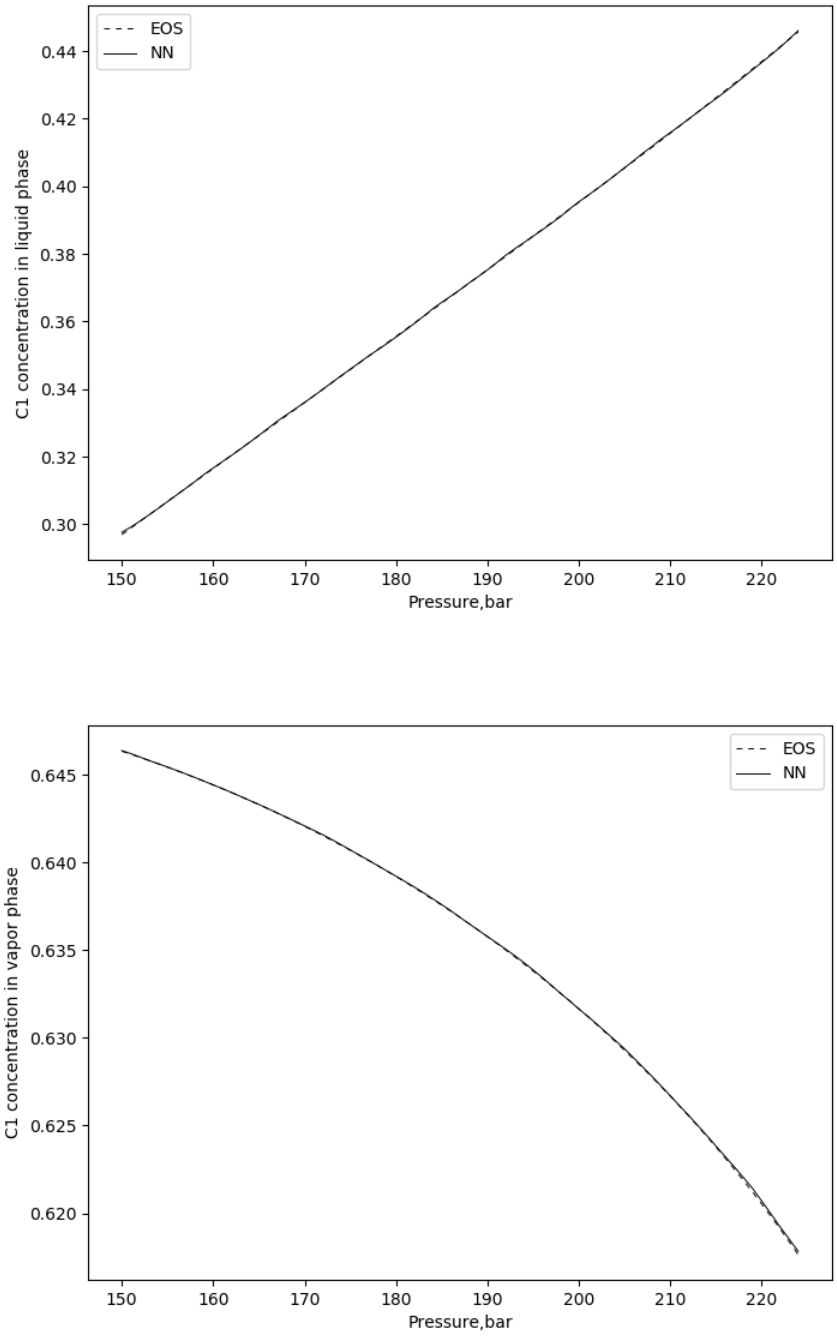


Figure 5.23 Phase mole fraction calculations comparison between EOS and ANN Top: Liquid phase for C_1. Bottom: Vapor phase for C_1.

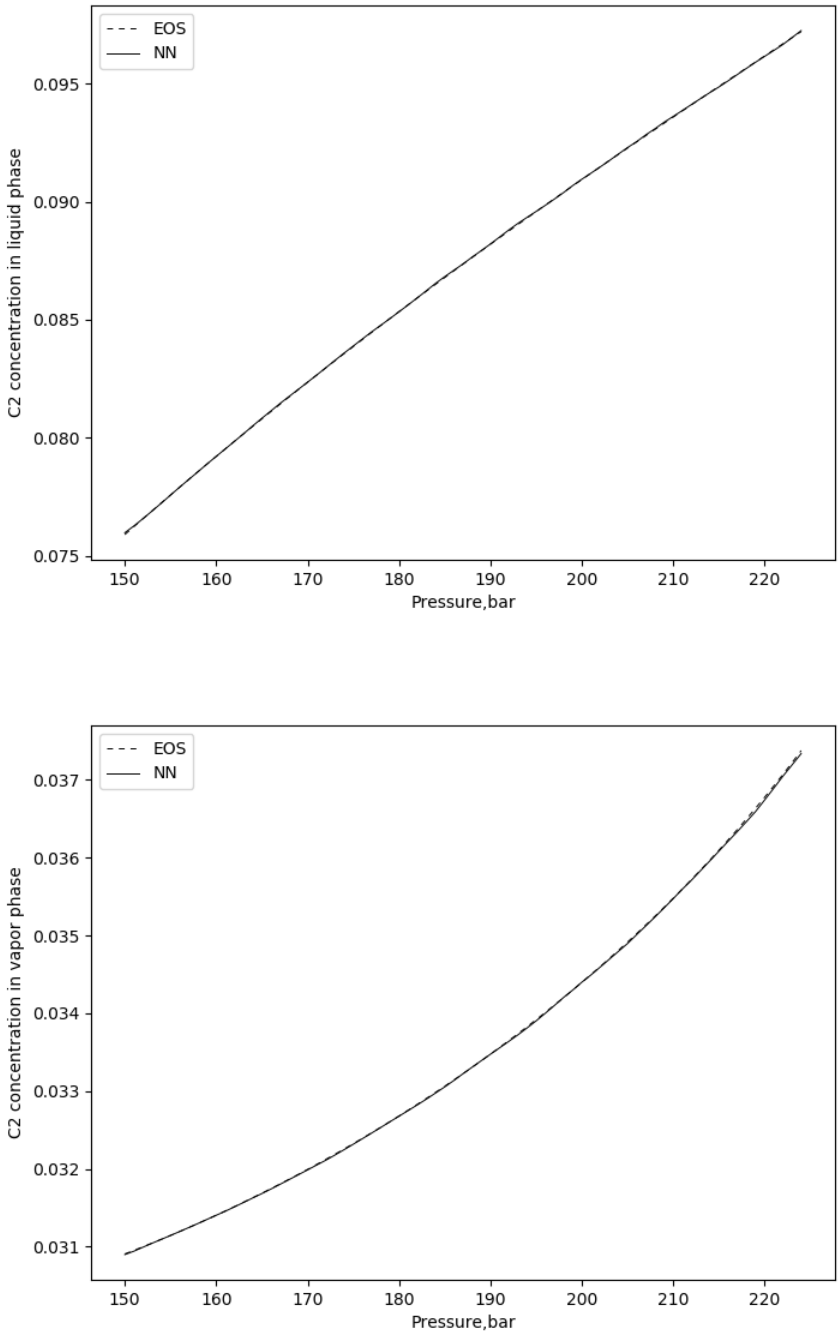


Figure 5.24 Phase mole fraction calculations comparison between EOS and ANN. Top: Liquid phase for $C_{2.}$ Bottom: Vapor phase for $C_{2.}$

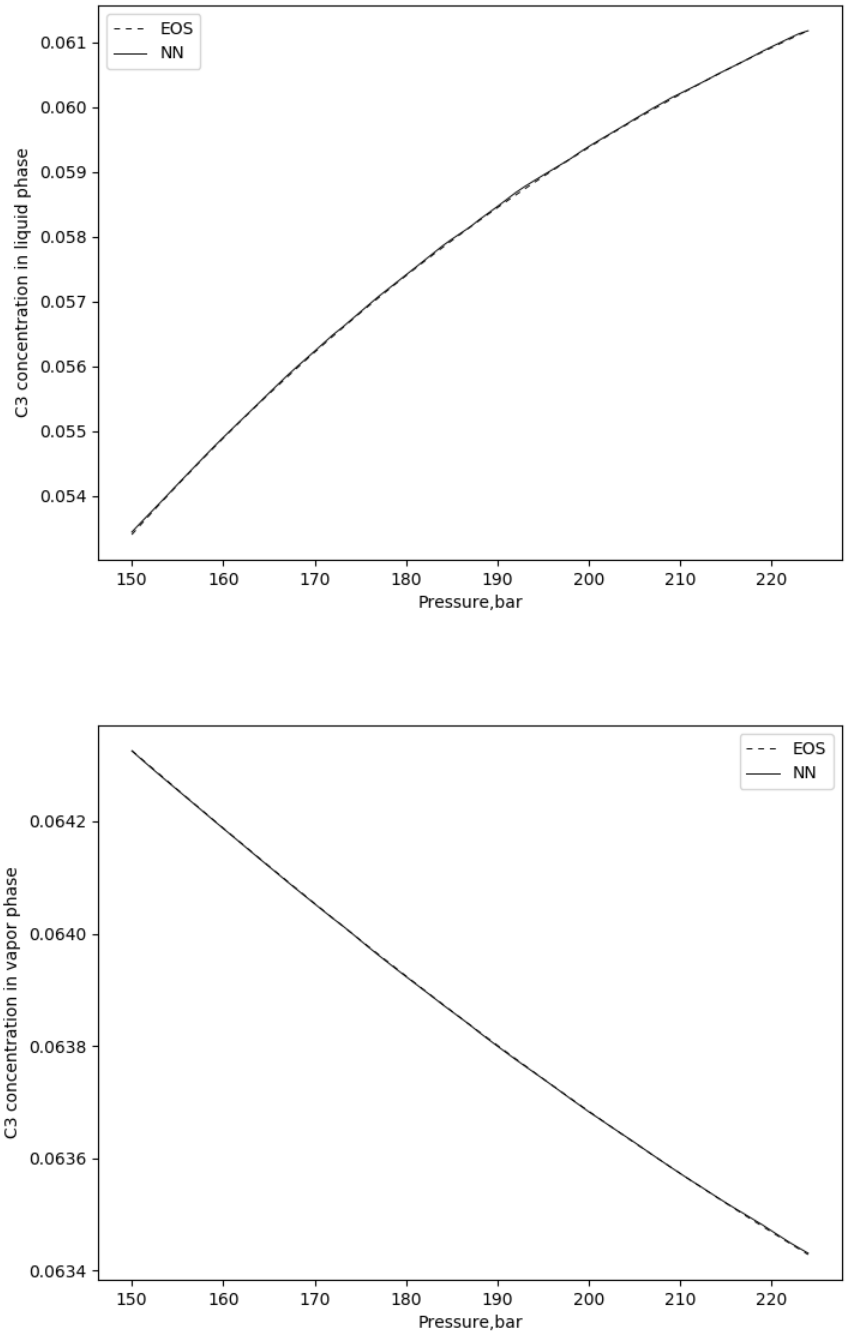


Figure 5.25 Phase mole fraction calculations comparison between EOS and ANN. Top: Liquid phase for $C_{3.}$ Bottom: Vapor phase for $C_{3.}$

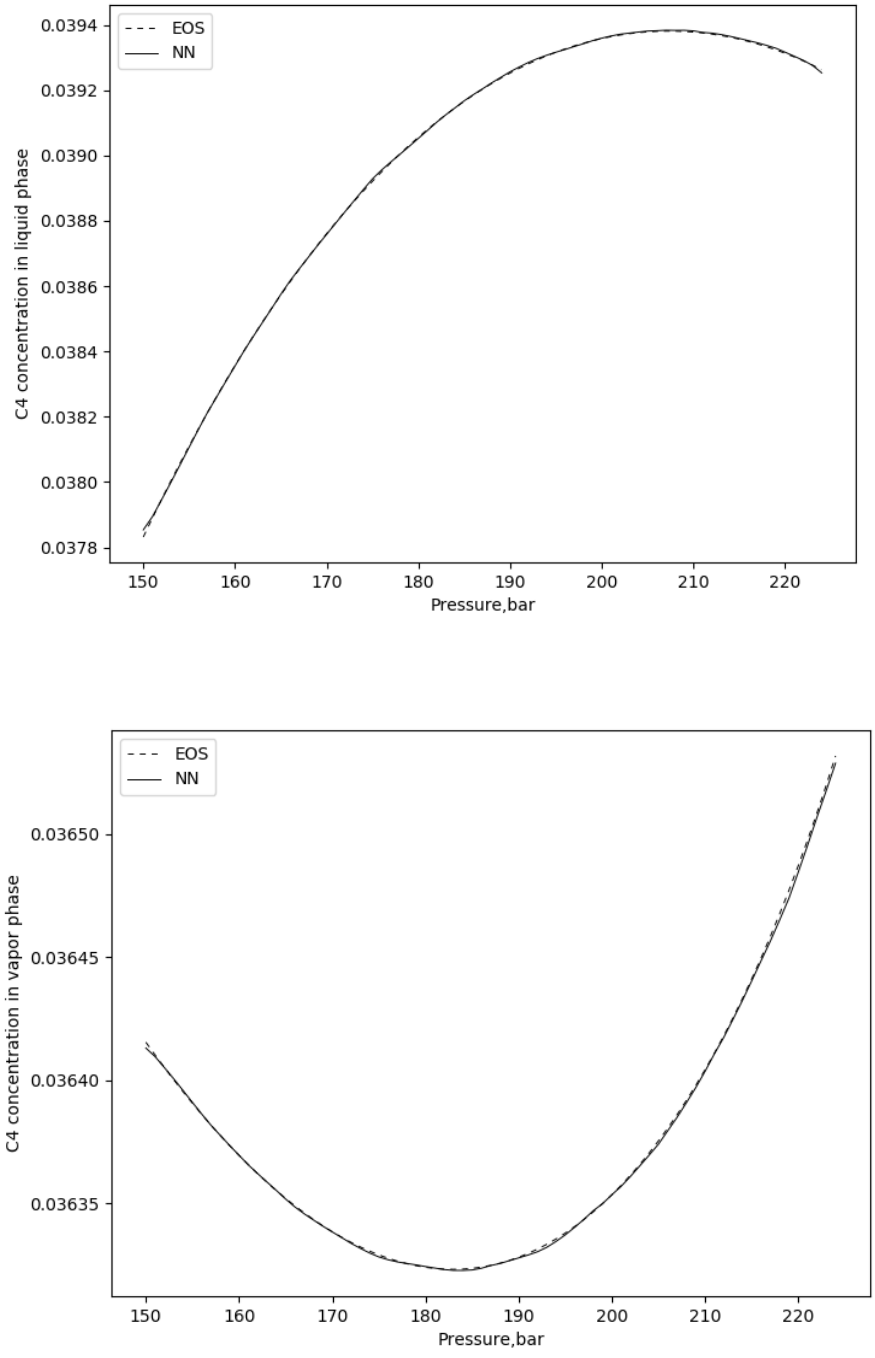
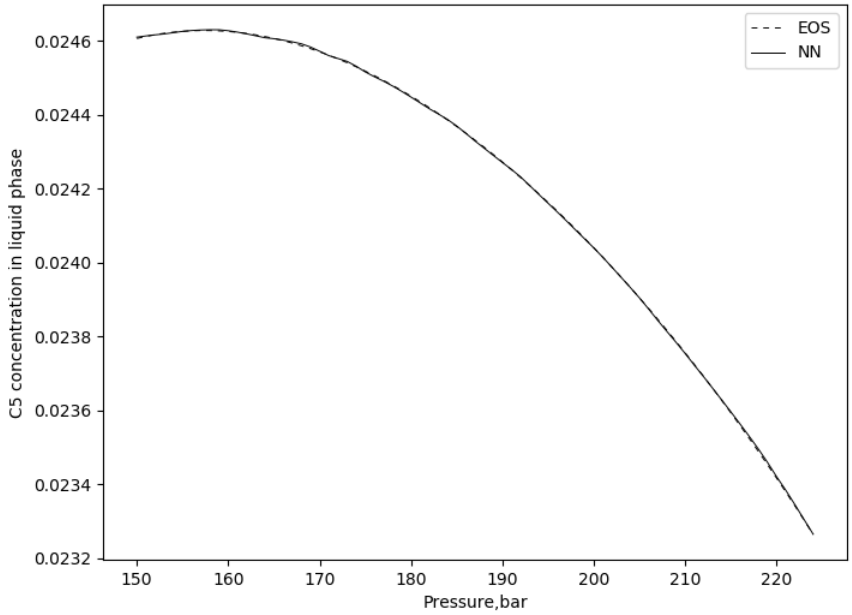


Figure 5.26 Phase mole fraction calculations comparison between EOS and ANN. Top: Liquid phase for $C_{4.}$ Bottom: Vapor phase for $C_{4.}$



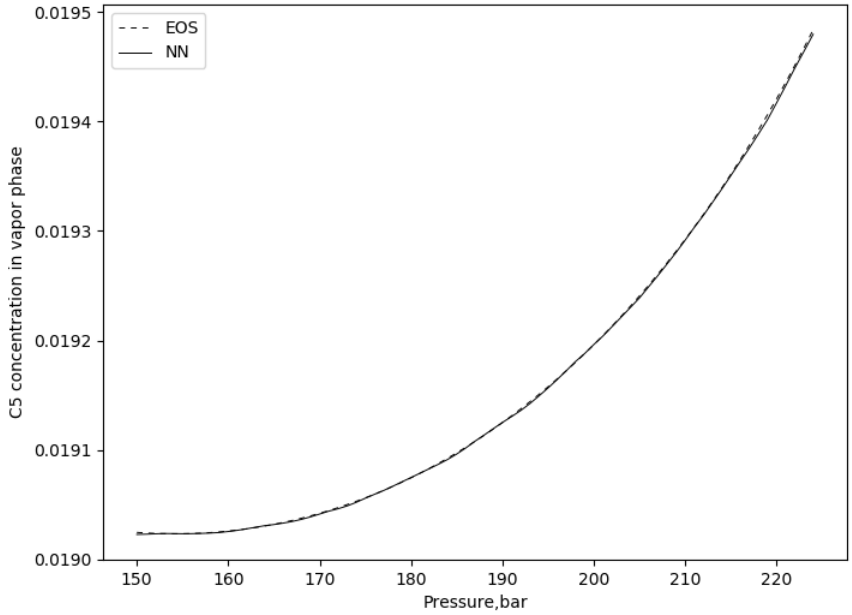


Figure 5.27 Phase mole fraction calculations comparison between EOS and ANN. Top: Liquid phase for $C_{5.}$ Bottom: Vapor phase for $C_{5.}$

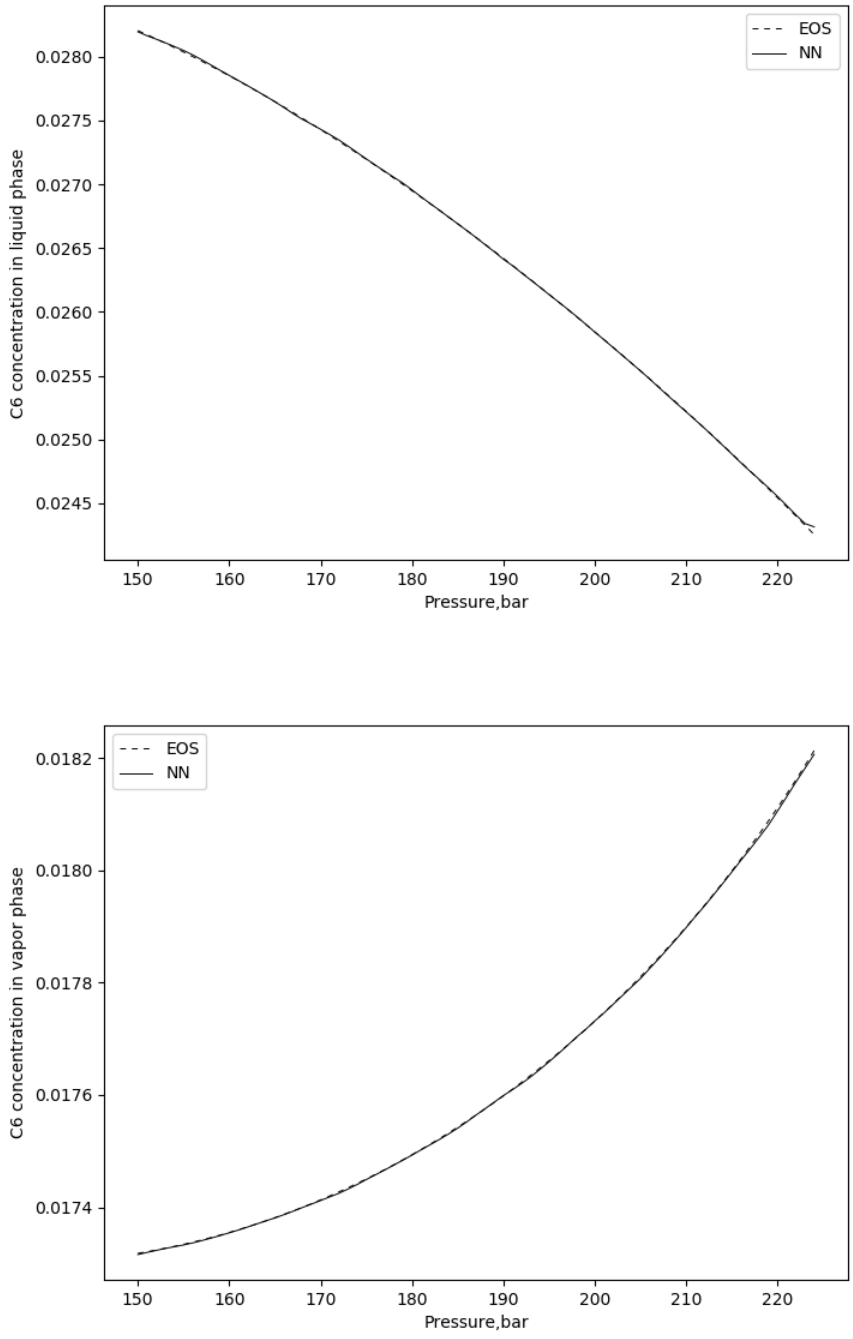


Figure 5.28 Phase mole fraction calculations comparison between EOS and ANN. Top: Liquid phase for $C_{6.}$ Bottom: Vapor phase for $C_{6.}$

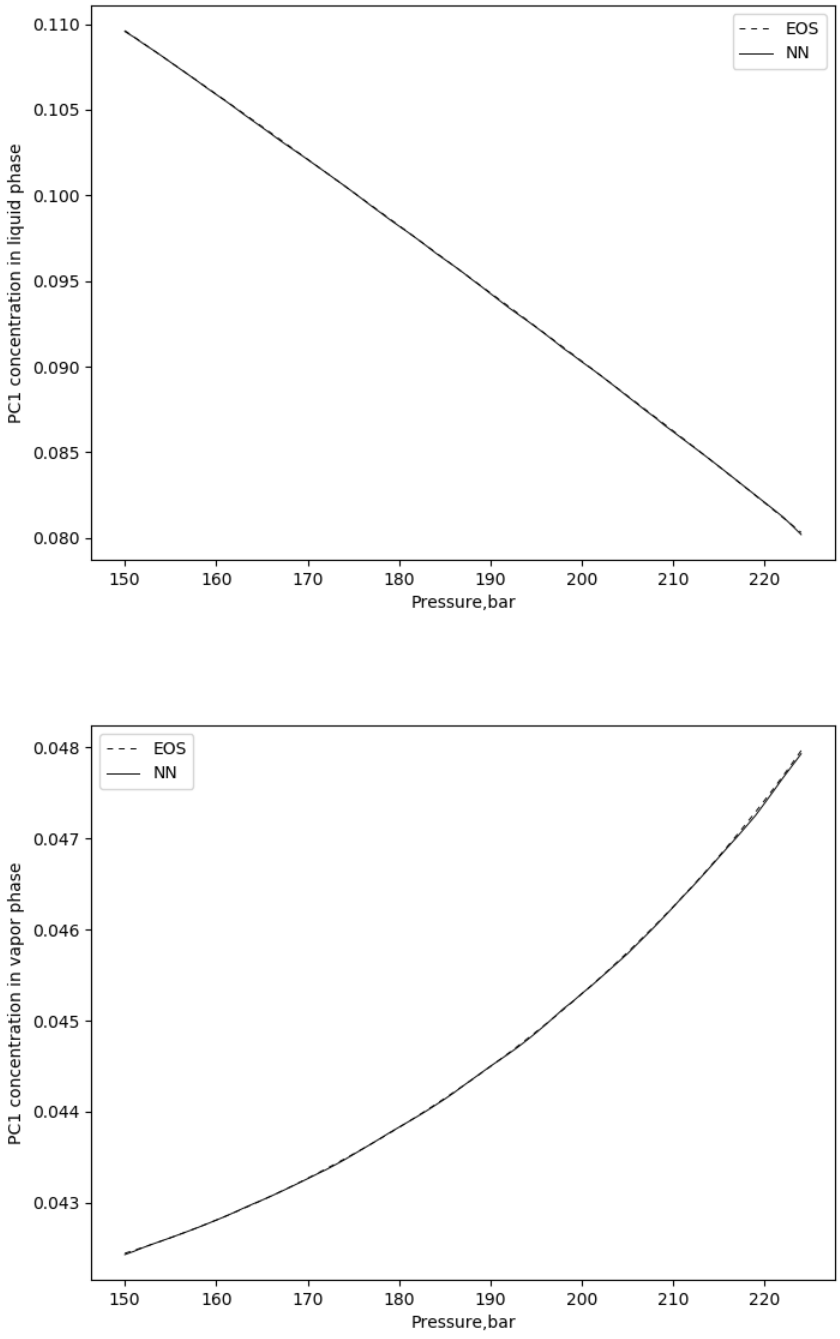


Figure 5.29 Phase mole fraction calculations comparison between EOS and ANN. Top: Liquid Pseudo Component 1. Bottom: Vapor Pseudo Component 1.

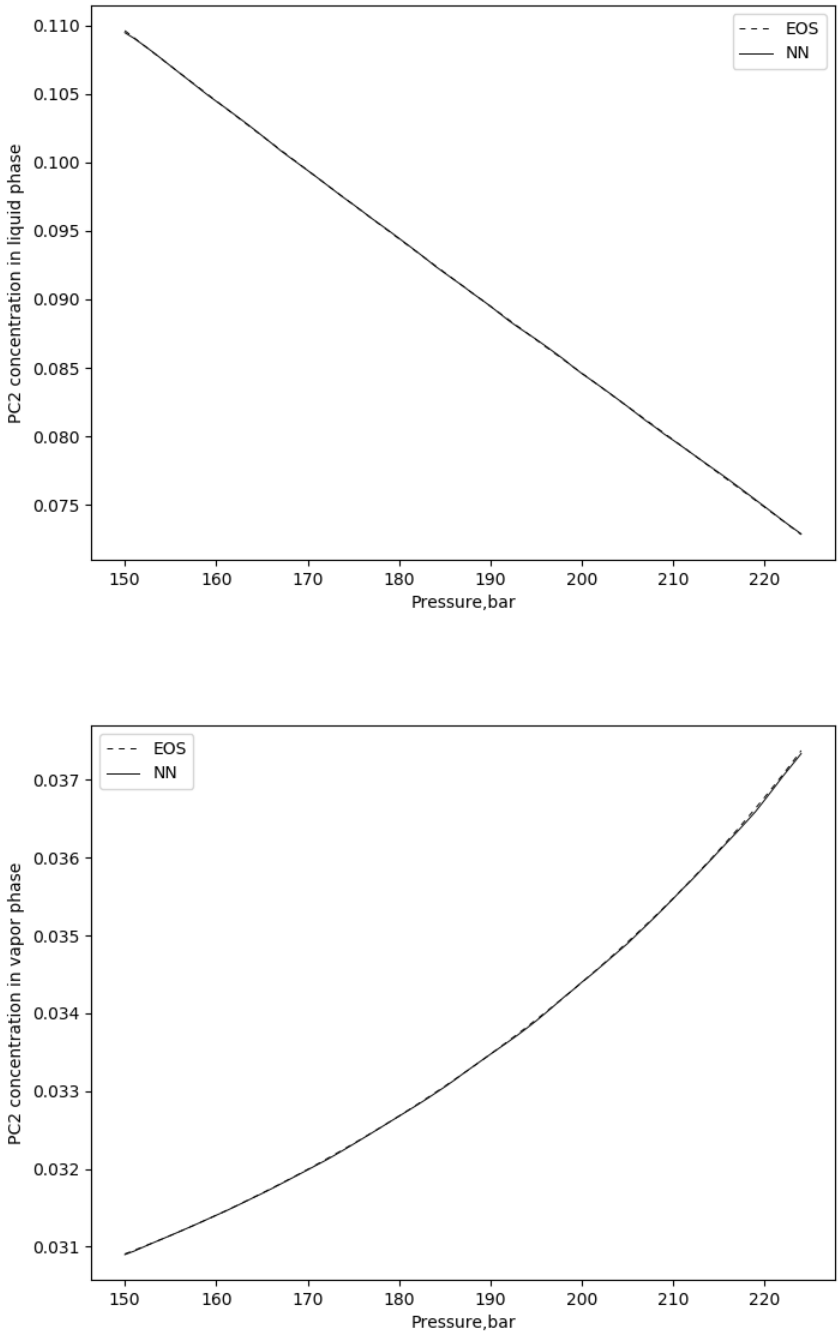


Figure 5.30 Phase mole fraction calculations comparison between EOS and ANN. Top: Liquid Pseudo Component 2. Bottom: Vapor Pseudo Component 2.

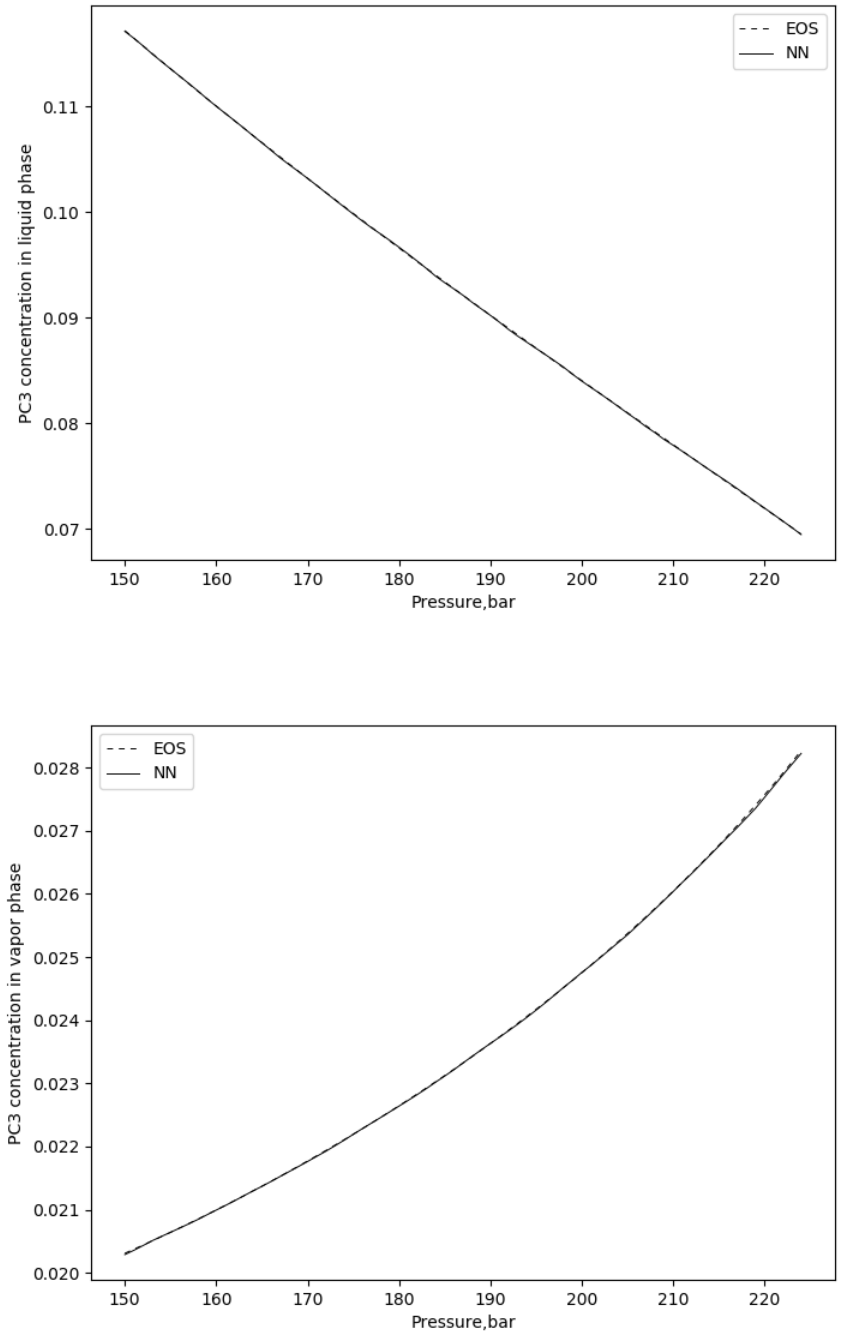


Figure 5.31 Phase mole fraction calculations comparison between EOS and ANN. Top: Liquid Pseudo Component 3. Bottom: Vapor Pseudo Component 3.

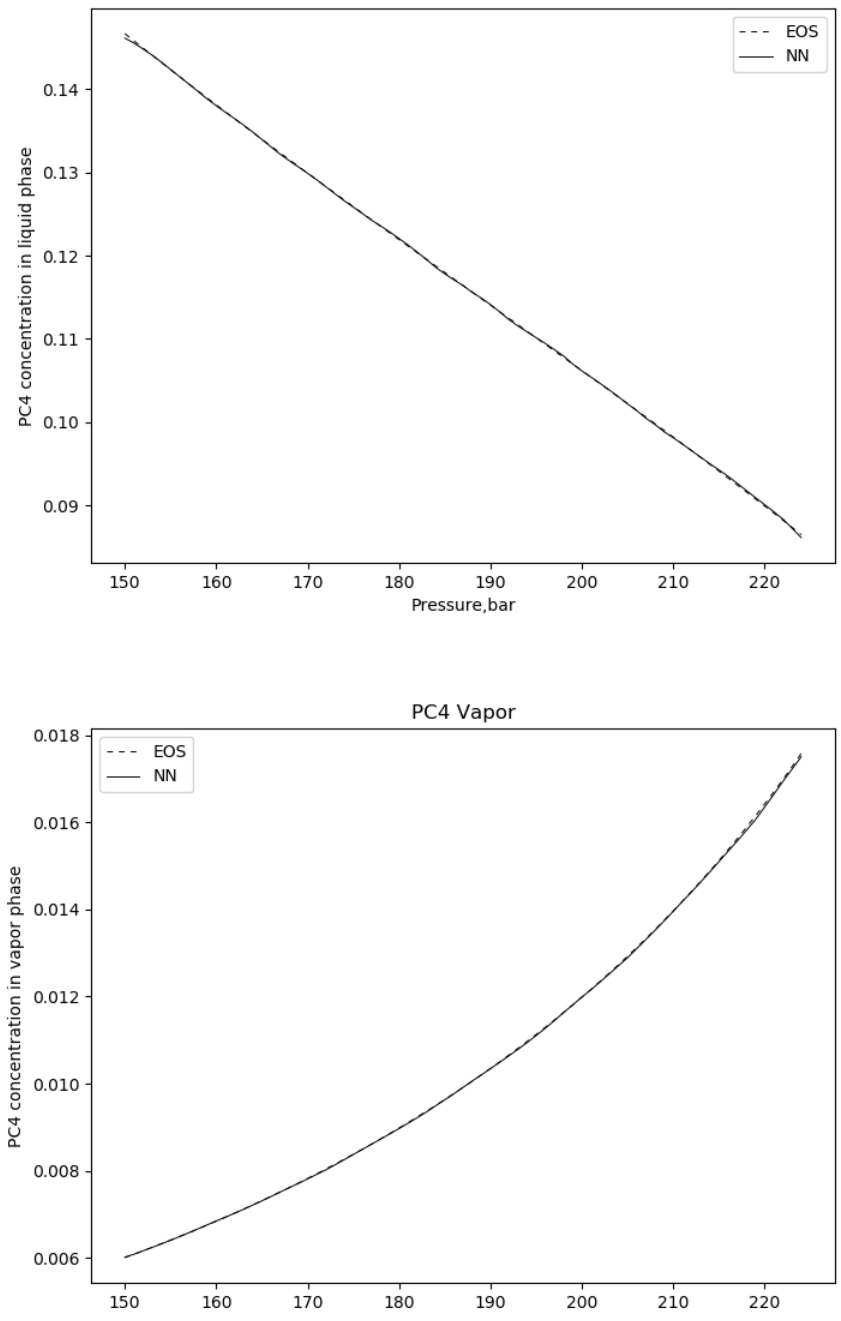


Figure 5.32 Phase mole fraction calculations comparison between EOS and ANN. Top: Liquid Pseudo Component 4. Bottom: Vapor Pseudo Component 4.

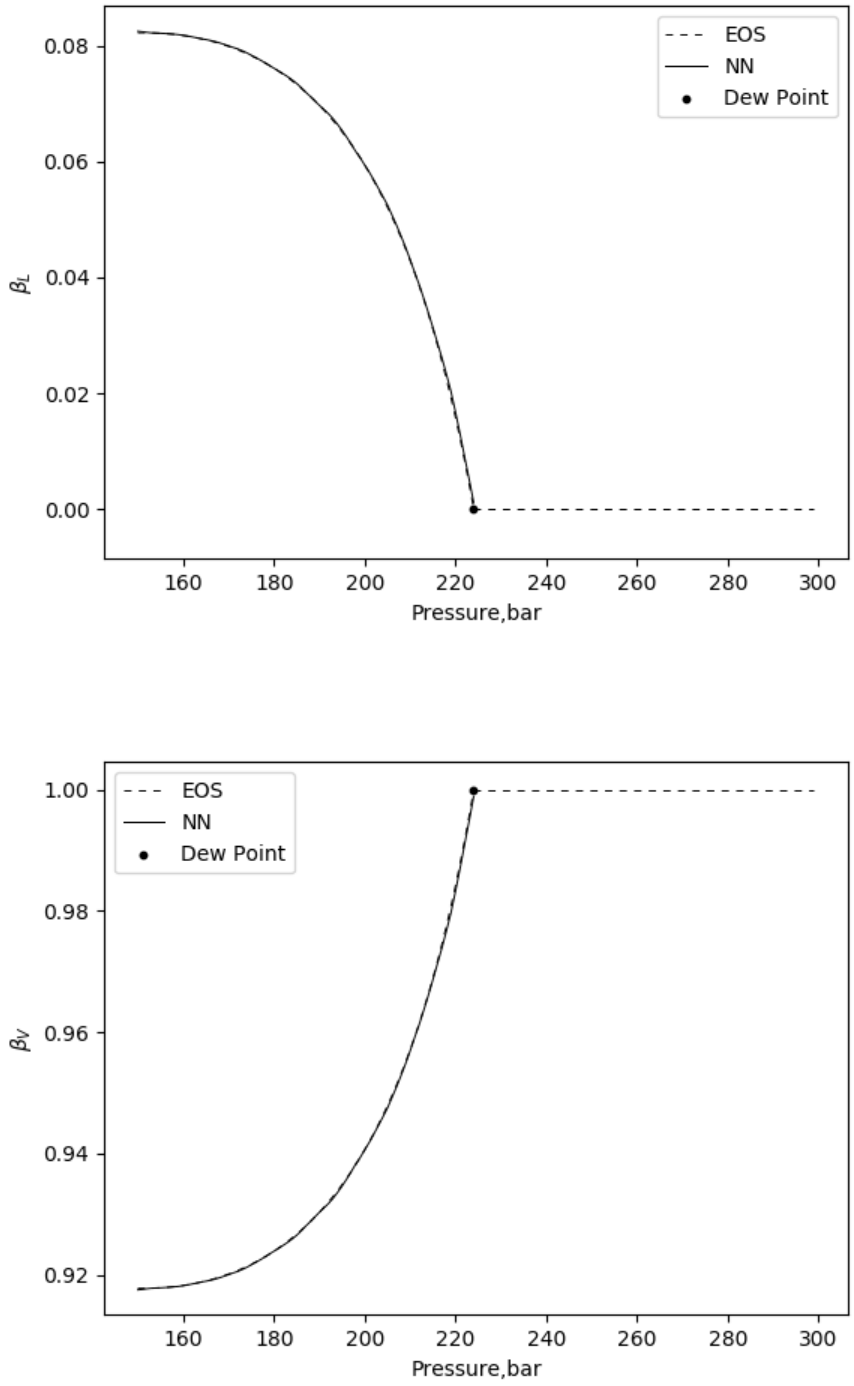


Figure 5.33 Phase mole fraction calculation comparison between EOS and ANN. Top: Liquid mole fraction. Bottom: Vapor mole fraction.

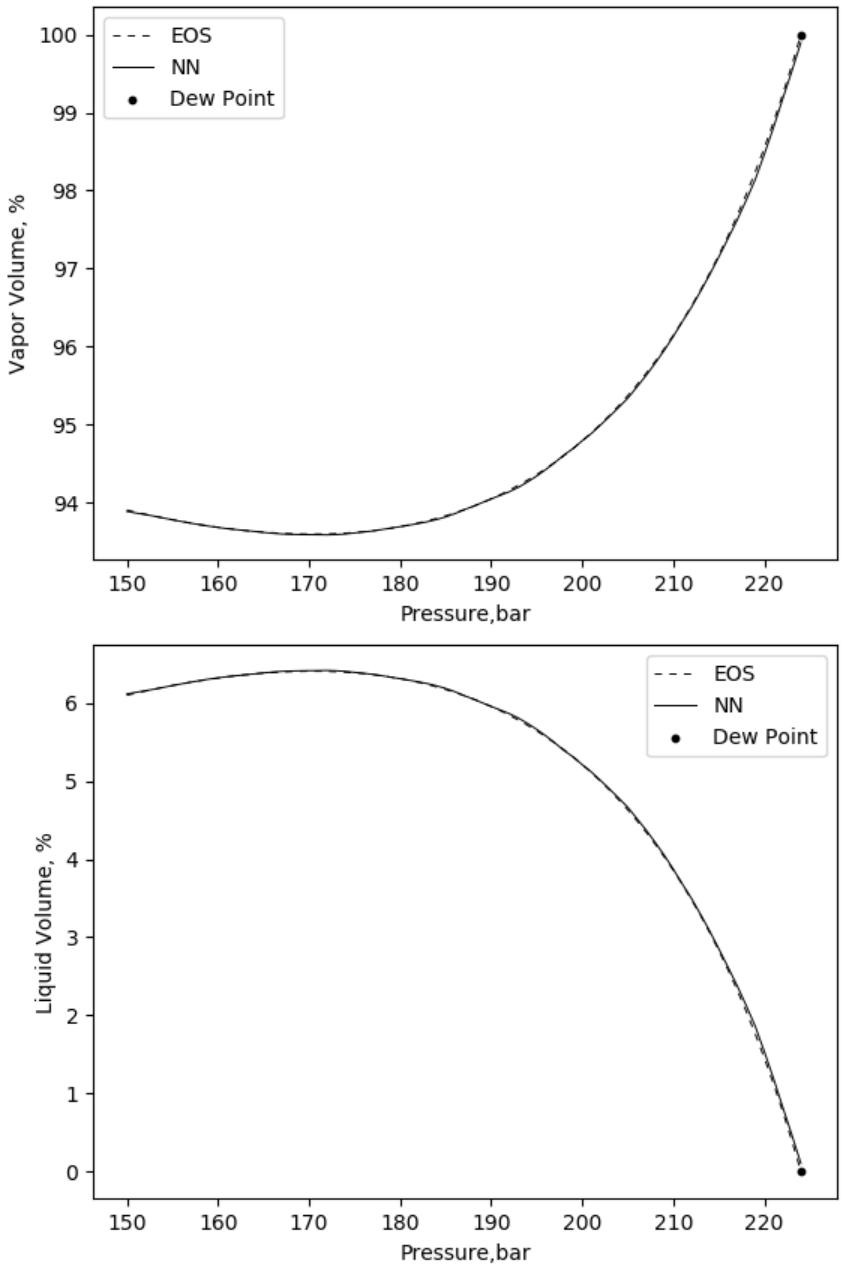


Figure 5.34 Fluid saturation comparison between EOS and ANN. Top: Vapor saturation. Bottom: Liquid Saturation.

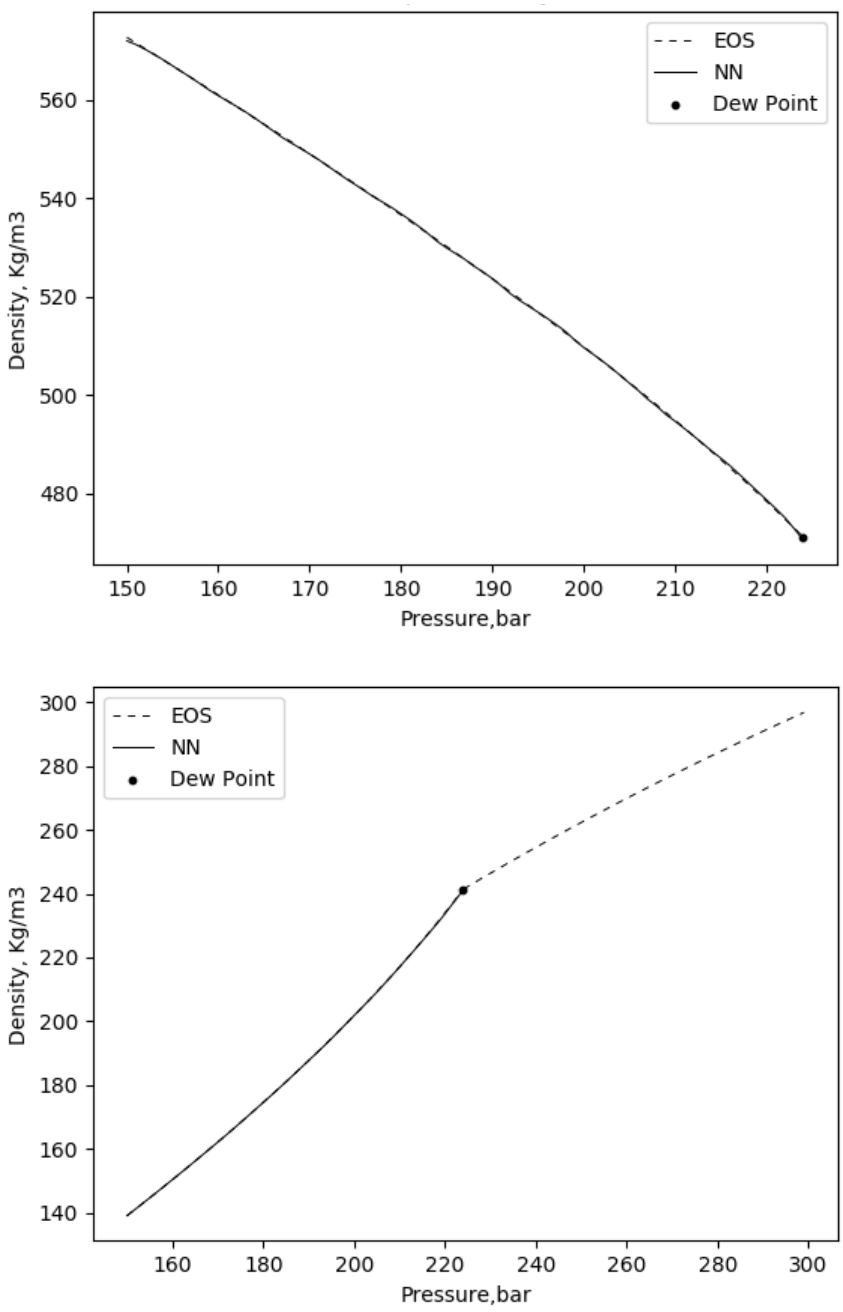


Figure 5.35 Fluid density calculation comparison between EOS and ANN. Top: Liquid density. Bottom: Vapor density.

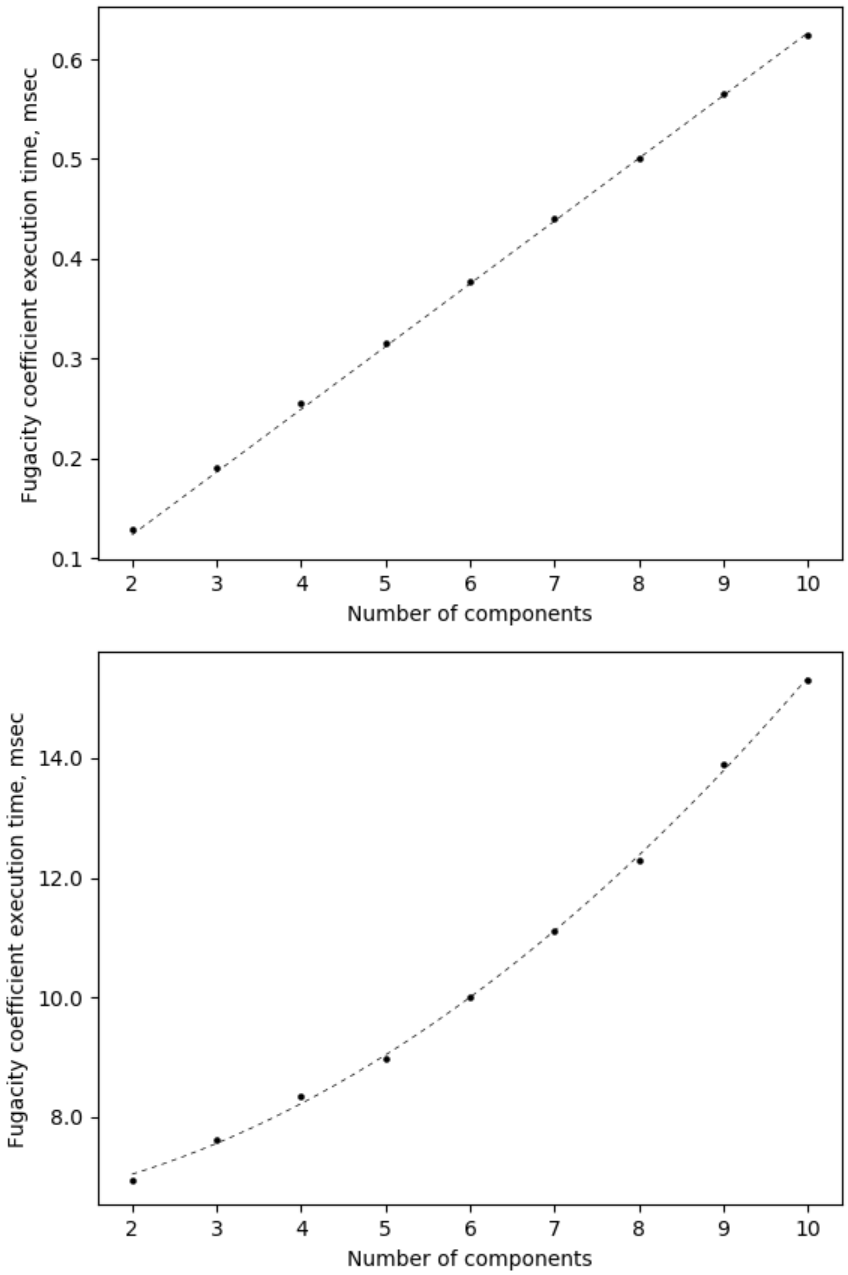


Figure 5.36 Fugacity coefficient execution time.
Top: ANNs. Bottom: EOS.

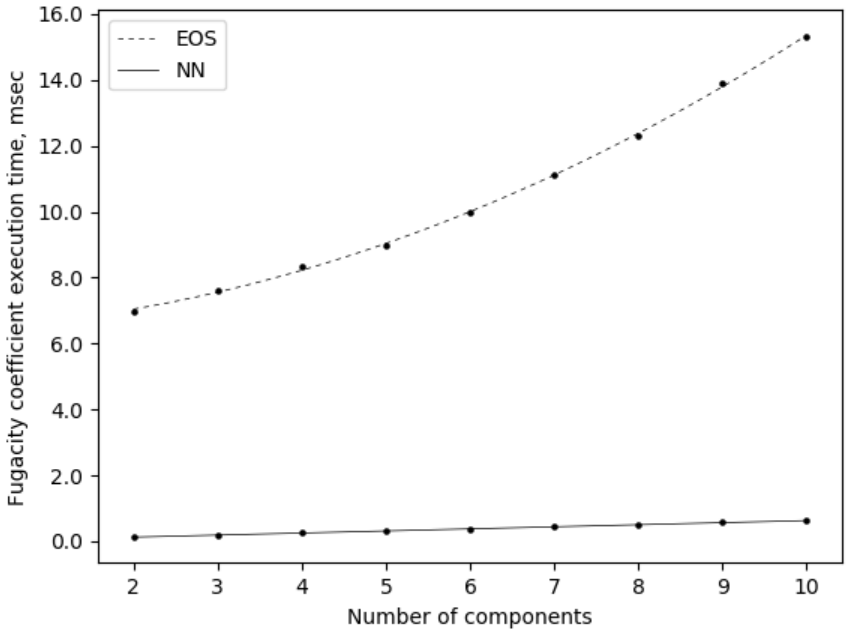


Figure 5.37 Fugacity coefficient execution time comparison between EOS and ANN.

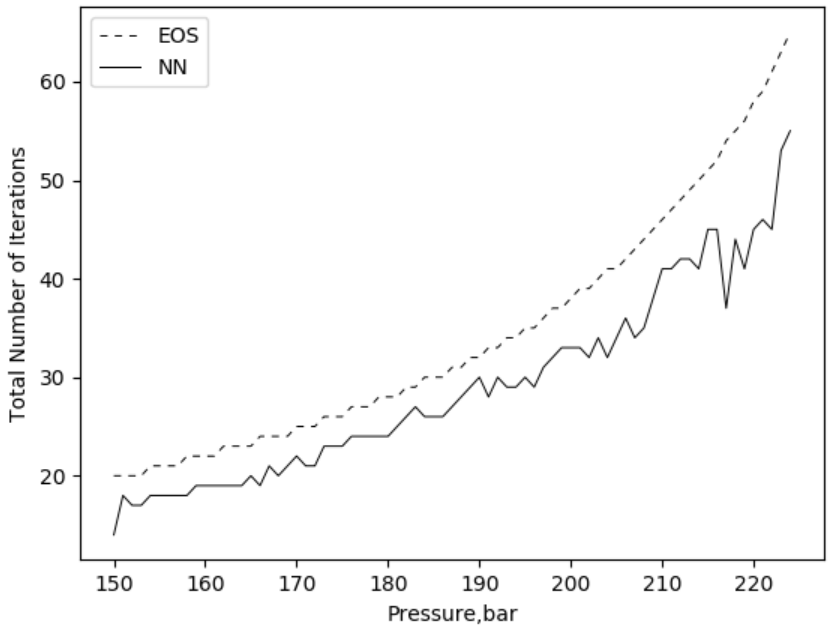


Figure 5.38 Total number of iterations needed to reach to the solution between EOS flash and ANN flash.

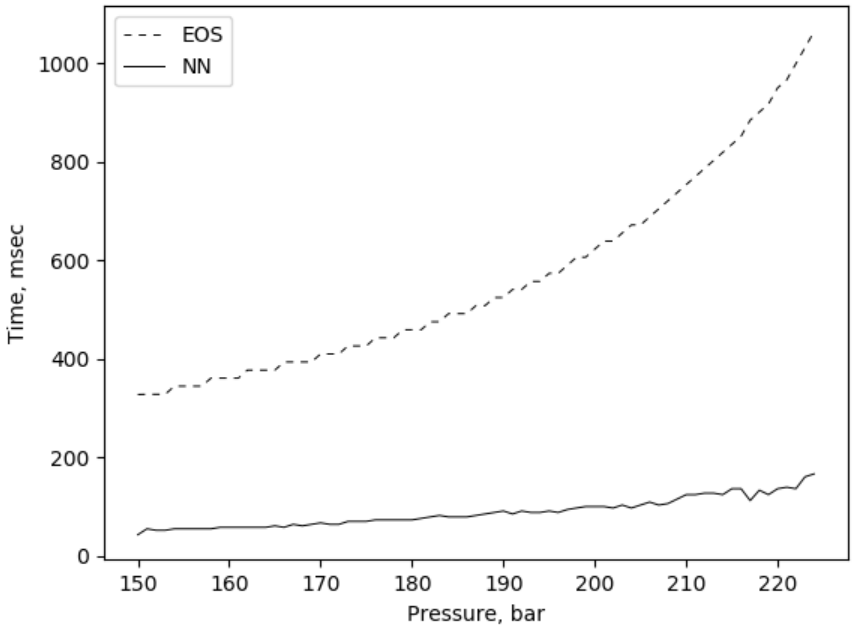


Figure 5.39 Total execution time to reach solution of conventional EOS method and ANN flash method.

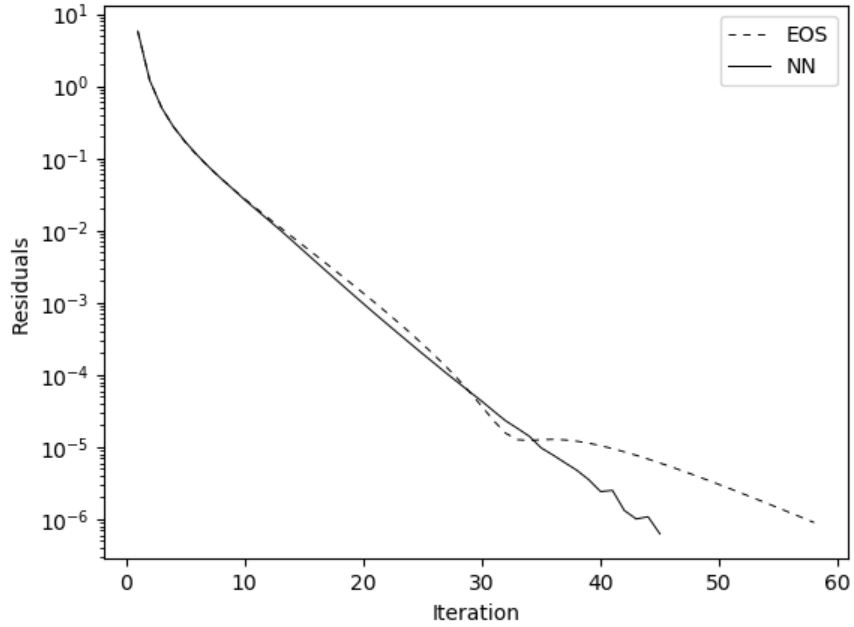


Figure 5.40 Convergence behavior comparison between EOS flash and ANN flash at a pressure of 220 Bar.

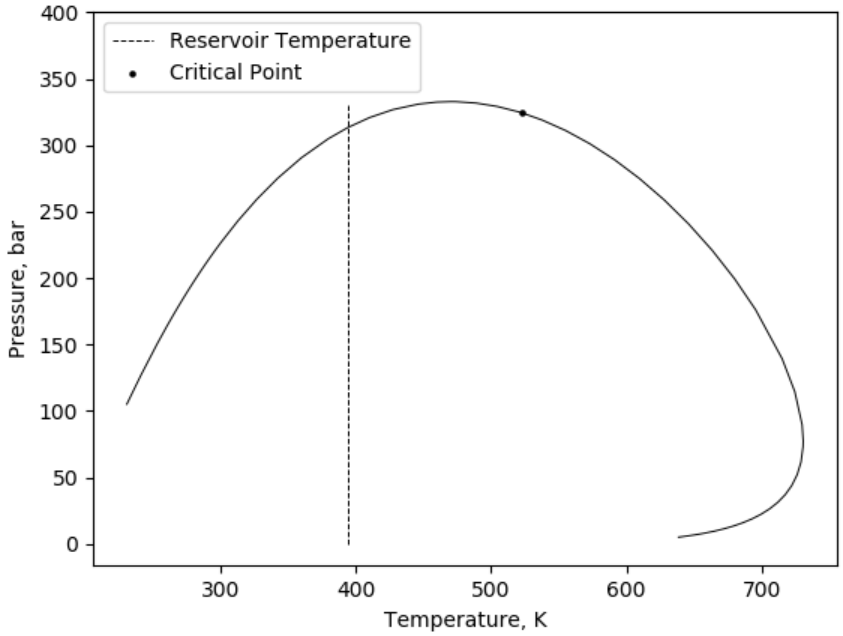


Figure 5.41. Phase diagram of fluid model in case 3.

Table 5.8 Fluid model for case 3.

Component	Overall Composition	MW, g/mol	Tc, K	Pc, Bar	Acentric Factor
N_2	0.0011	28.01	126.2	33.94	0.04
CO_2	0.0214	44.01	304.2	73.76	0.225
C_1	0.5559	16.04	190.6	46	0.008
C_2	0.087	30.07	305.4	48.84	0.098
C_3	0.0589	44.1	369.8	42.46	0.152
C_4	0.0405	58.12	419.46	37.49	0.1873
C_5	0.0253	72.15	465.35	33.79	0.2399
C_6	0.0197	86.18	507.4	29.69	0.296
PC-1	0.0765	119.41	617.17	28.42	0.1616
PC-2	0.0539	169.32	687.97	24.54	0.2245
PC-3	0.0379	240.96	789.61	20.43	0.3278
PC-4	0.0219	416.17	1011	15.43	0.5663

Table 5.9. BIPs for fluid in case 3.

	N2	CO2	CH4	C2H6	C3H8	C4H10	C5H12	C6H14	PC-1	PC-2	PC-3	PC-4
N2	0											
CO2	0	0										
CH4	0.1	0.1	0									
C2H6	0.1	0.145	0.042	0								
C3H8	0.1	0.2115	0.042	0.04	0							
C4H10	0.1	0.1834	0.042	0.04	0.03	0						
C5H12	0.1	0.1544	0.042	0.04	0.03	0.0116	0					
C6H14	0.1	0.1381	0.042	0.04	0.03	0.0155	0.0058	0				
PC-1	0.13	0.0294	0.0509	0.0415	0.0324	0.038	0.0231	0	0			
PC-2	0.13	0.0637	0.0536	0.0418	0.0339	0.0482	0.0356	0	0	0		
PC-3	0.13	0.1003	0.0581	0.0424	0.0378	0.0594	0.0521	0	0	0	0	
PC-4	0.13	0.1306	0.0698	0.0438	0.0532	0.0732	0.0774	0	0	0	0	0

Table 5.10. Generalization error of fugacity coefficients from ANNs.

Model	Mean Square Error	Average percentage error	R^2
N_2	3.44602E-08	0.022108731	0.999999619
CO_2	1.36382E-08	0.335540038	0.999998135
C_1	2.99343E-08	1.097797124	0.999999167
C_2	5.00523E-09	0.009843797	0.99999872
C_3	5.41057E-08	0.01605231	0.999992704
C_4	4.7724E-08	0.009767913	0.999997499
C_5	6.25513E-08	0.009573048	0.999998744
C_6	3.47057E-07	0.014598305	0.999997774
PC-1	1.52473E-07	0.00785658	0.999999562
PC-2	8.37175E-07	0.015457675	0.999998583
PC-3	1.14541E-06	0.012976682	0.999999024
PC-4	1.67559E-06	0.008772282	0.999999609

Table 5.11. Time per iteration comparison between EOS Flash and NN Flash.

Time per iteration, msec	EOS Flash	NN Flash
	19.96194709	1.194794771

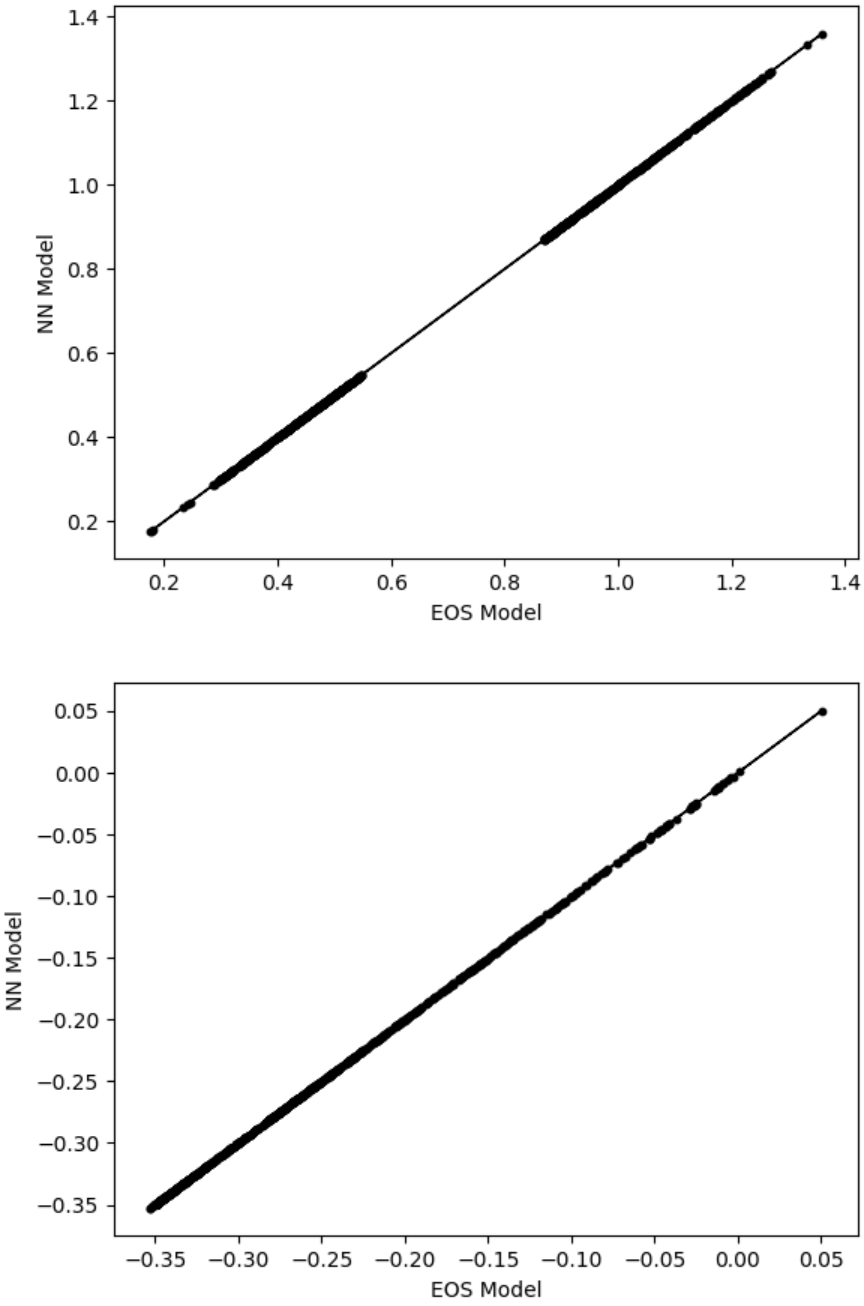


Figure 5.42 Accuracy comparison between fugacity coefficient calculated with EOS and ANN.
Top: N_2. Bottom: CO_2.

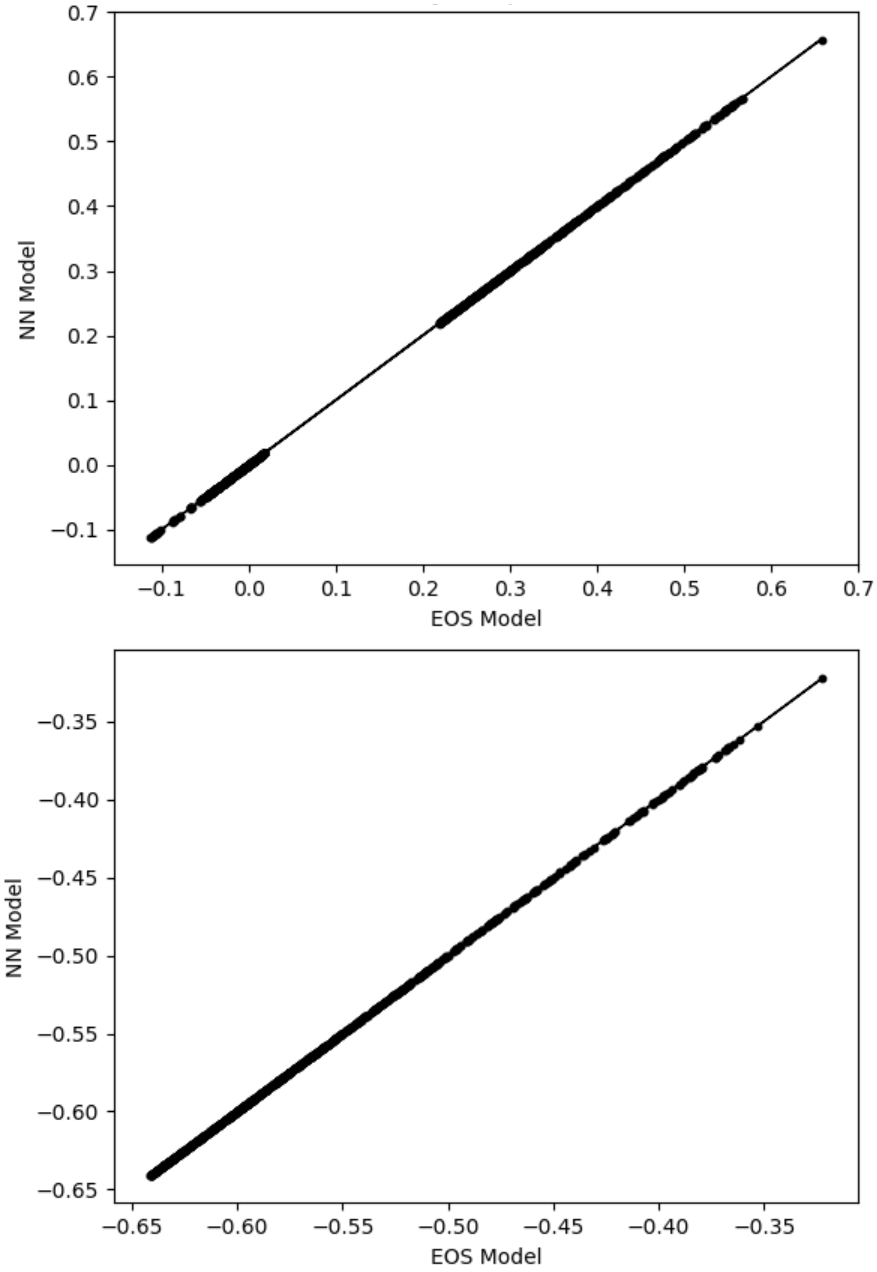


Figure 5.43 Accuracy comparison between fugacity coefficient calculated with EOS and ANN.
Top: Component C_1. Bottom: Component C_2.

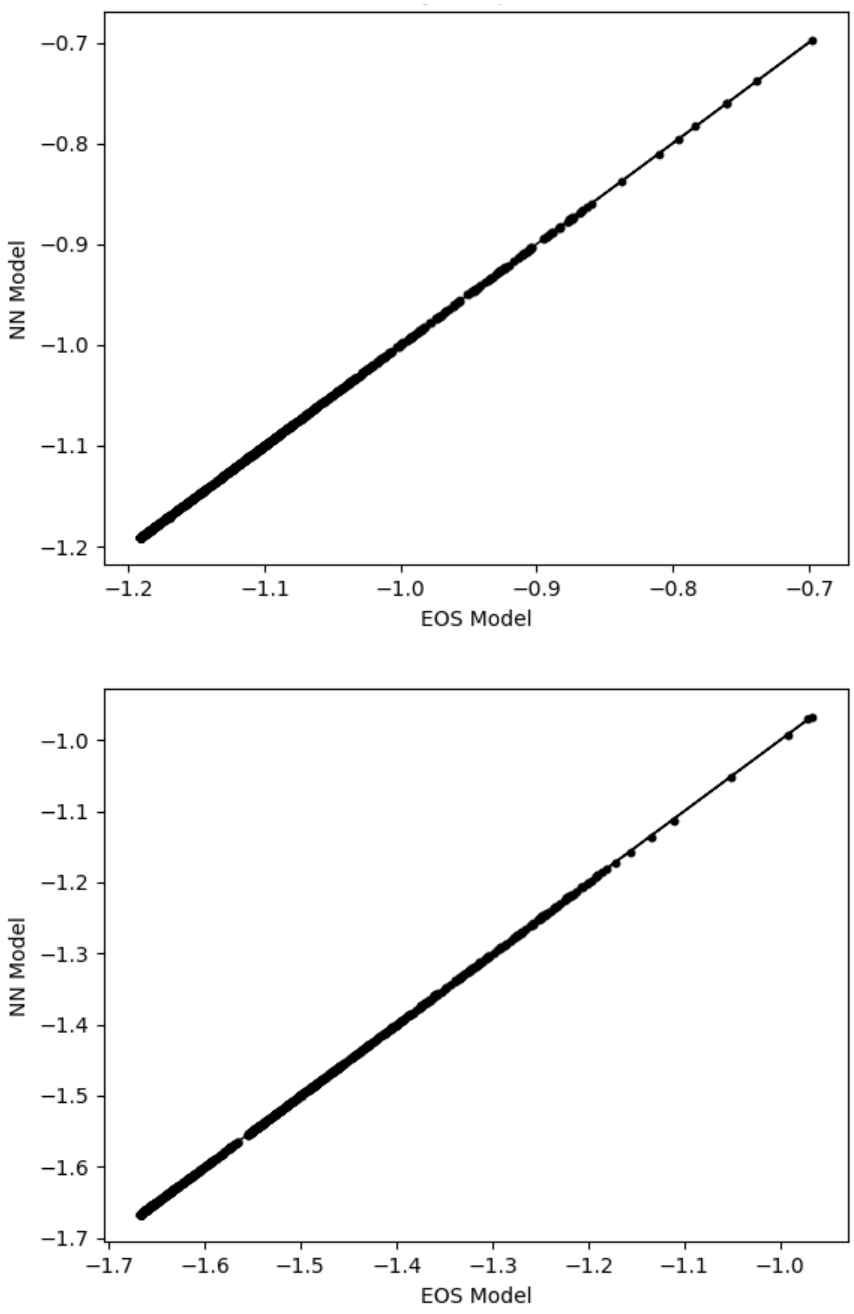


Figure 5.44 Accuracy comparison between fugacity coefficient calculated with EOS and ANN.
Top: Component C_3. Bottom: Pseudo-Component C_4.

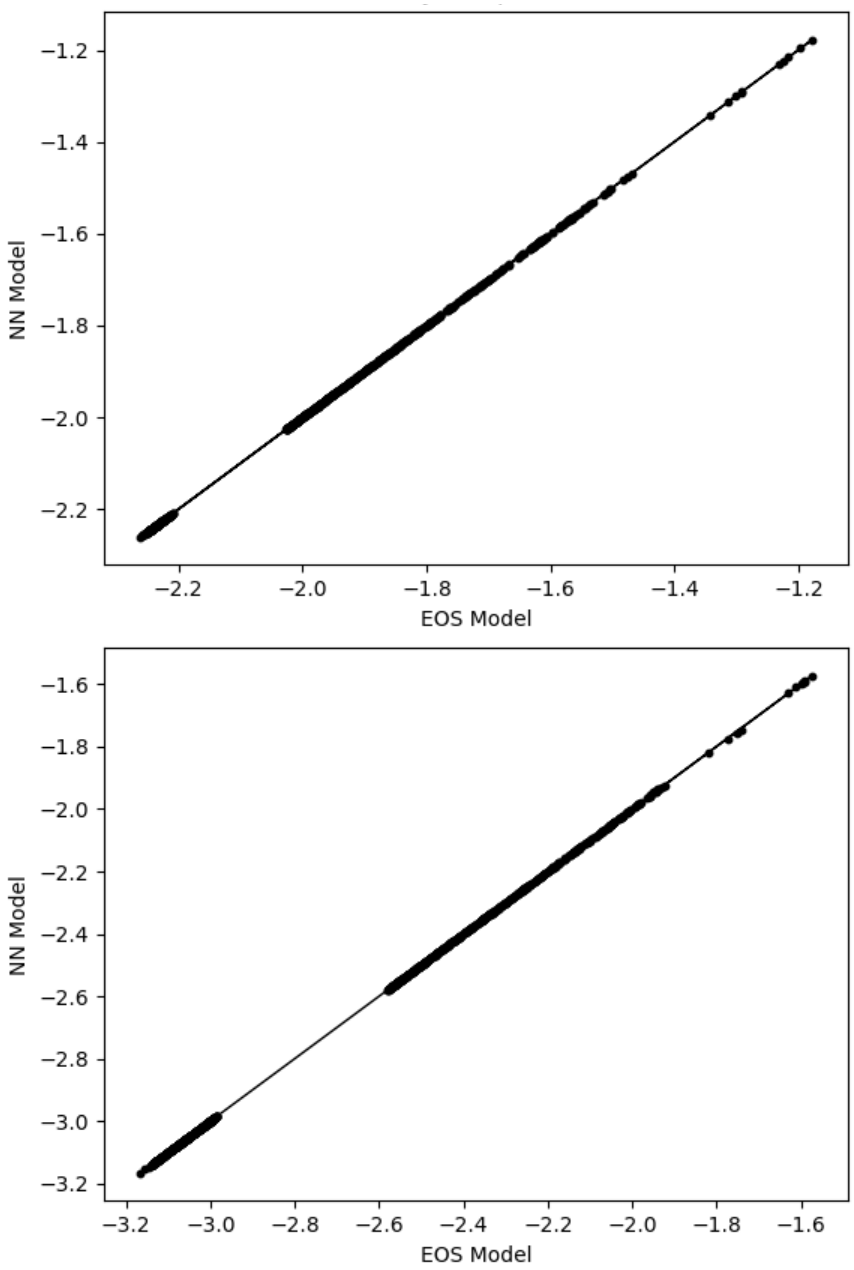


Figure 5.45 Accuracy comparison between fugacity coefficient calculated with EOS and ANN.
Top: Component C_5. Bottom: Pseudo-Component C_6.

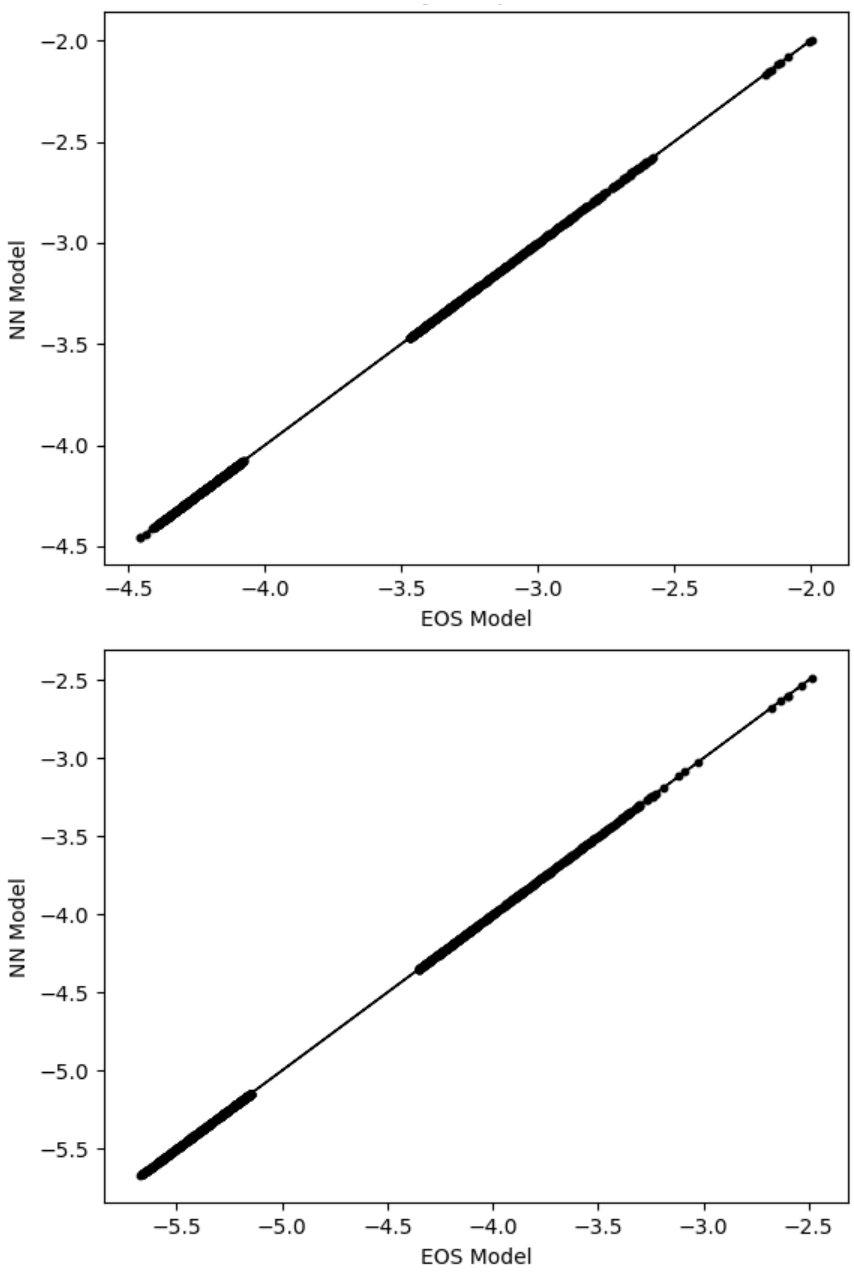


Figure 5.46 Accuracy comparison between fugacity coefficient calculated with EOS and ANN.
Top: Pseudo Component 1. Bottom: Pseudo Component 2.

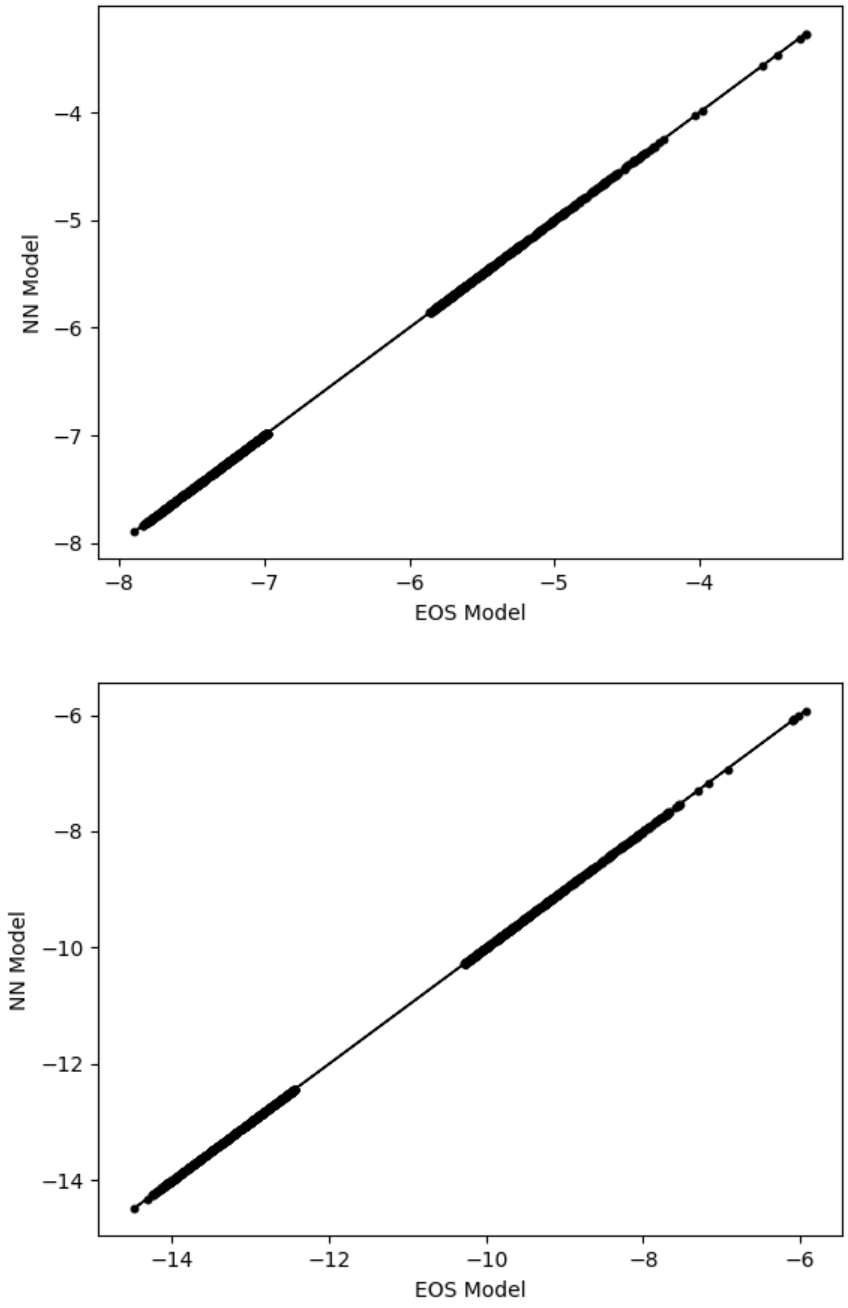


Figure 5.47 Accuracy comparison between fugacity coefficient calculated with EOS and ANN.
Top: Pseudo Component 3. Bottom: Pseudo Component 4.

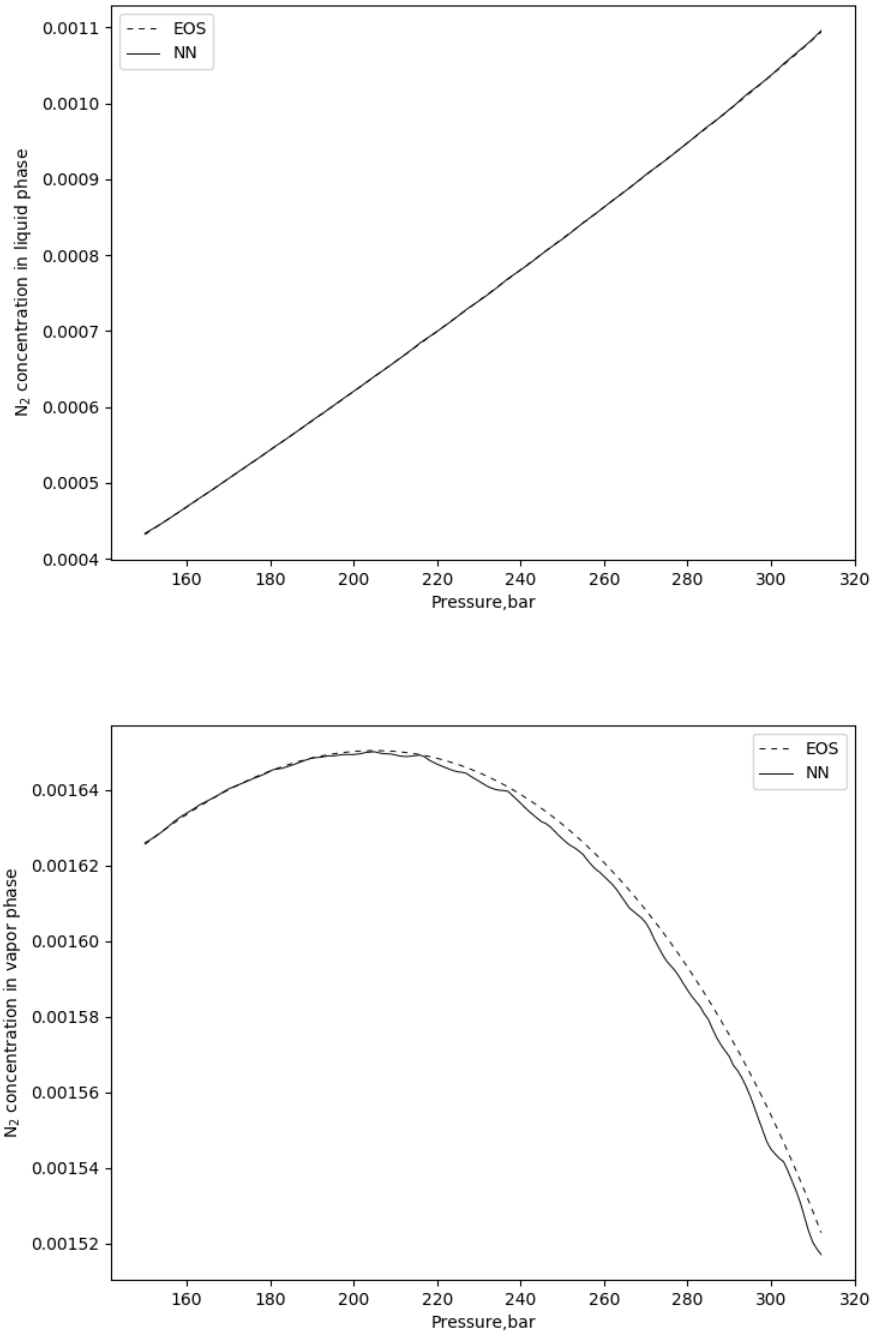


Figure 5.48 Phase mole fraction calculations comparison between EOS and ANN. Top: Liquid N_2. Bottom: Vapor N_2.

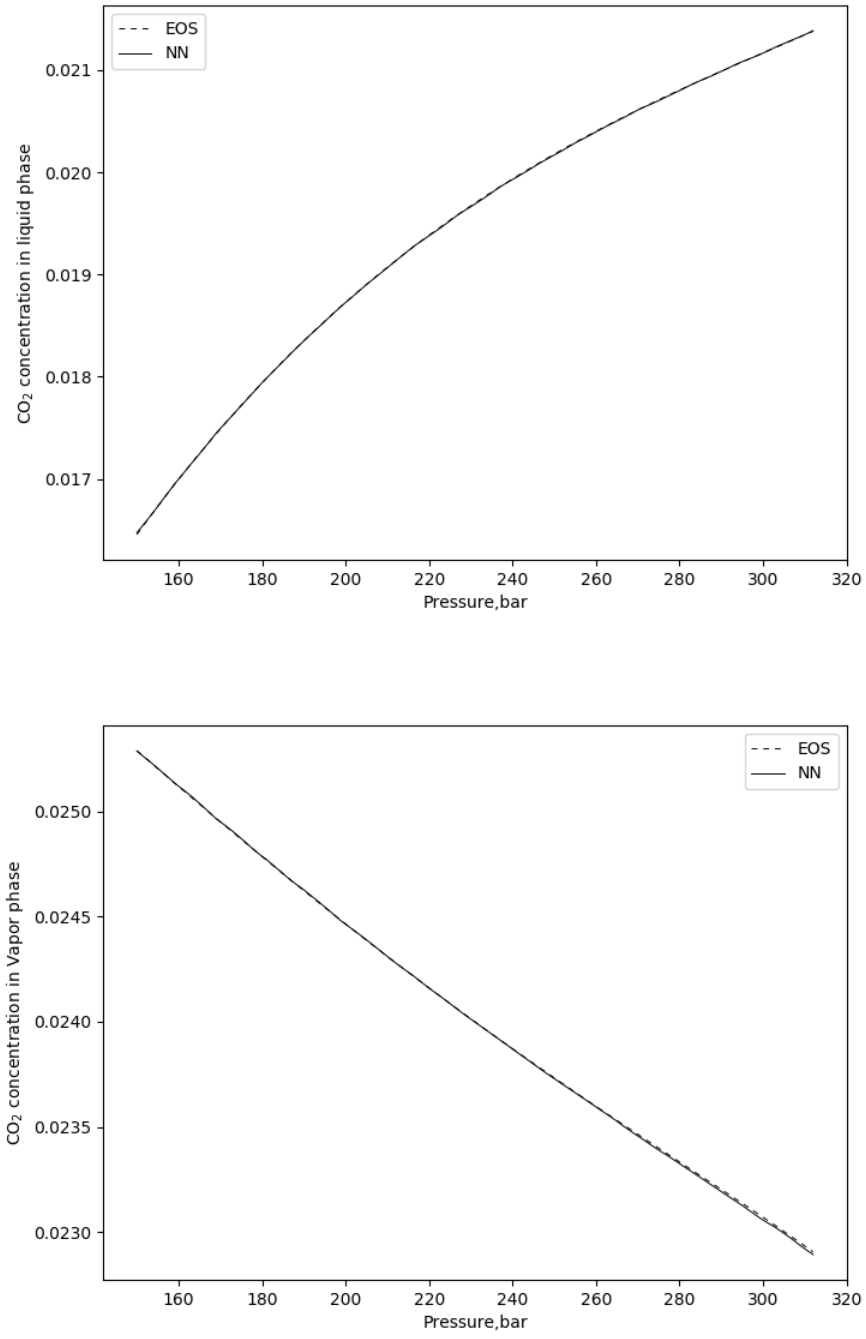


Figure 5.49 Phase mole fraction calculations comparison between EOS and ANN. Top: Liquid CO_2. Bottom: Vapor CO_2.

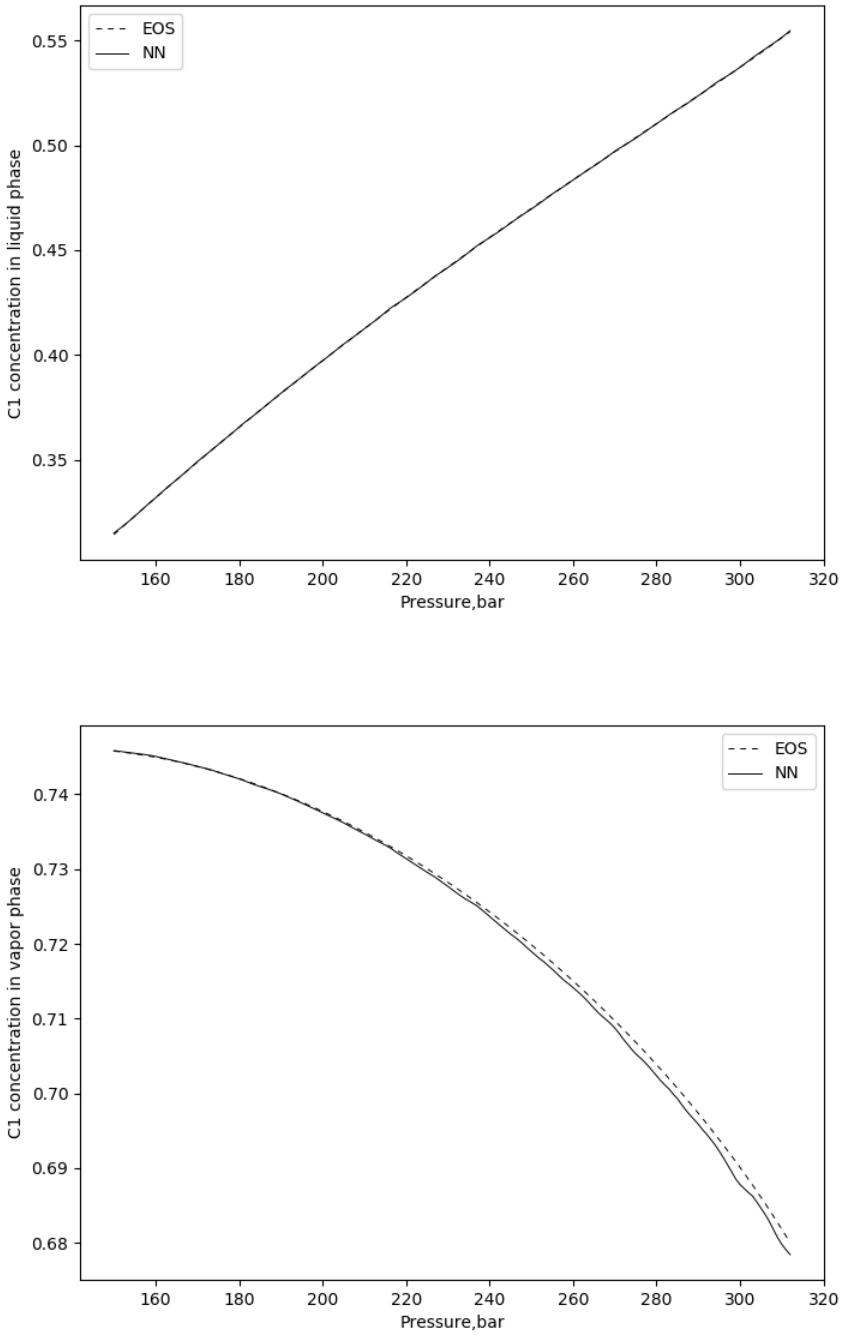


Figure 5.50 Phase mole fraction calculations comparison between EOS and ANN. Top: Liquid C_1. Bottom: Vapor C_1.

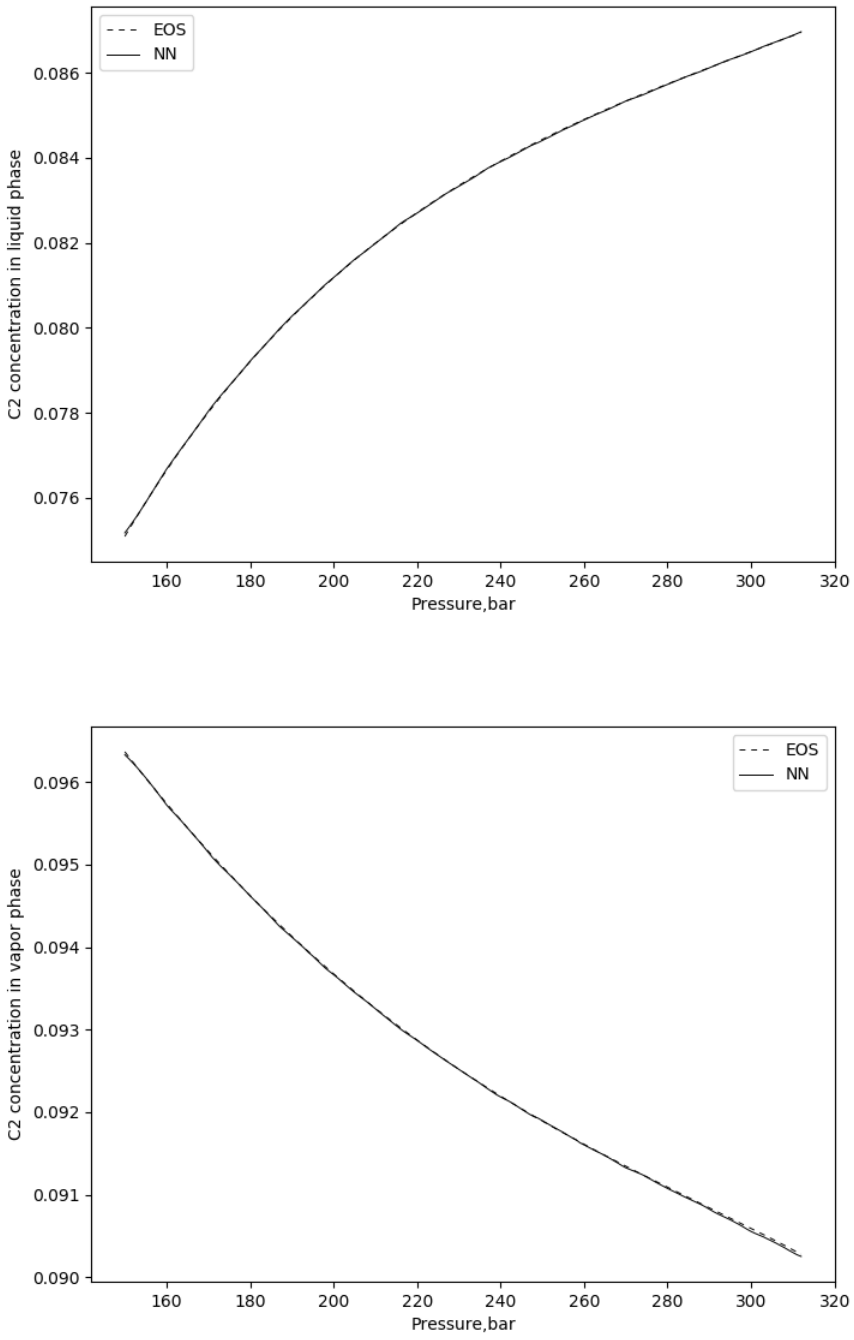


Figure 5.51 Phase mole fraction calculations comparison between EOS and ANN. Top: Liquid $C_{2.}$ Bottom: Vapor $C_{2.}$

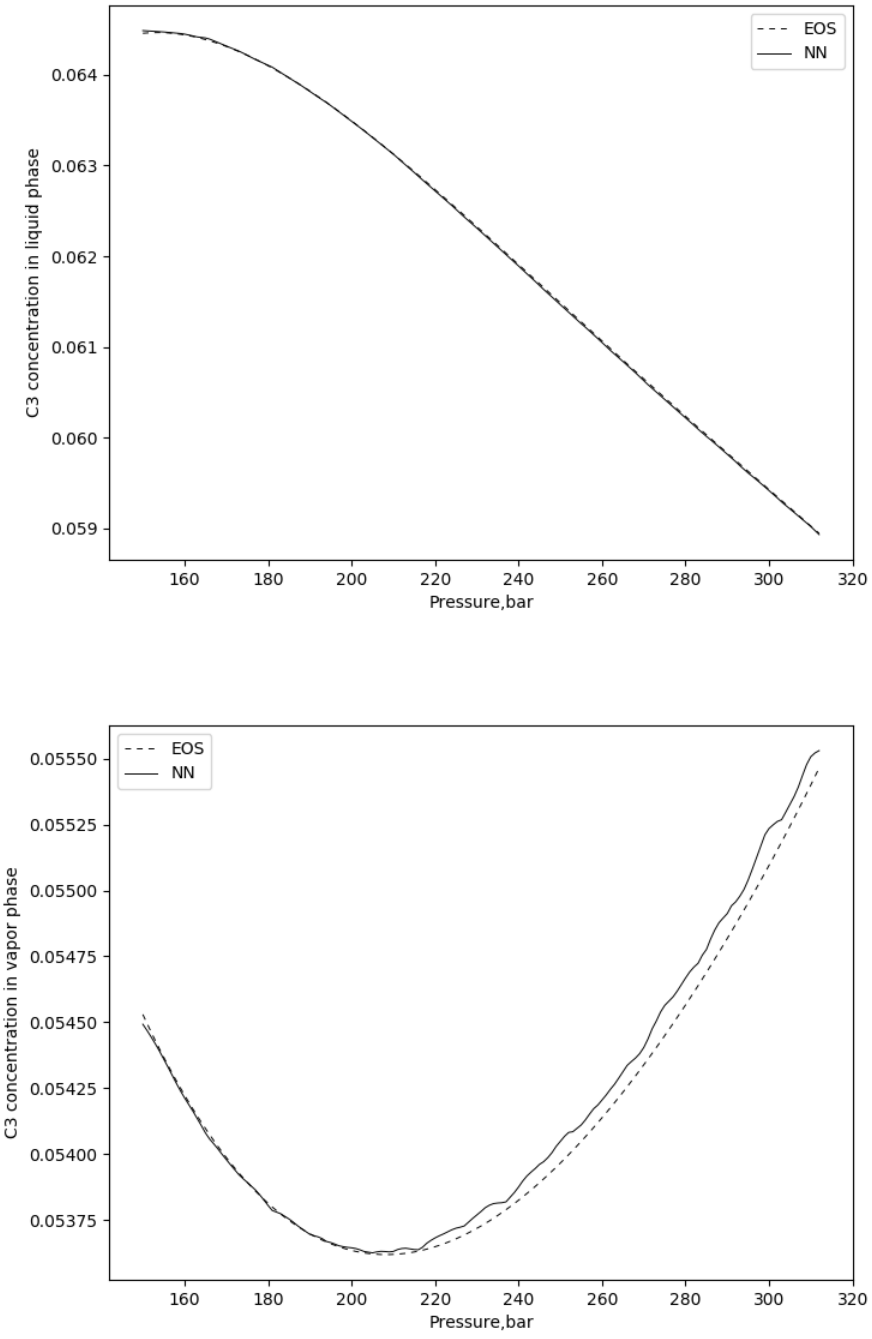


Figure 5.52 Phase mole fraction calculations comparison between EOS and ANN. Top: Liquid $C_{3.}$ Bottom: Vapor $C_{3.}$

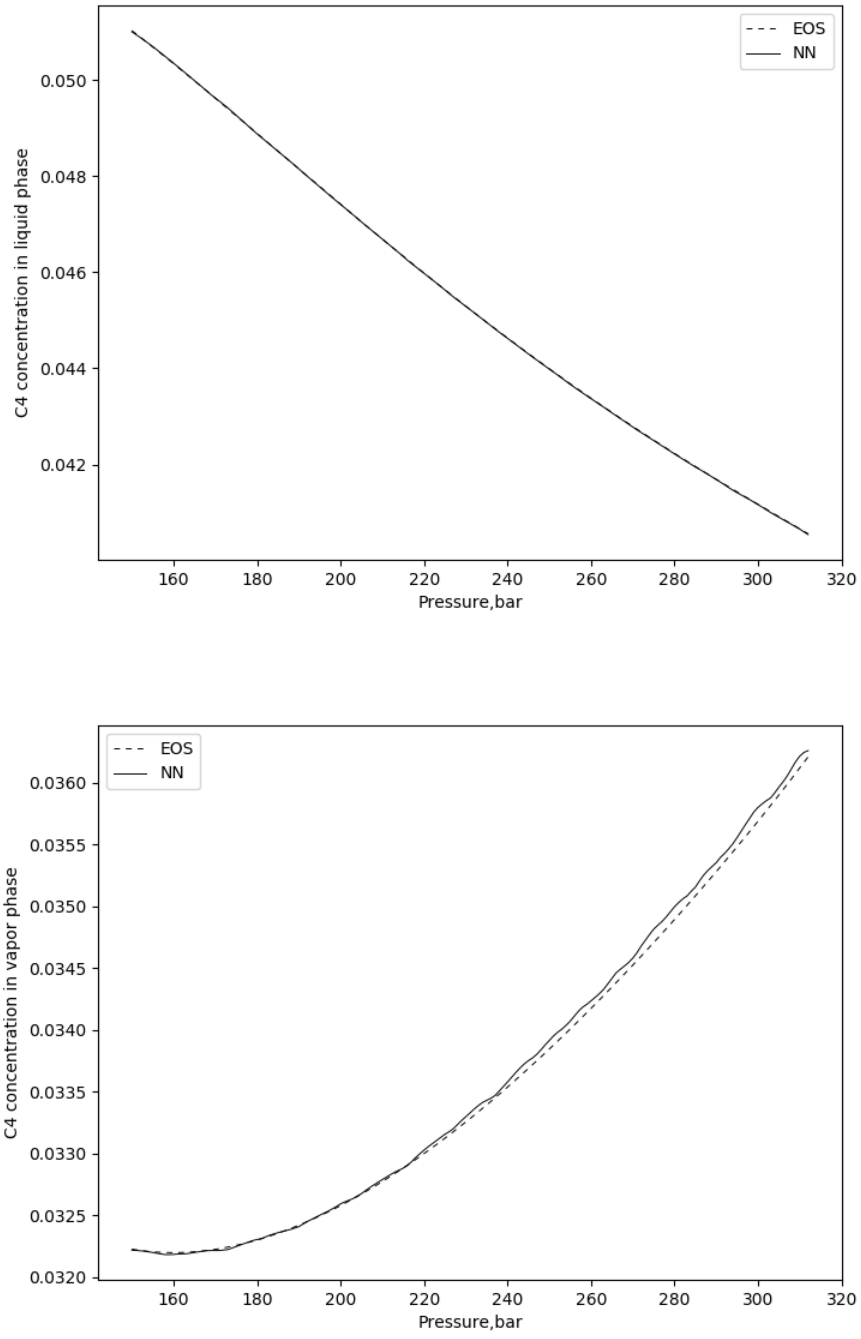


Figure 5.53 Phase mole fraction calculations comparison between EOS and ANN. Top: Liquid $C_{4.}$ Bottom: Vapor $C_{4.}$

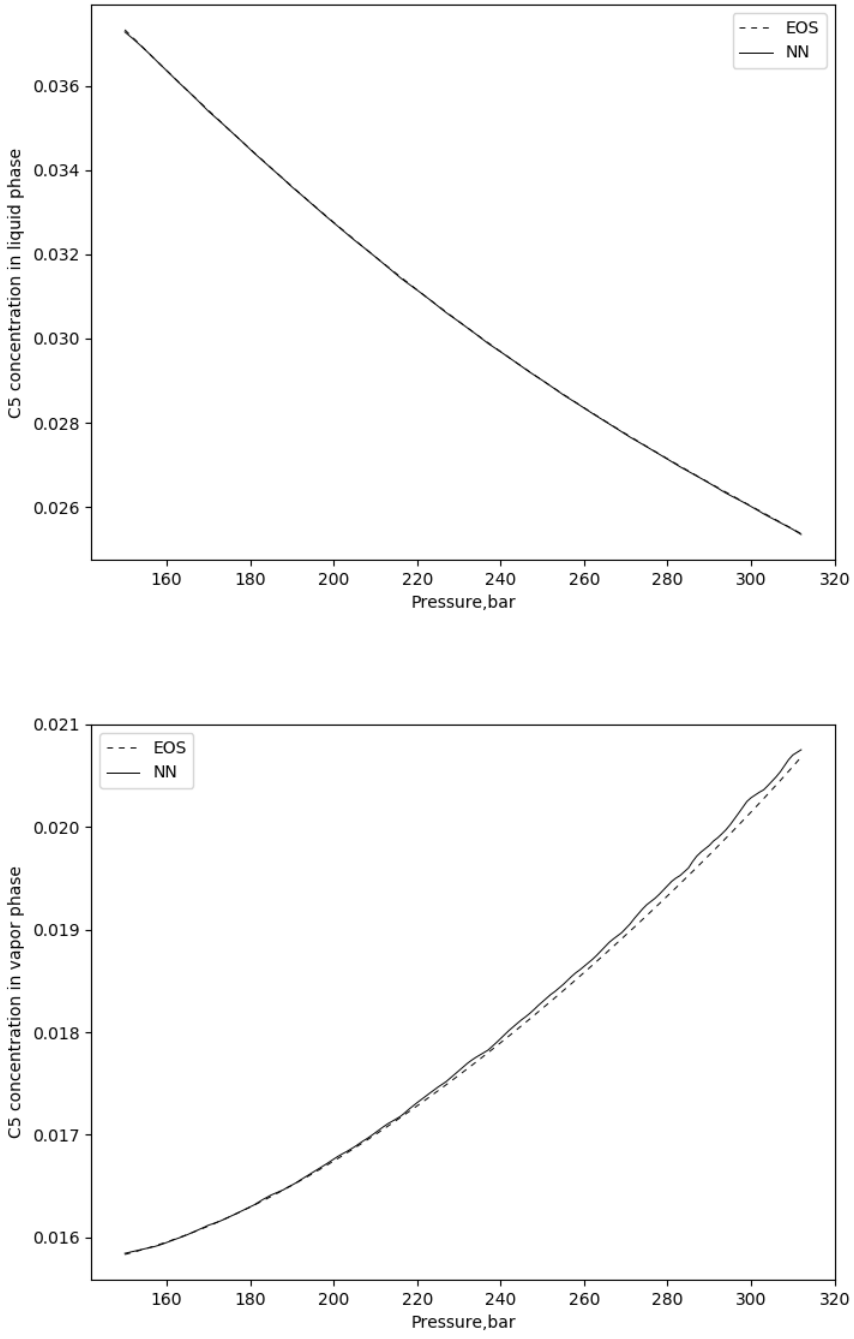


Figure 5.54 Phase mole fraction calculations comparison between EOS and ANN. Top: Liquid $C_{5.}$ Bottom: Vapor $C_{5.}$

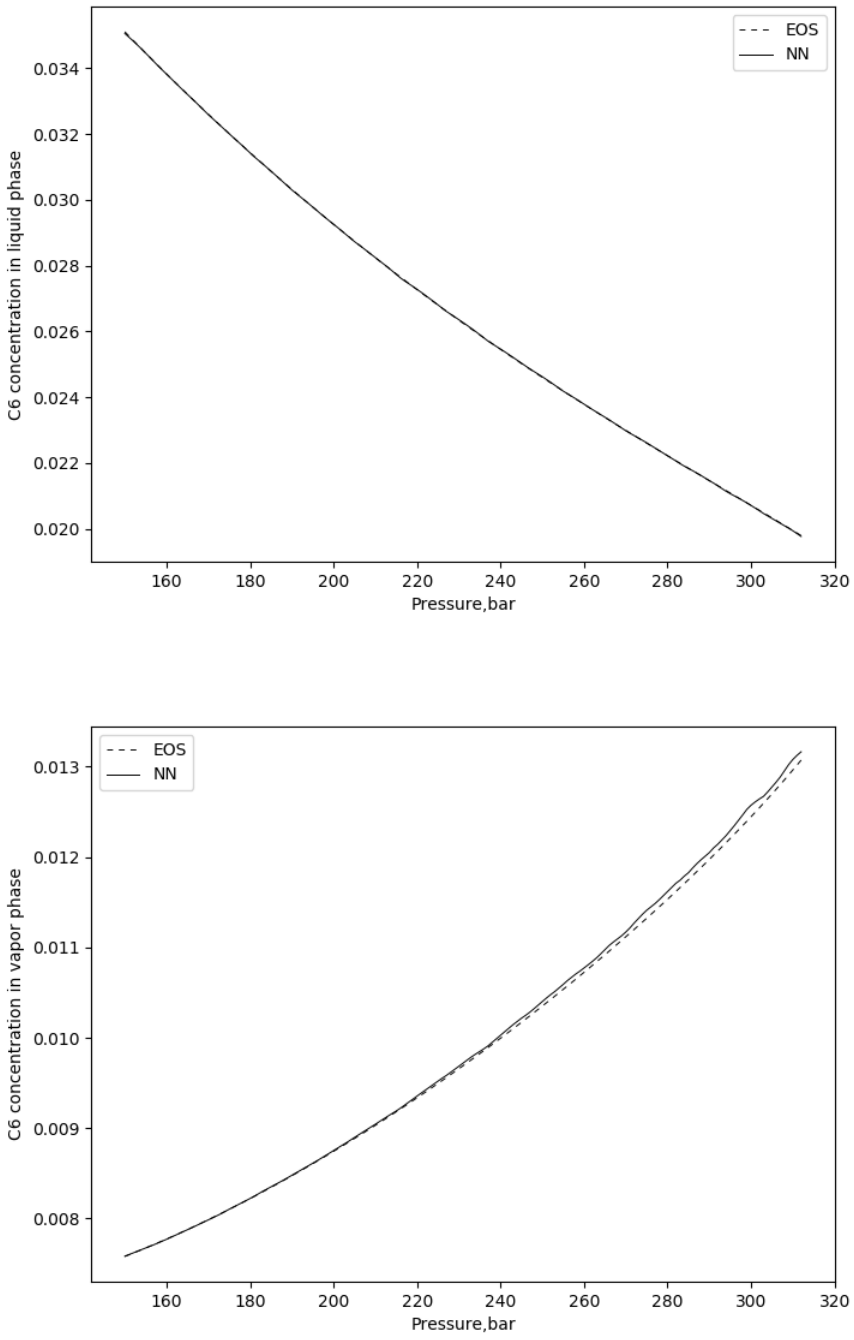


Figure 5.55 Phase mole fraction calculations comparison between EOS and ANN. Top: Liquid $C_{6.}$ Bottom: Vapor $C_{6.}$

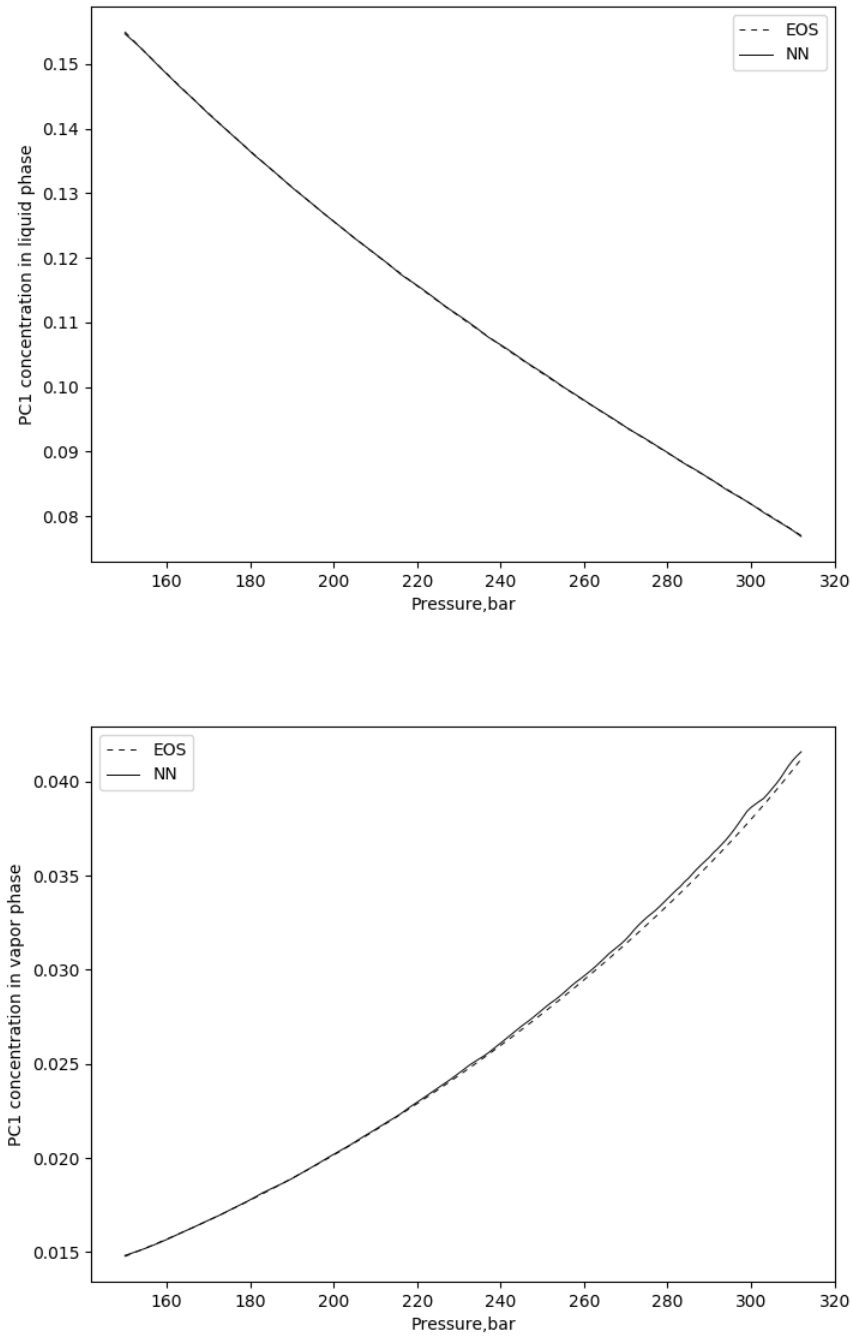


Figure 5.56 Phase mole fraction calculations comparison between EOS and ANN. Top: Liquid Pseudo Component 1. Bottom: Vapor Pseudo Component 1.

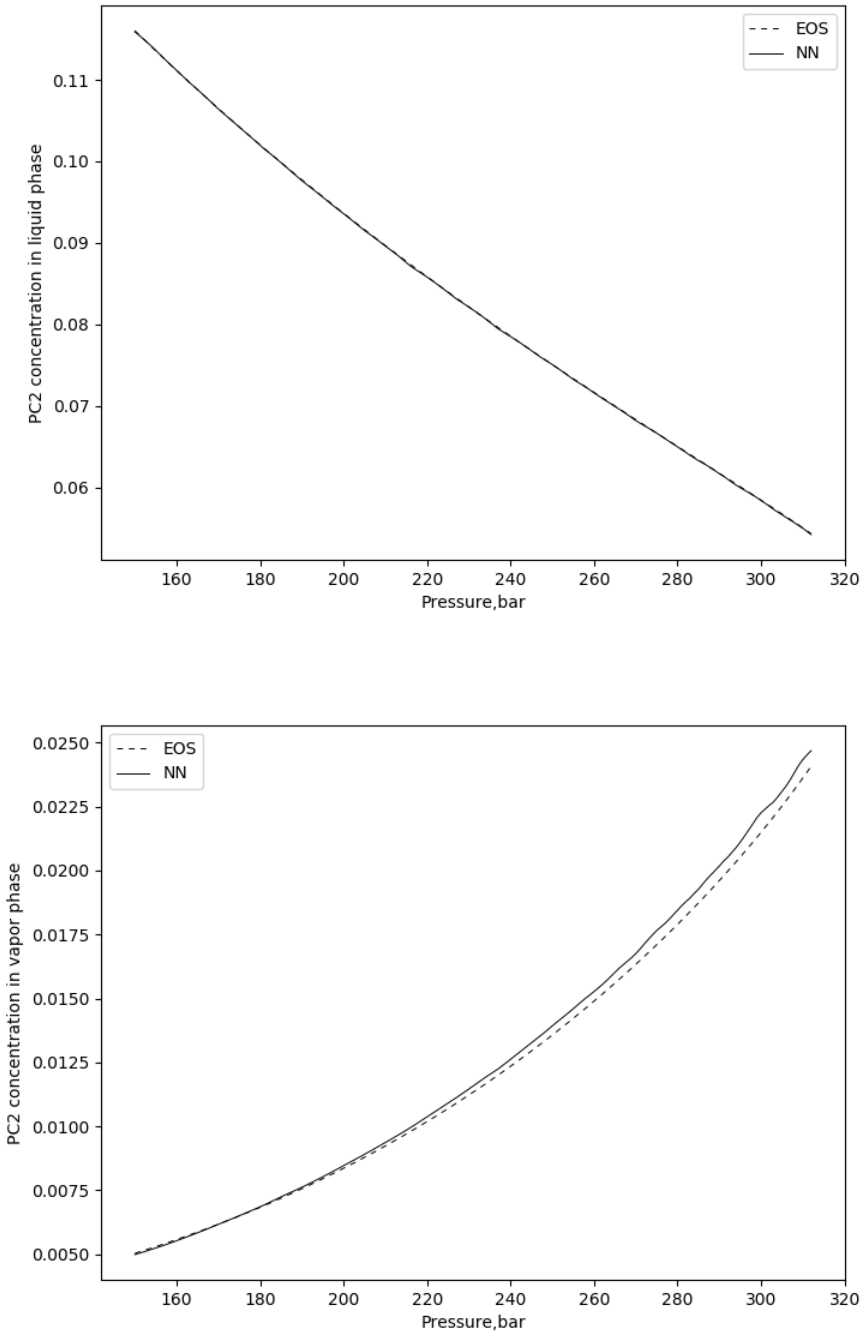


Figure 5.57 Phase mole fraction calculations comparison between EOS and ANN. Top: Liquid Pseudo Component 2. Bottom: Vapor Pseudo Component 2.

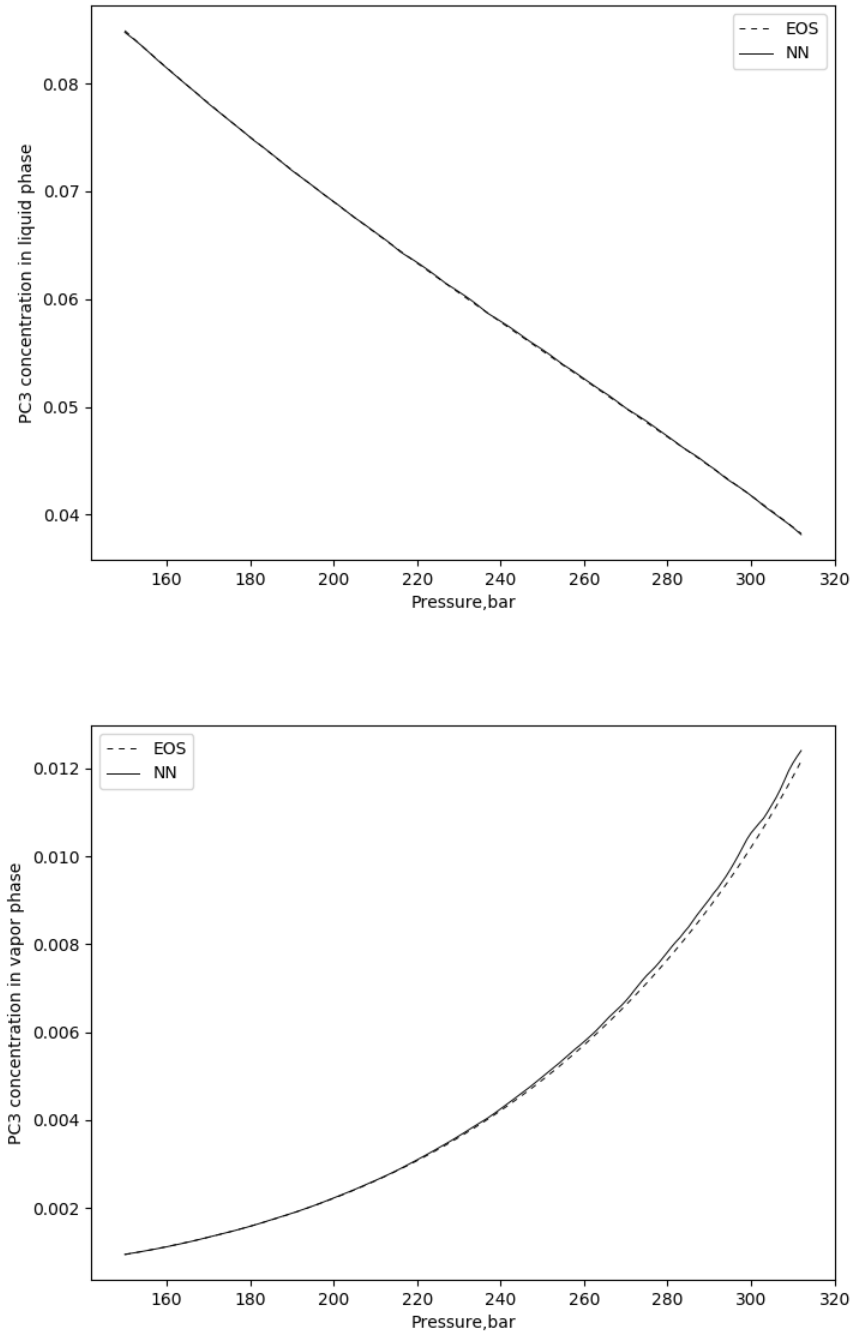


Figure 5.58 Phase mole fraction calculations comparison between EOS and ANN. Top: Liquid Pseudo Component 3. Bottom: Vapor Pseudo Component 3.

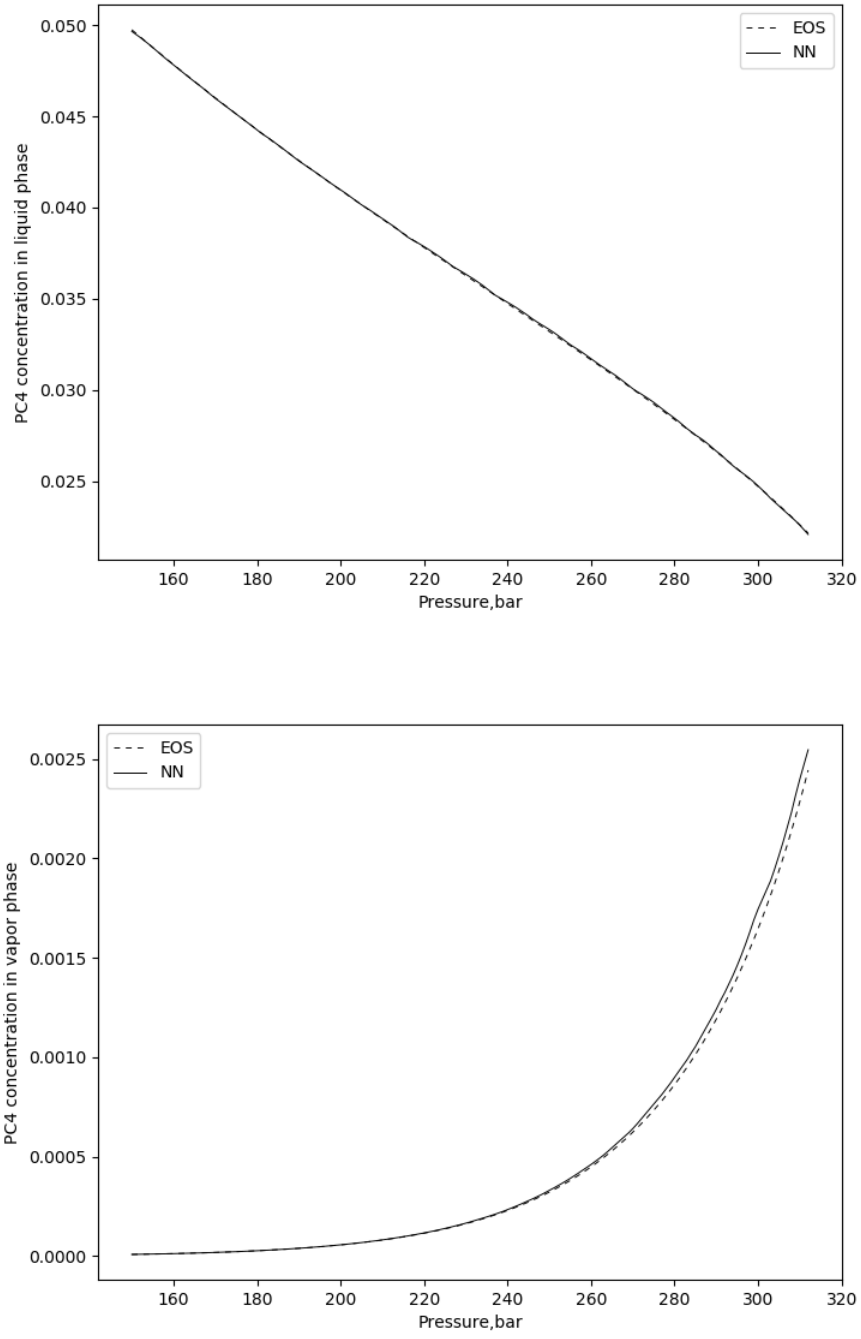


Figure 5.59 Phase mole fraction calculations comparison between EOS and ANN. Top: Liquid Pseudo Component 4. Bottom: Vapor Pseudo Component 4.

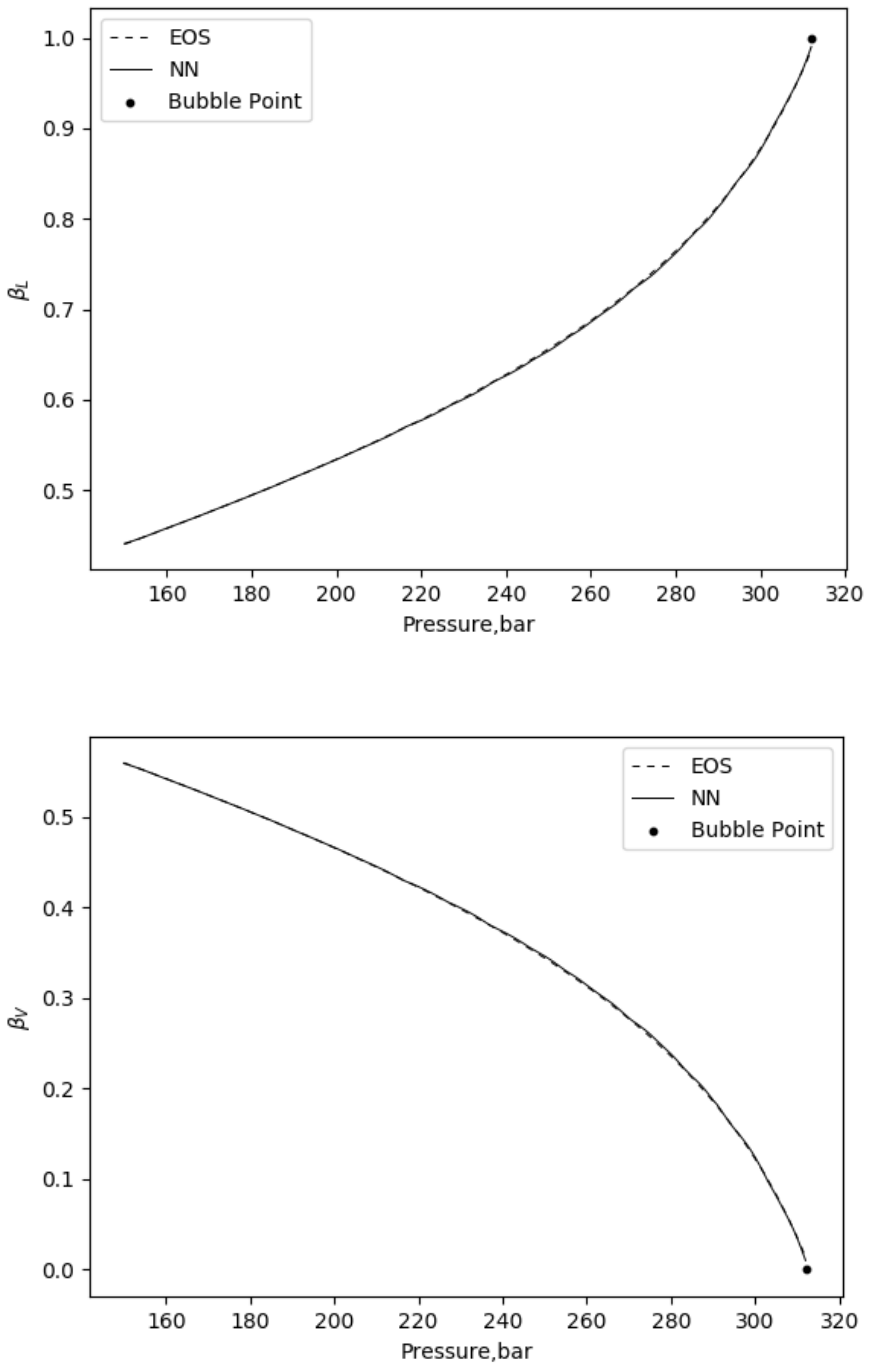


Figure 5.60 Phase mole fraction calculation comparison between EOS and ANN. Top: Liquid mole fraction. Bottom: Vapor mole fraction.

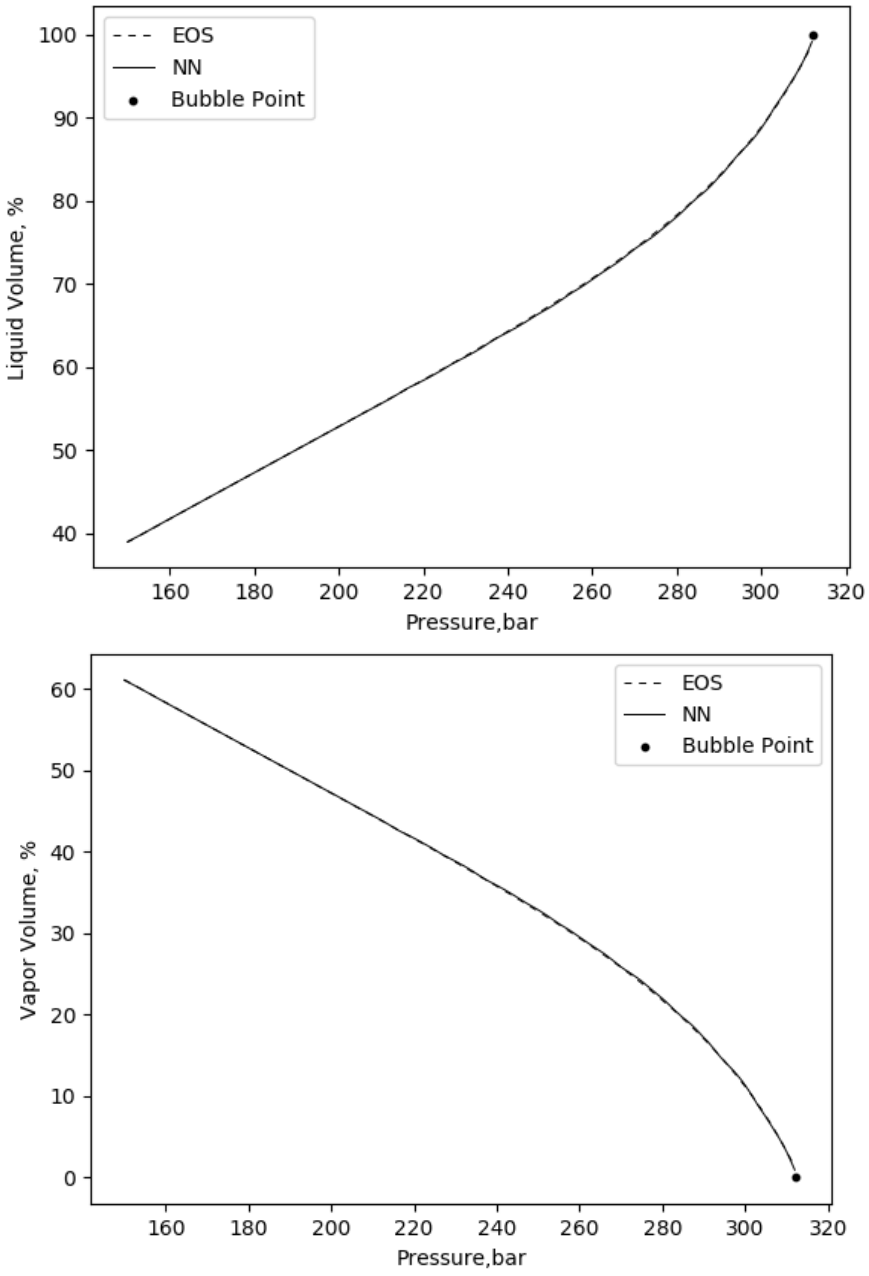


Figure 5.61 Fluid saturation comparison between EOS and ANN. Top: Vapor saturation. Bottom: Liquid Saturation.

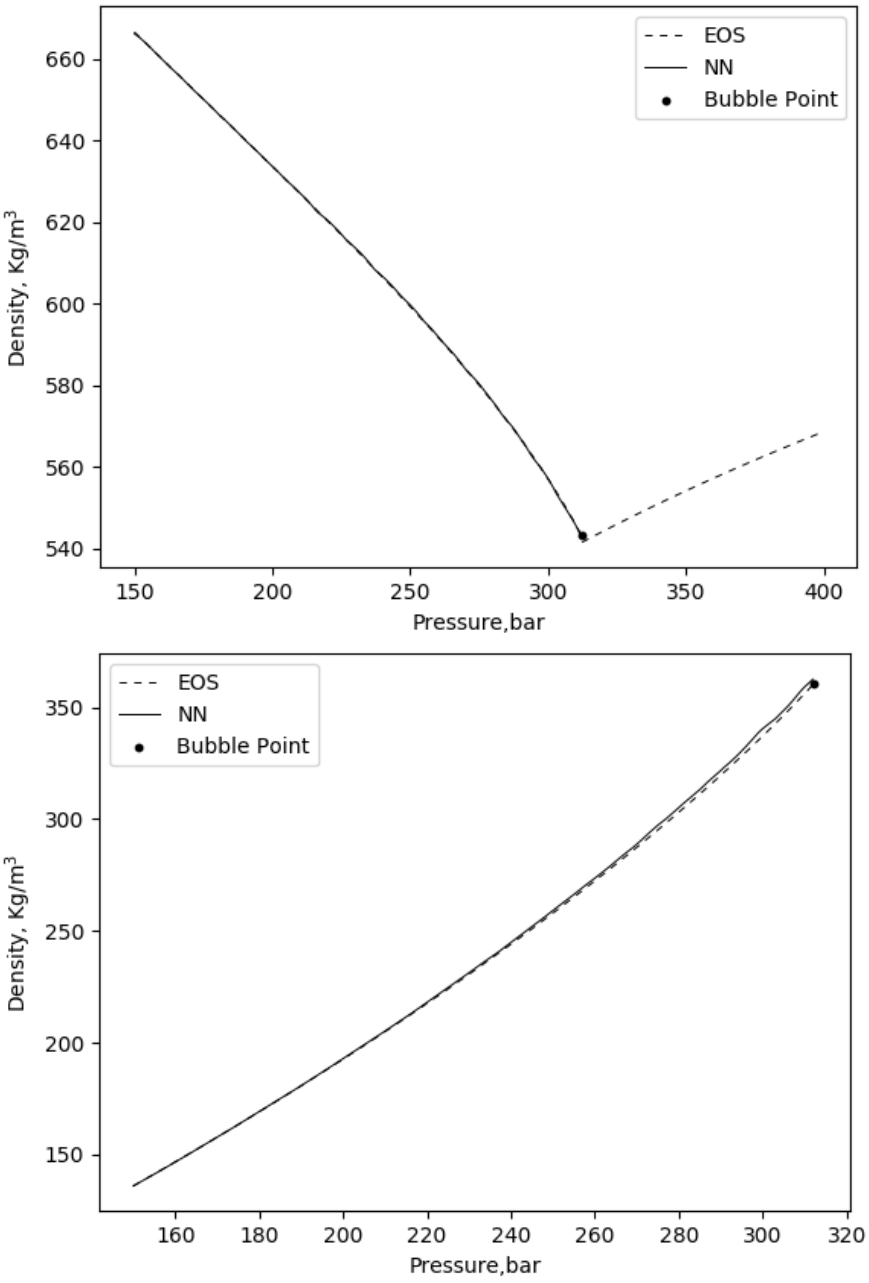


Figure 5.62 Fluid density calculation comparison between EOS and ANN. Top: Liquid density. Bottom: Vapor density.

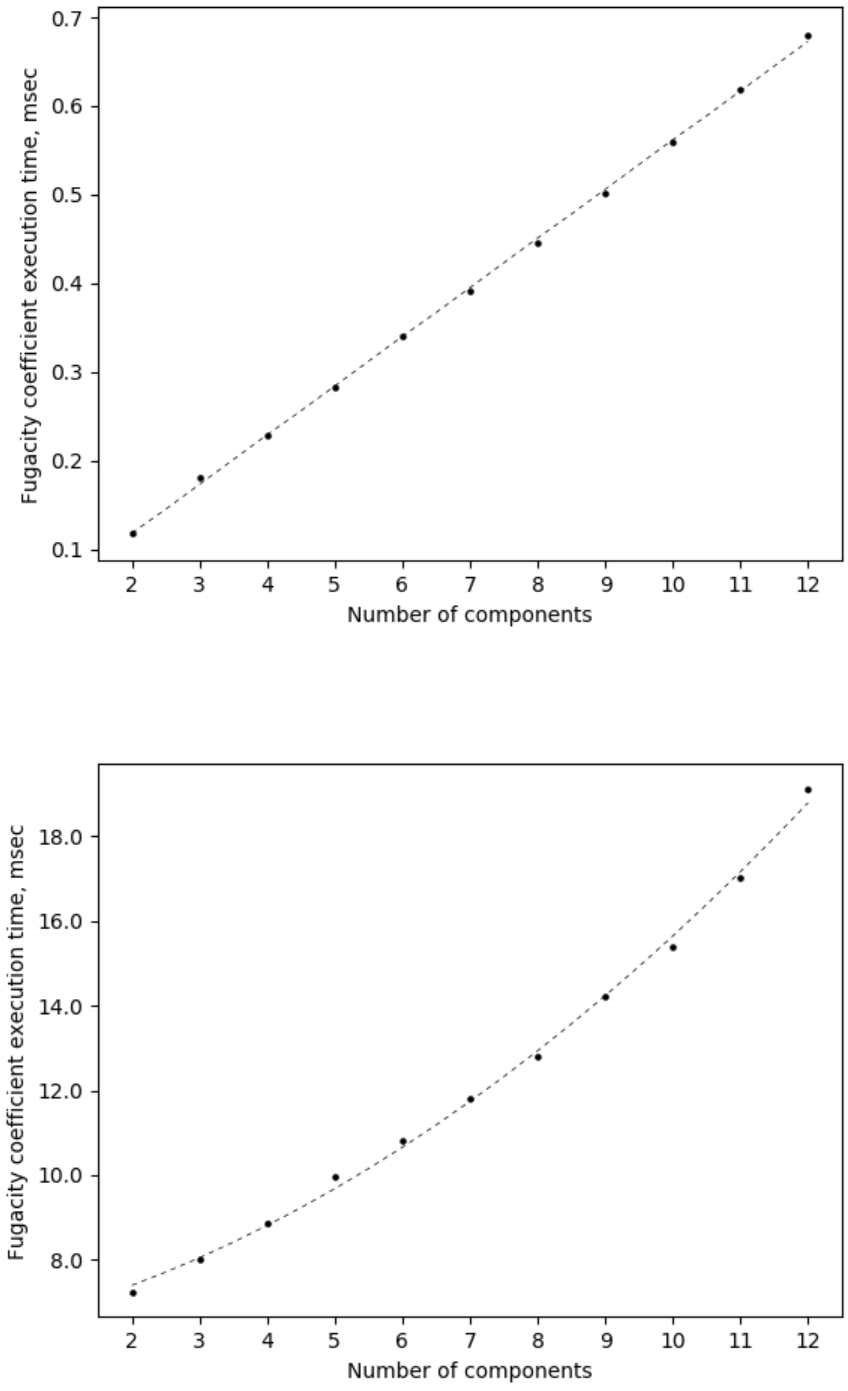


Figure 5.63 Fugacity coefficient execution time.
Top: ANNs. Bottom: EOS

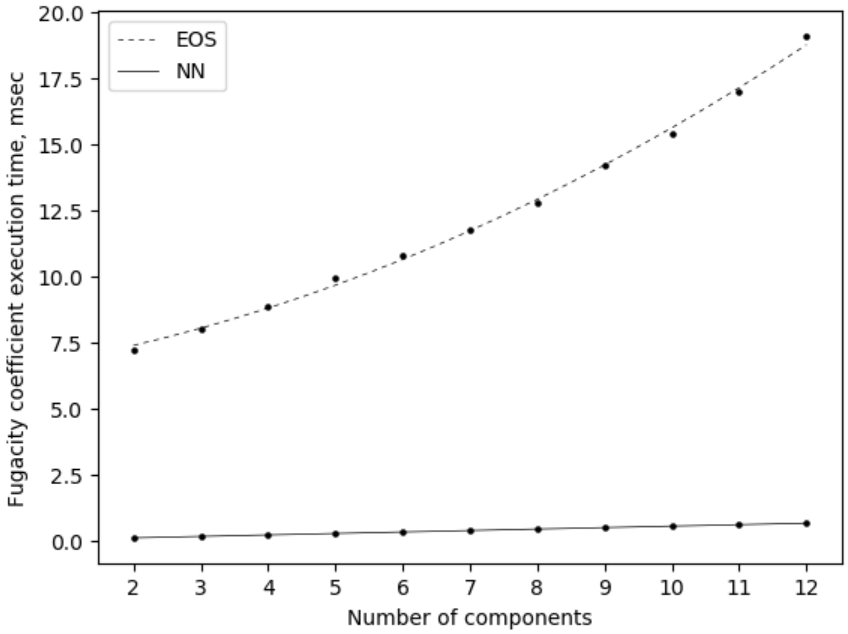


Figure 5.64 Fugacity coefficient execution time comparison between EOS and ANN

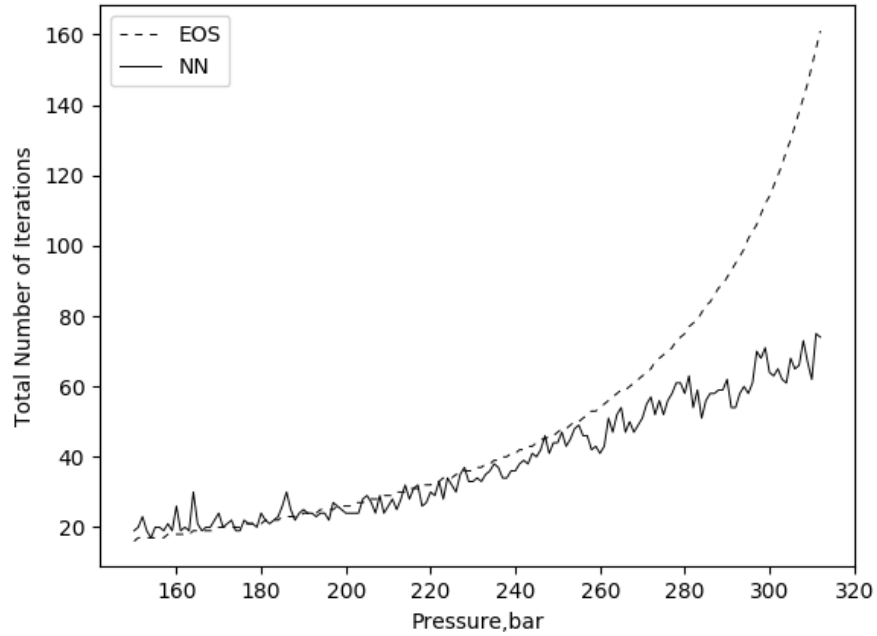


Figure 5.65 Total number of iterations to reach solution of conventional EOS method and ANN flash method.

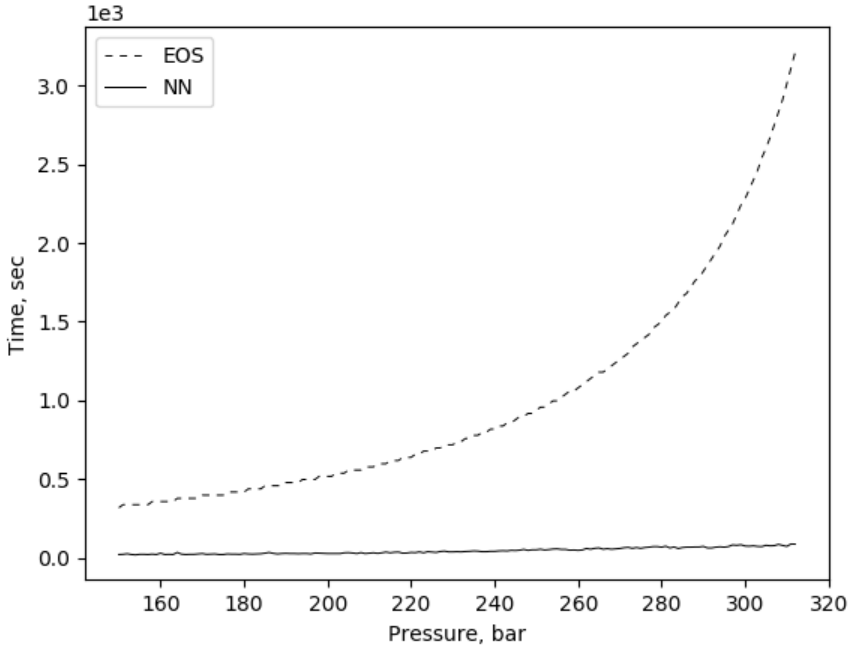


Figure 5.66 Total execution time to reach solution of conventional EOS method and ANN flash method.

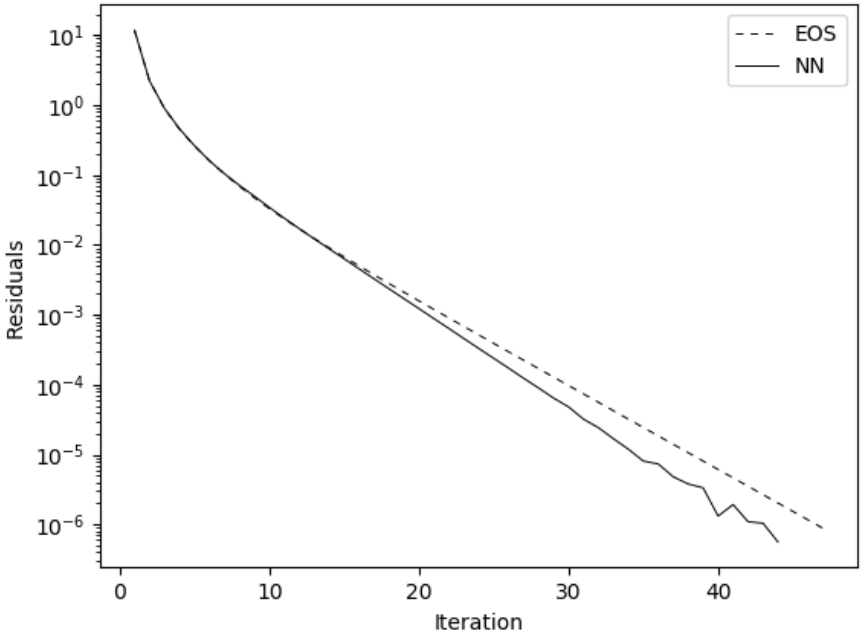


Figure 5.67 Convergence behavior comparison between EOS flash and ANN flash at a pressure of 220 Bar.

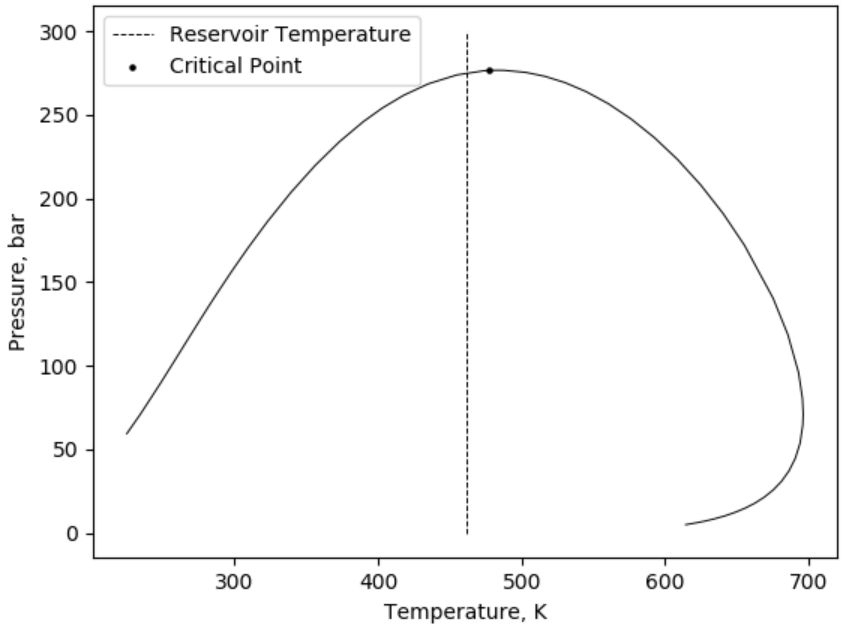


Figure 5.68. Phase diagram of study case number 4.

Table 5.12. Fluid model for case 4.

Component	Overall Composition	MW, g/mol	Tc, K	Pc, Bar	Acentric Factor
C_1	0.535520867	16.04	190.6	46	0.008
C_2	0.102474249	30.07	305.4	48.84	0.098
C_3	0.092917065	44.1	369.8	42.46	0.152
C_4	0.062546459	58.12	418.35	37.39	0.1862
C_5	0.031326325	72.15	464.67	33.79	0.2381
C_6	0.024317723	86.18	507.4	29.69	0.296
PC-1	0.058829776	117.36	618.51	28.86	0.1747
PC-2	0.043219709	159.75	689.76	25.04	0.2428
PC-3	0.030901561	223.2	773.46	21.68	0.3325
PC-4	0.017946267	385.02	988.48	16.64	0.5766

Table 5.13. Generalization error of fugacity coefficients from ANNs

Component	Mean Square error	Average percentage error	R^2
C_1	6.9050940309405e-09	0.02803580156022059	0.99999955691239
C_2	5.3506857562917e-09	0.06287203172845639	0.9999988518461894
C_3	4.6323974385562e-09	0.0065057960460917	0.9999985060862664
C_4	8.0573148746983e-09	0.0054783202854876405	0.9999985701122961
C_5	8.6815557873929e-09	0.004550404478930354	0.9999992904208203
C_6	2.94897871988366e-08	0.006768406455592809	0.9999987166076977
PC-1	6.1702269899789e-08	0.006320129632940961	0.9999990865435309
PC-2	7.9247650805181e-08	0.006278033830646585	0.9999993494488439
PC-3	1.42215462247054e-07	0.006249761794386665	0.9999993705621273
PC-4	1.22818904559822e-06	0.009917284658483893	0.9999985775998397

Table 5.14. Time per iteration comparison between EOS Flash and ANN Flash.

	EOS Flash	ANN Flash
Time per iteration, msec	16. 6958528948416	1.03413565577091

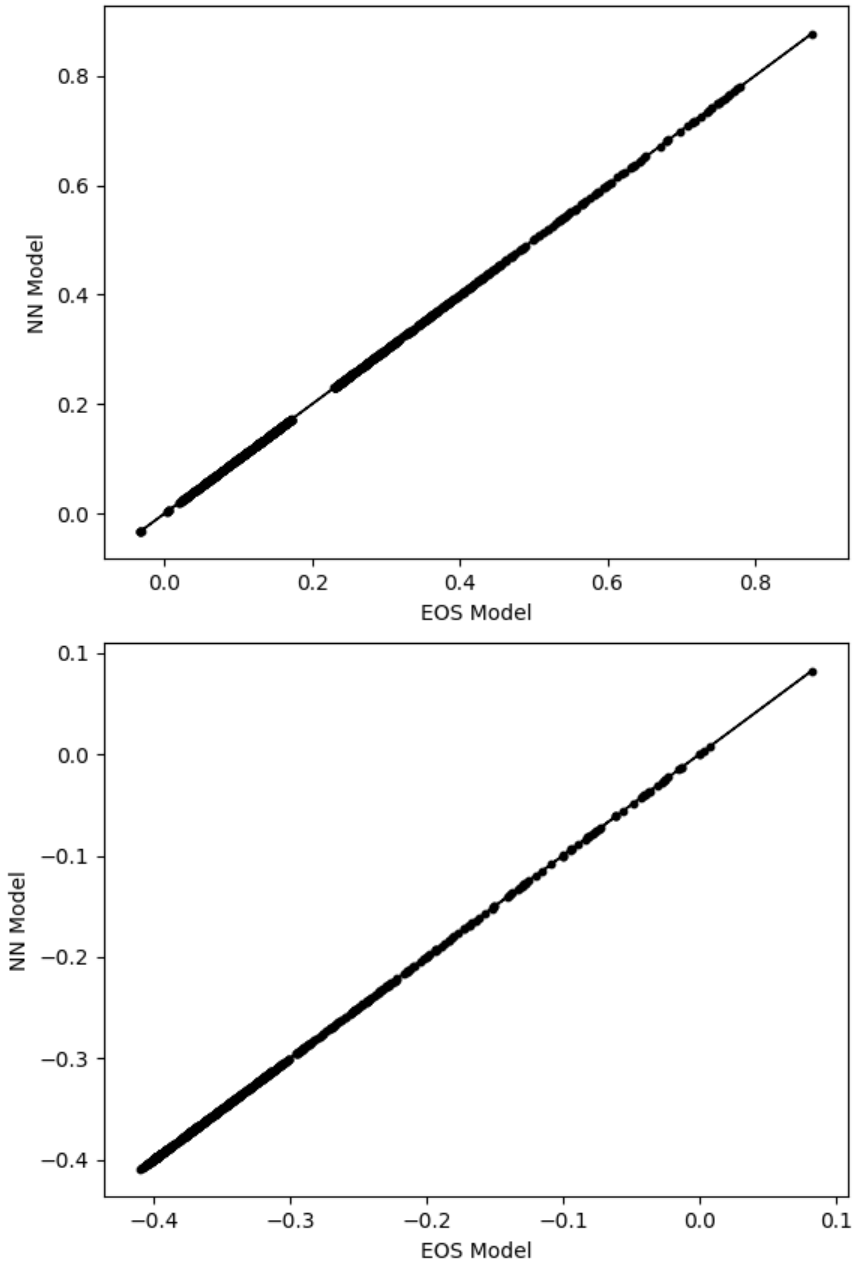
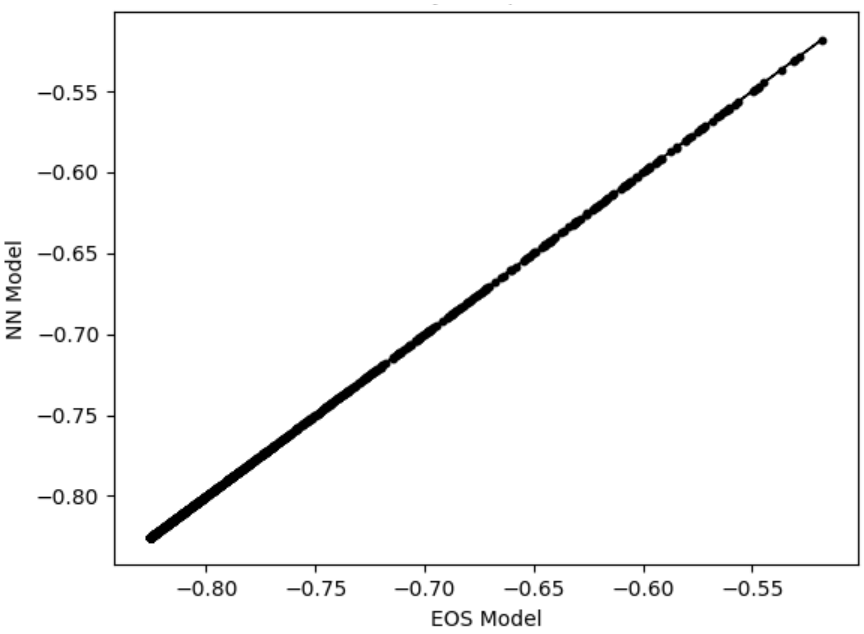


Figure 5.69 Accuracy comparison between fugacity coefficient calculated with EOS and ANN.
Top: Component $C_{1.}$ Bottom: Component $C_{2.}$



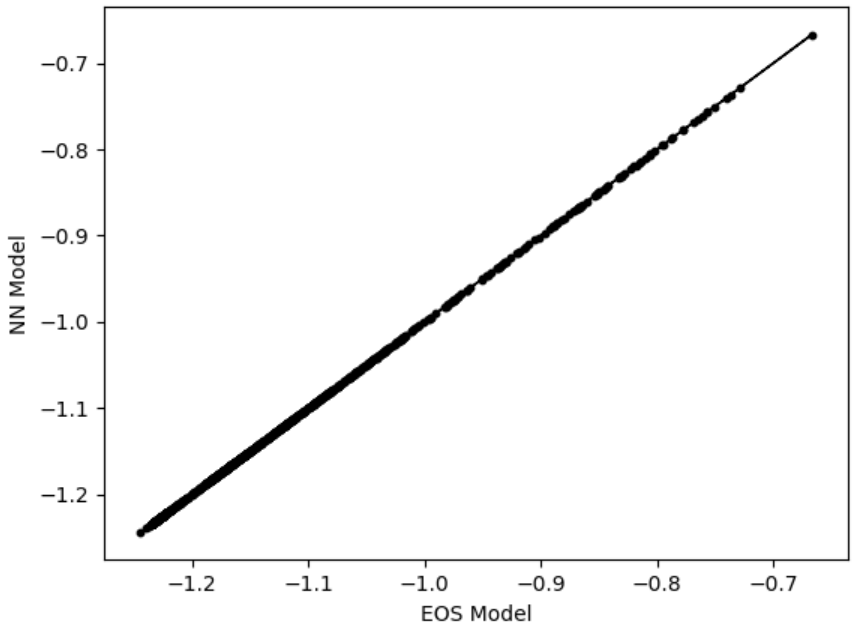


Figure 5.70 Accuracy comparison between fugacity coefficient calculated with EOS and ANN.
Top: Component C_3. Bottom: Component C_4.

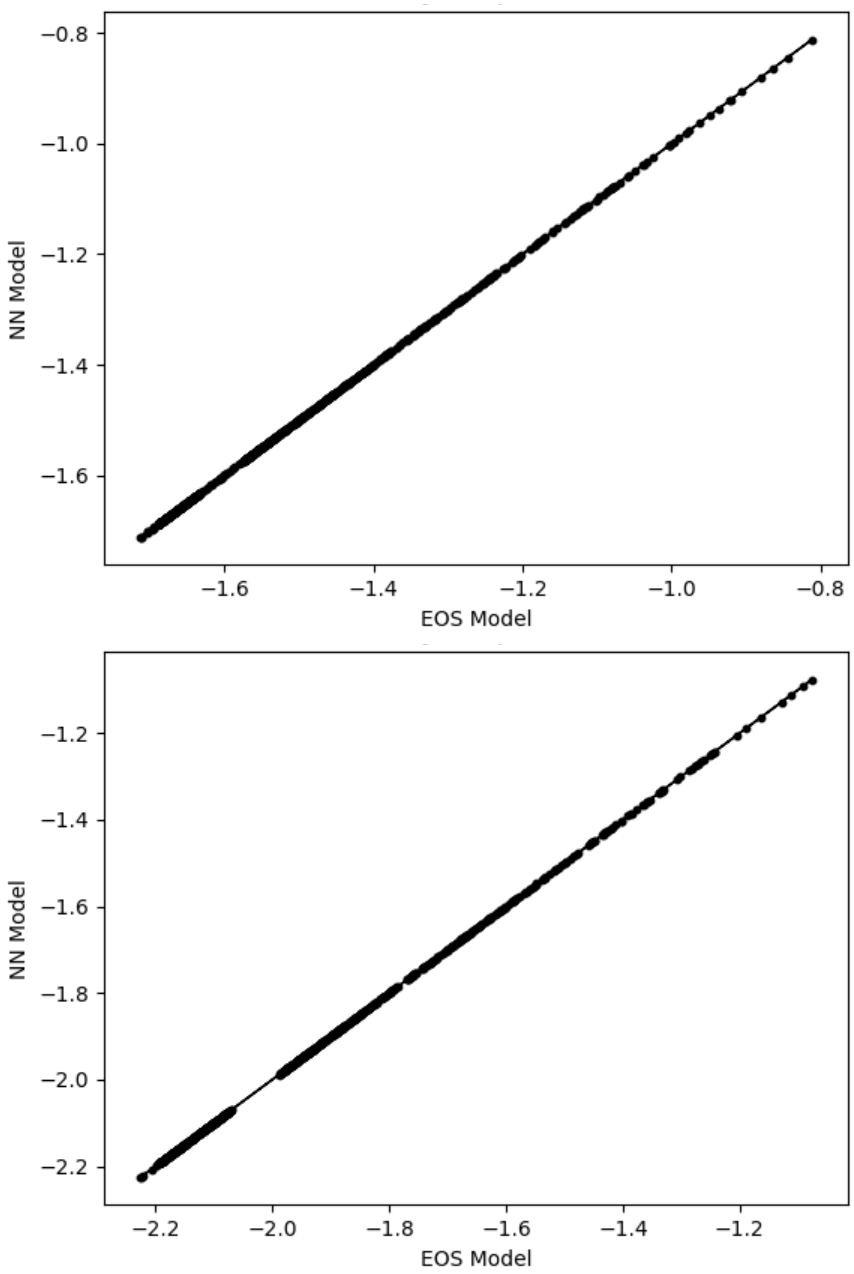


Figure 5.71 Accuracy comparison between fugacity coefficient calculated with EOS and ANN.
Top: Component C_5. Bottom: Component C_6.

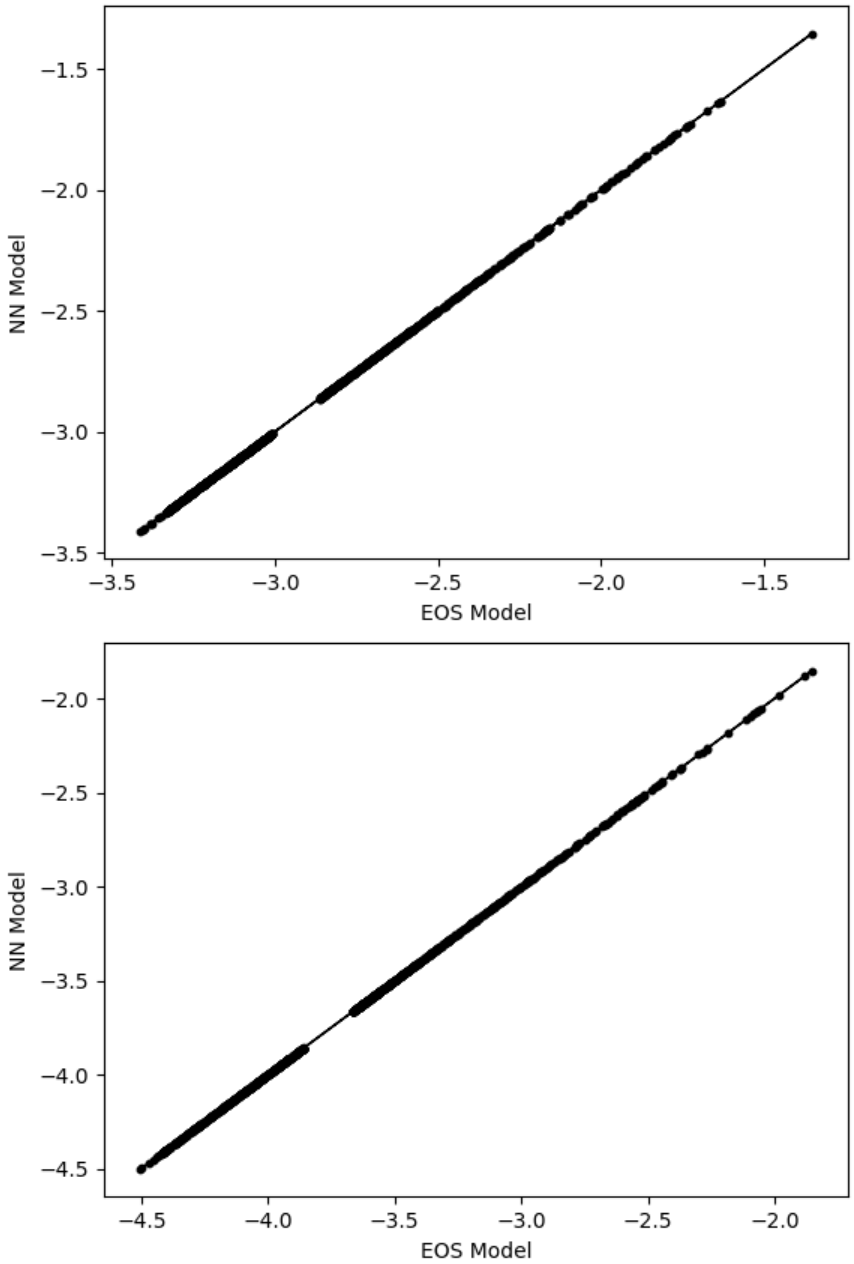


Figure 5.72 Accuracy comparison between fugacity coefficient calculated with EOS and ANN.
Top: Pseudo-Component 1. Bottom: Pseudo-Component 2.

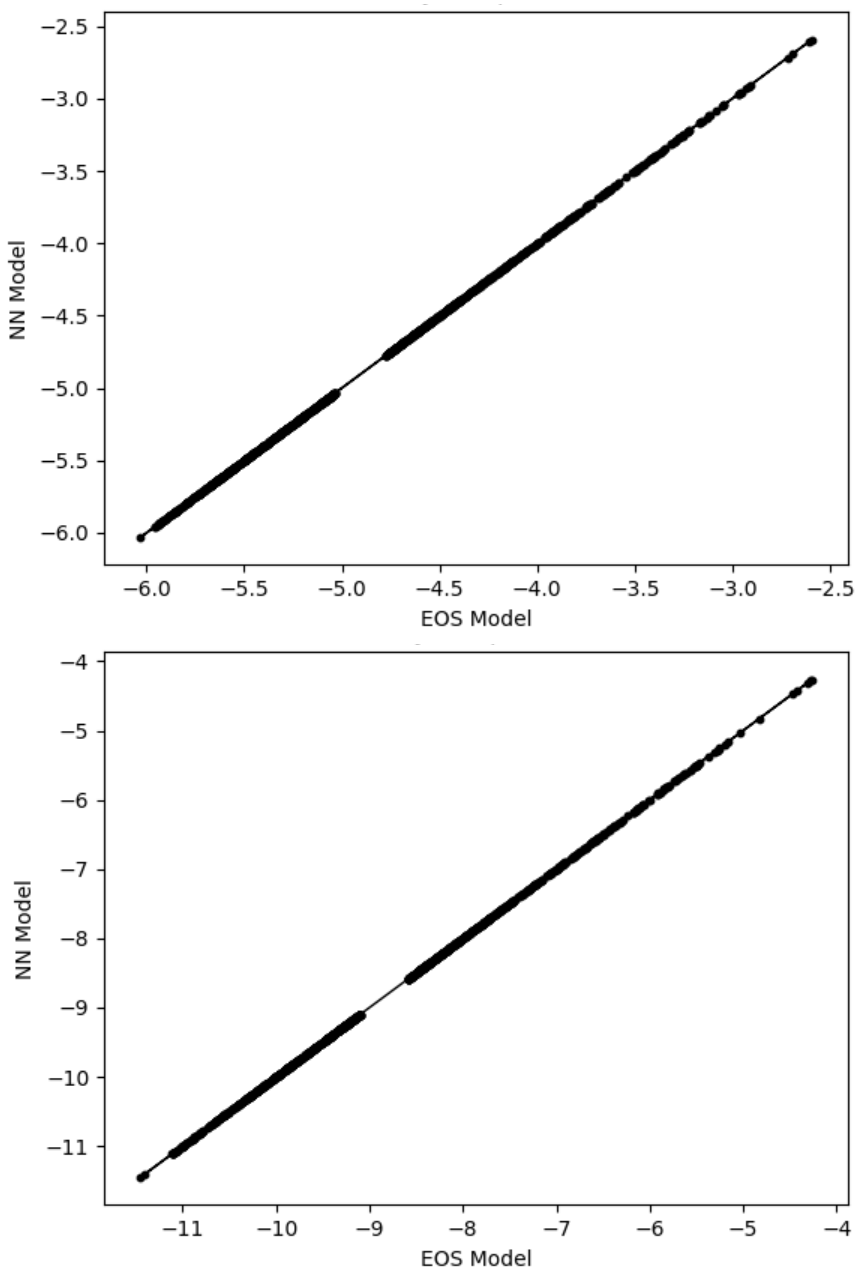


Figure 5.73 Accuracy comparison between fugacity coefficient calculated with EOS and ANN.
Top: Pseudo Component 3. Bottom: Pseudo Component 4.

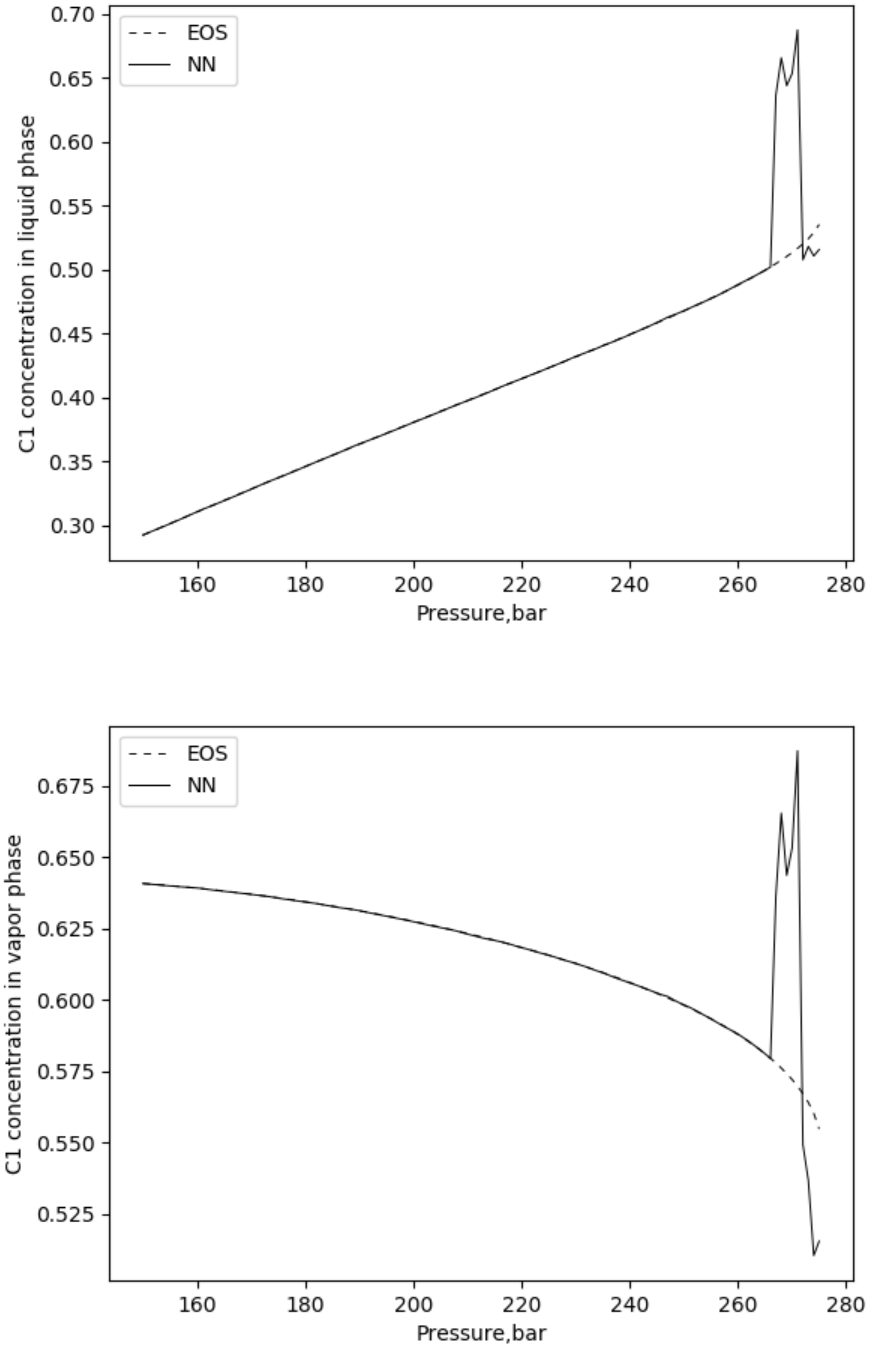


Figure 5.74 Phase mole fraction calculations comparison between EOS and ANN. Top: Liquid C_1. Bottom: Vapor C_1.

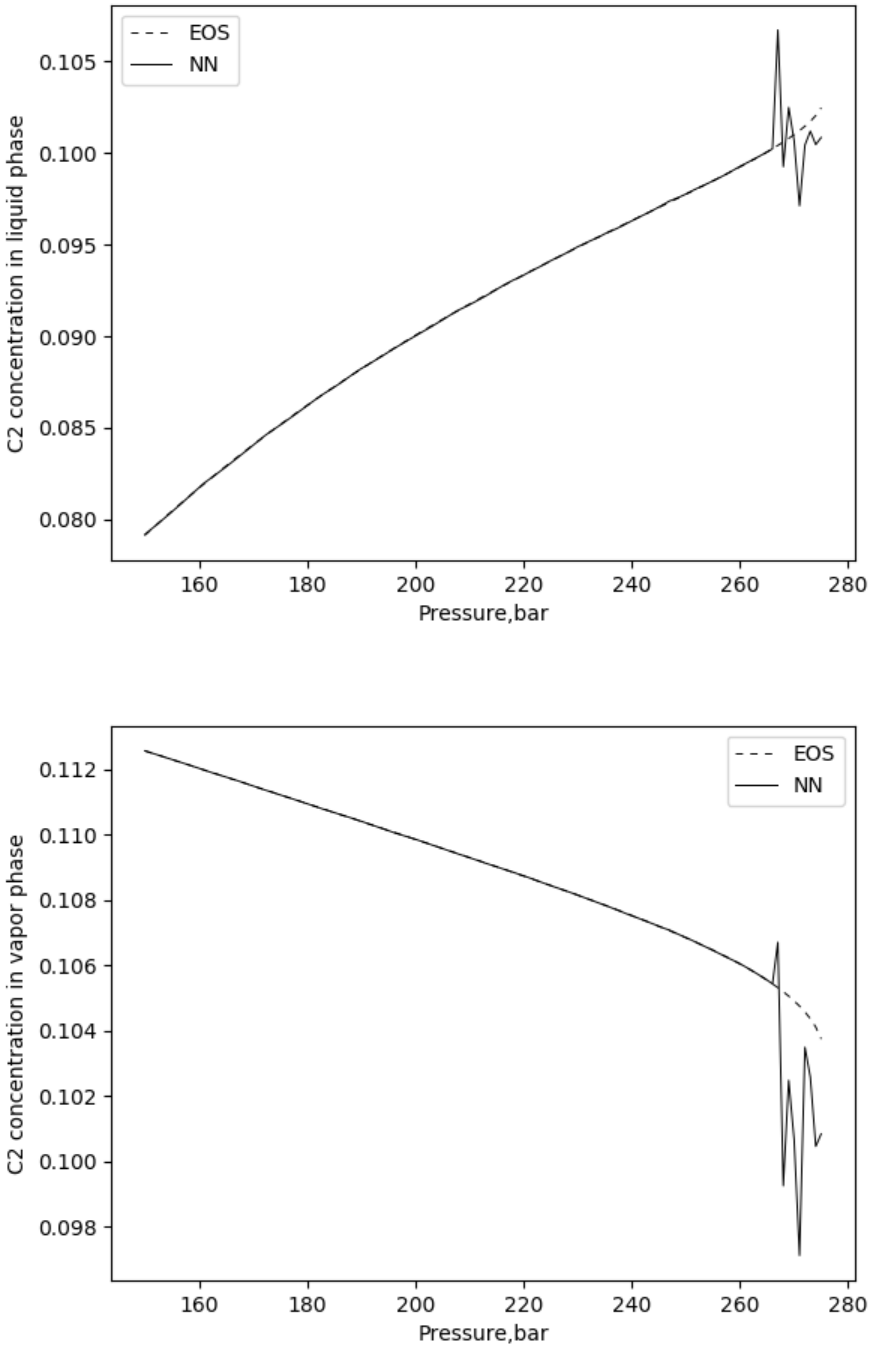


Figure 5.75 Phase mole fraction calculations comparison between EOS and ANN. Top: Liquid C_2 Bottom: Vapor C_2.

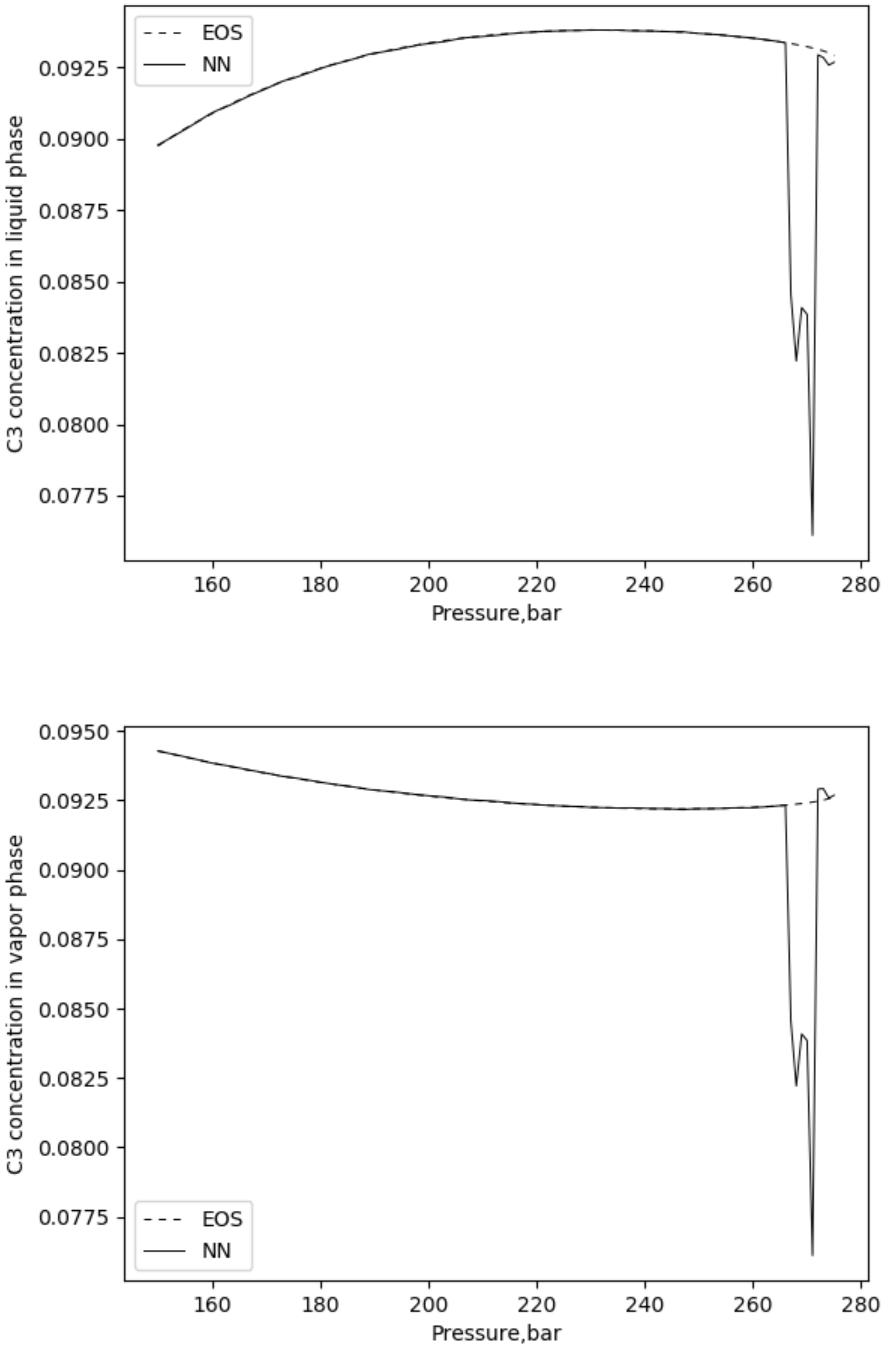


Figure 5.76 Phase mole fraction calculations comparison between EOS and ANN. Top: Liquid $C_{3.}$ Bottom: Vapor $C_{3.}$

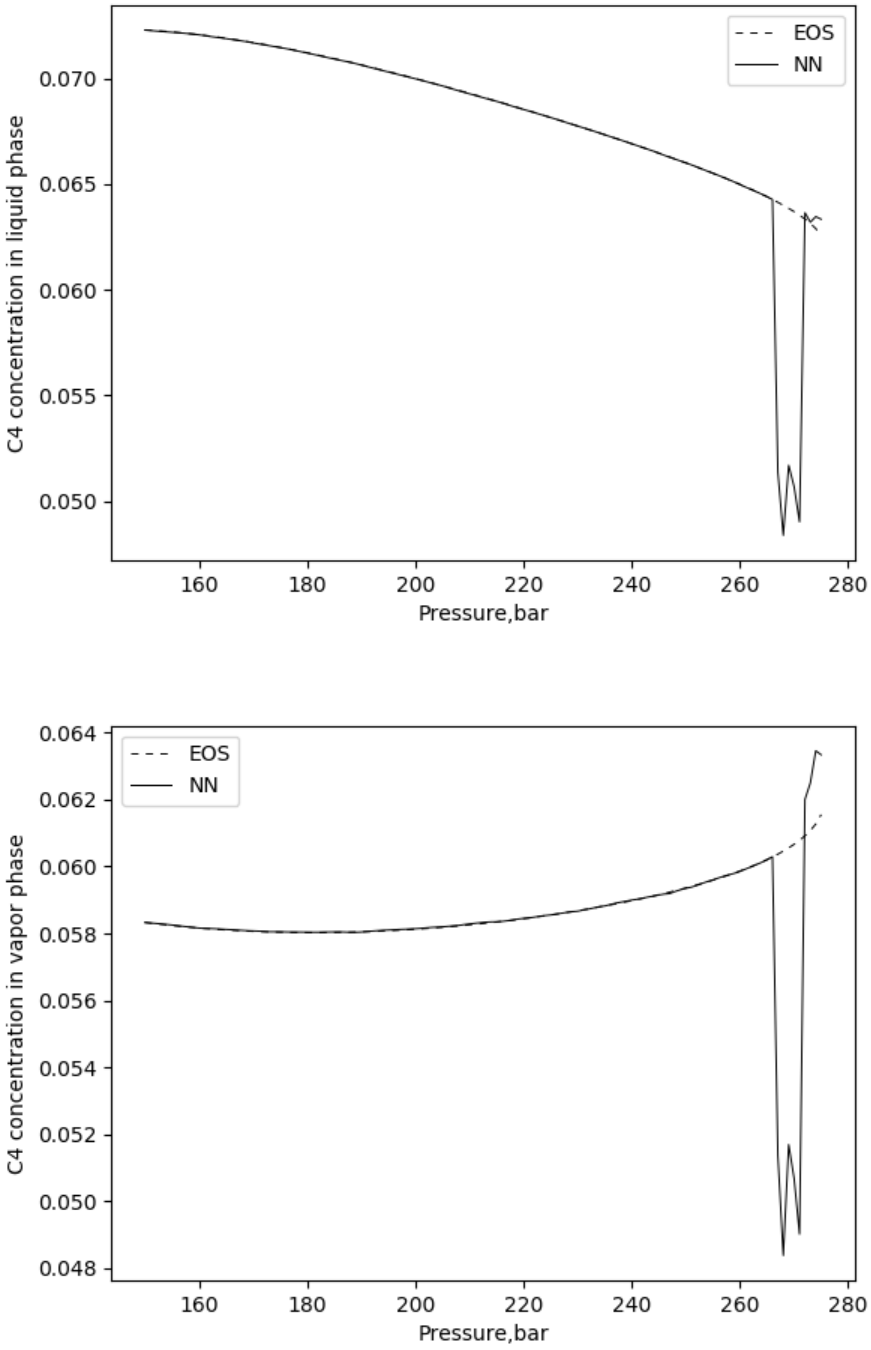


Figure 5.77 Phase mole fraction calculations comparison between EOS and ANN. Top: Liquid $C_{4.}$ Bottom: Vapor $C_{4.}$

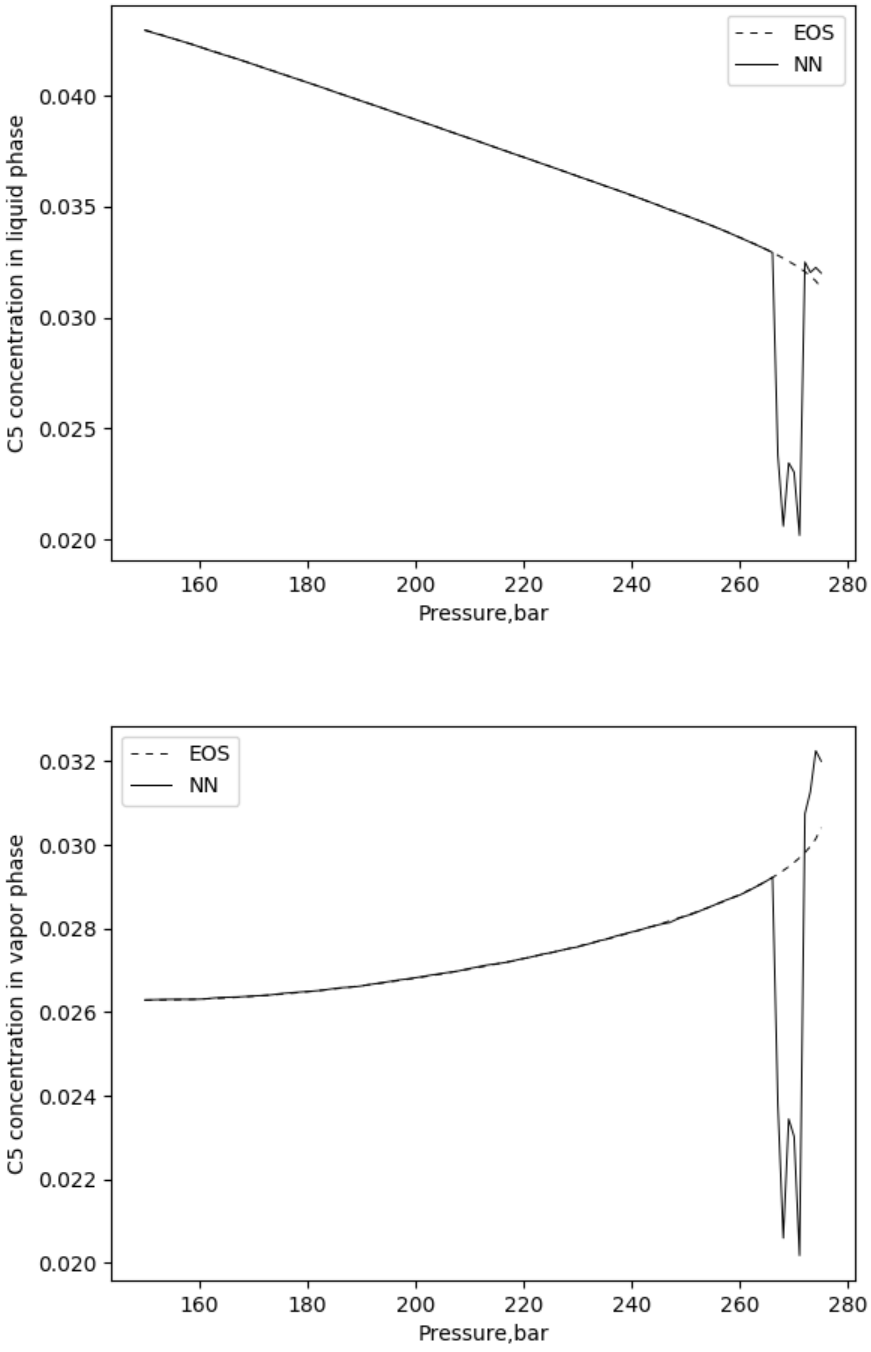


Figure 5.78 Phase mole fraction calculations comparison between EOS and ANN. Top: Liquid $C_{5.}$ Bottom: Vapor $C_{5.}$

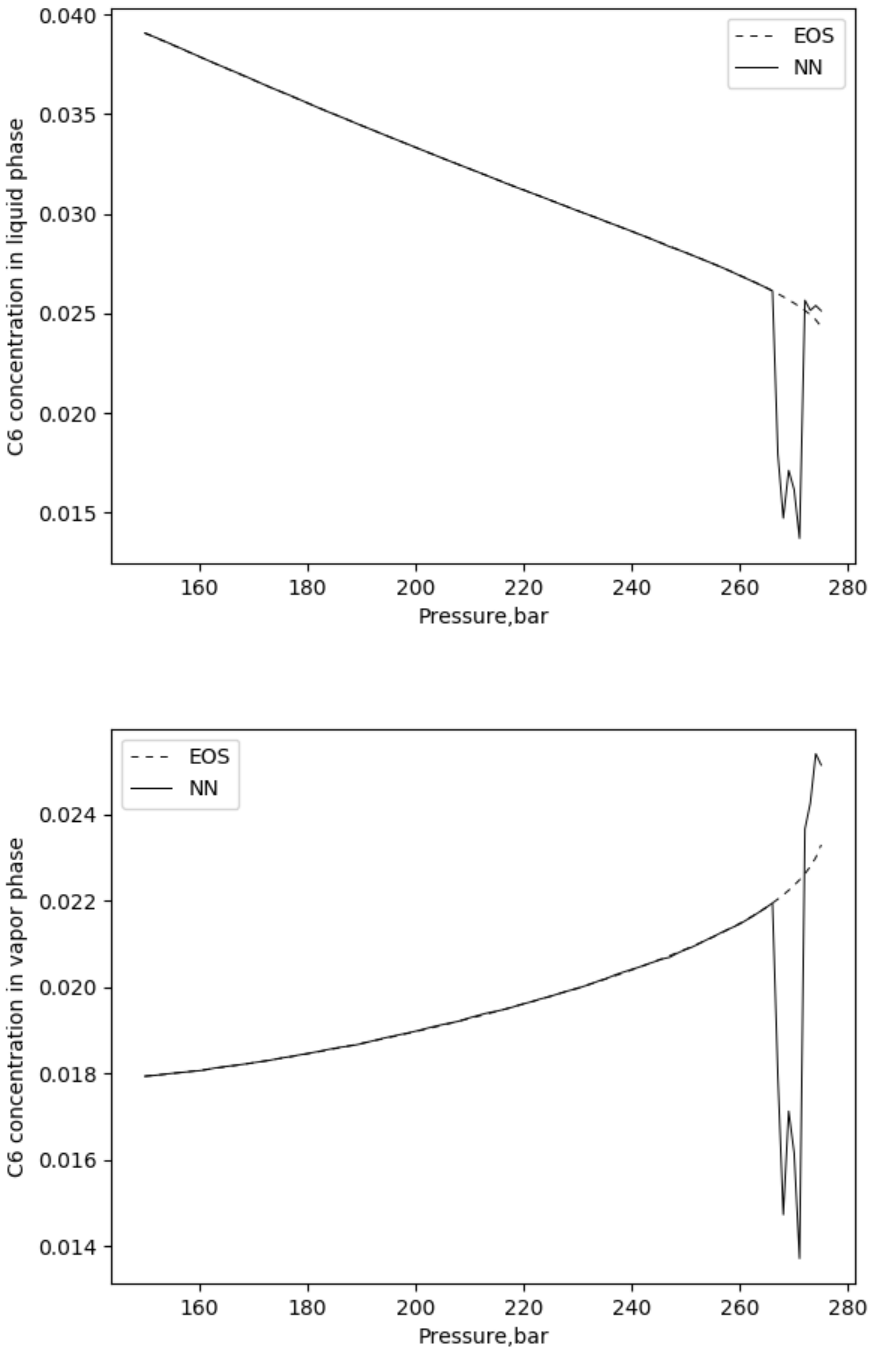


Figure 5.79 Phase mole fraction calculations comparison between EOS and ANN. Top: Liquid $C_{6.}$ Bottom: Vapor $C_{6.}$

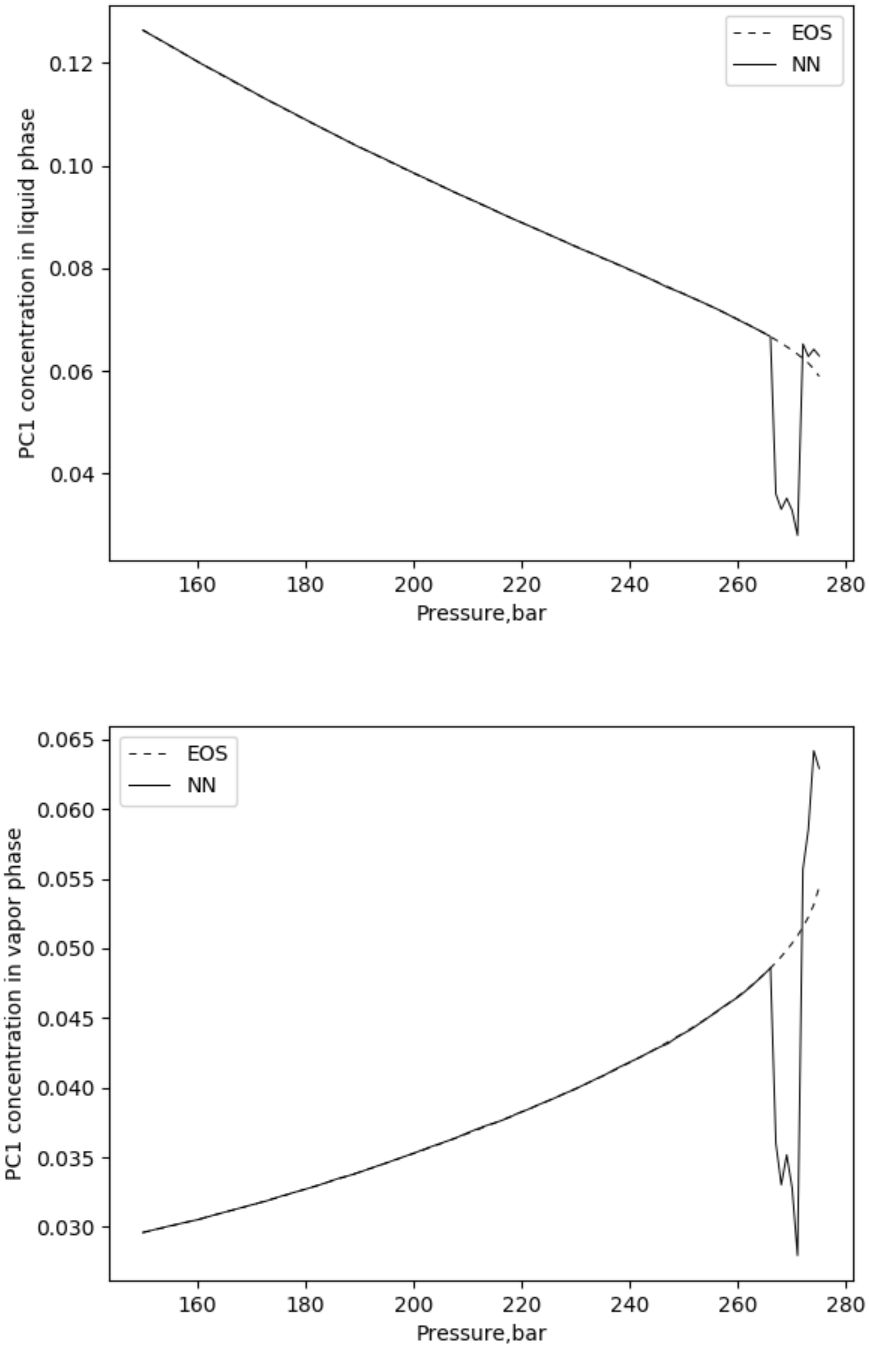


Figure 5.80 Phase mole fraction calculations comparison between EOS and ANN. Top: Liquid Pseudo Component 1. Bottom: Vapor Pseudo Component 1.

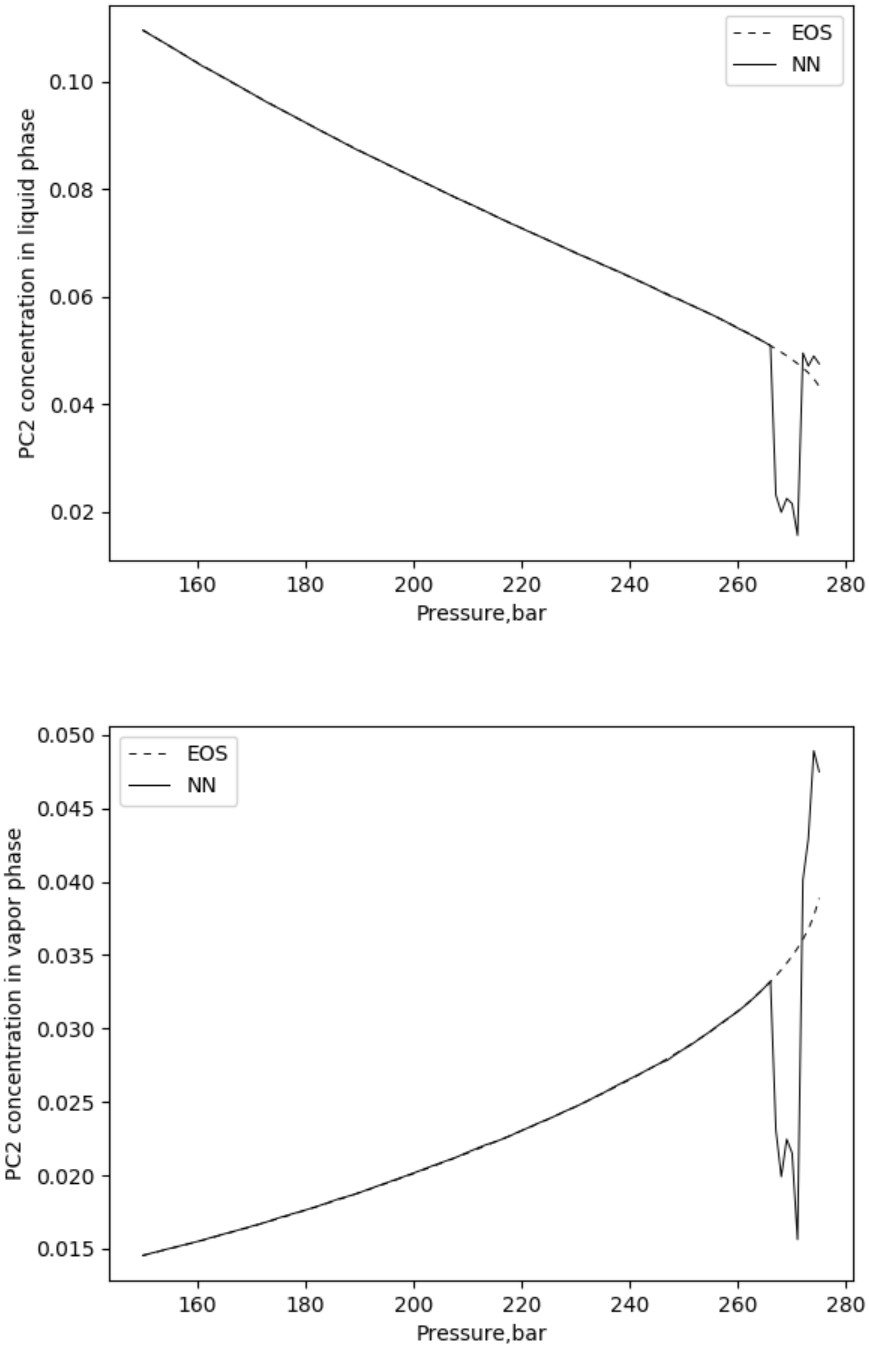


Figure 5.81 Phase mole fraction calculations comparison between EOS and ANN. Top: Liquid Pseudo Component 2. Bottom: Vapor Pseudo Component 2.

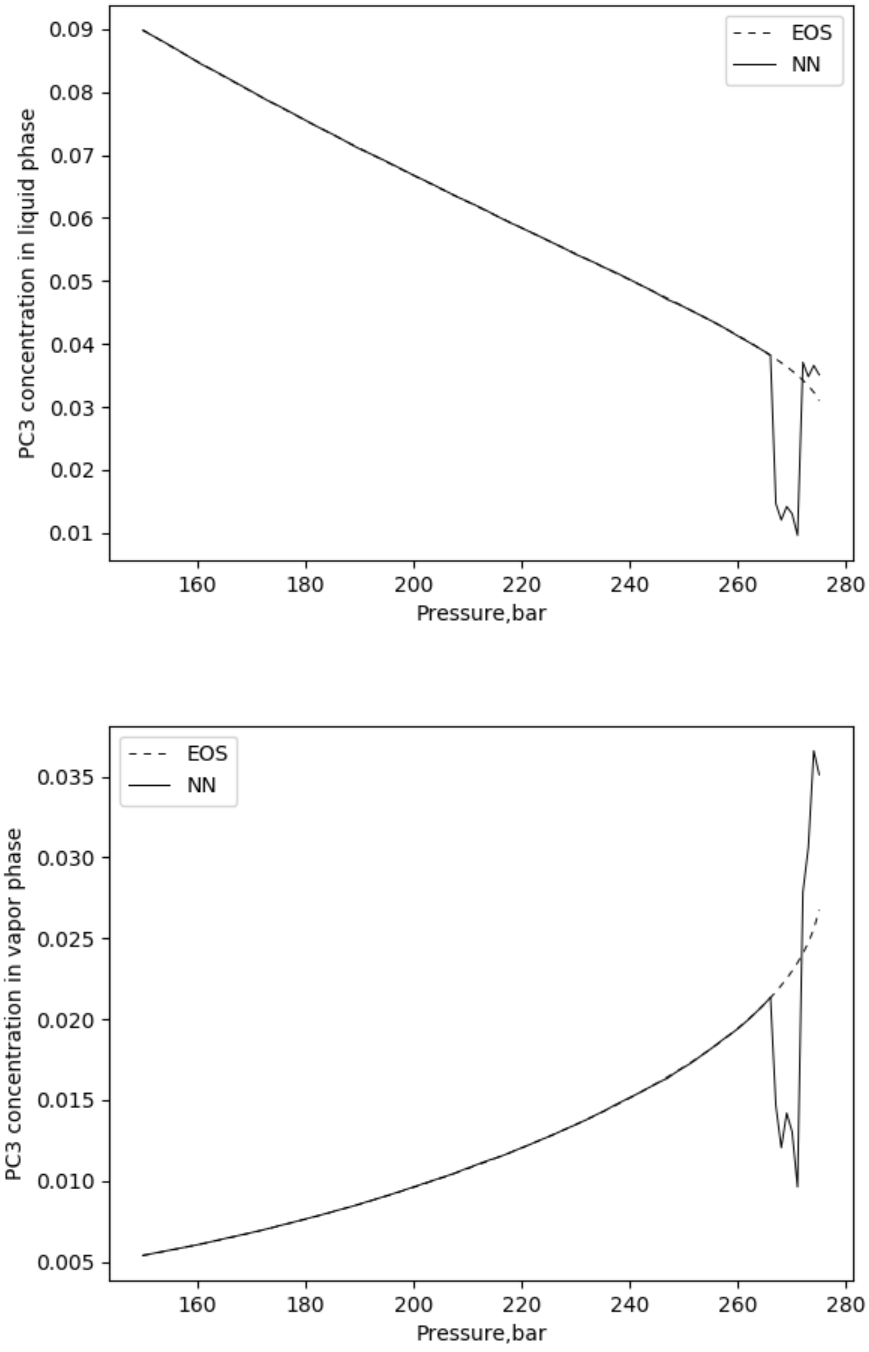


Figure 5.82 Phase mole fraction calculations comparison between EOS and ANN. Top: Liquid Pseudo Component 3. Bottom: Vapor Pseudo Component 3.

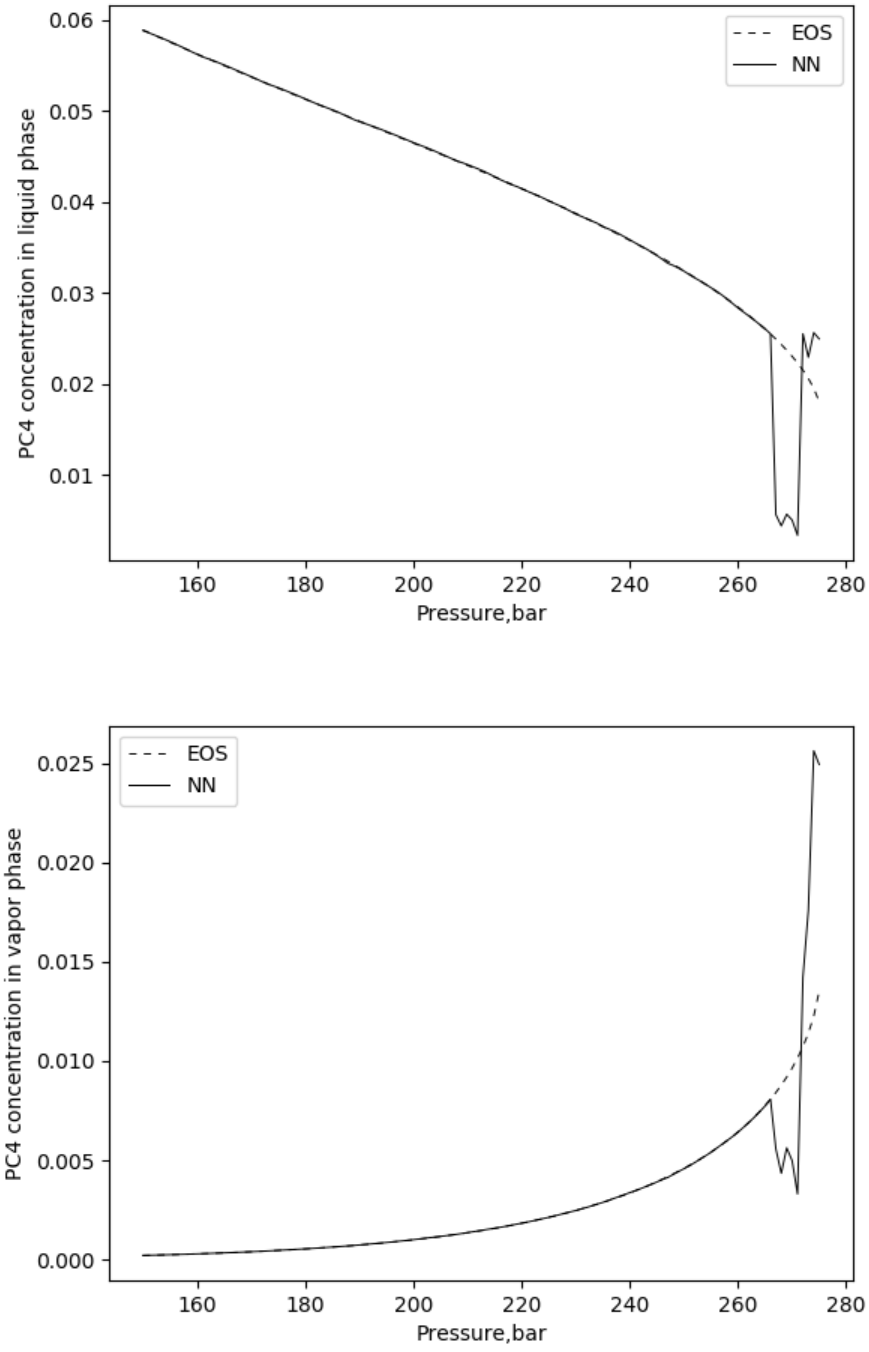


Figure 5.83 Phase mole fraction calculations comparison between EOS and ANN. Top: Liquid Pseudo Component 4. Bottom: Vapor Pseudo Component 4.

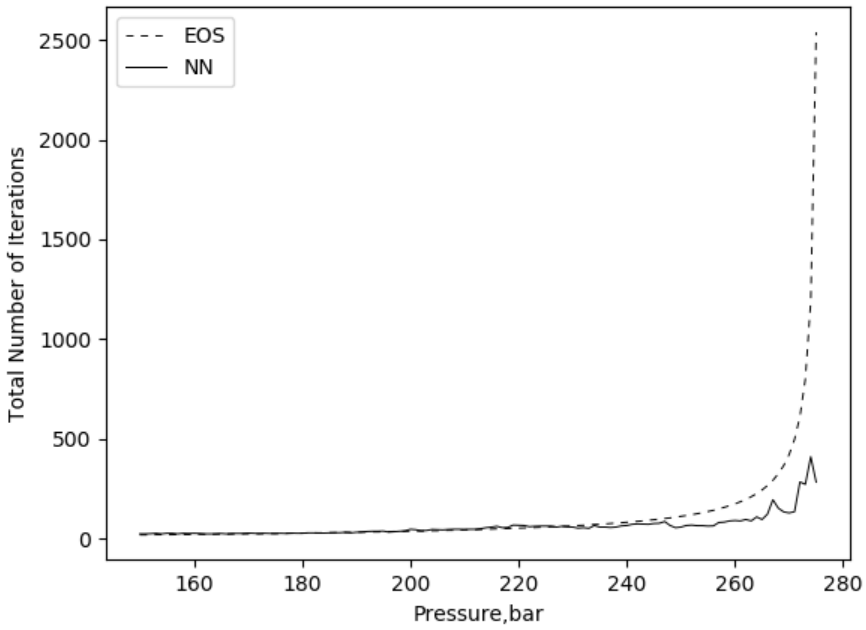


Figure 5.84 Total number of iteration comparison between EOS flash and ANN flash.

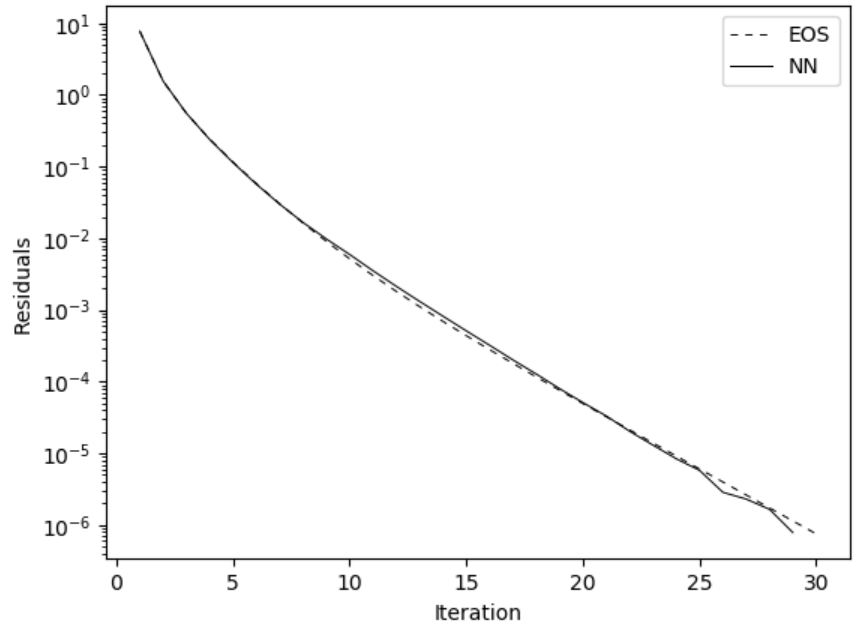


Figure 5.85 Convergence behavior at 180 bars (To correct solution).

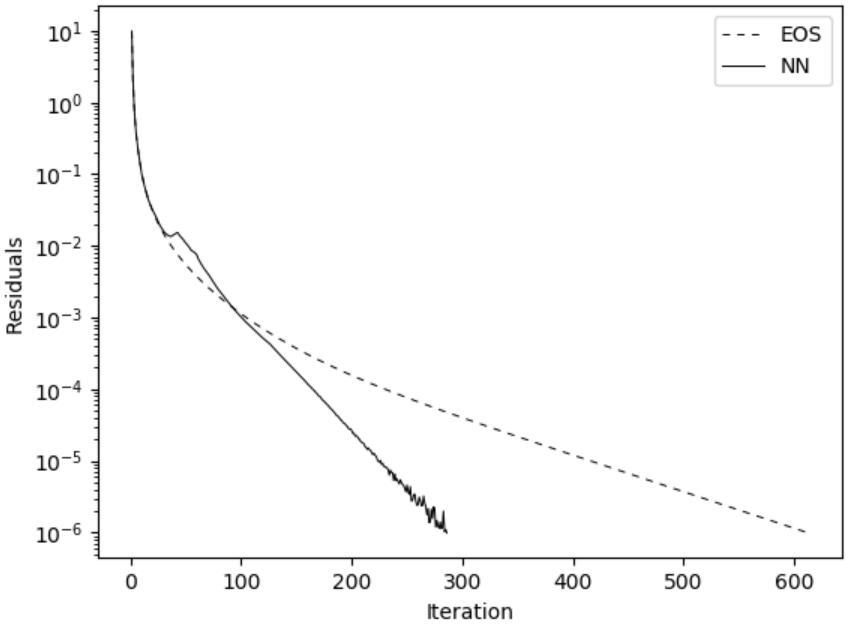


Figure 5.86 Convergence behavior at 272 bars (To incorrect solution).

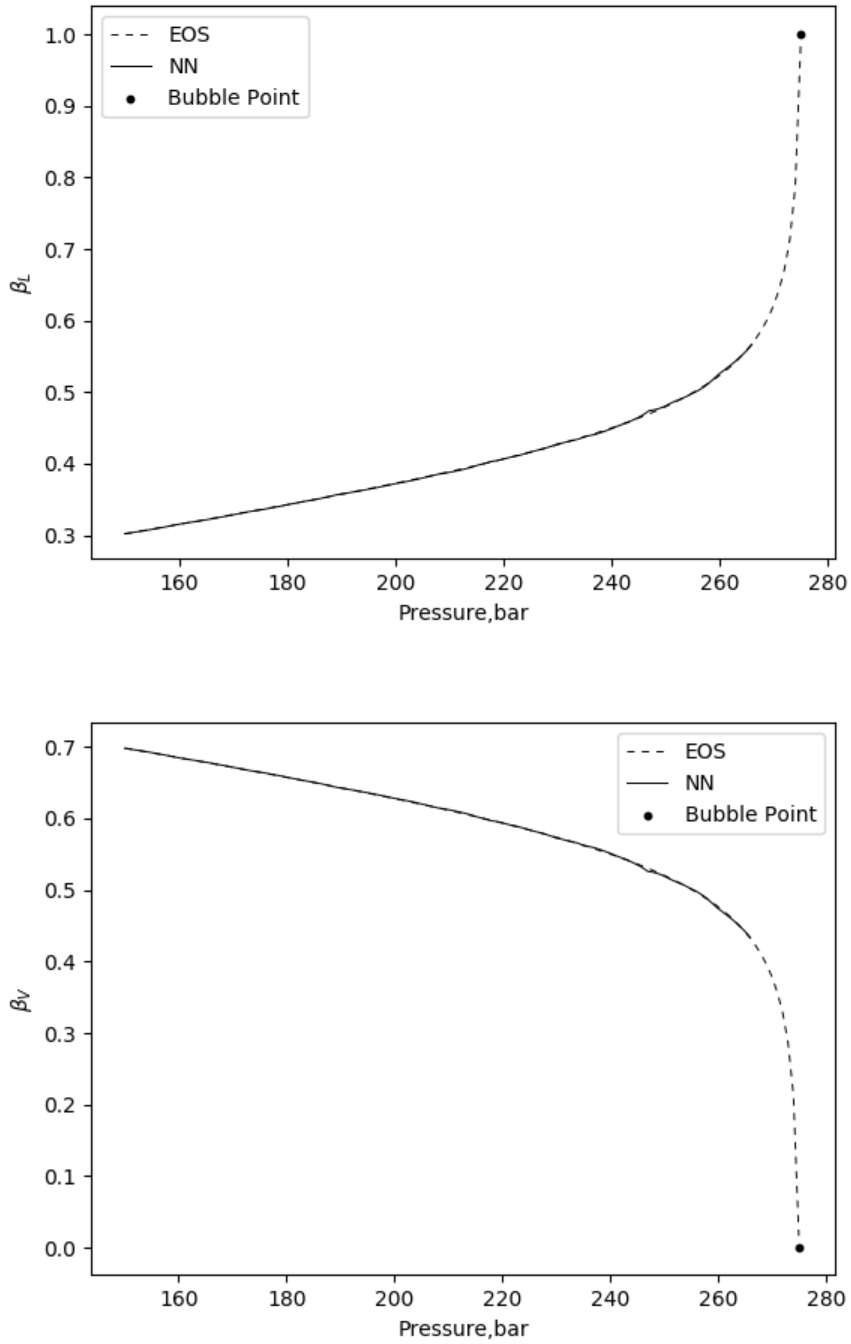


Figure 5.87 Phase mole fraction calculation comparison
Top: Liquid mole fraction. Bottom: Vapor mole fraction.

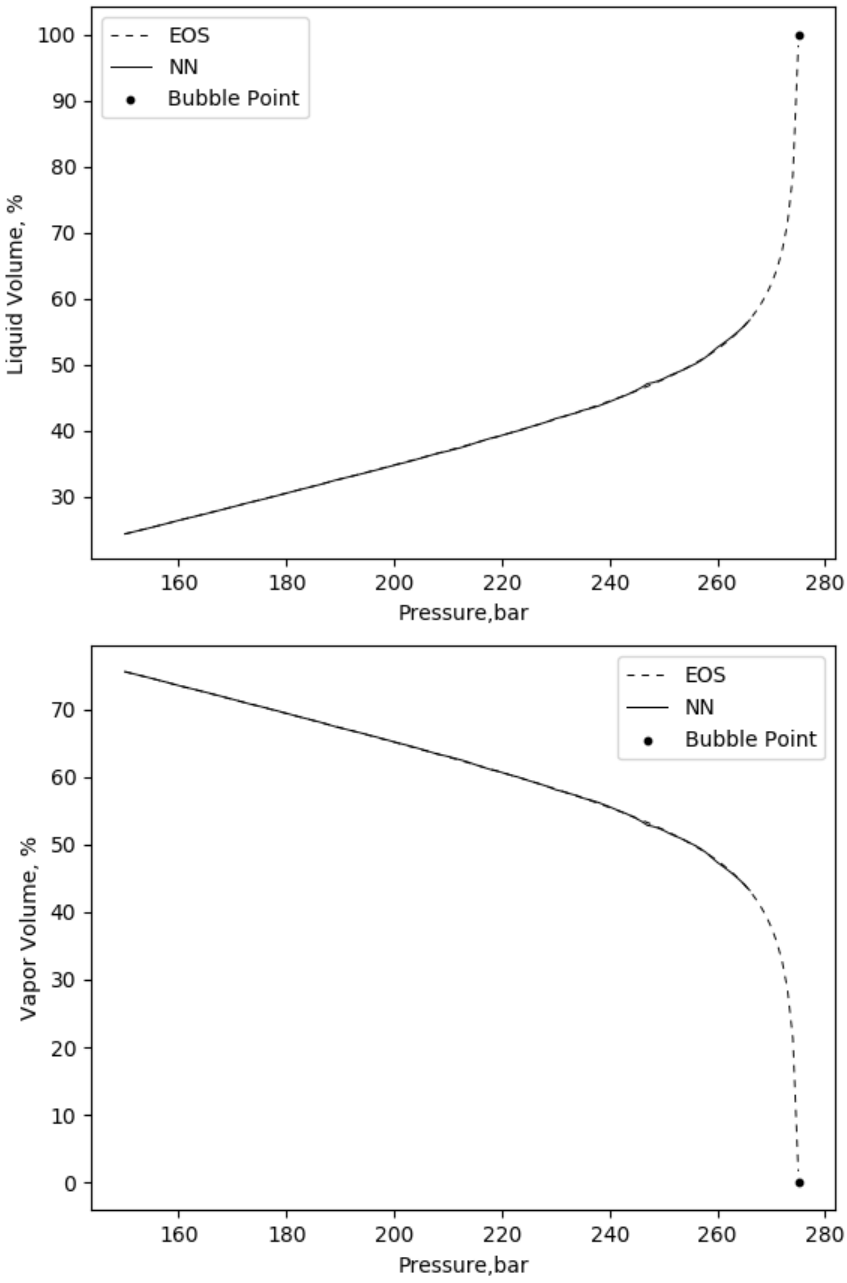


Figure 5.88 Liquid saturation comparison
Top: Liquid saturation. Bottom: Vapor saturation.

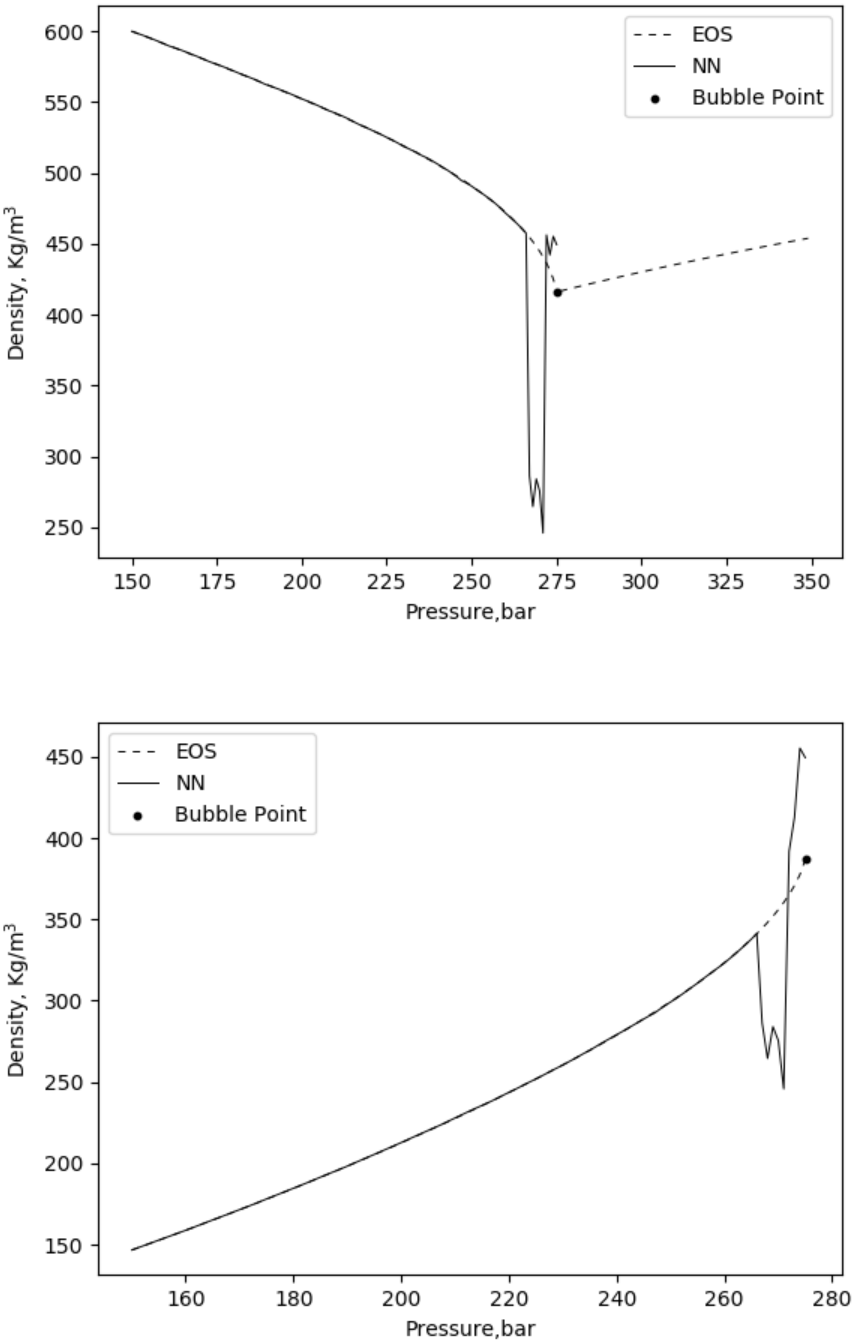


Figure 5.89 Fluid density calculation comparison between EOS and ANN. Top: Liquid density. Bottom: Vapor density.

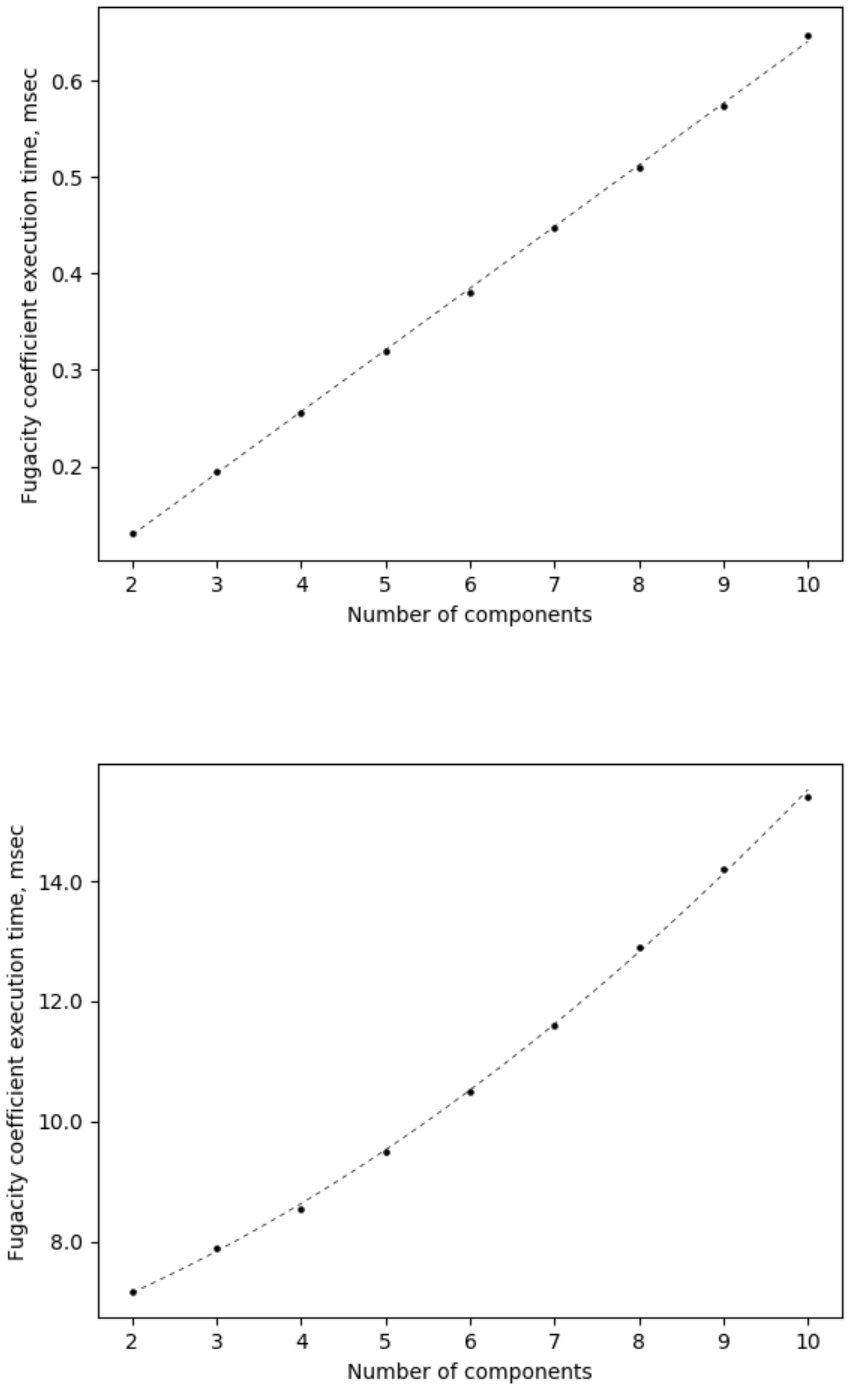


Figure 5.90 Fugacity coefficient execution time.
Top: ANN. Bottom: EOS.

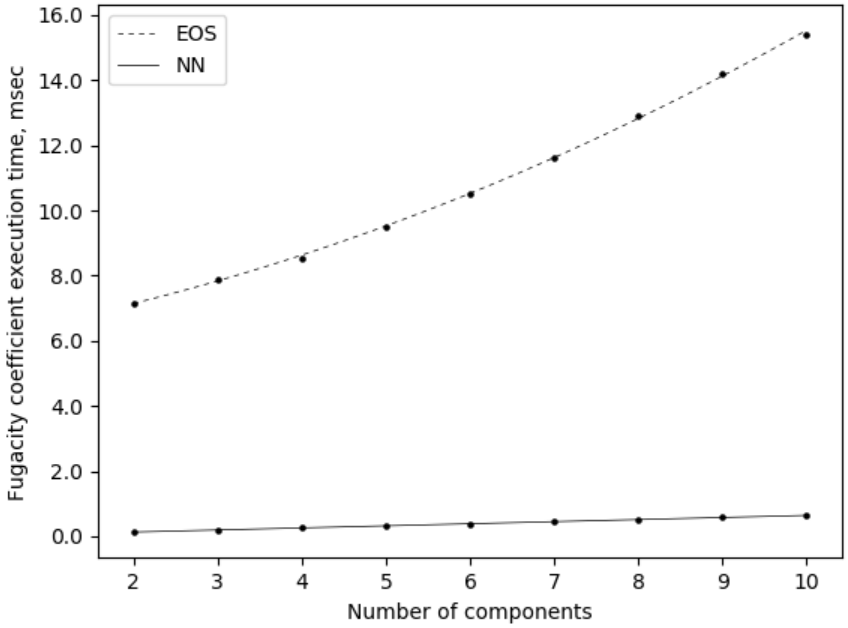


Figure 5.91 Fugacity coefficient execution time comparison between EOS and ANN.

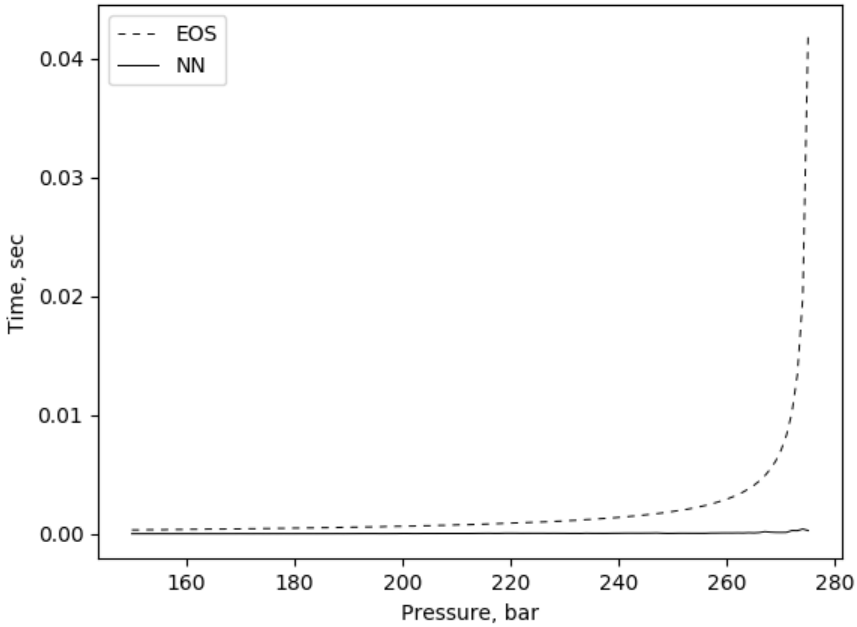


Figure 5.92 Total execution time to reach solution of conventional EOS method and ANN flash method.

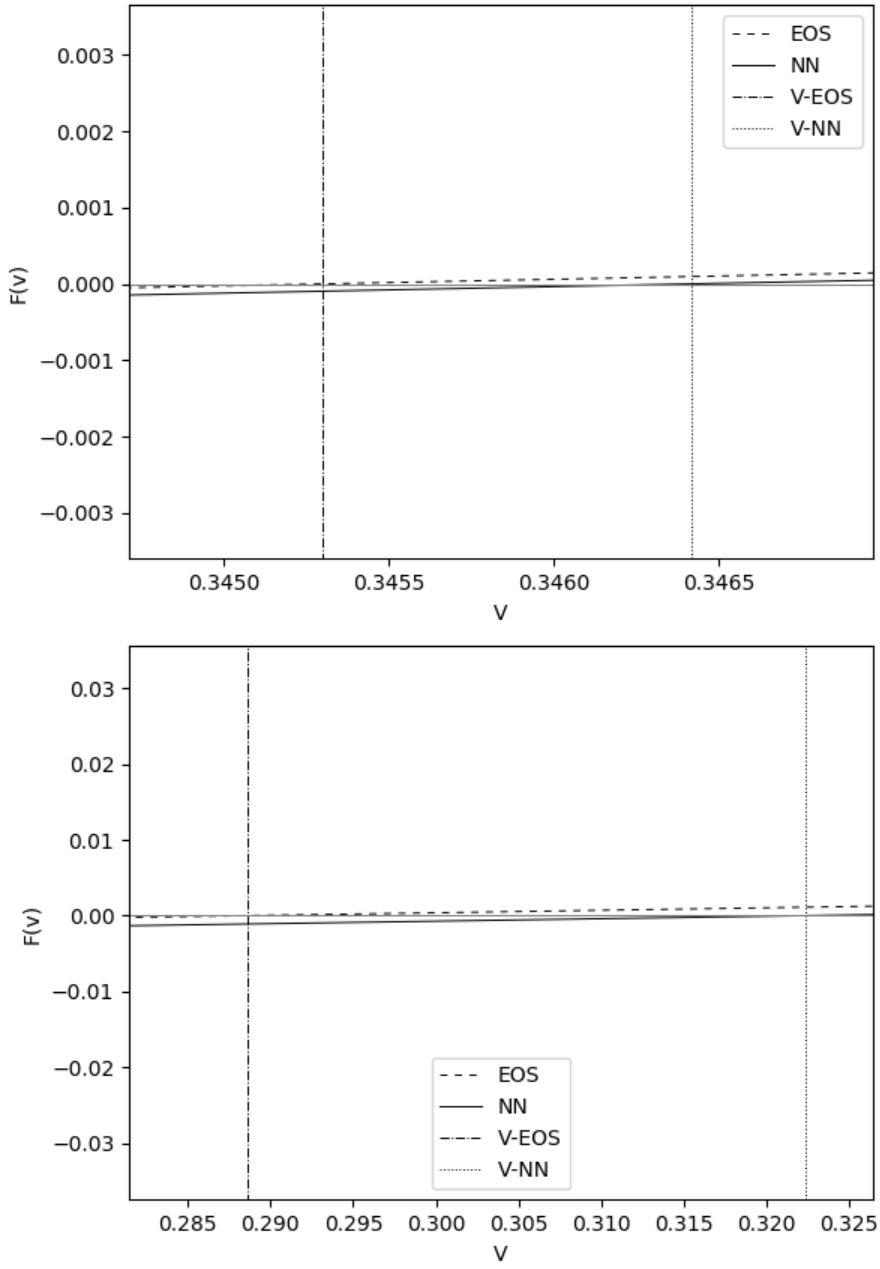


Figure 5.93 Solution of the RR function at 272 bars approaching the critical region Top: Iteration 8th Bottom: Iteration 32nd .

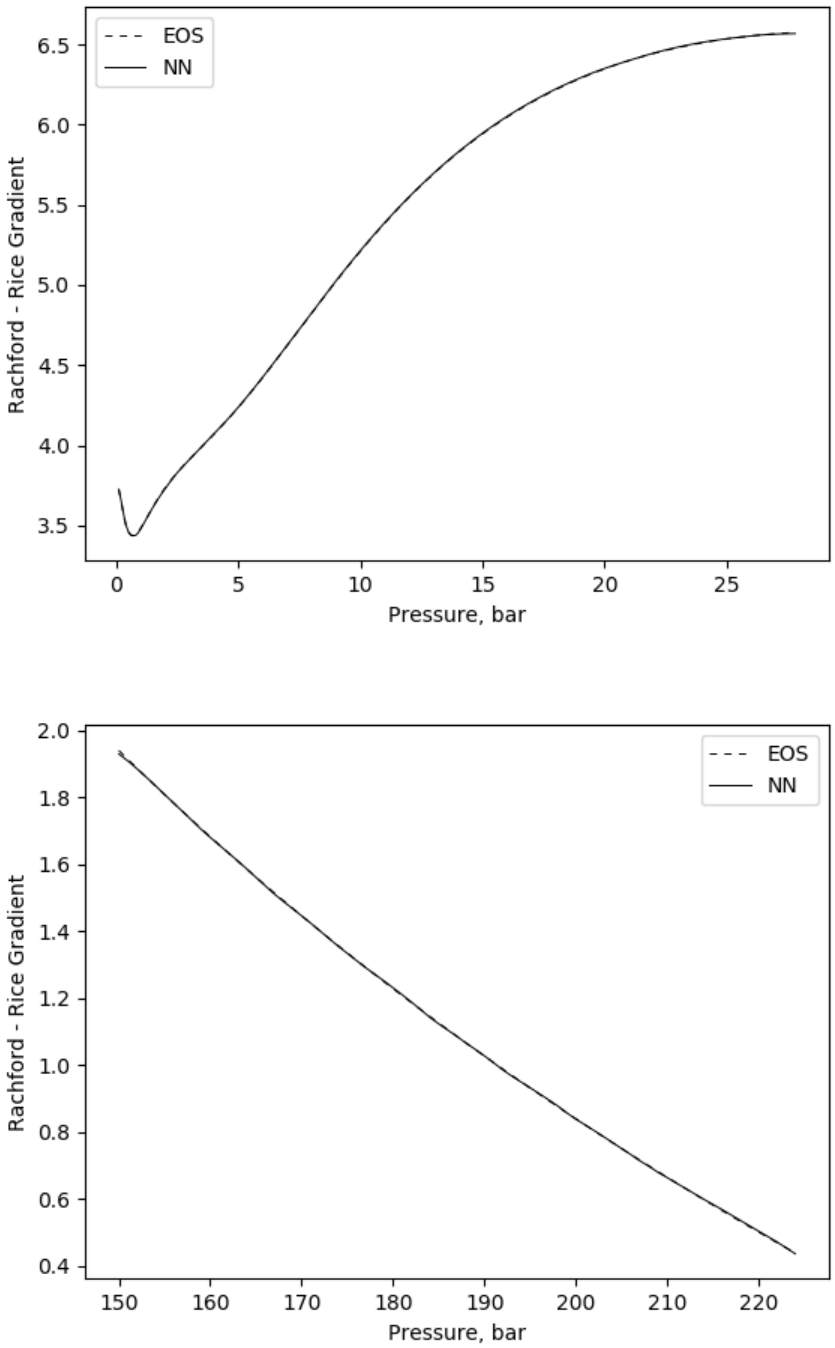


Figure 5.94 Gradient of the Rachford-Rice solution
Top: Case 1. Bottom: Case 2.

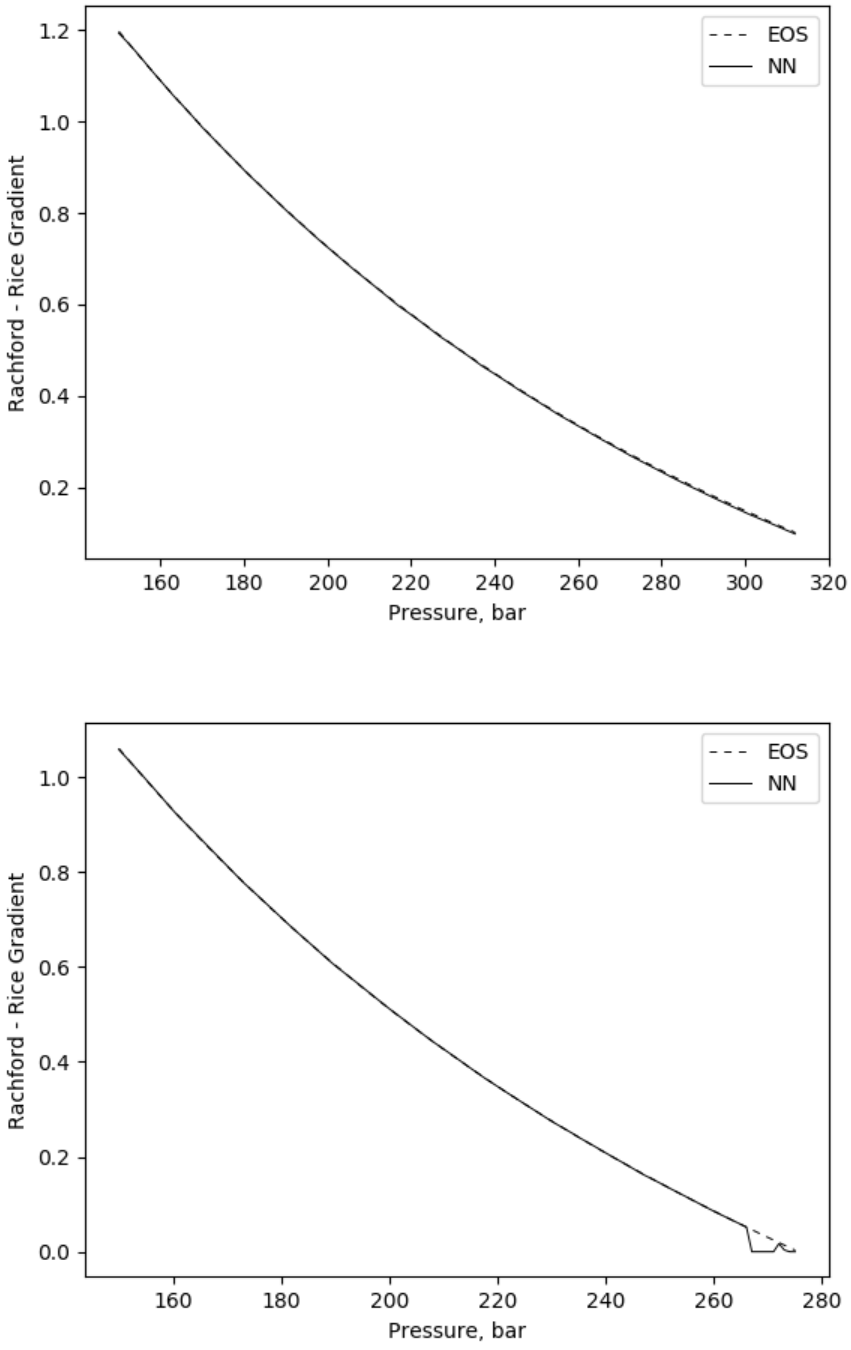


Figure 5.95 Gradient of the Rachford-Rice solution
Top: Case 3. Bottom: Case 4.

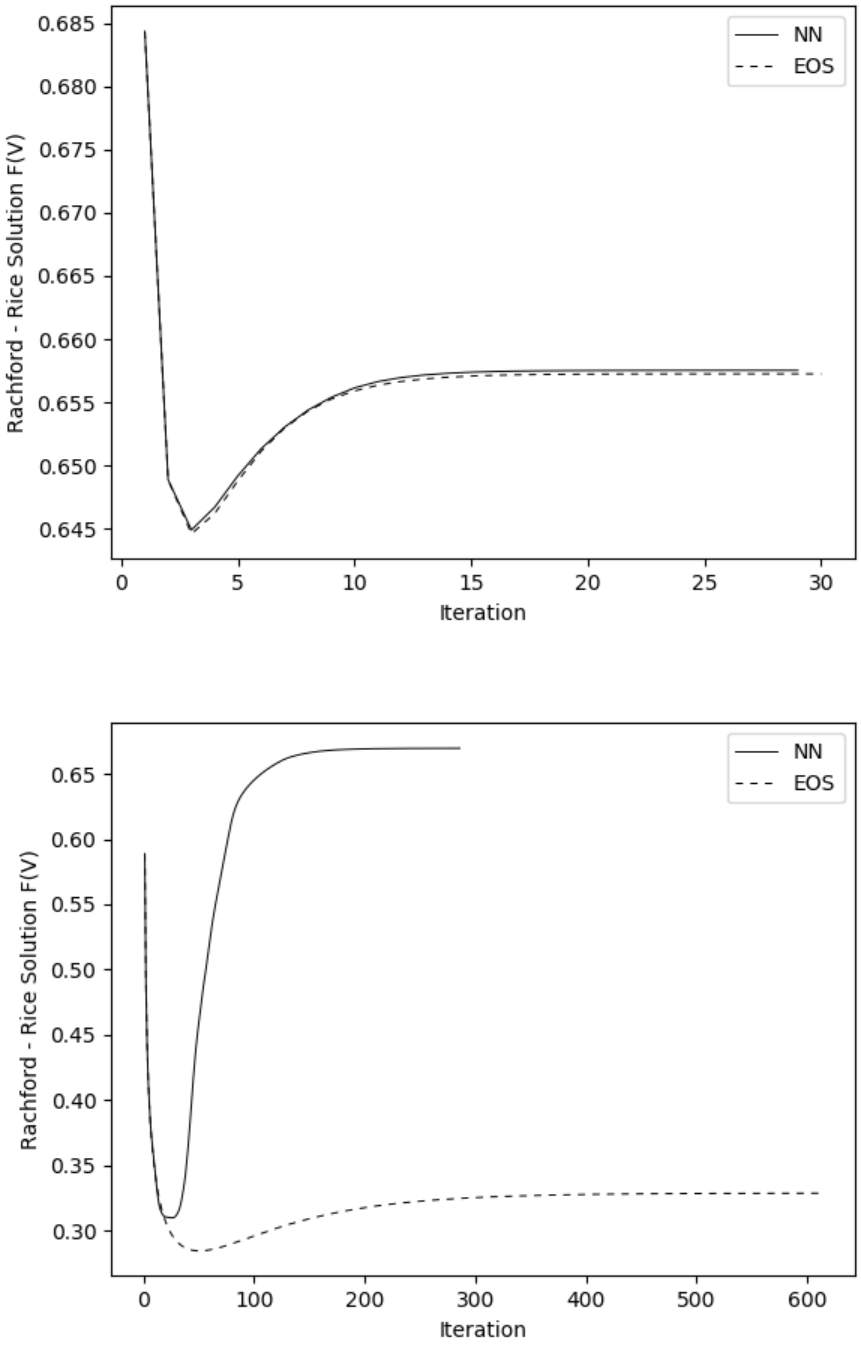


Figure 5.96 Rachford-Rice solution comparison between EOS and ANN flash Top: 180 Bar (away from critical point). Bottom: 272 Bar (Close to critical point).

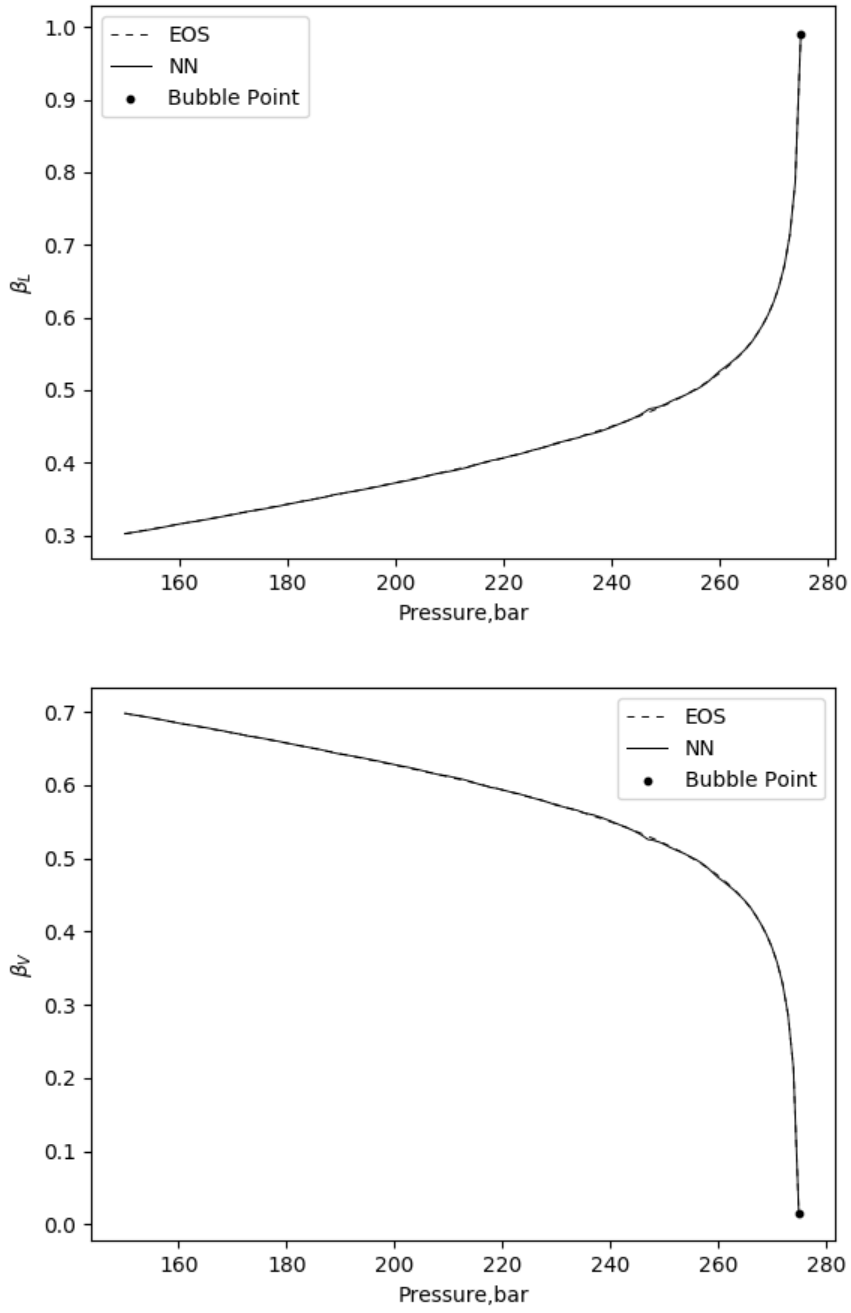


Figure 5.97 Phase amount calculation comparison between EOS and NN with switching criteria.
Top: Liquid density. Bottom: Vapor density.

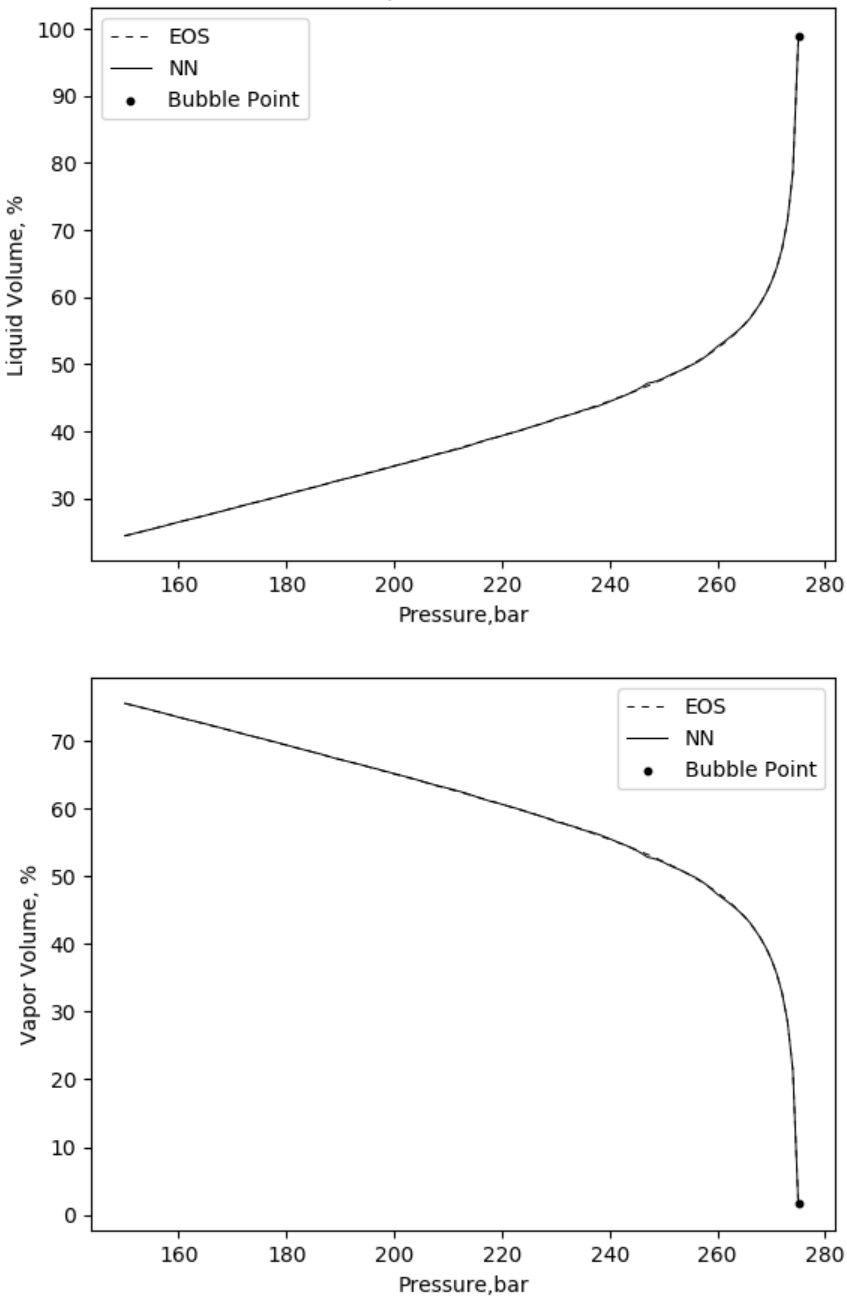


Figure 5.98 Fluid saturation calculation comparison between EOS and NN with switching criteria.
Top: Liquid density. Bottom: Vapor density.

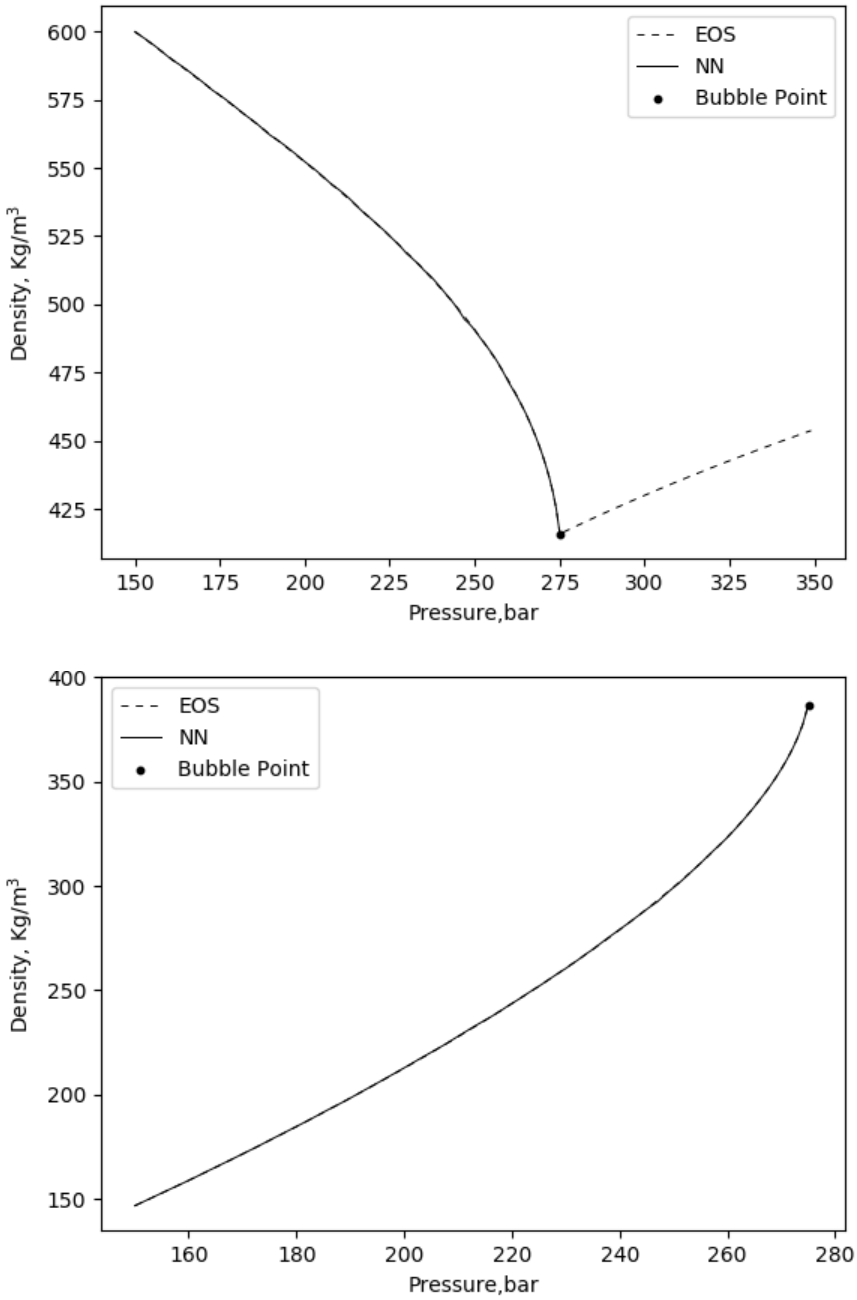


Figure 5.99 Fluid density calculation comparison between EOS and NN with switching criteria.
Top: Liquid density. Bottom: Vapor density.

CHAPTER 6

SUMMARY, CONCLUSIONS AND RECOMMENDATIONS FOR FUTURE RESEARCH

This chapter summarizes the objective of this work, the implementation of the new algorithm and the conclusions gathered from the study cases. Additionally, recommendations for future research are presented.

6.1 SUMMARY AND CONCLUSIONS

Compositional reservoir simulators are widely used in petroleum engineering. They are used to predict reservoir performance for different field development scenarios, to optimize well spacing and surface facilities locations, and to design and evaluate EOR projects. Compositional reservoir simulators must be able to predict the phase behavior of hydrocarbon-rich mixtures at different thermodynamic conditions. Phase behavior is calculated by the use of stability analysis and flash calculations. Stability analysis is to determine whether the hydrocarbon mixture is in a single phase or not. If instability of the fluid is detected, flash calculations are performed to calculate the phase mole fraction, composition, and fluid properties.

Flash calculations are performed at each reservoir grid-block and at each iteration step, making the EOS flash one of the most time-consuming calculations in compositional reservoir simulations. This limitation of compositional reservoir simulators restricts the application of refined reservoir grids to analyze in more detail the sweep efficiency of EOR techniques since the increased number of grid-blocks would increase the computational time significantly. Additionally, to reduce the computational time, reservoir fluid models are lumped into pseudo components to speed up the compositional simulation reducing the

accuracy of the phase behavior. For that reason, researchers have developed various methods to speed up EOS flash calculations in compositional simulation.

These methods include reductions methods, where the main idea is to reduce the number of nonlinear equations to be solved during flash calculations. Another method that aims to reduce the time for flash calculations is the use of look-up tables, where the flash calculation is performed as a pre-simulation step storing the results and then using them directly during reservoir simulations. Most recently, the development of ANNs to aid flash calculations has been proposed by several authors.

During flash calculations, the fugacity coefficient is used extensively to find the phase equilibrium. A key to obtaining an efficient flash calculation is to speed up the time-consuming fugacity coefficient calculation in flash calculations. In this thesis, ANNs were applied to make an accurate and efficient model for the fugacity coefficient for each component in a mixture. The ANN-based fluid model was used to solve for phase equilibrium conditions by use of the traditional successive substations method.

The replacement of the conventional EOS fugacity coefficient with the ANN-based one provides several advantages. First, there is no need to solve the cubic EOS for compressibility factor(s) during the iteration since the ANN fluid model gives the fugacity coefficients through a feedforward calculation. This also means that there is no need to evaluate the Gibbs free energy of different cubic roots when the solution of the EOS gives more than one root. The ANN-based fluid model is rapid in providing fugacity coefficients because the feedforward calculation is computationally efficient (additions and multiplications) in comparison to the EOS-based fugacity coefficient that contains logarithms, square roots, and exponentials.

The newly formulated flash calculation using an ANN-based fluid model was used for different reservoir fluids. The main conclusions are as follows:

1. Results showed that EOS-based fugacity coefficients can be accurately represented by ANNs for different reservoir fluids (e.g. up to 12 components, zero and non-zero BIPs, and non-hydrocarbon components, such as N_2 and CO_2, in the mixture).
2. The ANN-based fugacity coefficients were used to solve for phase compositions and amounts for different reservoir fluids. Flash calculations using the ANN-based fugacity coefficient model successfully converged, except for one fluid in the close vicinity of the critical point. The phase equilibrium conditions converged were essentially identical between EOS and ANN-based flash. In the near-critical case, the equilibrium solution was so sensitive to fugacity coefficients that the ANN-based flash converged to an inaccurate solution. This implementation issue was resolved by using a switching criterion to EOS flash as described in conclusion 5 below.
3. The ANN flash generally converged in fewer iterations in comparison with the conventional EOS. The reduced number of iterations also contributed to the algorithm efficiency.
4. The ANN flash provides a computational time reduction of 89.83% on average in comparison with the conventional EOS method as shown in the case studies.
5. ANN flash was robust for fluid models that are not in a critical region. Near a critical point, the ANN flash deviates from the correct solution when the gradient of the Rachford – Rice (RR) solution is extremely small, which makes the RR solution very sensitive to K values. To avoid this potential of inaccurate ANN flash in a near-critical region, the switching criterion from ANN flash to EOS flash was defined and applied by a gradient of the RR function, 0.05, below which the switching occurs. This switching from ANN to EOS flash does not waste the

calculations performed prior to the switching, and lets the iteration converge to the correct solution in the cases studied.

6. There are other applications of ANNs to flash calculations presented in the literature. However, this is the first time ANNs were applied to make simple and accurate models for the fugacity coefficients that speed up the iterative flash calculation.
7. Unlike reduced methods, the implementation and computational advantage of ANN-based flash are not affected by non-zero BIPs.
8. The execution time of ANN flash increases linearly with the number of components in the mixture, while the execution time of EOS flash increases quadratically with the number of components increases. That is, the advantage of ANN flash over EOS flash tends to be more pronounced for a larger number of components.

6.2 RECOMMENDATIONS FOR FUTURE RESEARCH

Recommendations for future research of ANNs for flash calculations in compositional reservoir simulations are as follows:

1. Standalone flash calculations evaluations are necessary but not sufficient for reservoir simulations applications. Generally, reservoir simulations require more robust methods for phase equilibrium calculations. Therefore, the application of ANN flash calculations in compositional reservoir simulation should be evaluated to verify its robustness and efficiency.
2. The scope of this work was to evaluate the application and performance of ANNs for flash calculations. However, stability analysis is an integral part of the phase equilibrium calculations. Therefore, the use of ANNs for the fugacity coefficient

should be evaluated in conjunction with stability analysis to create an integrated phase equilibrium algorithm based on ANNs

3. This work focused on two-phase flash calculations, but more than two phases can be present during compositional reservoir simulations. Therefore, the ANNs can be applied to more complex algorithms to look for more than two phases.
4. In this work, ANNs models were generated for each component in the mixture. This approach allows us to have more control over the accuracy and generalization error of the fugacity coefficient for each component in the mixture. However, other neural network architectures can be investigated to generate less artificial neural models and speed up the training process and the data preparation before implementation in flash calculations.

Appendix A. Database generation code

This code uses the fluid model described in case 1 and calculate the fugacity coefficient, determine the multiple concentration combinations for a given concentration increments and compute the fugacity coefficient database for a range of pressure, concentration and number of components. This code was developed in python language.

```
#Libraries
Import numpy as np


#Reduce pressure calculation
def Pred(P,Pc):
   Pr=np.divide(P,Pc)
   return Pr

#Reduced temperature calculation
def Tred(T,Tc):
   Tr=np.divide(T,Tc)
   return Tr

#Attraction parameter
def Aa(Pr,Tr,alph):
   A=np.divide((0.457235529)*(Pr)*alph,Tr**2)
   return A

#Covolume parameter
def Be(Pr,Tr):
   B=(0.077796074)*np.divide(Pr,Tr)
   return B


#Temperature dependent EOS parameter
def alpha(k,Tr):
   alph=(1+k*(1-np.sqrt(Tr)))**2
   return alph

#Accentric factor dependent parameter
def kk(Nc,af):
   for i in range(0,Nc):
```

```
        if af[i] <= 0.49:
            k=0.37464+1.54226*af-(0.26992)*(af**2.)
        elif af[i] > 0.49:
            k=0.379642+1.48503*af-0.164423*(af**2.)+(0.01666)*(af**3.)
    return k

#Peng robinson equation of state variables
def gammma(Amix,Bmix):
    gamma=np.multiply(-Amix,Bmix)+np.power(Bmix,2)+np.power(Bmix,3)
    return gamma

def betta(Bmix,Amix):
    beta=Amix-(3*np.power(Bmix,2))-np.multiply(2,Bmix)
    return beta


def phil(Bmix):
    phi2=-1.+Bmix
    return phi2

#Reduced attraction parameter
def Ami(X,Am,Nc):
    Amix=0.
    for i in range(0,Nc):
        for j in range(0,Nc):
            Amix+=X[i]*X[j]*Am[i,j]
    return  Amix

#Reduced covolume parameter
def Bm(X,B):
    Bmix2=np.sum(X*B)
    return Bmix2



#Van del Waals mixing rules
def Vw(Nc,A,bi):
    Am=np.zeros([Nc,Nc])
    for i in range(0,Nc):
        for j in range(0,Nc):
            Am[i,j]=np.sqrt(A[i]*A[j])*(1.-bi[i,j])
    return Am
```

```
#Terms of the fugacity coefficient equation for PR EOS
def fc(Amix,Bmix,z,Am,Nc,X,B):
    Term2=np.log(z-Bmix)
    Term3=np.divide(Amix,(2*np.sqrt(2)*Bmix))
    Term6=np.log(np.divide((z+(1+np.sqrt(2))*Bmix),(z+(1-np.sqrt(2))*Bmix)))
    mask=np.zeros((Nc,))
    for i in range(0,Nc):
        for j in range(0,Nc):
            mask[i]+=X[j]*Am[i,j]
    Term1=(B/Bmix)*(z-1)
    Term5=B/Bmix
    Term4=(2.*(mask))/Amix
    Fc=Term1- Term2-Term3*(Term4-Term5)*Term6
    return Fc

#Gibbs free energy root evaluation
def Dgg(X,FcL,FcV):
    Dg2=np.sum(X*(FcL-FcV))
    return Dg2

#Fugacity Coefficient calculation
def lnfug(af,T,Tc,P,Pc,Nc,bi,X):
    Tr=Tred(T,Tc)
    Pr=Pred(P,Pc)
    k=kk(Nc,af)
    alph=alpha(k,Tr)
    B=Be(Pr,Tr)
    A=Aa(Pr,Tr,alph)
    Am=Vw(Nc,A,bi)
    Bmix=Bm(X,B)
    Amix=Ami(X,Am,Nc)
    phi=phil(Bmix)
    beta=betta(Bmix,Amix)
    gamma=gammma(Amix,Bmix)
    z = Compressibility(phi,beta,gamma)
    RN= RootNumber(phi,beta,gamma)
    if RN ==2:
        Zv=np.max(z)
        Zl=np.min(z)
        FcL=fc(Amix,Bmix,Zl,Am,Nc)
        FcV=fc(Amix,Bmix,Zv,Am,Nc)
        Dg=Dgg(X,FcL,FcV)
        if Dg<0:
```

```
            z=Zl
        else:
            z=Zv
    lnfug=fc(Amix,Bmix,z,Am,Nc,X,B)
    return lnfug

#Cardano to solve cubic Equation of State
def Compressibility(A,B,C):
    D=(A/3)**3-(A*B/6)+(C/2)
    E=(B/3)-(A/3)**2
    Delta=(D**2)+(E**3)

    if Delta == 0:

        if D > 0:
            Z1 = 2*(-abs(-D)**(1/3))-(A/3)
            Z2 = -abs(-D)**(1/3)-(A/3)
            Z3 = -abs(-D)**(1/3)-(A/3)

        elif D <= 0:
            Z1=2*((-D)**(1/3))-(A/3)
            Z2=(-D)**(1/3)-(A/3)
            Z3=(-D)**(1/3)-(A/3)
            return Z1,Z2,Z3

    elif Delta > 0:
        global G
        if -D + (Delta)**(1/2) < 0:
          F= -(abs(-D + (Delta)**(1/2)))**(1/3)
        else:
            F = (-D + (Delta)**(1/2))**(1/3)

        if -D - (Delta)**(1/2) < 0:
            G= -(abs(-D - (Delta)**(1/2)))**(1/3)

        else:
            G = (-D - (Delta)**(1/2))**(1/3)

        Z = F + G-(A/3)
        return Z

    elif Delta < 0:
        TETA=np.acos(-D/np.sqrt(-E**3))
```

```
        Z1=2*np.sqrt(-E)*np.cos(TETA/3)-A/3
        Z2=2*np.sqrt(-E)*np.cos((TETA/3)+(2/3)*np.pi)-A/3
        Z3=2*np.sqrt(-E)*np.cos((TETA/3)+(4/3)*np.pi)-A/3
    return Z1,Z2,Z3

#Root number identification
def RootNumber(A,B,C):
    D=(A/3)**3-(A*B/6)+(C/2)
    E=(B/3)-(A/3)**2
    Delta=(D**2)+(E**3)
    if Delta == 0:
        RN=2
        return RN
    elif Delta > 0:
        RN=1
        return RN
    elif Delta < 0:
        RN=2
        return RN

#Mixture composition generation for material balance, N indicates the resolution
def Mixture(N):
    X1=np.linspace(0,1,N)
    X2=np.linspace(0,1,N)
    X3=np.linspace(0,1,N)
    X4=np.linspace(0,1,N)
    X5=np.linspace(0,1,N)
    X6=np.linspace(0,1,N)
    one=np.zeros(X1.size*X2.size*X3.size*X4.size*X5.size*X6.size)
    two=np.zeros(X1.size*X2.size*X3.size*X4.size*X5.size*X6.size)
    three=np.zeros(X1.size*X2.size*X3.size*X4.size*X5.size*X6.size)
    fourth=np.zeros(X1.size*X2.size*X3.size*X4.size*X5.size*X6.size)
    five=np.zeros(X1.size*X2.size*X3.size*X4.size*X5.size*X6.size)
    six=np.zeros(X1.size*X2.size*X3.size*X4.size*X5.size*X6.size)
    l=0
    for j in X1:
        for k in X2:
            for f in X3:
                for h in X4:
                    for o in X5:
                        for q in X6:
                            one[l]=j
                            two[l]=k
```

```
                three[l]=f
                fourth[l]=h
                five[l]=o
                six[l]=q
                l=l+1

    one=one.reshape(-1,1)
    two=two.reshape(-1,1)
    three=three.reshape(-1,1)
    fourth=fourth.reshape(-1,1)
    five=five.reshape(-1,1)
    six=six.reshape(-1,1)
    Summ=one+two+three+fourth+five+six
    Xarray=np.concatenate((one,two,three,fourth,five,six,Summ), axis=1)
    item = 1
    index =np.where(Summ==item)[0]
    Xarray1=Xarray[index]
    Mixcomp=np.delete(Xarray1, 6, axis=1)
    return Mixcomp

#Database generation
Pressure=np.arange(0,27.89,0.1)
Concentration=Mixture(0.01)
T=313.706
af=np.array([0.008,0.131,0.240,0.618,0.957,1.268])
Tc=np.array([190.6000,344.20556,463.222222,605.75,751.016667,942.4777778])
Nc=6
Pc=np.array([46.00173748,44.99234683,33.99591831,21.74878308,16.54049284,16.417
76638])
bi =
np.array([[0,0,0,0,0,0],[0,0,0,0,0,0],[0,0,0,0,0,0],[0,0,0,0,0,0],[0,0,0,0,0,0],[0,0,0,0,0,0]])
Fug=np.zeros(Pressure.size*Concentration[0].size)
l=0
for P in Pressure:
    for x in Concentration:
        Fug[l]=lnfug(af,T,Tc,P,Pc,Nc,bi,x)
        l=l+1
Data=np.concatenate((Pressure,Concentration,Fug), axis=1)
np.savetxt('Database.dat',Data)
```

Appendix B. Artificial Neural Network models

This code generates the ANNs for the fugacity coefficient using the database as described in Appendix A. The code generates the datasets for training, validation and testing, train the neural networks and evaluate its performance. Keras was used to train the neural network models with the backpropagation algorithm.

```
#Import libraries
import numpy as np
from keras.models import Sequential
from keras.layers import Dense
from keras.callbacks import ModelCheckpoint
import matplotlib.pyplot as plt
import random

#Load database
Data = np.loadtxt('Database.dat')
X= Data[:,[1,2,3,4,5,6]]
Pressure=[:,0].reshape(-1,1)
Min_P=np.min(Pressure)
Max_P=np.max(Pressure)

#Neural Networks for the fugacity coefficient
Nc=np.arange(7,12,1) #Number of components in the mixture
for i in Nc:
    N=6+i
    Fug= Data[:,N].reshape(-1,1) #Fugacity coefficient
    Min_Fug=np.min(Fug)
    Max_Fug=np.max(Fug)
    limits= np.array([[Min_Fug],[Max_Fug]])
    np.savetxt("Fuglimits"+str(i)+".txt",limits)

   #Randomly Database splitting for training, testing and validation
    test=np.array(random.sample(range(X.shape[0]),int(np.round(X.shape[0]*0.10))))
    P1=np.delete(Pressure,test,0)
    X1=np.delete(X,test,0)

    P_test=Pressure[test]
    X_test=X[test]

    Fugset1=np.delete(Fug,test,0)
```

```
    fug_test=Fug[test]


val=np.array(random.sample(range(Fugset1.shape[0]),int(np.round(Fugset1.shape[0]*0.1
0))))
    P_training=np.delete(P1,val,0)
    X_training=np.delete(X1,val,0)
    fug_training=np.delete(Fugset1,val,0)

    P_val=P1[val]
    X_val=X1[val]
    fug_val=Fugset1[val]

    plot=np.array(random.sample(range(Fug.shape[0]),1000))
    P_plot=Pressure[plot]
    X_plot=X[plot]
    fug_plot=Fug[plot]

    #Database normalization
    nP_training=np.divide((P_training-Min_P),(Max_P- Min_P))
    nP_test=np.divide((P_test-Min_P),(Max_P- Min_P))
    nP_val=np.divide((P_val-Min_P),(Max_P- Min_P))
    nP_plot=np.divide((P_plot-Min_P),(Max_P- Min_P))

    norm_data_training=np.concatenate((nP_training,X_training),axis=1)
    norm_data_test=np.concatenate((nP_test,X_test),axis=1)
    norm_data_val=np.concatenate((nP_val,X_val),axis=1)
    norm_data_plot=np.concatenate((nP_plot,X_plot),axis=1)

    #Label normalization
    nFug_training=np.divide((fug_training-Min_Fug),(Max_Fug- Min_Fug))
    nFug_test=np.divide((fug_test-Min_Fug),(Max_Fug- Min_Fug)).reshape(-1,1)
    nFug_val=np.divide((fug_val-Min_Fug),(Max_Fug- Min_Fug)).reshape(-1,1)
    nFug_plot=np.divide((fug_plot-Min_Fug),(Max_Fug- Min_Fug)).reshape(-1,1)

    #Neural network Keras assembly
    model=Sequential()
    model.add(Dense(20,activation = 'relu', use_bias=True, input_dim=7))
    model.add(Dense(20,activation = 'relu', use_bias=True))
    model.add(Dense(1))
    model.compile(optimizer='adam' ,loss='mse',metrics=['mae','mape'])
     #Model name
    filepath="Component_str(i)+".hdf5"
```

```
   checkpointer=ModelCheckpoint(filepath, monitor='val_loss', verbose=1,
save_best_only=True, save_weights_only=False, mode='min')
   #Neural Network training with backpropagation
history=model.fit(norm_data_training,nFug_training,epochs=20,batch_size=100,callback
s=[checkpointer],validation_data=(norm_data_val,nFug_val))

   # summarize history for loss
   plt.figure()
   plt.plot(history.history['loss'])
   plt.plot(history.history['val_loss'])
   plt.title('model loss')
   plt.ylabel('loss')
   plt.xlabel('epoch')
   plt.legend(['train', 'test'], loc='upper left')
   plt.show()

   #Model predicton on testing data
   from keras.models import load_model
   Best=load_model(filepath)
   standirized_fug=Best.predict(norm_data_plot,batch_size=1000).reshape(-1, 1)
   Fug_NN=(np.multiply(standirized_fug,(Max_Fug-Min_Fug))+Min_Fug).reshape(-1,1)
   Fug=Fug.reshape(-1, 1)
   plt.plot(fug_plot,fug_plot,color='Black',linewidth=1.0)
   plt.scatter(fug_plot,Fug_NN,s=5,color='gray')
   plt.title('Accuracy comparison C5')
   plt.xlabel('EOS Model')
   plt.ylabel('NN Model')

   #Model Evaluation
   standirized_fug_test=Best.predict(norm_data_test,batch_size=4346175).reshape(-1, 1)
   Fug_NN_test=(np.multiply(standirized_fug_test,(Max_Fug-
Min_Fug))+Min_Fug).reshape(-1,1)
   def mean_absolute_percentage_error(y_true, y_pred):
      y_true, y_pred = np.array(y_true), np.array(y_pred)
      return np.mean(np.abs((y_true - y_pred) / y_true)) * 100
   from sklearn.metrics import r2_score, mean_squared_error
   r2=r2_score(fug_test,Fug_NN_test)
   mape=mean_absolute_percentage_error(fug_test,Fug_NN_test)
   mse=mean_squared_error(fug_test,Fug_NN_test)
   metrics= np.array([[r2],[mape],[mse]])
   np.savetxt("Metrics"+str(i)+".txt",metrics)
```

Appendix C Artificial Neural Network Flash

This code uses ANNs to calculate the fugacity coefficient as described in chapter 4. The models used in this flash calculations correspond to the fluid model of case 1.

```
NN flash
import numpy as np
from keras.models import load_model
NN1=load_model(' Component1.hdf5')
NN2=load_model(' Component2.hdf5')
NN3=load_model('Component3.hdf5')
NN4=load_model('Component4.hdf5')
NN5=load_model('Component5.hdf5')
NN6=load_model('Component6.hdf5')

#Extract weights and bias from Keras

w1=[]
b1=[]
for layer in NN1.layers:
    weights= layer.get_weights()[0]
    bias=layer.get_weights()[1]
    w1.append(weights)
    b1.append(bias)

w2=[]
b2=[]
for layer in NN2.layers:
    weights= layer.get_weights()[0]
    bias=layer.get_weights()[1]
    w2.append(weights)
    b2.append(bias)




w3=[]
b3=[]
for layer in NN3.layers:
    weights= layer.get_weights()[0]
    bias=layer.get_weights()[1]
```

```
    w3.append(weights)
    b3.append(bias)

w4=[]
b4=[]
for layer in NN4.layers:
    weights= layer.get_weights()[0]
    bias=layer.get_weights()[1]
    w4.append(weights)
    b4.append(bias)

w5=[]
b5=[]
for layer in NN5.layers:
    weights= layer.get_weights()[0]
    bias=layer.get_weights()[1]
    w5.append(weights)
    b5.append(bias)

w6=[]
b6=[]
for layer in NN6.layers:
    weights= layer.get_weights()[0]
    bias=layer.get_weights()[1]
    w6.append(weights)
    b6.append(bias)

weights=[w1,w2,w3,w4,w5,w6]
bias=[b1,b2,b3,b4,b5,b6]

P=10
X=np.array([0.0917166666666667,0.155916666666667,0.172716666666667,0.3360166
66666667,0.166716666666667,0.0769166666666667])
T=313.706
af=np.array([0.008,0.131,0.240,0.618,0.957,1.268])
Tc=np.array([190.6000,344.20556,463.222222,605.75,751.016667,942.4777778])
Nc=6
Pc=np.array([46.00173748,44.99234683,33.99591831,21.74878308,16.54049284,16.417
76638])
Fugmax=np.array([0.8509604729089526,-0.035248585700825785,-
0.5206856081534699,-0.7108217921364027,-0.9097244029847029,-
1.127952250637442,-1.6578438405221156,-2.1894188142990103,-3.13814748798403,-
5.4719367618593715])
```

```
Fugmin=np.array([-0.060246650178461425,-0.49843491827529496,-
0.9844384647848845,-1.472792683003422,-2.0102446974914265,-
2.5768514689564306,-4.078493849514273,-5.561646380581826,-8.23884036197259,-
15.044247653802742])
Pmin=np.array([0.1])
Pmax=np.array([27.899000000000026])

def NNflash(P,T,Nc,X,af,Tc,Pc,weights,bias):
    Tr=Tred(T,Tc)
    Pr=Pred(P,Pc)
    Kw=will(Pr,af,Tr)
    Ress=1
    s=0
    while True:
        V=RR(Kw,X,Nc)
        XLiq=Xliq(X,V,Kw)
        Yvap=Yvapo(XLiq,Kw)
        Fugliq=lnfug(P,XLiq,weights,bias,Fugmax,Fugmin,Pmin,Pmax)
        Fugvap=lnfug(P,Yvap,weights,bias,Fugmax,Fugmin,Pmin,Pmax)
        Conv=conve(XLiq,Yvap,Fugliq,Fugvap)
        Ress=resi(Conv)
        Kw=kww(Fugliq,Fugvap)
        s=s+1
        if(Ress < 1e-6):
            break

def lnfug(P,XLiq,weights,bias,Fugmax,Fugmin,Pmin,Pmax):
    #Pressure normalization
    nP=np.divide((P-Pmin),(Pmax- Pmin))
    v=np.concatenate((nP,X),axis=0).reshape(1,-1)#Input data for neural network
    #Feedfoward calculation
    fugcoefficient=[]
    for i in range(0,Nc):
        w=weights[i]
        b=bias[i]
        f11 = np.dot(v,w[0])+b[0] #first layer
        f11a = np.maximum(f11, 0) #activation of first layer
        f12 = np.dot(f11a,w[1])+b[1] #second layer
        f12a=np.maximum(f12, 0) #activation second layer
        f13= np.dot(f12a,w[2])+b[2] #output layer
        fugcoeff= np.sum(f13) #normilized output
        lnfug=(np.multiply(fugcoeff,(Fugmax[i]-Fugmin[i]))+Fugmin[i]).reshape(-1,1) #Re-
scaled fugacity coefficient
```

```
        fugcoefficient.append(lnfug)
    return np.array(fugcoefficient).reshape(-1,1)

def will(Pr,af,Tr):
 Kw2=np.multiply(np.divide(1,Pr),np.exp((np.multiply(np.multiply(5.373,(1+af)),(1-
(1/Tr))))))
 return Kw2


def Pred(P,Pc):
    Pr=np.divide(P,Pc)
    return Pr


def Tred(T,Tc):
    Tr=np.divide(T,Tc)
    return Tr


def RR(Kw,X,Nc):
    Kmax=1/(1-np.max(Kw))
    Kmin=1/(1-np.min(Kw))
    V=np.divide((Kmax+Kmin),2)
    Err=1
    while True:
        fv = 0
        dfv = 0
        for i in range(0,Nc):
            fv = fv+(1-Kw[i])*(X[i])/(1-(1-Kw[i])*V)
            dfv=dfv+(Kw[i]-1)**2*(X[i])/((Kw[i]-1)*V+1)**2
        Vnew=V-(fv/dfv)
        Err=abs(fv)
        V=Vnew
        if(Err < 1e-6):
            break
    return V


def Xliq(X,V,Kw):
    XLiq2=np.divide(X,(V*Kw+(1-V)))
    return XLiq2
```

```
def Yvapo(XLiq,Kw):
    Yvap2=np.multiply(XLiq,Kw)
    return Yvap2


def conve(XLiq,Yvap,Fugliq,Fugvap):
    Conv=np.log(XLiq)-np.log(Yvap)+Fugliq-Fugvap
    return Conv


def resi(Conv):
    Ress=np.linalg.norm(Conv)
    return Ress


def kww(Fugliq,Fugvap):
    Kw2=np.exp(Fugliq-Fugvap)
    return Kw2
```

Glossary

Roman symbols

a	Attraction parameter for cubic EOS
A	Dimensionless attraction parameter for cubic EOS
b	Covolume parameter for cubic EOS or bias term
B	Dimensionless covolume parameter for cubic EOS
f_{ij}	Fugacity of component *i* in phase *j*
G	Gibbs free energy
$\underline{G}$	Molar Gibbs free energy
$\bar{G}_{ij}$	Partial Gibbs free energy of component *i* in phase *j*
g_i	Activation functions in first hidden layer
h_i	Activation function in second hidden layer
J	Loss function
k_{ij}	Binary interaction coefficient
K_{ij}	K-value of component *i* in phase *j*
Nc	Number of components
Np	Number of phases
Nv	Number of inputs in neural network
Nn	Number of neurons in hidden layer
N_i	Number of inputs of a neuron
P	Pressure
P_c	Critical pressure
R	Gas constant
$\bar{S}_i$	Partial molar entropy of component *i*
T	Temperature

T_c	Critical temperature
V	Volume or vapor phase mole fraction
$\bar{V}_i$	partial molar volume of component *i*
w_{ij}	Weight value of input *i* in neuron *j*
x_{ij}	Mole fraction of component *i* in phase *j* or output of input *i* in neuron *j*
$\overline{x_i}$	Normalized feature
x_i	Original feature
y_i	Mole fraction of component *i* vapor phase
y_i	Target value at instance i
$\overline{y}_i$	Predicted value at instance i
z_i	Mole fraction of component *i* in a mixture
Z_j	Compressibility factor of phase j

Greek letters

α	Output of a neuron
β_j	Mole fraction of phase *j*
φ_{ij}	Fugacity coefficient of component of component *i* in phase *j*
υ	net stimuli of a neuron
μ	Step size in the minimization problem.
$\alpha(T)$	Temperature dependent parameter in equation of state
ω	Acentric factor
ε	Parameter to express equation of state in cubic form
β	Parameter to express equation of state in cubic form
γ	Parameter to express equation of state in cubic form

Superscripts

IGM	Ideal gas mixture
IG	Ideal gas
k	Iteration step
ℓ	Layer number
L	Output layer

Subscripts

C	Critical property
max	Maximum
mix	Minimum

Abbreviations

BIP	Binary Interaction Parameter
EOS	Equation of State
EOR	Enhanced Oil Recovery
MSE	Mean square error
MAE	Mean absolute error
msec	Microseconds: Second to the 10^{-6}
NN	Neural Network
RR	Rachford – Rice equation
UTCOM	IMPEC multiphase reservoir simulation developed at the University of Texas at Austin
FNN	Feed Forward Neural Network

References

Al-Meshari, A. 2004. New Strategic Method to Tune Equation of State to Match Experimental Data for Compositional Simulation. PhD Thesis. Texas A&M University, Texas. USA.

Aggarwal, C. C. (2018). Neural networks and deep learning. *Cham: Springer International Publishing*.

Baker, L. E., Pierce, A. C., & Luks, K. D. (1982). Gibbs energy analysis of phase equilibria. *Society of Petroleum Engineers Journal*, *22*(05), 731-742.

Evelein, K. A., Moore, R. G., & Heidemann, R. A. (1976). Correlation of the phase behavior in the systems hydrogen sulfide-water and carbon dioxide-water. *Industrial & Engineering Chemistry Process Design and Development*, *15*(3), 423-428.

Gaganis, V., & Varotsis, N. (2013). An improved BIP matrix decomposition method for reduced flash calculations. *Fluid Phase Equilibria*, *340*, 63-76.

Gaganis, V., & Varotsis, N. (2012, January). Machine learning methods to speed up compositional reservoir simulation. In *SPE Europec/EAGE Annual Conference*. Society of Petroleum Engineers.

Gaganis, V., & Varotsis, N. (2014). An integrated approach for rapid phase behavior calculations in compositional modeling. *Journal of Petroleum Science and Engineering*, *118*, 74-87.

Goodfellow, I., Bengio, Y., & Courville, A. (2016). *Deep learning*. MIT press.

Gulli, A., & Pal, S. (2017). Deep Learning with Keras. Packt Publishing Ltd.

Haykin, S. (1994). *Neural networks: a comprehensive foundation*. Prentice Hall PTR.

Hendriks, E. M. (1988). Reduction theorem for phase equilibrium problems. *Industrial & engineering chemistry research*, *27*(9), 1728-1732.

Hendriks, E. M., & Van Bergen, A. R. D. (1992). Application of a reduction method to phase equilibria calculations. *Fluid Phase Equilibria*, *74*, 17-34.

Khan, S.A., Pope, G.A., and Sepehrnoori, K. (1992). Fluid Characterization of Three-Phase CO2/Oil Mixtures. Paper presented at the SPE/DOE Enhanced Oil Recovery Symposium, Tulsa, Oklahoma, 22-24 April.

Kashinath, A., Szulczewski, M. L., & Dogru, A. H. (2018). A fast algorithm for calculating isothermal phase behavior using machine learning. *Fluid Phase Equilibria*, *465*, 73-82.

Kaul, P., & Thrasher, R. L. (1996). A parameter-based approach for two-phase-equilibrium prediction with cubic equations of state. *SPE Reservoir Engineering*, *11*(04), 273-279.

Kumar, A. 2016. Characterization of Reservoir Fluids based on Perturbation on n-Alkanes. PhD dissertation, University of Alberta, Edmonton, Alberta, Canada.

Kumar, A. and Okuno, R. (2016). A new algorithm for multiphase fluid characterization for solvent injection. *SPE Journal,* 21(05), 1688-1704.

Keras version 2.2.4, François Chollet, MIT

Jensen, B.H. and Fredenslund, A. 1987. A Simplified Flash Procedure for Multicomponent Mixtures Containing Hydrocarbons and One Non-Hydrocarbon Using Two-Parameter Cubic Equations of State. Industrial and Engineering Chemistry Research 26(10): 2129-2134.

Li, Y., & Johns, R. T. (2006). Rapid flash calculations for compositional simulation. *SPE Reservoir Evaluation & Engineering*, *9*(05), 521-529.

Michelsen, M. L. (1982b). The isothermal flash problem. Part II. Phase-split calculation. *Fluid phase equilibria*, *9*(1), 21-40.

Michelsen, M. L. (1986). Simplified flash calculations for cubic equations of state. *Industrial & Engineering Chemistry Process Design and Development*, *25*(1), 184-188.

Nichita, D. V., & Minescu, F. (2004). Efficient phase equilibrium calculation in a reduced flash context. *The Canadian Journal of Chemical Engineering*, *82*(6), 1225-1238.

Nichita, D. V., Broseta, D., & de Hemptinne, J. C. (2006). Multiphase equilibrium calculation using reduced variables. *Fluid phase equilibria*, *246*(1-2), 15-27.

Nichita, D. V., & Graciaa, A. (2011). A new reduction method for phase equilibrium calculations. *Fluid Phase Equilibria*, *302*(1-2), 226-233.

Okuno, R., Johns, R. T., & Sepehrnoori, K. (2010a). Three-phase flash in compositional simulation using a reduced method. *SPE Journal*, *15*(03), 689-703.

Okuno, R., Johns, R. T., & Sepehrnoori, K. (2010b). Application of a reduced method in compositional simulation. *SPE Journal*, *15*(01), 39-49.

Okuno, R., Johns, R.T., and Sepehrnoori, K. (2010c). A new algorithm for Rachford-Rice for multiphase compositional simulation. *SPE Journal,* 15(02), 313-325.

Okuno, R., 2009. Modeling of phase behavior for gas flooding simulation. PhD dissertation, the University of Texas at Austin, Austin, Texas, U.S.A.

Okuno, R., 2017. Class notes for PGE 384 "Advanced Thermodynamics in Petroleum Engineering", Department of Petroleum and Geosystems Engineering, The University of Texas at Austin, Austin, Texas, USA.

Pan, H. and Firoozabadi, A. 2003. Fast and Robust Algorithm for Compositiona Modeling: Part II-Two-Phase Flash Computations. SPE Journal 8(4): 380-391.

Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., ... & Vanderplas, J. (2011). Scikit-learn: Machine learning in Python. *Journal of machine learning research*, *12*(Oct), 2825-2830.

Perschke, D. R., Chang, Y., Pope, G. A., & Sepehrnoori, K. (1989). Comparison of phase behavior algorithms for an equation-of-state compositional simulator.

PVTsim version 17.3.0 Calsep A/S, Lyngby, Denmark.

Priddy, K. L., & Keller, P. E. (2005). Artificial neural networks: an introduction (Vol. 68). SPIE press.

Pao, Y. (1989). Adaptive pattern recognition and neural networks.

Rachford Jr, H. H., & Rice, J. D. (1952). Procedure for use of electronic digital computers in calculating flash vaporization hydrocarbon equilibrium. *Journal of Petroleum Technology*, *4*(10), 19-3.

Robinson, D. B., & Peng, D. Y. (1978). *The characterization of the heptanes and heavier fractions for the GPA Peng-Robinson programs*. Gas processors association.

Rummelhart, D. E., McClelland, J. L., & PDP Research Group. (1986). Parallel distributed processing.

Sandler, S. I., & Sandler, S. I. (2006). Chemical, biochemical, and engineering thermodynamics.

Van der Waals, JD (1873). *About the Continuity of the Gas and Liquid State* (Vol. 1). Sijthoff.

Voskov, D., & Tchelepi, H. A. (2007, January). Compositional space parameterization for flow simulation. In *SPE Reservoir Simulation Symposium*. Society of Petroleum Engineers.

Whitson, C. H., & Brulé, M. R. (2000). *Phase behavior* (Vol. 20). Richardson, TX: Henry L. Doherty Memorial Fund of AIME, Society of Petroleum Engineers.

Wang, K., Luo, J., Yan, L., Wei, Y., Wu, K., Li, J., ... & Chen, Z. (2019a, March). Artificial Neural Network Accelerated Flash Calculation for Compositional Simulations. In *SPE Reservoir Simulation Conference*. Society of Petroleum Engineers.

Wang, S., Sobecki, N., Ding, D., Wu, Y.-S., & Zhu, L. (2019b, March 29). Accelerated Compositional Simulation of Tight Oil and Shale Gas Reservoirs Using Proxy Flash Calculation. Society of Petroleum Engineers. doi:10.2118/193878-MS